

Harnessing Neural Dynamics as a Computational Resource

by

Andreas Stöckel

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2021

© Andreas Stöckel 2021

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner	Wulfram Gerstner Professor, School of Life Sciences and School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne
Supervisor	Chris Eliasmith Professor, Department of Philosophy and Department of Systems Design Engineering, University of Waterloo
Internal Member	Robin Cohen Professor, Department of Computer Science, University of Waterloo
Internal Member	Jeff Orchard Professor, Department of Computer Science, University of Waterloo
Internal-External Member	Sue Ann Campbell Professor, Department of Applied Mathematics, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Andreas Stöckel is the sole author of Chapters 1, 4, and 6. These chapters were either not written for publication or are based on self-published technical reports. The remaining chapters are, in heavily edited and extended form, based on the following published works:

- [1] Andreas Stöckel and Chris Eliasmith (2021). “Passive Nonlinear Dendritic Interactions as a Computational Resource in Spiking Neural Networks”. In: *Neural Computation* 33.1, pp. 96–128. DOI: 10.1162/neco_a_01338
- [2] Andreas Stöckel, Terrence C. Stewart, and Chris Eliasmith (2021). “Connecting Biological Detail with Neural Computation: Application to the Cerebellar Granule-Golgi Microcircuit”. In: *Topics in Cognitive Science* 13.3, pp. 515–533. DOI: 10.1111/tops.12536

Use of these publications in this thesis is with permission of the publishers. All figures in this thesis, with the sole exception of Figure 2.17, were created by the author. Annotations such as “inspired by” or “adapted from” merely indicate the source of the presented information.

Contributions to the individual publications.

- [1] This research was conducted by Andreas Stöckel. Chris Eliasmith supervised this work and helped designing the experiments. Andreas Stöckel wrote all the underlying code, conducted the experiments, created the figures, and wrote the manuscript. Chris Eliasmith edited the manuscript.
- [2] This research was conducted by Andreas Stöckel and Terrence C. Stewart. Andreas Stöckel was responsible for conducting the experiments, contributing the underlying theoretical work, writing, creating all figures, and conducting the literature review. Terrence C. Stewart helped with designing the final network, exploring the parameter space, and writing the paper. Chris Eliasmith supervised this work, provided helpful feedback, and edited the manuscript.

Use of these publications in this thesis.

Chapter 2: Small parts of this chapter are based on [1], in particular the description of the Neural Engineering Framework in Sections 2.3.3-2.3.5.

Chapter 3: The introduction of Chapter 3, and Sections 3.2-3.4, 3.6 are in part based on [1]. The experiments in Section 3.3 and 3.4 and the corresponding figures are, with minor changes, as previously published.

Chapter 4: This chapter is in large parts based on [2].

Abstract

Researchers study nervous systems at levels of scale spanning several orders of magnitude, both in terms of time and space. While some parts of the brain are well understood at specific levels of description, there are few overarching theories that systematically bridge low-level mechanism and high-level function. The Neural Engineering Framework (NEF) is an attempt at providing such a theory. The NEF enables researchers to systematically map dynamical systems—corresponding to some hypothesised brain function—onto biologically constrained spiking neural networks. In this thesis, we present several extensions to the NEF that broaden both the range of neural resources that can be harnessed for spatiotemporal computation and the range of available biological constraints. Specifically, we suggest a method for harnessing the dynamics inherent in passive dendritic trees for computation, allowing us to construct single-layer spiking neural networks that, for some functions, achieve substantially lower errors than larger multi-layer networks. Furthermore, we suggest “temporal tuning” as a unifying approach to harnessing temporal resources for computation through time. This allows modellers to directly constrain networks to temporal tuning observed in nature, in ways not previously well-supported by the NEF.

We then explore specific examples of neurally plausible dynamics using these techniques. In particular, we propose a new “information erasure” technique for constructing LTI systems generating temporal bases. Such LTI systems can be used to establish an optimal basis for spatiotemporal computation. We demonstrate how this captures “time cells” that have been observed throughout the brain. As well, we demonstrate the viability of our extensions by constructing an adaptive filter model of the cerebellum that successfully reproduces key features of eyeblink conditioning observed in neurobiological experiments.

Outside the cognitive sciences, our work can help exploit resources available on existing neuromorphic computers, and inform future neuromorphic hardware design. In machine learning, our spatiotemporal NEF populations map cleanly onto the Legendre Memory Unit (LMU), a promising artificial neural network architecture for stream-to-stream processing that outperforms competing approaches. We find that one of our LTI systems derived through “information erasure” may serve as a computationally less expensive alternative to the LTI system commonly used in the LMU.

Acknowledgements

Researching and writing a thesis can at times be a rather solitary affair, and a global pandemic surely does little to improve this situation. More than usual, I am thus grateful to all the wonderful people who have accompanied me throughout this journey. Foremost, I would like to extend my gratitude to my supervisor Chris Eliasmith. Chris has been a tremendous source of inspiration and has offered his guidance in countless discussions, while always giving me the freedom to explore what I found most interesting. Aside from research, Chris went above and beyond to make our lab a fun and rewarding place to be in; our pre-pandemic summer schools and cottage weekends are some of the most memorable times I had during my stay in Canada.

Next, I would like to thank Terry Stewart, who has been an outstanding scientific mentor and source of support throughout the years. Terry’s unparalleled enthusiasm was pivotal in sparking my interest in the Neural Engineering Framework. It is safe to say that without him, I would not have come to Waterloo and this thesis would not exist. Similarly, I thank Aaron Voelker for taking the time to read my work on denritic computation and sliding-window transformations, suggesting improvements, and providing instrumental feedback. Prof. Javier Medina’s (Baylor College of Medicine) CTN seminar served as the inspiration for our cerebellum model. He furthermore provided helpful feedback on an initial version of the model, which shaped the direction of this work.

All other former members of the CNRG, whom I had many thoughtful discussions with, shall of course not remain unmentioned: Sean Aubin, Peter Blouw, Narsimha Chilkuri, Xuan Choo, Peter Duggins, Nicole Dumont, Michael Furlong, Jan Gosmann, Eric Hunsberger, Pawel Jaworski, Ivana Kajić, Brent Komer, Thomas Lu, Ryan Orr, Sugandha Sharma, Mariah Shein, Sverrir Thorgerisson, and Natarajan Vaidyanathan. Unfortunately, due to the pandemic, our traditional board game nights had to come to a sudden end!

Special thanks go to my current and former house mates Brent, Peter, Sean, Xuan, and frequent guest Nicole. You have bravely endured my unprompted outbursts of talking about my research in the most inappropriate moments, and have in turn pushed me to, at times, actually leave my desk and to have some fun. If there is any objective measure for what makes marvellous house mates, then it should be “still talking to each other after being stuck at home for two years”, and you pass this test with flying colours.

On a very fundamental level, this research would not have been possible without thousands of people—most of them volunteers—building the software used for producing this document and for conducting the experiments therein. Just to name a few of those projects: Inkscape, Matplotlib, Numpy, Scipy, \LaTeX , \TeX studio, Poppler, OSQP, CVXOPT, Linux, GNOME, Fedora, GCC, Eigen, Meson, Autograd, Tensorflow, and many more.

Furthermore, I would like to thank all the students that I had the honour of teaching, either as a teaching assistant, or as the instructor of SYDE 556/750 “Simulating Neurobiological

Systems” in Winter 2020. Thank you for being so curious, and for asking so many fantastic questions! Even if you have only learned a fraction of what I have learned while teaching you, I would still count this as a great success. Not only has preparing weekly lecture notes for SYDE 556 made me a more confident writer, but those notes have since become a basis for Chapter 2 of this thesis.

My thesis committee shall of course not remain unmentioned either. Wulfram Gerstner, Chris Eliasmith, Robin Cohen, Jeff Orchard, and Sue Ann Campbell have all bravely combed through this rather voluminous thesis, provided important feedback on areas that required clarification, and helped to eliminate remaining typos. Despite these heroic efforts, many mistakes and inconsistencies surely still lurk among the pages to follow. Those are solely my own responsibility.

On a personal note, I extend my thanks to friends and family—to my brother Daniel Stöckel, and my friend Benjamin Paaßen for their words of wisdom on academia; to Jan and Christina Göpfert for keeping me updated on my former home in Bielefeld; and to Tosca Lechner and Shanna Yang for accompanying me on many walks and hikes. Last but not least, I owe thanks to my parents Sabine and Thomas Stöckel. They have taught me to be curious, kindled my interest in technology, supported and encouraged my studies, and, overall, have made me the person I am today.

—Andreas Stöckel, Waterloo, October 2021

Table of Contents

Table of Contents	ix
List of Figures	xiii
List of Tables	xviii
1 Introduction	1
2 Modelling Neurobiological Systems	7
2.1 Morphology and Physiology of Individual Neurons	10
2.1.1 Functional and Structural Overview	11
2.1.2 Ion Channels and the Membrane Potential	13
2.1.3 Neural Dynamics and the Hodgkin-Huxley Model	18
2.1.4 Compartmental Neuron Models	21
2.2 From Individual Neurons to Spiking Neural Networks	23
2.2.1 Synaptic Transmission	23
2.2.2 Simplified Neuron Models	27
2.2.3 Spiking Neural Networks	31
2.2.4 Neural Tuning and Population Codes	32
2.3 The Neural Engineering Framework	36
2.3.1 Model Validation and Hypothesis Generation	36
2.3.2 Applications to Neuromorphic Computing	38
2.3.3 Principle 1: Representation	40
2.3.4 Principle 2: Transformation	44
2.3.5 Principle 3: Dynamics	46
2.3.6 Some Limitations of the Neural Engineering Framework	49
3 Computational Properties of Multi-Compartment LIF Neurons	55
3.1 Theoretical Aspects of Dendritic Computation	59
3.1.1 Additive Multivariate Networks	59
3.1.2 Two-Layer Networks and Intermediate Populations	62
3.1.3 Dendritic Computation	63
3.1.4 Numerical Exploration	65
3.2 Extending the Neural Engineering Framework	70
3.2.1 Decoding the Current-Translation Function	70
3.2.2 Nonnegative Weights and Dale's Principle	73
3.2.3 Subthreshold Relaxation	80

Table of Contents

3.2.4	Extension Toward Dendritic Nonlinearities	84
3.3	A Family of Multi-Compartment LIF Neurons	86
3.3.1	Mathematical Description of n -LIF Neurons	86
3.3.2	Analysis of the Subthreshold n -LIF Dynamics	89
3.3.3	Deriving a Surrogate Model of the Dendritic Nonlinearity H	90
3.3.4	Some Worked Examples	93
3.3.5	Theoretical Properties of the n -LIF Nonlinearity	95
3.4	Networks of Two-Compartment LIF Neurons	98
3.4.1	Estimating Model Parameters	98
3.4.2	Experiment 1: Precision of the Model Parameter Estimation	100
3.4.3	Solving for Synaptic Weights	103
3.4.4	Experiment 2: Theoretical Analysis of Two-Compartment LIF Neurons	107
3.4.5	Experiment 3: Dendritic Computation in Spiking Neural Networks	109
3.5	Weight-Optimisation for Arbitrary Dendritic Trees	113
3.5.1	Solving for n -LIF Surrogate Model Parameters	113
3.5.2	Assessing Optimiser Performance and Model Quality	118
3.5.3	Solving for n -LIF Synaptic Weights	122
3.5.4	Computational Properties of the n -LIF Dendritic Nonlinearity Model	128
3.5.5	n -LIF Neurons in a Spiking Network Context	130
3.6	Summary	134
4	Temporal Tuning	137
4.1	Temporal Tuning Curves	140
4.1.1	Time Cells and Temporal Tuning	141
4.1.2	Temporal Tuning Curves and the Neural Engineering Framework	144
4.1.3	Implementing Linear Time-Invariant Systems	147
4.1.4	Realising LTI Systems in Networks with Complex Recurrence	152
4.2	Realising Temporal Bases	158
4.2.1	Computing Sliding-Window Spectra Using Linear Time-Invariant Systems	158
4.2.2	Continuous and Discrete Orthonormal Function Bases	161
4.2.3	Numerically Solving for Linear Time Invariant Systems	164
4.2.4	Closed-Form Solution for the Legendre Basis	168
4.2.5	Decoding Delays as a Benchmark for Temporal Bases	170
4.3	Accounting for Neural Nonlinearities	174
4.3.1	Sampling Input Signals and Temporal Encoders	174
4.3.2	Realising and Comparing Temporal Bases in Spiking Neural Networks	179
4.3.3	Spatiotemporal Neuron Populations	184
4.3.4	Adaptive Filters	188
4.3.5	Accounting for Intrinsic Neural Dynamics	191
4.4	Applications to Artificial Neural Networks	193
4.4.1	The Legendre Memory Unit	193
4.4.2	Computational Efficiency of the LDN LTI System	196

4.4.3	Efficient Sliding-Window Spectrum Transformations	200
4.4.4	Experiments	201
4.5	Conclusion	210
5	An Adaptive Filter Model of the Cerebellum	213
5.1	Background	216
5.1.1	Eyeblink conditioning	216
5.1.2	Cerebellar Microcircuitry	217
5.1.3	Hypothesised Mechanisms Supporting Delay Learning	218
5.2	Modelling the Granule-Golgi Circuit	219
5.2.1	Levels of Biological Detail	219
5.2.2	Impact of Biological Detail on Temporal Representations	222
5.2.3	Parameter Exploration	225
5.3	Extension Toward a Model of Eyeblink Conditioning	226
5.4	Discussion	229
6	Conclusion	231
	Bibliography	237
A	Mathematical Detail	263
A.1	Proofs for Chapter 3	263
A.1.1	Proof of Theorem 3.1	263
A.1.2	Proof of Theorem 3.2	263
A.1.3	Proof of Theorem 3.3	264
A.1.4	Discussion of Theorem 3.4	265
A.1.5	Proof of Theorem 3.6	266
A.1.6	Proof of Theorem 3.7	268
A.1.7	Proof of Theorem 3.8	271
A.2	Conditioning the n -LIF system	272
A.2.1	Suggested Preconditioning Procedure	272
A.2.2	Impact of Conditioning in Practice	273
A.3	Proofs and Mathematical Detail for Chapter 4	274
A.3.1	Proof for Theorem 4.1	274
A.3.2	Temporal Tuning and Nonlinear Dynamics	276
A.3.3	Heterogeneous Time Constants Require Access to the Derivative	278
A.3.4	Homogeneous Recurrent Networks with Interneurons	279
A.3.5	Proof of Lemma 4.2	281
A.3.6	Proof of Lemma 4.3	281
A.3.7	Proof of Lemma 4.4	282
B	Additional Data	283

Table of Contents

B.1	Quantitative Analysis of Publications Related to the NEF	283
B.2	Additional Data for Chapter 3	285
B.2.1	Regularisation Factor and Pre-Filter Sweep	285
B.3	Additional Data for Chapter 4	297
B.3.1	Comparison Between the LDN and Modified Fourier Systems	297
B.3.2	Realising Spatiotemporal Networks Without Temporal Tuning Curves	298
B.3.3	Code and Data for the Machine Learning Experiments	299
C	Software	303
C.1	An n -LIF Weight Solver and Simulator: libnlif	304
C.1.1	Describing n -LIF Neurons	304
C.1.2	Simulating n -LIF Neurons	305
C.1.3	Predicting Somatic Currents and Solving for Parameters and Weights .	307
C.2	More Biologically Plausible Nengo Models: NengoBio	309
C.2.1	Accounting for Dale's Principle	309
C.2.2	Inhibitory Interneuron Populations and Communication Channels . .	310
C.2.3	Sparsity Constraints	311
C.2.4	Dendritic Computation	312

List of Figures

1.1	Neural networks can be faster than their fastest time-constant	3
2.1	Illustration of temporal and spatial scales in neuroscience	8
2.2	Drawings of neurons in different brain regions	10
2.3	Overview of neural function	11
2.4	Membrane potential as a result of selectively permeable ion channels	13
2.5	Schematic illustration of ion channels embedded into the cell membrane	14
2.6	Electrical circuit model of the cell membrane	17
2.7	Hodgkin-Huxley-type model neuron simulations of action potential generation	19
2.8	Equivalent circuit diagram of the Hodgkin-Huxley model	19
2.9	Equivalent circuit of an exemplary compartmental neuron model	21
2.10	Illustration of potential propagation within a cylindrical cable.	22
2.11	Synaptic transmission in a chemical synapse	24
2.12	The conductance-based synapse model	25
2.13	Second- and first-order exponential synaptic low-pass filter dynamics	26
2.14	Equivalent circuit diagram and membrane potential trace of an LIF neuron	27
2.15	Voltage traces for the LIF and Hodgkin-Huxley neuron model	28
2.16	Response curves for the LIF and Hodgkin-Huxley-type neuron model	28
2.17	Diversity of neural dynamics for constant or ramp input currents	29
2.18	Illustration of different neural network types	31
2.19	Neural tuning in the visual cortex	32
2.20	Examples of randomly generated Gabor filters	33
2.21	Model of the Hubel and Wiesel experiment	34
2.22	Probabilistic decoding of scalar quantities using population codes	35
2.23	Overview of the Neural Engineering Framework	37
2.24	Obtaining bell-shaped tuning curves when using two-dimensional encoders	41
2.25	Representation in the Neural Engineering Framework	42
2.26	Examples of the function decoding scheme	44
2.27	Neuron populations in the NEF as a single-hidden-layer artificial neural network	45
2.28	Estimating the instantaneous firing rate using a synaptic filter	46
2.29	Integrator and synaptic filter realisations of an LTI system	47
2.30	Synaptic filters and LTI systems in NEF networks	48
2.31	Neuron parameter variability only accounts for small bias currents	50
2.32	Constrained connectivity and Dale’s principle	51
2.33	Illustration of the “Parisien transform”	52
2.34	Neural dynamics for different maximum firing rates	53
3.1	Using dendritic computation to approximate multivariate functions in the NEF	59

List of Figures

3.2	Additive networks are a generalisation of the perceptron	60
3.3	Visualisation of the XOR decision problem for different classifiers	61
3.4	Sketch of a two-layer neural network	62
3.5	Overview of our notion of dendritic computation.	64
3.6	Overview of our procedure for generating random 2D functions	65
3.7	Function approximation using different network setups	67
3.8	Computing functions with different spatial frequencies using different network types	68
3.9	Number of decodable basis functions over the number of neurons	69
3.10	Decoding for currents instead of represented values	71
3.11	Bias decoding and post-population tuning curve accuracy	72
3.12	Accounting for multiple pre-populations when decoding the current-translation function	73
3.13	Impact of the ratio between excitatory to inhibitory neurons on network function	75
3.14	Rank-reduced factorisation of nonnegative weight matrices	76
3.15	Comparison of weight optimisation schemes in terms of sparsity	77
3.16	Inhibitory interneurons	78
3.17	Inhibitory communication channels	79
3.18	Illustration of the goal of subthreshold relaxation	80
3.19	Current decoding with and without subthreshold relaxation	82
3.20	Reduction in decoding error achieved with subthreshold relaxation	83
3.21	Neural response curve decomposition	85
3.22	“Ball-and-stick” illustration of multi-compartment LIF neurons	86
3.23	Circuit and membrane potential trace of a two-compartment LIF neuron	87
3.24	Impulse response of the n -LIF neuron	89
3.25	Mean somatic membrane potential over the neuron output rate	91
3.26	Dendritic nonlinearity models H for different n -LIF neurons	94
3.27	Effects of the relative location of input channels on the dendritic nonlinearity H .	96
3.28	Comparison between the actual and substitute loss function	99
3.29	Two-compartment nonlinearity model for constant input	101
3.30	Two-compartment nonlinearity model for noisy input	102
3.31	Computing addition in single- and two-compartment neurons	104
3.32	Computing multiplication in single- and two-compartment neurons	105
3.33	Error and weight statistics for computing addition and nonnegative multiplication	106
3.34	Two-compartment neuron decoding error for random multivariate current functions	108
3.35	Computing nonnegative multiplication in a spiking neural network using two- compartment LIF neurons	110
3.36	Median error for computing random bandlimited functions in a feed-forward network over 100 trials	111
3.37	Illustration of gradient descent using a soft trust-region	117
3.38	Comparing different optimisers performing parameter optimisation	120
3.39	Calibrated n -LIF dendritic nonlinearity model for constant input	121
3.40	Approximating four-quadrant multiplication in different n -LIF neurons	125

3.41	Comparing different optimisers performing synaptic weight optimisation	127
3.42	Multi-compartment current decoding error for random target functions	129
3.43	Positive intercepts induce large current decoding errors	132
3.44	Computing four-quadrant multiplication in a spiking neural network using three-compartment LIF neurons	133
4.1	Illustration of possible in situ temporal response patterns	140
4.2	Temporal properties of cells in visual cortex	141
4.3	Illustration of time cells	142
4.4	Example of a space-time receptive field tuned to downwards motion	143
4.5	Solving for desired dynamics in a feed-forward linear network	147
4.6	Chaining populations with diverse temporal tuning	149
4.7	Principal component analysis of a set of low-pass filters	149
4.8	Solving for some desired temporal tuning in a recurrent neural network	150
4.9	Realising LTI systems using temporal tuning curves	151
4.10	Realising LTI systems in networks with heterogeneous second-order filters	153
4.11	Approximation errors in heterogeneous networks with second-order synaptic filters	154
4.12	Effect of the training signal distribution on the system error	154
4.13	Approximating dynamical systems across intermediate populations	156
4.14	Realising an integrator in a recurrent network with an intermediate population	157
4.15	Using an LTI system to compute a sliding basis transformation	160
4.16	Orthonormal basis functions	161
4.17	Visualisation of the discrete Fourier, cosine, and Legendre basis	163
4.18	LTI systems generating the Fourier, cosine and Legendre basis	165
4.19	Windowed LTI systems generating the Fourier, cosine and Legendre basis	165
4.20	Effect of the “information erasure” technique on LTI systems generating function bases	167
4.21	Feedback matrices generating LTI systems and their eigenvalues	167
4.22	Feedback matrices and impulse responses of the Legendre system	169
4.23	Computing delays as a function basis benchmark	171
4.24	Systematic analysis of different bases and windowing methods	172
4.25	Mean delay decoding error statistics	173
4.26	Mean delay decoding error over the input signal frequency	173
4.27	Linearly combining impulse responses to obtain temporal encoders	175
4.28	Example of non-orthogonal basis functions emulating time cells	176
4.29	Illustration of uniform activation sampling	178
4.30	Impact of uniform activation sampling on realising an integrator	178
4.31	Realising temporal bases and decoding delays in a spiking neural network	180
4.32	Systematic comparison of different temporal bases and synaptic weight optimisation methods	180
4.33	Analysing the feedback weight matrices obtained using different optimisation methods	181

List of Figures

4.34	Temporal fields and activity vector similarity of networks generating temporal bases	182
4.35	Compensating for heterogeneous synapses using temporal tuning	183
4.36	Two-dimensional quantities over time in a rank one spatiotemporal network	185
4.37	Decoding delayed multiplication in a network with full rank spatiotemporal encoders	186
4.38	Decoding the recently travelled distance from a spatiotemporal network	187
4.39	An adaptive filter learning pendulum dynamics	189
4.40	Adaptive filter error over time	190
4.41	Accounting for ALIF dynamics	192
4.42	Overview of the Legendre Memory Unit	194
4.43	Exploring the effect of Euler discretisation on the LDN impulse response and stability	198
4.44	Orthogonality of the ZOH and Euler LDN basis transformation matrix	198
4.45	Exploring the effect of higher order discretisation on the LDN basis quality	199
4.46	Effect of higher order discretisation on the modified Fourier basis	199
4.47	Integrating the Modified Fourier Basis using Runge-Kutta	199
4.48	Visualisation of the orthonormal discrete Haar wavelets	200
4.49	Comparing the orthogonality of different methods for generating sliding-window spectra	202
4.50	Overview of the psMNIST task and our network architecture	203
4.51	Classification accuracies for the psMNIST dataset using different sliding-window transformations	204
4.52	Visualisation of the Mackey-Glass system	206
4.53	Overview of the Mackey-Glass prediction network and dataset	207
4.54	Prediction errors for the Mackey-Glass dataset using different sliding-window transformations	209
5.1	Marr and Poggio’s levels of description	214
5.2	Illustration of eyeblink conditioning	216
5.3	Schematic of the cerebellar microcircuit	217
5.4	Network types used in the cerebellum experiments	219
5.5	Spatial connectivity constraints between the Golgi and granule cells	221
5.6	Granule cell EPSC distributions and PCN tuning curves	221
5.7	Temporal response of the Granule-Golgi networks	223
5.8	Delayed signal reconstruction errors in the Cerebellum model	224
5.9	Examples showing the delayed input signals decoded from the granule layer in the detailed model	224
5.10	Cerebellum model parameter exploration	225
5.11	Overview of the final eyeblink conditioning network	227
5.12	Model and experimental data for the eyeblink conditioning task	228
6.1	NEF networks with oscillator dynamics produce tonically bursting neurons	232
A.1	Visualisation of the proof of Theorem 3.3	264

A.2	Illustration of Theorem 3.4	265
A.3	Impact of conditioning on gradient-based model parameter optimisation	273
A.4	Implementing arbitrary nonlinear dynamics in the NEF	276
A.5	Block-diagram of a recurrent networks with homogeneous and heterogeneous filters	278
A.6	Block-diagram of recurrent networks with an interneuron population	279
B.1	Publications related to the NEF in the S2 ORC dataset	284
B.2	Regularisation sweep for the theoretical analysis of the two-compartment LIF nonlinearity	285
B.3	Regularisation factor and pre-filter time-constant sweeps without subthreshold relaxation	290
B.4	Regularisation factor and pre-filter time-constant sweep with subthreshold relaxation	291
B.5	Uncalibrated rate predictions for different n -LIF neurons	294
B.6	Example delay decodings for strictly bandlimited input	297
B.7	Comparing the modified Fourier and the LDN systems for band-limited inputs . .	297
B.8	Results of a delay decoding experiment in a spatiotemporal NEF network	298
B.9	FIR filters used in the Mackey-Glass experiment	300
B.10	Learning curves for the psMNIST and Mackey-Glass experiments	301
B.11	Prediction errors for the Mackey-Glass dataset with perfect rectangle windows . .	302
C.1	Output of the <i>libnlif</i> two-compartment LIF simulation code example	305
C.2	Output of the <i>libnlif</i> “rate_empirical” function for different noise parameters . . .	306
C.3	Two-compartment LIF parameter optimisation using <i>libnlif</i>	307
C.4	Nonnegative multiplication in two-compartment LIF neurons using <i>libnlif</i>	308
C.5	Spatial connectivity constraints in NengoBio	311

List of Tables

1.1	Comparison between spatial and spatiotemporal computation	3
2.1	Ion concentrations and reversal potentials in the squid and mammals	15
3.1	Matrix representations of the first four neuron models in Figure 3.22	88
3.2	Reduced matrix representations of the first four neuron models in Figure 3.22	91
3.3	Spiking neural network approximation errors	112
3.4	Function approximation errors for spiking networks using various n -LIF neurons	131
4.1	Time and space complexity of different sliding transformations	202
4.2	Test accuracies for the psMNIST experiment	205
4.3	Statistical significance of the psMNIST test accuracies	205
4.4	Prediction errors and statistical analysis for the Mackey-Glass dataset	209
B.1	Neuron and synaptic parameters for the two-compartment LIF experiments	286
B.2	Fitted two-compartment LIF dendritic nonlinearity model parameters	286
B.3	Functions analysed in the n -LIF network experiments	287
B.4	Complete results for the benchmark experiment	288
B.5	Regularisation factors and pre-filter time-constants	292
B.6	Complete results for the two-compartment LIF benchmark experiment with pre-filter	293
B.7	Calibrated and conditioned reduced n -LIF system matrices	295
B.8	Detailed function approximation errors for spiking networks using various n -LIF neurons	296
B.9	Prediction errors and statistical analysis for the Mackey-Glass dataset	302

List of Abbreviations

ALIF adaptive leaky integrate-and-fire.	LRGF locally recurrent, globally feed-forward.
ANN artificial neural network.	LSTM long short-term memory.
API application programming interface.	LTI linear time-invariant.
BFGS Broyden-Fletcher-Goldfarb-Shanno algorithm.	MSE mean square error.
CR conditioned response.	NEF Neural Engineering Framework.
CS conditioned stimulus.	NLP natural language processing.
DLOP discrete Legendre orthogonal polynomial.	NNLS nonnegative least squares.
DOF degrees of freedom.	NRMSE normalised root mean square error.
EBN Efficient, Balanced Spiking Networks.	PCN pre-cerebellar nucleus <i>or</i> pre-cerebellar neuron.
EEG electroencephalography.	PES prescribed error sensitivity.
EPSC excitatory post-synaptic current.	PSP post-synaptic potential.
EPSP excitatory post-synaptic potential.	QP quadratic program.
ESN Echo State Network.	ReLU rectified linear unit.
FCT fast cosine transform.	RMS root mean square.
FFT fast Fourier transform.	RMSE root mean square error.
FIR finite impulse response.	SDCT sliding discrete cosine transform.
GPU graphics processing unit.	SDFT sliding discrete Fourier transform.
GUI graphical user interface.	SGD stochastic gradient descent.
IO inferior olive.	SNN spiking neural network.
IPSP inhibitory post-synaptic potential.	SQP sequential quadratic program.
L-BFGS-B limited-memory BFGS with bounds.	SVD singular value decomposition.
LDN Legendre Delay Network.	UR unconditioned response.
LIF leaky integrate-and-fire.	US unconditioned stimulus.
LMU Legendre Memory Unit.	XOR exclusive or.
	ZOH zero-order hold.

Eventually, all things merge into one, and a river
runs through it. The river was cut by the world's
great flood and runs over rocks from the basement
of time. On some of the rocks are timeless
raindrops. Under the rocks are the words, and
some of the words are theirs.

I am haunted by waters.

— NORMAN MACLEAN
A River Runs Through It and Other Stories (1979)

Chapter 1

Introduction

Network models [must] reflect the fundamental and essential temporal nature of actual nervous systems. External processes and events are extended in time and for a nervous system successfully to recognize and respond may require temporally extended representation; movements required for behavior involve sets of bodily motions sequenced in time; short-term memory is a trick for allowing present access to the recent past, and longer-term memory to the more remote past.

— PARTICIA S. CHURCHLAND AND TERRENCE J. SEJNOWSKI
The Computational Brain (1992)

Biological neural networks are dynamical systems. Of course, taken at face value, this is quite an understatement—just like all physical systems, biological systems evolve through time and are thus “dynamic”. However, dynamics are of paramount interest when studying the brain (Churchland and Sejnowski, 1992, Section 3.9). At the lowest level, dynamics shape computation within individual neurons and dendritic structures. Simultaneously, at the highest level, dynamics determine the ability of an organism to survive. Variations in response-time on the order of milliseconds can make the difference between life and death, as avid viewers of nature documentaries and operators of motor vehicles will surely attest.

From the perspective of computer engineering it seems almost implausible that biological systems should be capable of rapid coordinated action—especially considering that the elements that make up nervous systems compute asynchronously and possess strong intrinsic dynamics. After all, when constructing artificial computers, asynchronicity and dynamics are mere annoyances of physical reality that engineers seek to eliminate. Typical microprocessors spend a sizeable portion of their energy budget on distributing clock signals (e.g., Zhang et al., 2008), and digital circuits are meticulously designed to switch between clearly interpretable voltage states as quickly as possible (e.g., Weste and Harris, 2011, Chapter 4). Similarly, on an algorithmic level, computer scientists assume that neurons in artificial neural networks compute instantaneously, while time progresses in discrete, synchronous steps (e.g., Goodfellow, Bengio, and Courville, 2016, Chapter 10).

This juxtaposition of artificial and biological systems gives rise to an old, but still common (and rather subtle) mischaracterisation—namely, the idea that biological information processing is slow and that, compared to digital computers, biological brains compensate for their slowness with massive parallelism (e.g., Von Neumann, 1958, pp. 50f). There is, for sure, a kernel of

truth to this statement. For example, signal propagation in electronic circuits is between six and eight orders of magnitudes faster than axonal action potential propagation.¹ Still, stating that neurons and synapses are “slow” is misleading at best. On the one hand, neural *networks* can be much faster than their slowest elements (cf. Figure 1.1), and, on the other hand, the “slow” neural dynamics are an *intrinsic* part of the computation that is being performed. Put differently, and rephrasing the above Churchland and Sejnowski quote, computation that depends on the history of a signal on short time-scales is possible *because* information about the past is implicitly retained by the “slow” temporal dynamics, and not *despite* this fact.

While it is undisputed that dynamics play an important role in the brain, there are few *overarching* theories that suggest how dynamics should be integrated into models of biological and cognitive systems. Cybernetics (Wiener, 1948) and dynamicism (van Gelder, 1998; Eliasmith, 1996) stand out as attempts to establish dynamics-focused research programs, but, as of writing, have either been subsumed by other fields or faded to obscurity. Instead, the temporal evolution of biological and cognitive systems is modelled, often separately, at every possible level of abstraction—from individual neurons and their subcomponents (e.g., Gerstner and Kistler, 2002; Izhikevich, 2007), over small- and large-scale networks (e.g., Gerstner, Kistler, et al., 2014; Bassett and Sporns, 2017), to behaviour and language (e.g., Anderson, Matessa, and Christian Lebiere, 1997; De Bot, Lowie, and Verspoor, 2007).

What is missing are *bridging laws*, that describe in broad strokes how low-level mechanism gives rise to high-level behaviour and cognition. Building upon earlier work by Eliasmith and Anderson (2003), this thesis is an attempt at providing such laws.

In particular, our goal is to study ways in which low-level neural dynamics can be harnessed as a resource for high-level function. We approach this from two different directions. First, we analyse the dynamics intrinsic to passive dendritic trees, and derive a static model of dendritic nonlinearity that can be systematically exploited for high-level, non-temporal computation. Second, we investigate “temporal tuning” as a low-level characterisation of individual model neurons. This allows us to construct networks that can approximate arbitrary spatiotemporal functions (cf. Table 1.1), while outperforming common methods from machine learning.

Importantly, we focus on being as integrative as possible. That is, our goal is *not* to model biological phenomena at the greatest possible detail, but to think about mathematically tractable ways in which a broad range of neurophysiological and behavioural phenomena can be captured. This way, we hope to provide a useful tool for researchers wishing to model high-level function, while constraining their work to physiological data.

Similarly important, and returning to our comparison between brains and computers, our approach is not only useful for cognitive modellers, but also for scientists working on brain-inspired computers (so called *neuromorphic hardware*), and to the field of machine learning. Systematically thinking about ways in which the neural substrate connects to function informs

¹Electrical signals travel at about two thirds of the speed of light along conductors (i.e., about $200 \times 10^6 \text{ m s}^{-1}$). Axonal action potential propagation reaches between 1 m s^{-1} to 100 m s^{-1} (Kandel et al., 2012, Chapter 2, p. 23).

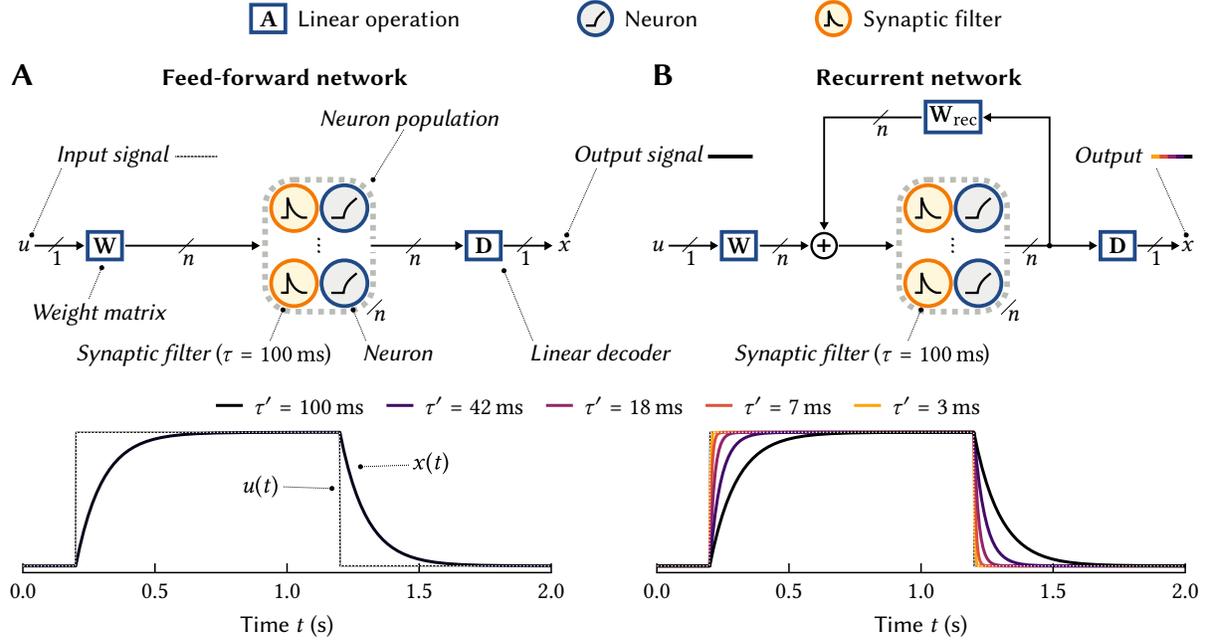


Figure 1.1: Neural networks can be faster than their fastest feed-forward time-constant. See Sections 2.3 and 4.1.3 for more information. **(A)** Feed-forward computation with 100 ms first-order low-pass filters (*synaptic filters*) in its forward path. Sending a pulse signal $u(t)$ into this network, we receive a low-pass filtered output $x(t)$ that *slowly* converges to $u(t)$. **(B)** By adding recurrent connections and solving for weight matrices \mathbf{W} , \mathbf{W}_{rec} , we can (in the limit of $n \rightarrow \infty$) realise any dynamical system, including low-pass filters with time-constants $\tau' < \tau$ (coloured lines; cf. Eliasmith and Anderson, 2003, Chapter 8).

Table 1.1: Comparison between spatial and spatiotemporal computation. Spatial (static, or non-temporal) computation can be modelled as a first-order functions f , spatiotemporal computation relies on second-order functions processing the entire input history $\mathfrak{x}(t)$ for $t \leq 0$. Listed are the resources used to establish the function basis, i.e., tuning curves, in neural networks (see also Chapter 4).

	Spatial computation	Spatiotemporal computation
	$f : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ $f(\mathbf{x}) \mapsto \mathbf{y}$	$f : (\mathbb{R}^- \rightarrow \mathbb{R}^d) \rightarrow \mathbb{R}^{d'}$ $f(\mathfrak{x}) \mapsto \mathbf{y}$
<i>Domain</i>	Input $\mathbf{x} \in \mathbb{R}^d$	Signal $\mathfrak{x} \in \{f \mid f : \mathbb{R}^- \rightarrow \mathbb{R}^d\}$
<i>Codomain</i>	Output $\mathbf{y} \in \mathbb{R}^{d'}$	Output $\mathbf{y} \in \mathbb{R}^{d'}$
<i>Function basis</i>	Tuning curves $a_i(\mathbf{x})$	Temporal tuning curves $a_i(\mathfrak{x})$
<i>Neural resources</i>	• Somatic & dendritic nonlinearities	• Somatic & dendritic nonlinearities • Synaptic filters & axonal delays • Intrinsic dynamics • Recurrent connections

both the architecture of neuromorphic computers, and suggests a “neural compiler” for realising desired algorithms on these devices (Boahen, 2017; Voelker and Eliasmith, 2021). Furthermore, our approach lends itself to thinking about the “best possible” representation that should be established in neural networks, and can thus pave the way for artificial neural networks that are faster to train, more data-efficient, and higher performing (Voelker, Kajić, and Eliasmith, 2019; Chilkuri and Eliasmith, 2021).

Structure and overview of this thesis. The remainder of this thesis is structured as follows. In Chapter 2 we review the aforementioned techniques for modelling and simulating neurobiological systems. For readers with a mathematical, but no further biology background, we first introduce fundamental concepts from neuroscience, and then discuss our main modelling technique, the *Neural Engineering Framework* (NEF; Eliasmith and Anderson, 2003). We close with a discussion of shortcomings of the NEF that we hope to resolve in this thesis.

We continue in Chapter 3 by describing novel methods for integrating complex neuron models with multiple input channels and conductance-based synapses into NEF networks. Specifically, we focus on a mathematically tractable extension to the LIF neuron, namely the n -compartment LIF, or n -LIF, neuron. The purpose of this is twofold. First, our extensions to the NEF allow modellers to better constrain their models to neurophysiological data (we explore this in more detail in Chapter 5). Second, we demonstrate—both in theory, and in simulation—that it is possible to systematically exploit the dynamics intrinsic to the passive dendritic tree of the n -LIF neuron to perform multivariate, non-temporal computation in a spiking neural network context. That is, NEF networks with n -LIF neurons can be made to automatically employ a simple form of *dendritic computation* (Mel, 1994; London and Häusser, 2005) and to thus approximate a larger range of functions well compared to similar networks with standard LIF neurons, sometimes outperforming two-layer networks. In particular, we can model gain modulation, an important phenomenon in the central nervous system (Salinas and Thier, 2000; Chance, Abbott, and Reyes, 2002). Our results may inform the design of future neuromorphic hardware and enable a better utilisation of existing neuromorphic computers.

Next, in Chapter 4, we introduce the notion of *temporal tuning curves* to the NEF. Based on a model of spatiotemporal receptive fields in visual cortex (Carandini, Heeger, and Anthony Movshon, 1999), temporal tuning curves provide a systematic way to compute functions *through time* (Table 1.1). This approach is a generalisation of the NEF dynamics principle, and a paradigm shift that offers a unified view onto resources for temporal computation, including synaptic filters, time-delays, and, to some degree, intrinsic neural dynamics.

Interestingly, we can, with small modifications, reuse a convex optimisation problem derived in the previous chapter to solve for weights that realise desired temporal tuning. The weight solving procedure does not distinguish between feed-forward and feedback connections, and we can employ feedback connections to realise diverse² temporal tuning. Building upon previous work by Voelker and Eliasmith (2018), we show that mapping linear time-invariant

²We use the term “diverse” to refer to a highly linearly independent basis.

(LTI) systems approximating sliding-window transformations onto spiking neural networks generates temporal tuning that resembles time cells (Pastalkova et al., 2008; Howard et al., 2014; Tiganj et al., 2016). We present a general method for constructing such LTI systems.

Interestingly, our spatiotemporal NEF populations are equivalent to a layer of Legendre Memory Units (LMUs; Voelker, Kajić, et al., 2019). LMUs are an exciting artificial neural network architecture for stream-to-stream processing. We find evidence that one of our LTI systems may possess more favourable computational properties compared to the standard LTI system used in the LMU, while offering a higher performance in some experiments.

As a final experiment in Chapter 5, we combine the approaches outlined in the previous chapters to construct an adaptive filter model of the cerebellum (Fujita, 1982). We apply our methods for building NEF networks with more biological constraints (including Dale’s principle; cf. Chapter 3) in the context of the recurrent Granule-Golgi circuit. This circuit can in principle implement an LTI system generating a temporal basis as discussed in Chapter 4. Building upon the Granule-Golgi circuit we construct a network that learns to decode arbitrary delays and reproduces basic features of eyeblink conditioning (e.g., Heiney et al., 2014).

We conclude with a short discussion of our contributions and potential future work in Chapter 6. Proofs and additional information may be found in Appendices A and B. We present some software libraries written for this thesis, namely *libnlif* and *NengoBio*, in Appendix C.

Notation

Some effort has been made to be as consistent as possible in the mathematical notation. Bold lower-case letters (e.g., \mathbf{x} , \mathbf{y}) denote vectors, while bold upper-case letters (e.g., \mathbf{X} , $\mathbf{\Gamma}$) denote matrices. Double-struck letters denote sets, with \mathbb{N} , \mathbb{Z} , \mathbb{Q} , \mathbb{R} , \mathbb{C} being the usual number systems. \mathbb{R}^+ and \mathbb{R}^- are the positive and negative real numbers (including zero).

We generally assume that vectors are column vectors, i.e., $\mathbf{x} \in \mathbb{R}^d$ is implicitly a matrix $\mathbf{x} \in \mathbb{R}^{d \times 1}$. Correspondingly, $\mathbf{x}^T \mathbf{y} = \langle \mathbf{x}, \mathbf{y} \rangle$ forms an inner product, and $\mathbf{x} \mathbf{y}^T = \mathbf{x} \otimes \mathbf{y}$ is an outer product.

When describing an entry in the i th row and the j th column of a matrix \mathbf{M} , we write $(\mathbf{M})_{ij}$; $(\mathbf{M})_i$ corresponds to the i th row, whereas $(\mathbf{M}^T)_j$ corresponds to the j th column. All indices start with one.

We use the standard notation for the L^p vector norm and the Frobenius matrix norm. Let $\mathbf{x} = (x_1, \dots, x_d)$ and $\mathbf{M} \in \mathbb{R}^{m \times n}$ with $(\mathbf{M})_{ij} = m_{ij}$. Then

$$\|\mathbf{x}\|_0 = |\{x_k | x_k \neq 0\}|, \quad \|\mathbf{x}\|_p = \sqrt[p]{\sum_{k=1}^d |x_k|^p}, \quad \|\mathbf{x}\|_\infty = \max\{x_1, \dots, x_d\}, \quad \|\mathbf{M}\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n m_{ij}^2}.$$

We define the inner product $\langle f, g \rangle$ and the convolution $f * g$ of two functions $f, g : \mathbb{R} \rightarrow \mathbb{R}$ as

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(\tau)g(\tau) d\tau, \quad (f * g)(t) = \int_{-\infty}^{\infty} f(t - \tau)g(\tau) d\tau.$$

Chapter 2

Modelling Neurobiological Systems

First, it is an overstatement to describe what I am attempting here as an “approach toward the understanding”; it is merely a somewhat systematized set of speculations as to how such an approach ought to be made. That is, I am trying to guess which of the—mathematically guided—lines of attack seem, from the hazy distance in which we see most of them, a priori promising, and which ones have the opposite appearance.

— JOHN VON NEUMANN
The Computer and the Brain (1958)

The desire to understand cognition is by no means a new one—wondering about one’s own thought processes has surely been part of the human condition since prehistoric times. Today, we know that nerve cells in the brain are the primary functional and structural units that give rise to biological cognition. This is referred to as the *neuron doctrine*, a view that garnered support through observations made by Santiago Ramón y Cajal and Charles Scott Sherrington at the turn of the nineteenth century (Yuste, 2015; Bear, Connors, and Paradiso, 2016, Chapter 2).

Although the nature of neurons as independent units is undisputed, the neuron doctrine continues to be scrutinised. There is an argument to be made that neural circuits are better described from the perspective of neural ensembles instead of individual neurons (Yuste, 2015; Churchland et al., 1992)—an idea that we revisit in the context of population codes. Furthermore, it is still a matter of debate in how far structures such as Glial cells, must be taken into account to accurately model brain function (e.g., Verkhatsky and Steinhäuser, 2000).

The fact that, after more than a century of research, such fundamental debates still persist, illustrates that neuroscience (and, by extension, the other cognitive sciences) are still in their early days. This is not due to a lack of talent or dedication; rather, the gaps in our knowledge are a testament to the intrinsic difficulty of understanding complex systems such as the brain.

Material in this chapter from prior publications

Parts of this chapter, particularly Section 2.3, are based on Stöckel and Eliasmith (2021). Some figures in this chapter were adapted from material previously prepared for the University of Waterloo course SYDE 556/750, “Simulating Neurobiological Systems” taught by the author in Winter 2020.

Chapter 2. Modelling Neurobiological Systems

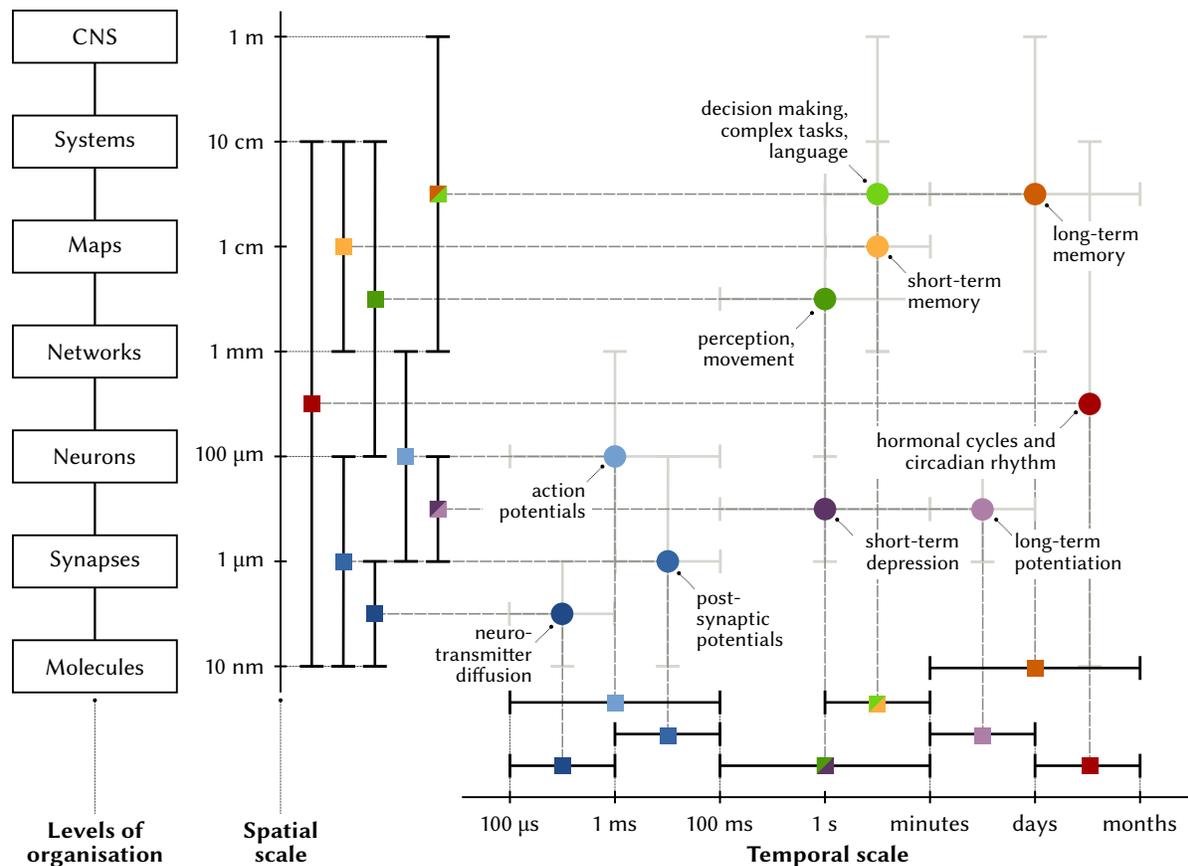


Figure 2.1: Illustration of temporal and spatial scales in neuroscience. Placement of individual concepts is deliberately coarse and, in some cases, open to debate. Spatial scale and levels of organisation adapted from Churchland and Sejnowski (1992, Figure 1.4, p. 11). Temporal scales inspired by Sejnowski, Churchland, and Movshon (2014, Figure 1).

In some regard, neuroscience may be compared to physics. Both fields study and predict the behaviour of natural systems, and both study phenomena that span vast spatial and temporal scales (cf. Figure 2.1). Unlike physics, neuroscience is concerned with the behaviour of living objects consisting of billions of complex elements.¹ The evolved nature of brains in particular makes it challenging to disentangle structures merely responsible for metabolic function, from those orchestrating behaviour. Therefore, as demanded by many in the field (e.g., Marr, 1982; Churchland et al., 1992; Eliasmith and Anderson, 2003), neuroscience should be guided by predictive computational modelling and theory.

Here, the term “theory”, refers to overarching concepts applicable to different models (e.g., Stevens, 2000). Continuing our physics analogy, successful physical theories such as Newtonian or Lagrangian mechanics, describe phenomena at different scales—from falling

¹The human brain has $67\text{-}86 \times 10^9$ neurons and $\sim 85 \times 10^9$ glial cells (von Bartheld, Bahney, and Herculano-Houzel, 2016). Each neuron in cerebral cortex possesses $10^3\text{-}10^4$ synapses (Braitenberg and Schüz, 2013, Chapter 6).

applies to solar systems orbiting their galactic centre. Translating this to neuroscience, we would like our theories to connect low-level mechanisms (e.g., neurons, synapses, action potentials) to high-level behaviour (e.g., motor control, decision making, language). As of now, there is no widely accepted theory that accomplishes this (Eliasmith, 2013, Chapter 9).

All this is not to say that neuroscience has stagnated over the past decades. To the contrary. New recording methods—such as fMRI (functional magnetic resonance imaging), high-density multi-electrode arrays, calcium imaging, and optogenetics—have widened our perspective on brain function (Sejnowski, Churchland, and Movshon, 2014). Researchers have mapped out individual brain circuits (e.g., Shepherd and Grillner, 2012), can describe neurons at a molecular level (e.g., Sobolevsky, Rosconi, and Gouaux, 2009), and determined the roles specific brain regions play in cognition (e.g., Kanwisher and Yovel, 2006).

Paradoxically, recent progress in neuroscience has made finding theories that bridge multiple levels of analysis *more*, and not less important. The central challenge is that each dataset on its own is fairly limited, mostly because different recording techniques possess dissimilar temporal and spatial characteristics (Sejnowski et al., 2014). The data we have access to is not detailed enough to directly inform large-scale models, and our theories are not sophisticated enough to combine heterogeneous datasets. As pointed out by Churchland et al. (1992), there is an argument to be made that neuroscience has been, and still is, “data poor *and* theory poor”. A major difficulty in modelling brain function hence lies in building models that can be constrained by, and that make predictions compatible with, the different scales at which current recording techniques operate (Eliasmith, 2013, Chapter 9).

As of now, there are a few attempts at developing such theories, or, less ostentatiously, *modelling frameworks*, that facilitate describing the nervous systems at different scales. Examples include the Neural Engineering Framework (NEF; Eliasmith and Anderson, 2003), Efficient, Balanced Spiking Networks (EBN; Boerlin and Denève, 2011; Boerlin, Machens, and Denève, 2013), and, to some degree, FORCE (Sussillo and Abbott, 2009; Nicola and Clopath, 2017). Generally speaking, these approaches describe how to translate dynamical systems—corresponding to some hypothesized behavioural model—into an idealised spiking neural network that adheres to desired neurophysiological constraints. Depending on the specific method, this can include neural tuning, firing rate distributions, and population-level connectivity (Komer and Eliasmith, 2016; Nicola et al., 2017). The resulting networks can then be analysed using the same techniques as biological systems, bridging the gap between empirical data and theory.

As we mentioned in Chapter 1, a goal of this thesis is to extend the Neural Engineering Framework to better take mechanistic constraints often found in the neuroscience literature into account. To this end, we first review fundamental neuroscientific concepts, and then describe the NEF along with list of biological constraints currently not well-captured by the NEF. We then, in subsequent chapters, propose extensions to the NEF that, to some degree, alleviate these issues. Finally, we demonstrate constructing a detailed biological model of eyeblink conditioning in the cerebellum.

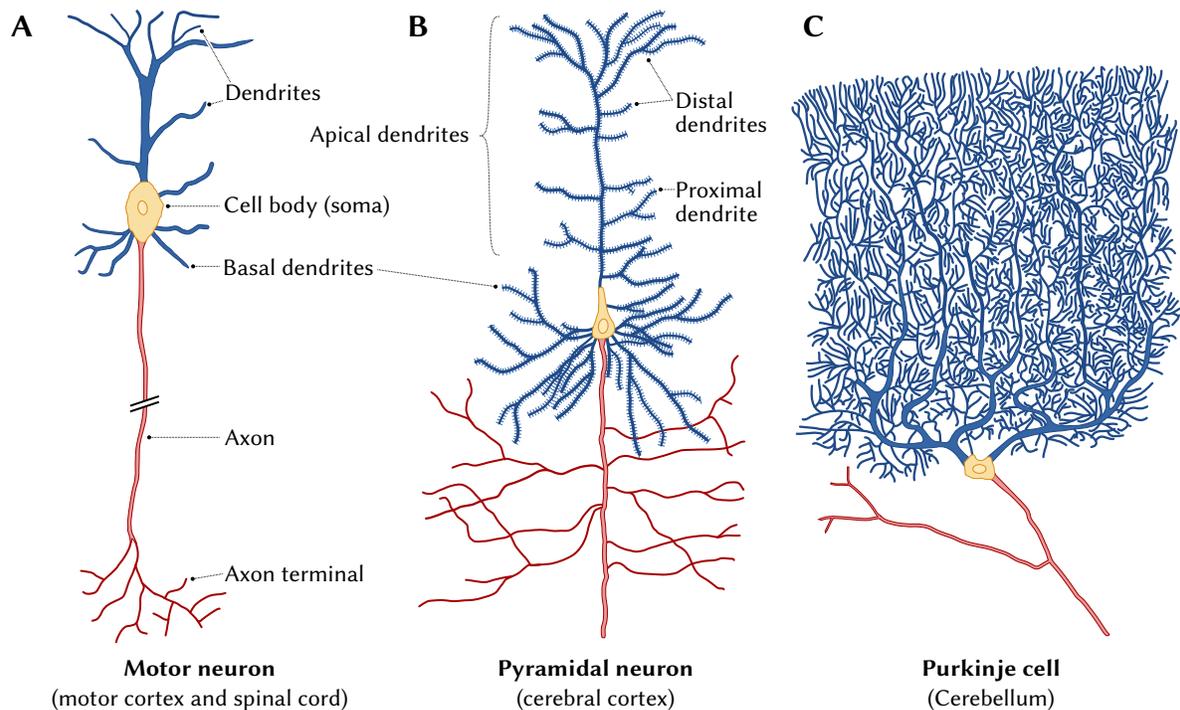


Figure 2.2: Drawings of neurons in different brain regions. Although key structural elements can be identified across neurons, they can have very different morphologies. Redrawn from Ramón y Cajal. Overall presentation from Kandel et al. (2012), Figure 2-3D, p. 25. **(A)** Schematic drawing of a motor neuron (Ramón y Cajal, 1894, Figure 6, p. 25). **(B)** Human cortical pyramidal neuron (Howell, 1916, Figure 84, p. 187). **(C)** Human purkinje cell in the cerebellum (Ramón y Cajal, 1909, Figure 9, p. 61).

2.1 Morphology and Physiology of Individual Neurons

Neurons come in various sizes, shapes, and degrees of interconnectedness. Ramón y Cajal’s drawings (schematically redrawn in Figure 2.2) offer a glimpse of the morphological diversity found in the nervous system. Some neurons possess only few branches (also *projections*) and have a comparably simple structure, whereas others, such as cerebellar Purkinje cells, feature sprawling dendritic trees more deserving of the name “dendritic forest”.

It should come as no surprise that the following review of neural morphology and physiology cannot do justice to the incredible complexity observed in nature. Our goal is to instead focus on structural and functional elements commonly observed across all neurons. As a side-effect, the resulting characterisation of the nervous system lends itself well to computational models. We open with a high-level overview of neural function and structure, and then focus on aspects of neural electrophysiology that will be relevant in later sections, including the generation of the membrane potential, action potentials, and synaptic transmission. Readers interested in a more thorough introduction to neuroscience are encouraged to consult neuroscience textbooks such as Bear et al. (2016), Purves et al. (2017), or Kandel et al. (2012).

2.1. Morphology and Physiology of Individual Neurons

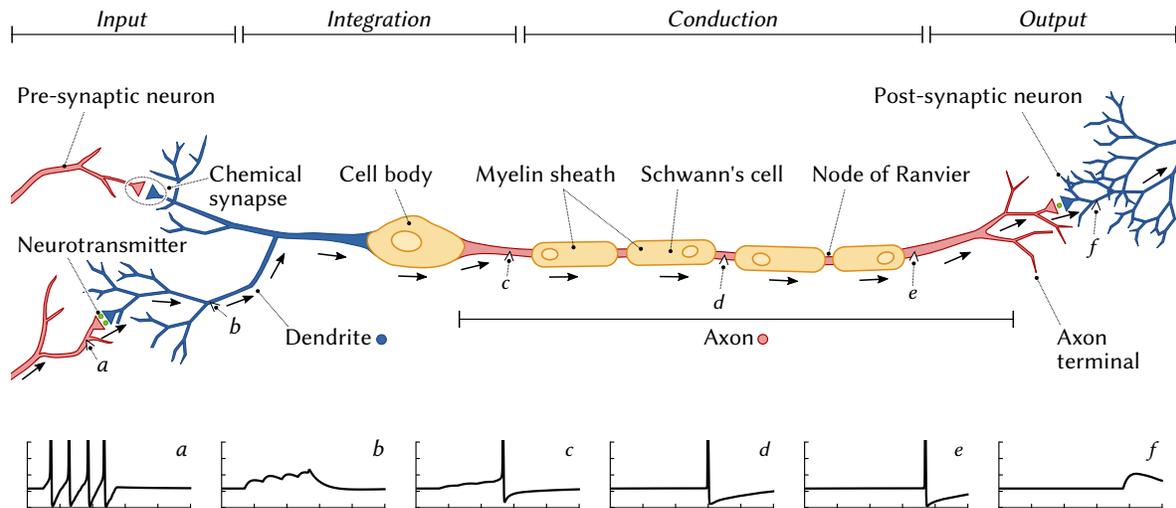


Figure 2.3: Overview of neural function. Inset diagrams illustrate the membrane potential over time at the indicated locations (data from a compartmental Hodgkin-Huxley-type model neuron). Arrows indicate the predominant flow of information. *Input:* Neurons receive signals from pre-synaptic neurons through synapses. For chemical synapses, action potentials (“spikes”) arriving at the synapse (*a*) cause a release of neurotransmitter that induces a post-synaptic potential (PSP) in the post-synaptic neuron (*b*). *Integration:* PSPs from multiple pre-synaptic neurons interact in the dendrites. Under certain conditions the neuron produces its own action potential (*c*). *Conduction:* The action potential travels along the axon (*d-e*). *Output:* The neuron induces a PSP in a post-synaptic neuron (*f*).

2.1.1 Functional and Structural Overview

From the perspective of computer science—and as suggested by the neuron doctrine—neurons are fundamental units of computation in biological systems. Specifically, neurons employ weak bioelectric signals, variations in their membrane potential, to compute. As depicted in the lower portion of Figure 2.3, these variations can either be gradual, as is the case when the neuron is in its *subthreshold* regime, or manifest themselves as rapid swings between the highest and lowest voltages that can be produced by a neuron; in this case the neuron is in its *superthreshold* regime. These rapid voltage changes are called *action potentials*, or, more colloquially, “*spikes*”, harking back to their prominent appearance on oscillograms.

Typically, gradual subthreshold potentials are not directly accessible from other parts of the network; this analogue code is solely part of a neuron’s internal state. Hence, from the perspective of other neurons in the network, a neuron is either “silent” or “spiking”. This binary nature of neurons fascinated early computer scientists such as Von Neumann (1958) and led to the development of the first artificial neural networks by McCulloch and Pitts (1943).

Neuroscientists typically divide neural information processing into four stages: input, integration, conduction, and output (Kandel et al., 2012, Chapter 2). These stages roughly correspond to different features of neural structure, as depicted in Figure 2.3.

The *input region* corresponds to the dendrites, and the synapses embedded therein.² Dendrites are tree-like structures protruding out of the cell body, typically less than two millimetres long (Bear et al., 2016, Chapter 1). They are usually classified according to their location relative to the cell body. For example, in pyramidal cells (cf. Figure 2.2B) dendrites directly connected to the soma are called *basal*; dendrites indirectly connected to the soma through longer projections are called *apical*. Apical dendrites are sometimes additionally divided—using standard anatomical terminology—into parts referred to as *proximal* or *distal*; that is, they are either closer or farther away from the soma (e.g., Seamans, Gorelova, and Yang, 1997, Figure 5).

Synapses form the coupling sites between neurons and—in the case of chemical synapses discussed below—establish a unidirectional flow of information. This ensures that pre-synaptic neurons only influence the state of the post-synaptic neuron, but not vice-versa (Kandel et al., 2012, Chapter 8). Some neurons in the periphery of the nervous system act as sensors; they transduce modalities such as temperature, pressure or light and do not rely on synaptic inputs (Kandel et al., 2012, Chapter 22).

The neuron's *integration region* is formed by the cell body and, to some degree, the dendrites themselves. Signals from several pre-synaptic neurons interact here. This interaction may elicit the generation of an output signal. As we mentioned before, this output takes the form of an *action potential* in most vertebrate neurons (Kandel et al., 2012, Chapter 2). Crucially, and in contrast to artificial neural networks, the integration process has a temporal component; the input *history* is relevant for the generation of the output, and not just the momentary input.

The *conductive region* corresponds to a neuron's axon. The axon carries signals generated by the neuron to other parts of the nervous system. Two features of the axon work in tandem to ensure signal integrity (Kandel et al., 2012, Chapter 7). First, the axon is electrically well insulated, being tightly wrapped in Schwann's cells, a type of glial cell. Second, axons actively renew action potentials at nodes of Ranvier, gaps between neighbouring Schwann's cells. Conduction is unidirectional in the sense that action potentials only travel away from the origin of excitation. Notably, in larger animals such as humans, individual neurons conduct signals across micrometres (between neighbouring neurons), centimetres (connecting different parts of the brain), and metres (motor neurons in the spinal cord).

Finally, the *output region* corresponds to the area surrounding the synapses that lies between the axon terminals and post-synaptic dendrites. In the case of chemical synapses, the arrival of an action potential at the axon terminal causes the release of neurotransmitter molecules. Receptors in the post-synaptic neuron transduce neurotransmitters into an input signal, a post-synaptic potential. In the periphery, axon terminals may be coupled to muscle fibres, where the arrival of action potentials generates movement (Kandel et al., 2012, Chapter 8 & 9).

²The difference between input and output regions of a neuron is not always clear-cut. Invertebrate unipolar cells only possess a single branch that protrudes from the cell body. This branch is referred to as an "axon" although it carries both input and output (Kandel et al., 2012, Chapter 2).

2.1. Morphology and Physiology of Individual Neurons

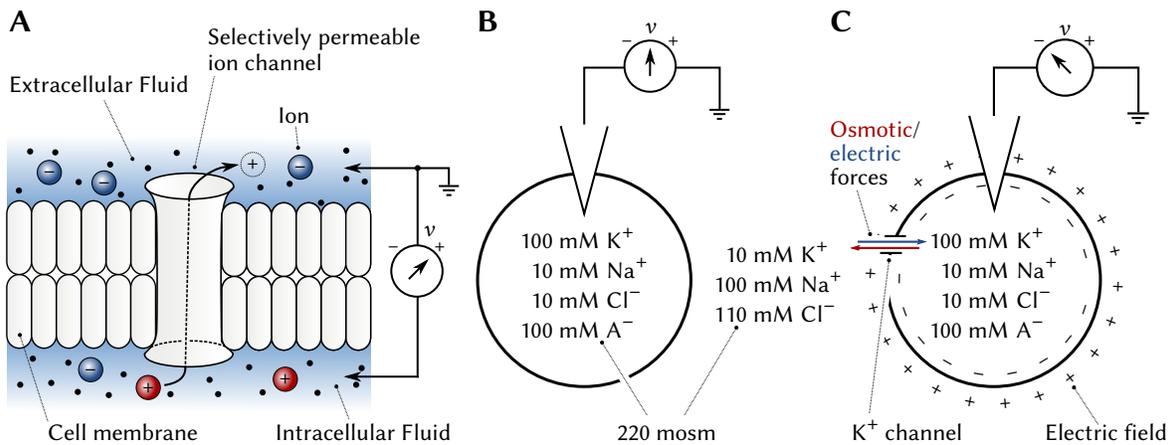


Figure 2.4: Membrane potential as a result of selectively permeable ion channels. **(A)** The cell membrane separates the intra- and extracellular fluid. Charge carriers (e.g., ions) are in solution in the fluids. The membrane potential is the voltage v across the membrane. Specific ions may pass through selectively permeable channels. **(B)** If the membrane was a perfect insulator, there would be no membrane potential. The individual fluids are electrically neutral. Molar concentration in $\text{mM} = \text{mmol l}^{-1}$; values illustrative. A^- corresponds to charged anionic molecules. **(C)** Adding a selectively permeable ion channel results in a membrane potential specific to the ion species at which electric and osmotic forces cancel out. Illustrations (B, C) adapted from Reichert (2000), Figures 2.8 and 2.10.

2.1.2 Ion Channels and the Membrane Potential

As mentioned above, neurons process information by systematically varying their membrane potential v (more precisely, their *transmembrane* potential). The membrane potential is the electrical potential between the inside and outside of a cell. The boundary between these regions is formed by the cell membrane, a double-layer of lipids.³ By convention, if the *inside* is more positively charged than the outside, we report a positive voltage (Figure 2.4A).

Measuring the membrane potential when a neuron is at rest—that is, when the cell does not receive any input—reveals the so-called resting potential v_{rest} . In mammals, v_{rest} ranges from -50 mV to -85 mV depending on the cell type (Moorhouse, 2016).

Upon closer investigation, the presence of a non-zero resting potential is rather surprising. The inside and outside of the cell are filled with watery solutions: the intracellular fluid (“cytoplasm”) and the extracellular fluid. Both fluids are electrically neutral. That is, although the fluids contain ions and anionic charge carriers in solution, the overall positive and negative

³To be clear, *all* biological cells possess bioelectrical properties, including a membrane potential. This was for example analysed in detail by Julius Bernstein in the early 1900s (Bernstein, 1912). Membrane potentials are important for homeostasis and cell-to-cell communication (Moorhouse, 2016). For example, electrical gradients between cells are crucial for laying out the body plan of organisms (Levin, 2014). Neurons should be thought of as cells shaped by evolution to excel at bioelectrical signalling, but are not unique in their use of bioelectricity.

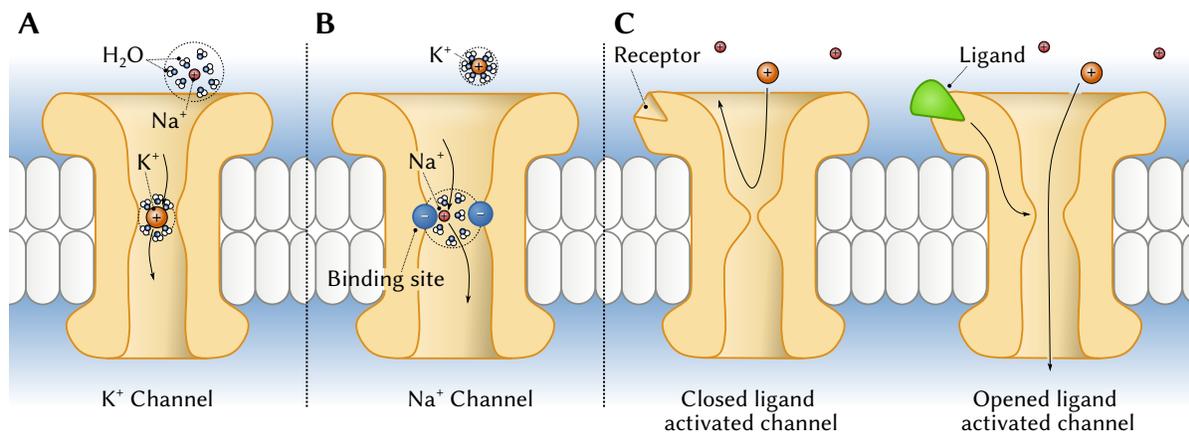


Figure 2.5: Schematic illustration of ion channels embedded in the cell membrane. Ion channels are pores that selectively let ions of a certain species pass. Selectivity is achieved by exploiting the radius and charge distributions of the ion and its watery hull. **(A)** Although the K^+ ion is larger than the Na^+ ion, its watery hull is more compact, and it can pass through narrower channels. **(B)** Na^+ selectivity is achieved by a negative binding site in the channel that stabilises the K^+ ion and its hull to let it pass. **(C)** Ion channels can change their conformation depending on external factors (e.g. mechanical forces, chemicals, electrical potentials); here, a chemical ligand opens a channel. Inspired by Kandel et al. (2012, Figure 5-1, p. 102 and Figure 5-6, p. 109); see *ibid.* for more information.

charges are balanced. There should be no measurable electrical potential (Figure 2.4B).

Selectively permeable ion channels generate the membrane potential. The resting potential can be explained if we assume that the cell membrane is selectively permeable to some ions through so-called *ion channels*. The presence of ion channels has been postulated for a long time, though it is only thanks to relatively recent studies that we understand the molecular machinery that underpins selective permeability (Kandel et al., 2012, Chapter 5).

Ion channels are porous proteins embedded in the cell membrane. They accomplish the impressive feat of acting as an “atomic sieve”. This “sieve” only lets a single ion of a certain kind (in chemical parlance: *ion species*) pass at a time, as is illustrated in Figures 2.5A and 2.5B. Furthermore, as we will see both in the context of action potential generation and synaptic transmission, these proteins can change their shape, or *conformation*, depending on various external circumstances, effectively opening or closing the channel (Figure 2.5C).

Of course, the mere presence of selectively permeable channels does not explain the membrane potential. For this, we have to consider that despite being electrically neutral, the intra- and extracellular fluids contain different concentrations of the individual charge carriers. In particular, there are large concentration gradients for potassium (K^+), sodium (Na^+) and chloride (Cl^-) ions. For example, the concentration of potassium ions K^+ is about 140 mmol l^{-1} in the cytoplasm, compared to 3 mmol l^{-1} in the extracellular fluid. If the cell membrane was permeable to K^+ ions, an osmotic force would act on the ions, driving them outside.

Table 2.1: Ion concentrations and reversal potentials in the squid and mammals. Data from McCormick (2014, Table 12.1, p. 353). The squid data are based on Hodgkin and Katz (1949), and reported similarly in Kandel et al. (2012, Table 6-1, p. 128). Reversal potentials computed using eq. (2.1).

Ion species	Concentrations		Reversal potentials	
	Intracellular	Extracellular	$T = 20^\circ\text{C}$	$T = 36^\circ\text{C}$
<i>Squid giant axon</i>				
Potassium K^+	400 mmol l^{-1}	20 mmol l^{-1}	-76 mV	-80 mV
Sodium Na^+	50 mmol l^{-1}	440 mmol l^{-1}	55 mV	58 mV
Chloride Cl^-	40 mmol l^{-1}	560 mmol l^{-1}	-67 mV	-70 mV
<i>Mammalian neuron</i>				
Potassium K^+	140 mmol l^{-1}	3 mmol l^{-1}	-97 mV	-102 mV
Sodium Na^+	18 mmol l^{-1}	145 mmol l^{-1}	53 mV	56 mV
Chloride Cl^-	7 mmol l^{-1}	120 mmol l^{-1}	-72 mV	-76 mV

This ionic flow disturbs the charge balance of the intra- and extracellular fluids and results in a non-zero membrane potential. In turn, this potential results in an electric force that counters the osmotic pressure. In our K^+ example, the intracellular fluid becomes negatively charged and the ensuing electric attraction reduces the net force acting on the K^+ ions. The system converges to an equilibrium point where the electric and osmotic forces cancel out (Figure 2.4C). The membrane potential at this point is called the *equilibrium* or *reversal potential*, as the ionic flow direction reverses at this point. For clarity, we use the term reversal potential when talking about the equilibrium potential for a membrane permeable to a *single* ion.

A single selectively permeable ion channel: The Nernst equation. For a single charge carrier X , the reversal potential E_X can be computed using the Nernst equation (Nernst, 1888):⁴

$$E_X = -\frac{RT}{zF} \log \left(\frac{[X]_{\text{in}}}{[X]_{\text{out}}} \right), \quad (2.1)$$

where R is the gas constant, T is the temperature, z is the charge number, or valence, of the particle (e.g., $z = 2$ for Ca^{2+} and -1 for Cl^-) and F is the Faraday constant. $[X]_{\text{out}}$ and $[X]_{\text{in}}$ are the concentrations (count per volume) of particles X outside and inside the cell, respectively. Note that the number of ions flowing through the cell membrane is rather minuscule compared to the total number of ions in solution; $[X]_{\text{out}}$ and $[X]_{\text{in}}$ essentially do not change over time.⁵

⁴The Nernst equation was published in 1888, before there were established theories about bioelectricity. Nernst studied the electrochemistry of electrolytes separated by membranes and this research was incorporated into analyses of biological cell membranes by Ostwald and later Bernstein (see Bernstein, 1912, Chapter 5).

⁵The ion concentrations in the intra- and extracellular fluid are maintained by ion pumps in the cell membrane. These pumps are secondary for a cell's short-term bioelectrical properties, but are indispensable in the long run.

Multiple selectively permeable ion channels: The Goldman-Huxley-Katz equation. Table 2.1 lists the ion concentrations of the squid giant axon (a model system common in the early days of neuroscience) as well as mammalian cells. Although there are large differences in the ion concentrations, the reversal potentials (when computed using eq. 2.1) are similar across species. Still, we find that, in general, no individual reversal potential exactly matches the resting potential. For example, in the squid, the resting potential is close to -62 mV at a temperature of 20°C (McCormick, 2014). This potential is slightly more positive than E_{K^+} and E_{Cl^-} , but much more negative than E_{Na^+} .

This suggests that the cell membrane is permeable to several ion species at the same time, but at different “permeabilities”. The membrane potential converges to a new equilibrium state E somewhere between the original reversal potentials. Mathematically, we summarise the relative permeability of the cell membrane for an ion species X as a quantity P_X . Instructively, P_X can be interpreted as the total number of open ion channels for an ion species X .

We can compute the overall equilibrium potential using the Goldman-Huxley-Katz equation (Goldman, 1943; Hodgkin and Katz, 1949). For the three ion species listed above we have

$$E = \frac{RT}{F} \log \left(\frac{P_{\text{K}^+}[\text{K}^+]_{\text{out}} + P_{\text{Na}^+}[\text{Na}^+]_{\text{out}} + P_{\text{Cl}^-}[\text{Cl}^-]_{\text{in}}}{P_{\text{K}^+}[\text{K}^+]_{\text{in}} + P_{\text{Na}^+}[\text{Na}^+]_{\text{in}} + P_{\text{Cl}^-}[\text{Cl}^-]_{\text{out}}} \right). \quad (2.2)$$

For the squid giant axon, the permeability ratios $P_{\text{K}^+} : P_{\text{Na}^+} : P_{\text{Cl}^-}$ can be experimentally determined to be about $1 : 0.04 : 0.45$ —in its resting state, the membrane is strongly permeable to potassium, but only weakly so for sodium (McCormick, 2014). Plugging these numbers into eq. (2.2) results in the resting potential $v_{\text{rest}} \approx -62$ mV.

Equivalent circuit model. Alternatively, the equilibrium potential can be modelled in terms of the equivalent circuit depicted in Figure 2.6. The idea is the following. An ion species X is “driven” by the voltage difference between the membrane potential v and the reversal potential E_X (eq. 2.1 and Table 2.1). Each ion channel provides a conductive path for a specific ion species X ; however, the channels are tiny, and ionic movement is influenced by thermal Brownian noise. Hence, there is a chance that ions will collide with the cell interior before passing through a channel. This provides a resistance R_X to the ionic current (Enderle, 2011).

Each ion channel can thus be modelled electrically as a resistor-voltage-source pair. The voltage source corresponds to the driving forces that move ions in or out of the cell, whereas the resistor models the stochastic resistance ions encounter while moving through the cell.

Intuitively, the more ion channels that are open—and the larger the permeability P_X —the smaller this resistance. Assuming that P_X represents the number of open channels for an ion species X , and that ion channels behave Ohmically, the total ionic current J_X is (Figure 2.6A)

$$J_X(v) = \sum_{i=1}^{P_X} \frac{v - E_X}{R_X} = \frac{P_X}{R_X} (v - E_X) = g_X(v - E_X), \quad (2.3)$$

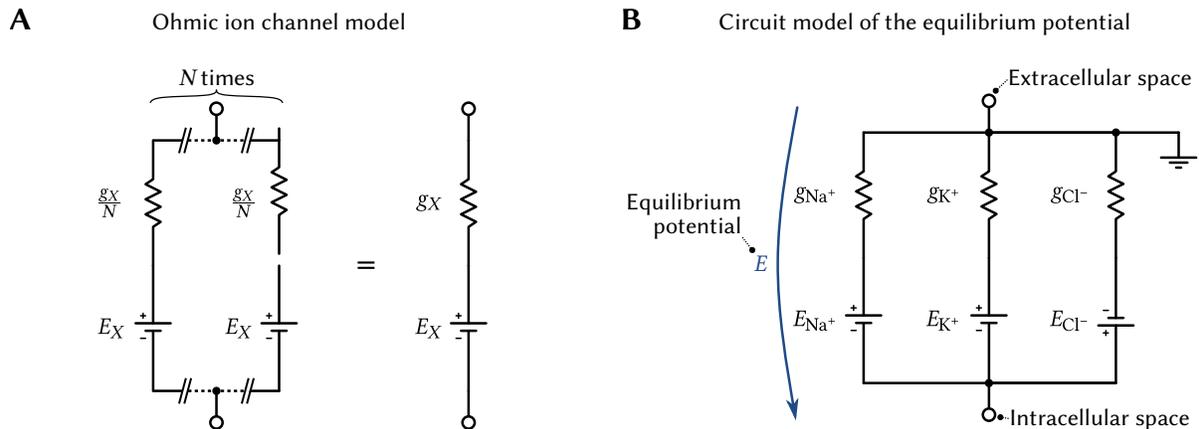


Figure 2.6: Electrical model circuit of the cell membrane. **(A)** Individual ion channels can be modelled as a resistor-voltage-source pair (left). Instead of modelling all ion channels for the same ion species X independently, they can be mathematically combined into a single pair (right). **(B)** Circuit model of a neuron at rest. The ratios of the conductances g_{K^+} , g_{Na^+} , and g_{Cl^-} determine the cell’s equilibrium potential E . Arrows indicate the current direction relative to $E = 0$ V (not physical ionic flow).

where g_X is the *conductance* of a “virtual” channel summarising all P_X channels.⁶ To obtain the equilibrium potential E for multiple ion channel types, we arrange the resistor-voltage-source pairs in parallel (Figure 2.6B). According to Kirchoff’s circuit laws we have

$$\sum_X J_X(E) = \sum_X g_X(E - E_X) = 0 \iff E = \frac{\sum_X g_X E_X}{\sum_X g_X}, \quad \text{for } \sum_X g_X > 0. \quad (2.4)$$

And for the ion species discussed above we get

$$E = \frac{g_{K^+} E_{K^+} + g_{Na^+} E_{Na^+} + g_{Cl^-} E_{Cl^-}}{g_{K^+} + g_{Na^+} + g_{Cl^-}}, \quad \text{for } g_{K^+} + g_{Na^+} + g_{Cl^-} > 0. \quad (2.5)$$

Put differently, the equilibrium potential is modelled as a weighted sum of reversal potentials, with the weights being the channel conductances. However, note that eqs. (2.3) and (2.4) assume a linear relationship between permeabilities and conductances. Mathematically, conductance and permeability are two distinct concepts (Enderle, 2011). Still, although linearity does not follow from the empirically well-tested Goldman equation, the linear approximation preserves the overall qualitative behaviour.

Correspondingly, the above “equivalent circuit” forms the basis of most neuron models. The model is simple, conductances can be fit to experimental data, and—as we will see next—the equivalent circuit lends itself to a dynamical description of the cell membrane by simply adding a capacitor to the model circuit.

⁶The conductance g of a resistor (measured in Siemens; unit symbol $S = \Omega^{-1}$) is the inverse of its resistance R . This unit is commonly used in neuroscience to simplify some equations. For example, a conductance of zero can be used to describe an open circuit, whereas resistances would take on unwieldy infinite values in this case.

2.1.3 Neural Dynamics and the Hodgkin-Huxley Model

We can now describe the steady-state membrane potential of a neuron at rest. However, the mechanisms discussed so far neither tell us how the membrane potential evolves over time, nor do they describe the action potential—the phenomenon distinguishing most neurons from other cells in the first place.

Fundamentally, neural dynamics can be summarised as follows. Assume that we inject short positive current pulses $J(t)$ into a neuron using a microelectrode. Measuring the membrane potential $v(t)$ over time t we observe the following:

1. *Subthreshold dynamics.* Small current pulses charge the cell membrane over time. The membrane potential v rises while the input persists, and then decays back to v_{rest} (Figure 2.7A).
2. *Superthreshold dynamics.* If the current pulse is energetic enough for the membrane potential to reach a soft threshold value v_{th} , the neuron generates an action potential; v suddenly rises towards positive numbers (*depolarisation*), and then falls to voltages below v_{rest} (*repolarisation*)—the neuron is *hyperpolarised*. Over time, v converges back to v_{rest} . This is depicted in Figure 2.7B.
3. *Refractory period.* While the neuron is hyperpolarised, it is significantly harder to evoke another action potential. This phase is referred to as the *refractory period* (Figure 2.7C).⁷

The first model to provide a detailed mechanistic account of these observations—and indeed what we used as a stand-in for a real neuron in the above exploration—is the Hodgkin-Huxley model (Hodgkin and Huxley, 1952). This model has been exceptionally successful in predicting neural behaviour quite accurately, and forms a basis for a whole family of neuron models (Meunier and Segev, 2002; McCormick, Shu, and Yu, 2007).

The important idea of Hodgkin-Huxley-type models is that the channel conductances g_{K^+} and g_{Na^+} possess dynamics that nonlinearly depend on the current membrane potential.⁸ That is, the permeability of the membrane for potassium and sodium changes depending on the membrane potential due to voltage-dependent ion channels. Hodgkin and Huxley model the dynamics of these channels using three dimensionless “gating” variables $m(t)$, $h(t)$, $n(t)$. Revised models differ from the original in terms of the concrete dynamics of these variables. In our examples, we use dynamics adapted from a model of pyramidal cells in the hippocampus described by Traub and Miles (1991, Chapter 4, pp. 92-94).

Conveniently, Hodgkin-Huxley-type models are an extension of the equivalent cell-membrane circuit from the previous subsection (Figure 2.6). We just need to make two conceptual changes to account for the linear subthreshold and nonlinear superthreshold dynamics.

⁷Technically, modellers distinguish between *absolute* and *relative* refractory periods. During the absolute refractory period, the neuron cannot produce action potentials; during the relative refractory period, merely large input currents are required (Izhikevich, 2007, Section 2.3.2).

⁸Hodgkin-Huxley-type models are sometimes also called “conductance-based neurons”. This is not to be confused with “conductance-based synapses”—synapse models are independent of the neuron model.

2.1. Morphology and Physiology of Individual Neurons

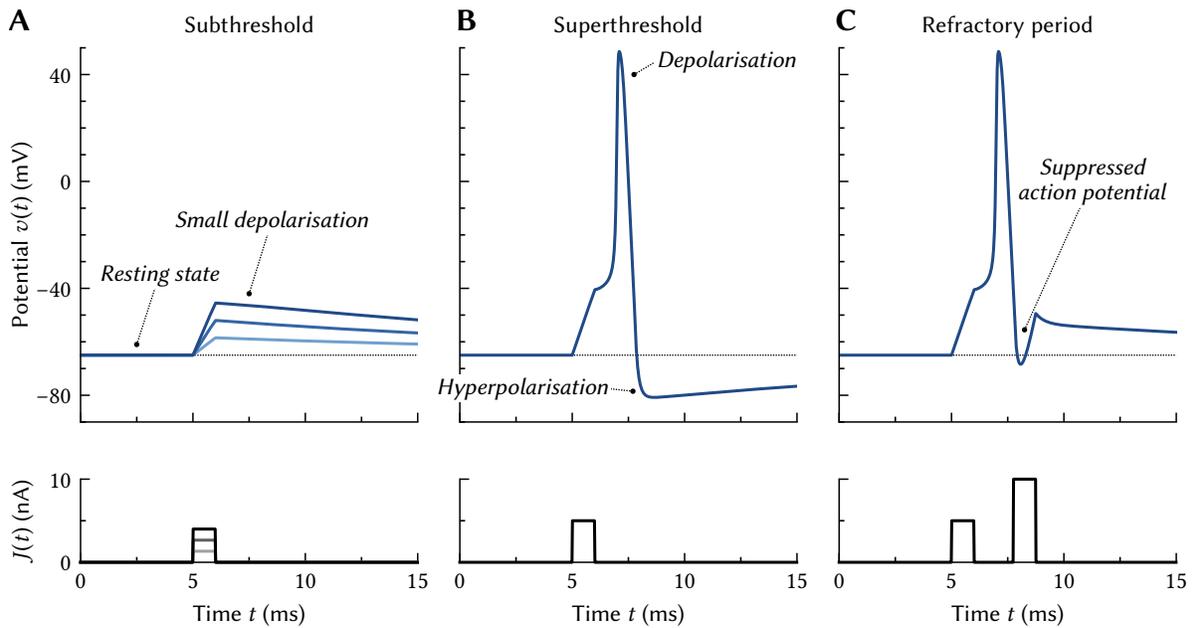


Figure 2.7: Hodgkin-Huxley-type model neuron simulations of action potential generation (using the dynamics described by Traub and Miles, 1991). **(A)** Injecting a current J into the neuron (*bottom*) charges the cell membrane. As long as the membrane potential v (*top*) approximately stays below a threshold, the neuron does not generate action potentials; the membrane acts linearly. **(B)** Above a certain threshold, the neuron suddenly generates an action potential. **(C)** Directly after an action potential, during the *refractory period*, even large input currents cannot evoke action potentials.

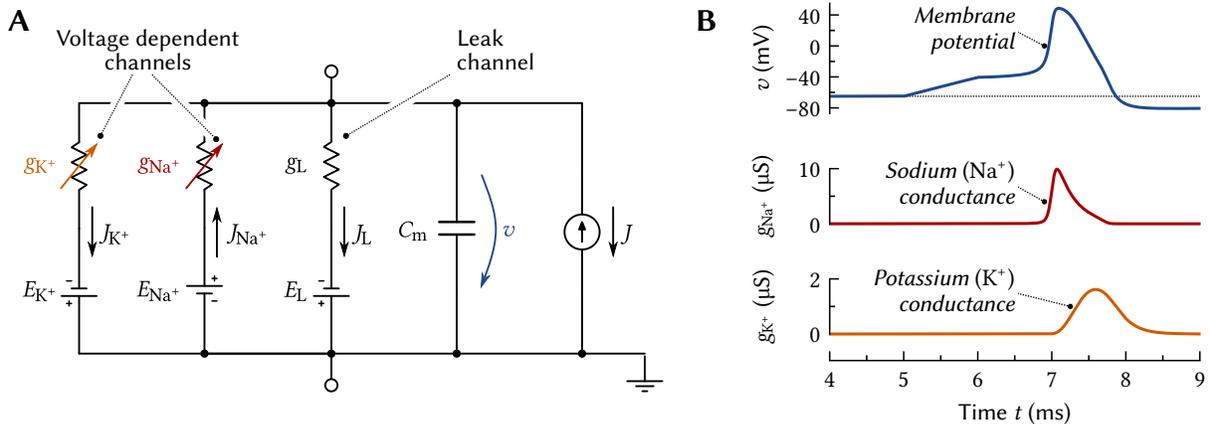


Figure 2.8: Equivalent circuit diagram of the Hodgkin-Huxley model. **(A)** Equivalent circuit of the Hodgkin-Huxley model. The resting potential is generated by a virtual “leak” channel; the K^+ and Na^+ channels vary over time and depend on the membrane potential. **(B)** Conductance traces for a single action potential (slice from Figure 2.7B). Sodium drives depolarisation, potassium repolarisation.

Subthreshold dynamics. To model the linear subthreshold dynamics, we simply add a capacitor with capacitance C_m to the circuit model. This *membrane capacitance* is physically motivated by the cell membrane separating two electrically charged bodies. Any such system inherently possesses capacitive properties.⁹ Given an input current $J(t)$, the sub-threshold dynamics can be described in terms of the following differential equation

$$\begin{aligned} C_m \dot{v}(t) &= g_{K^+} (E_{K^+} - v(t)) + g_{Na^+} (E_{Na^+} - v(t)) + g_{Cl^-} (E_{Cl^-} - v(t)) + J(t) \\ &= g_L (E_L - v(t)) + J(t), \end{aligned} \quad (2.6)$$

where the constant conductances and reversal potentials are summarised in terms of a *leak conductance* g_L and *leak potential* E_L that form a basic resistor-capacitance (RC) circuit (right half of Figure 2.8A) with the time-constant $\tau_m = C_m g_L^{-1}$. We have

$$g_L = g_{K^+} + g_{Na^+} + g_{Cl^-}, \quad \text{and} \quad E_L = \frac{g_{K^+} E_{K^+} + g_{Na^+} E_{Na^+} + g_{Cl^-} E_{Cl^-}}{g_{K^+} + g_{Na^+} + g_{Cl^-}}.$$

The equation for E_L is exactly the equilibrium potential as per eq. (2.5). In the absence of an input current $J(t)$, E_L is a stable attractor in the subthreshold dynamics. It typically holds $v_{\text{rest}} = E_L$; still, we use E_L in this context to emphasise its use as a conductance-based channel.

Superthreshold dynamics. The superthreshold dynamics are responsible for action potential generation. We describe these nonlinear dynamics in terms of potassium and sodium channels with time-dependent conductances $g_{K^+}(t)$, $g_{Na^+}(t)$ (Figure 2.8A):

$$C_m \dot{v}(t) = g_{K^+}(t) (E_{K^+} - v(t)) + g_{Na^+}(t) (E_{Na^+} - v(t)) + g_L (E_L - v(t)) + J(t).$$

Hodgkin and Huxley define the time-course of these conductances in terms of a dynamical system of gating variables $m(t)$, $h(t)$, and $n(t) \in [0, 1]$ that in turn depend on $v(t)$. We have

$$g_{K^+}(t) = \hat{g}_{K^+} n(t)^4, \quad g_{Na^+}(t) = \hat{g}_{Na^+} m(t)^3 h(t),$$

where \hat{g}_{K^+} and \hat{g}_{Na^+} are the maximum conductances. The dynamics of the gating variables produce a tight feedback loop between $v(t)$ and the conductances.

Figure 2.8B depicts the resulting conductances over time. In particular, $g_{Na^+}(t)$ rises rapidly once a certain threshold potential is exceeded. This drives the cell towards the positive sodium reversal potential E_{Na^+} ; the cell becomes depolarised. In turn, the positive membrane potential triggers a rise in potassium conductivity $g_{K^+}(t)$, while the gating variable $h(t)$ shuts the sodium current off. This drives the cell towards the negative potassium reversal potential E_{K^+} ; the cell becomes hyperpolarised, while the potassium conductance remains relatively high for a short time. This accounts for refractoriness, as large J are required to counter the potassium current. Interested readers may find a thorough discussion of the dynamics of Hodgkin-Huxley-like neurons in Izhikevich (2007).

⁹To be clear, the membrane capacitance is not specific to the Hodgkin-Huxley model. In fact, this assumption preceded the Hodgkin-Huxley model by several decades (e.g., Lapicque, 1907). We discuss the corresponding LIF neuron model first proposed by Lapicque later in this chapter.

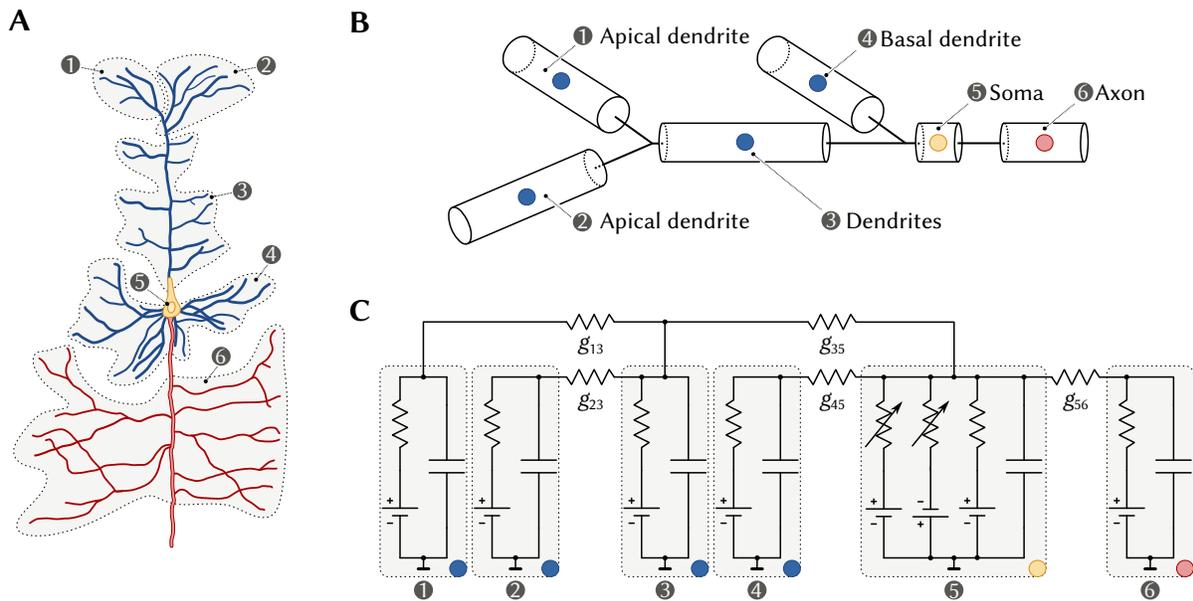


Figure 2.9: Equivalent circuit diagram of an exemplary compartmental neuron model. **(A)** A biological neuron is divided into $N = 6$ individual compartments (neuron redrawn from Howell, 1916, Figure 84, p. 187). **(B)** Compartments are often approximated as cylinders; using “cable theory” one can compute how potentials propagate along the membrane. **(C)** Cylinder models can be further simplified by modelling each compartment as a simple equivalent circuit. Here, the somatic compartment (orange) possesses Hodgkin-Huxley-like voltage-dependent dynamics. Individual model circuits are resistively coupled with conductances g_{ij} .

2.1.4 Compartmental Neuron Models

As we saw at the beginning of this section, neurons possess intricate morphologies (cf. Figure 2.9A). Up to this point, we have minimised this fact and instead summarised neural electrophysiology in terms of a single membrane potential over time, $v(t)$. Models based on this assumption are referred to as *point neurons* or *single compartment models*.

As we discuss in more detail in Chapter 3, the spatial organisation of a neuron can have a significant impact on its function. This is because the intracellular fluid is not a particularly good electrical conductor. Combined with the capacitive properties of the membrane, we can, at a single point in time, measure different membrane potentials throughout the neuron. These voltage differences cause currents injected at different points of the neuron to interact nonlinearly, which supports computation. Perhaps confusingly, the neural *dynamics* we describe here are mostly linear, but the *average effect over time* is nonlinear.

A quite faithful model of this can be constructed using *cable theory*. To this end, individual neurons are approximated as an arrangement of cylinders that represent individual segments of the neuron (Figure 2.9B). Voltages spatially propagate through these cylinders. That is, the model takes the longitudinal resistance of the individual neuron segments as well as their

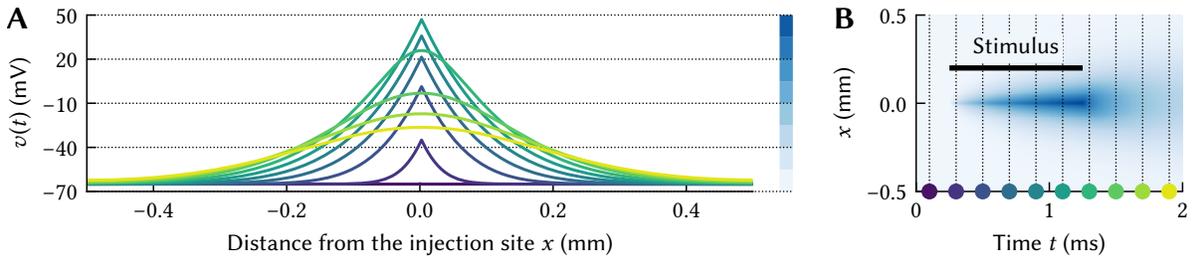


Figure 2.10: Illustration of potential propagation within a cylindrical cable. Injecting a 1 nA current at the centre of a 1 mm long cable (diameter $1\ \mu\text{m}$, capacitance $1\ \mu\text{F cm}^{-2}$, longitudinal resistance $150\ \Omega\ \text{cm}^{-1}$, leak conductance $0.5\ \text{mS cm}^{-2}$). **(A)** Potential at different points x and times t (see **(B)** for a legend) in a linear cable. **(B)** Continuous representation of the same data; see **(A)** for the mapping between colours and voltages.

capacitive properties into account. Given an input impulse, we can use *cable equations* to predict the voltage $v(x, t)$ at a certain position x along the segment at a time t (cf. Figure 2.10). Mathematically, this can be accomplished by assuming an infinite number of resistively coupled RC circuits and solving a partial differential equation (Koch, 1999, Chapter 2).

Instead of modelling spatial propagation of voltages through cylinders, individual cylinders can be approximated as one or multiple instances of the underlying equivalent circuit. Each of these instances is referred to as a *compartment* (Figure 2.9C). Such neuron models are correspondingly called *multi-compartment* or *compartmental* neuron models. In theory, the number of compartments N determines how closely the model can be made to match empirical data—some detailed neuron models rely on thousands of individual segments. However, for most network-level research, compartments can often be joined into reduced models that capture most of the original neural behaviour (Herz et al., 2006).

Depending on the modelling assumptions, compartments may either be *active* or *passive*. Active compartments—such as the “somatic” compartment in Figure 2.9C—model superthreshold dynamics and are capable of producing spikes, for example by including voltage-dependent Hodgkin-Huxley type models. In contrast, passive compartments are typically modelled as the simple linear subthreshold RC-circuit. A more detailed description of such models, including the passive dendritic trees we discuss later, can be found in Koch (1999, Chapter 3) and Gerstner and Kistler (2002, Chapter 2).

If we assume that individual neural compartments are resistively coupled, the current $J_i(t)$ flowing into the i th compartment is, according to Kirchhoff’s circuit laws, given as

$$J_i(t) = \sum_{j=1}^N c_{ij} (v_j(t) - v_i(t)). \quad (2.7)$$

Here, $c_{ij} = c_{ji}$ is the conductance between the i th and j th compartment. A value of $c_{ij} = 0$ indicates no connection. Mathematically, the conductances c_{ij} correspond to the symmetric adjacency matrix of an undirected connectivity graph describing the compartmental model.

2.2 From Individual Neurons to Spiking Neural Networks

The equations from the previous section model the basic electrophysiological properties of individual neurons. Of course, a single neuron on its own does not give rise to animal behaviour. Hence, in this section, we discuss how individual neurons can be used to form neural networks. We provide an overview of synaptic transmission, the process underpinning neural information transfer, followed by a discussion of idealised neuron and synapse models that lessen the computational burden of simulating biological systems. We close with a short discussion of spiking neural networks and population tuning—the latter shedding some light onto the role that individual neurons play in a network.

2.2.1 Synaptic Transmission

So far, our descriptions of neural dynamics assumed that neurons receive inputs through artificially injected currents $J(t)$. Of course, this does not explain how neurons receive inputs *in vivo*. As we mentioned in our overview in Section 2.1.1, the interface between two neurons is called a *synapse*. Neuroscientists distinguish two synapse types: electrical and chemical.

Electrical synapses. In adult vertebrates, electrical synapses are much less common than chemical synapses. Still, they play an important role in some mammalian brain areas, including the retina and the inferior olive in the cerebellum (Meriney and Fanselow, 2019, Chapter 5).

Electrical synapses directly connect two neurons through specialised ion channels, so-called *gap junctions*. These channels typically allow a bidirectional exchange of ions between the intracellular fluids. Correspondingly, neurons connected via gap junctions can be modelled using the same techniques as multi-compartment models (eq. 2.7; Kandel et al., 2012, Chapter 8).

Chemical synapses. Neurons coupled via chemical synapses are electrically isolated. The synapse establishes a unidirectional information flow and only passes on superthreshold signals. The transmitting and receiving neurons are generally referred to as “pre-synaptic” and “post-synaptic neurons”, respectively. To save some space, we use the terms pre- and post-neurons. Similarly, the synapse is divided into the “pre-” and “post-synapse”.

As illustrated in Figure 2.11, action potentials arriving at axon terminal, trigger the release of *neurotransmitter* molecules. These traverse the nanometer-scale gap between the pre- and post-synapse, the *synaptic cleft*, and temporarily bind to post-synaptic receptors specific to that neurotransmitter type. This—directly or indirectly—causes ion channels in the post-neuron to open. In turn, the permeability of the post-neuron for that particular ion species changes.

The resulting synaptically mediated membrane potential fluctuations are called *post-synaptic potentials* (PSPs). Depending on the type of ion channels opened or closed, the PSP can either drive the neuron towards more positive voltages (if sodium channels are opened), or negative voltages (e.g., if potassium or chloride channels are opened). Positive PSPs are referred to as *excitatory* (EPSP), negative changes as *inhibitory* (IPSP).

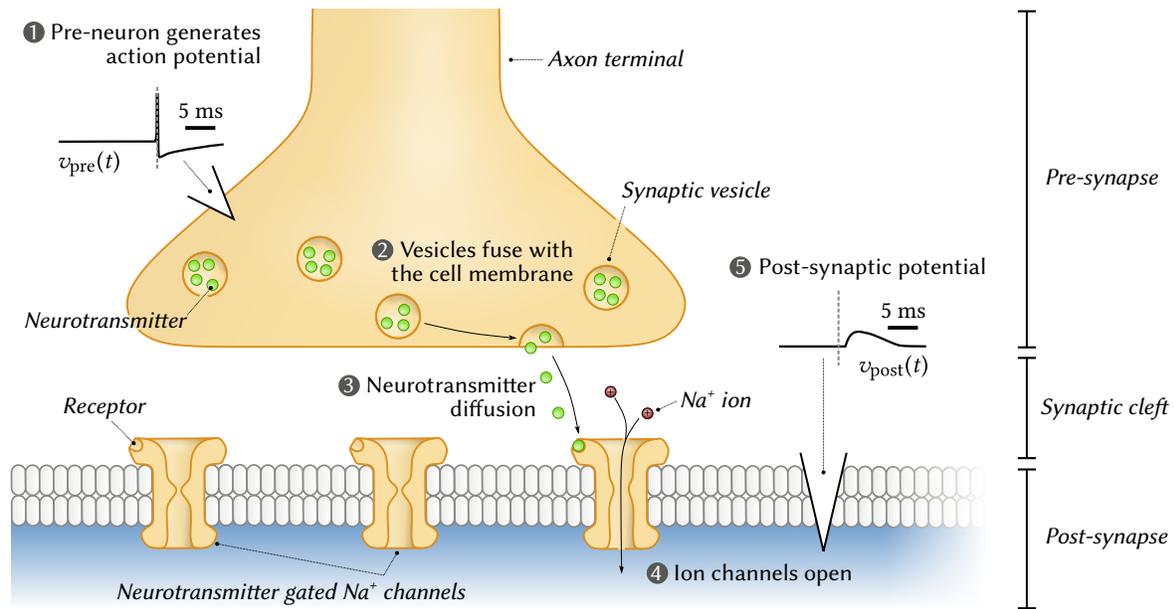


Figure 2.11: Synaptic transmission in a chemical synapse. Grey line in the membrane potential traces is the spike onset. See text for a description. Inspired by Kandel et al. (2012, Figure 8-8B-C, p. 185).

Whether a pre-neuron acts excitatorily or inhibitorily on a post-neuron depends on the kind of neurotransmitter released by the pre-neuron, as well as the specific receptors and ion channels in post-neuron. For example, the Gamma-Aminobutyric acid (GABA) neurotransmitter is primarily inhibitory (via $GABA_A$ or $GABA_B$ receptors), whereas Glutamic acid (Glutamate) is mostly excitatory (via AMPA or NMDA receptors; Kandel et al., 2012, Chapter 10).

Importantly, the mixture of neurotransmitters is the same across all axonal branches of a neuron. This is known as *Dale's principle* (Strata and Harvey, 1999; Eccles, 1986). Put simply, individual neurons typically either act excitatorily or inhibitorily on their post-neurons.¹⁰ We hence refer to neurons—and not just individual synapses—as being excitatory or inhibitory.

Conductance-based chemical synapse model. Most gated ion channels can be modelled as a voltage-resistor pair with variable conductance per receptor type (Roth and Rossum, 2009). This is depicted in Figure 2.12 for generic “excitatory” and “inhibitory” receptors. Mathematically, the post-synaptic current $J_{syn}(t)$ flowing from the synapse into the neuron is given as

$$J_{syn}(t) = g_E(t)(E_E - v(t)) + g_I(t)(E_I - v(t)). \quad (2.8)$$

Here, E_E and E_I are the equilibrium potentials of the gated ion channels. These potentials are not necessarily equal to ion reversal potentials (Table 2.1). For example, NMDA and AMPA channels conduct both Na^+ and K^+ , resulting in $E_E \approx 0$ mV (Kandel et al., 2012, Chapter 10).

¹⁰This characterisation of Dale's principle is useful, but technically incorrect. For example, a pre-synapse releasing glutamate typically excites post-neurons, but inhibits special neurons with inhibitory glutamate receptors (Cleland, 1996). Additionally, synapses could release both excitatory and inhibitory neurotransmitters.

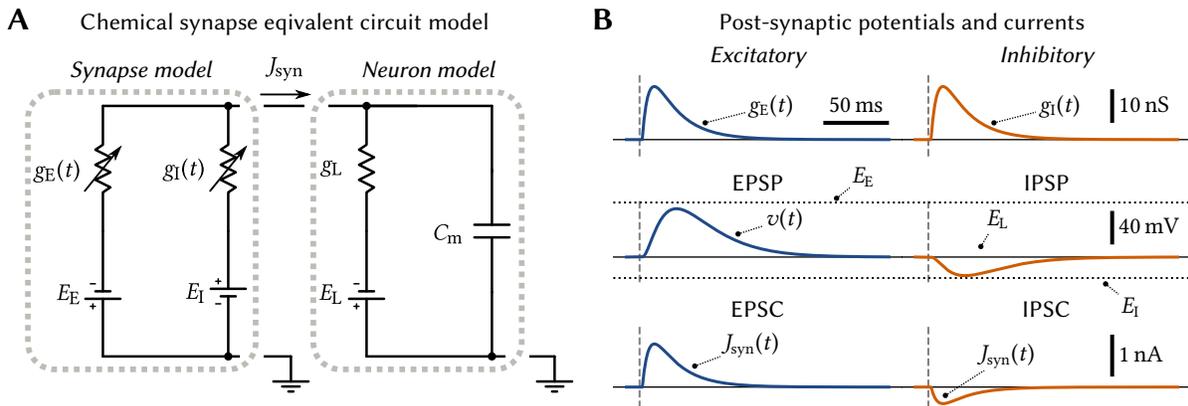


Figure 2.12: The conductance-based synapse model. **(A)** Circuit diagram of a passive cell membrane with synaptically controlled excitatory and inhibitory conductance-based channels. **(B)** Illustrative voltage, current, and conductance traces. *Top:* Excitatory and inhibitory channel conductances after a pre-synaptic action potential (grey dashed line). *Middle:* Resulting excitatory and inhibitory post-synaptic potentials. *Bottom:* Corresponding excitatory and inhibitory post-synaptic currents.

Synaptic dynamics and weights. The time course of the conductance $g(t)$ depends on the neurotransmitter and receptor type. Generally speaking, as illustrated in Figure 2.12B, the conductance rises quickly after a spike is received, and decays slowly as the neurotransmitter unbinds from the receptors. Receptors directly controlled by neurotransmitters acting as ligands (Figure 2.11) tend to have relatively short time-constants between 1 and 100 ms (Jones and Bekolay, 2014). Some receptors (e.g., “metabotropic receptors”) rely on indirect chemical cascades, leading to longer time-constants (Meriney et al., 2019, Chapter 14); this is not captured well by the models we present here (Roth et al., 2009).

Ligand-gated receptor dynamics are often modelled as a second-order linear dynamical system. Let t_{ij} denote the time of the j th pre-synaptic spike arriving from the i th pre-neuron, and let τ_1 , τ_2 denote the rise and delay times (Figure 2.13A). We have (Roth et al., 2009):

$$\dot{g}_1(t) = -\frac{g_1(t)}{\tau_1} + g_2(t), \quad \dot{g}_2(t) = -\frac{g_2(t)}{\tau_2} + \sum_i \alpha w_i \sum_j \delta(t_{ij} - t), \quad (2.9)$$

where $\delta(t)$ is the Dirac-delta, and α is a scaling factor such that the scalar w_i corresponds to the peak value of $g_1(t)$ for a single action potential received from pre-neuron i (Figure 2.13B). For $\tau_1 = \tau_2$ this type of synaptic dynamics follows the so-called *alpha function*.

Notably, the scalar w_i is referred to as *synaptic weight* in computational modelling, or as *synaptic strength* in neuroscience. This scalar summarises various biological processes that determine the average response of the post-neuron to an action-potential received from the i th pre-neuron. For example, the synaptic strength can depend on the amount of neurotransmitter released in the pre-synapse, or the ion-channel density in the post-synapse (Kandel et al., 2012, Chapter 12). In this context, the term *synaptic plasticity* refers to a change in synaptic strength over time. This plays a key role in learning—we discuss this in more detail in Chapter 5.

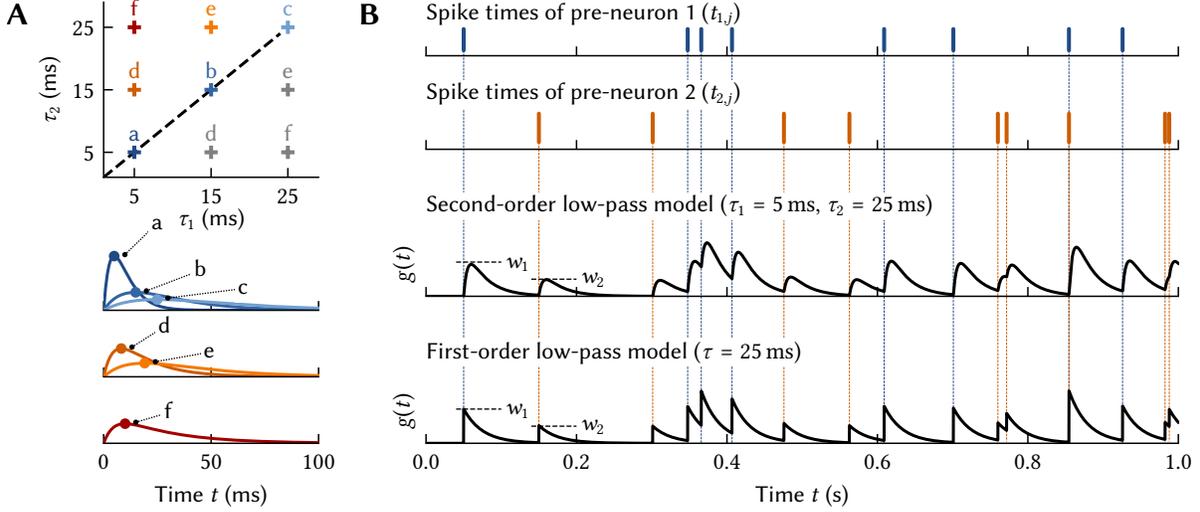


Figure 2.13: Illustration of the second- and first-order exponential synaptic low-pass filter dynamics. **(A)** Impulse response of the second-order dynamics for different τ_1 and τ_2 . *Top:* visualisation of the time-constants used below. Swapping τ_1 and τ_2 does not change the dynamics (grey crosses). *Bottom:* Impulse response of the filters (responses normalised to equal area), circles are the peaks. **(B)** Illustration of post-synaptic conductance traces (*bottom*) according to Equations (2.9) and (2.10) for two pre-neurons producing action-potentials (*top*) connected with two different synaptic weights w_1, w_2 .

Typically, the rise-time of $g(t)$ is relatively short. Correspondingly, the second-order low-pass filter can be approximated by a first-order dynamical system with decay-time τ . Mathematically, and as depicted the lower portion of Figure 2.13B, the dynamics are

$$\dot{g}(t) = -\frac{g(t)}{\tau} + \sum_i w_i \sum_j \delta(t - t_{ij}). \quad (2.10)$$

Synaptic dynamics as filters. The above synaptic dynamics (eqs. 2.9, 2.10) are linear time-invariant (LTI) systems. LTI systems are fully characterised by their *impulse response* $h(t)$ and can be replaced by a convolution (“*”) between $h(t)$ and the weighted input spike-train $u(t)$:

$$g(t) = (u * h)(t) = \int_0^\infty u(t - t')h(t') dt' = \int_0^\infty h(t') \sum_i w_i \sum_j \delta(t - t_{ij} - t') dt'. \quad (2.11)$$

For the first-order dynamics in eq. (2.10), the impulse response is $h(t) = \exp(-t/\tau)$ for $t \geq 0$ (and $h(t) = 0$ if $t < 0$); impulse responses of the second-order system are depicted in Figure 2.13A. Convolutions such as eq. (2.11) can alternatively be expressed as multiplication of the frequency contents of the impulse response $h(t)$ and the input spike-train $u(t)$:

$$g(t) = \mathcal{F}^{-1}(\mathcal{F}(h)\mathcal{F}(u))(t), \quad (2.12)$$

where \mathcal{F} is the forward Fourier transform, and \mathcal{F}^{-1} its inverse. From this signal-processing perspective, the synaptic dynamics attenuate certain frequencies in the input spike-train; they act as a *filter*. The synaptic dynamics discussed so far attenuate low frequencies far less than higher frequencies; they are thus referred to as *low-pass filters*.

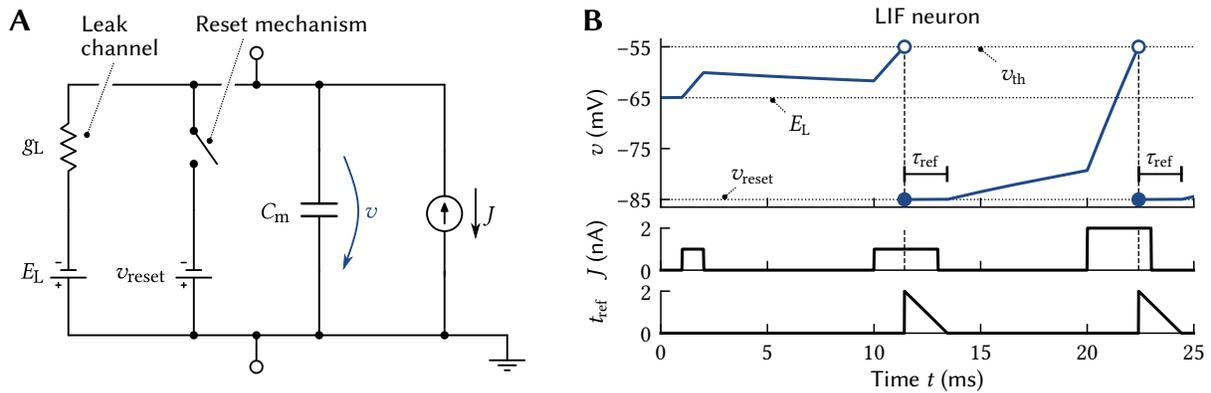


Figure 2.14: Equivalent circuit diagram and membrane potential trace of an LIF neuron. **(A)** The LIF model consists of a capacitive membrane with a “leak” channel. A reset mechanism clamps v to the reset potential v_{reset} . **(B)** Membrane potential trace (*top*) for rectangle pulse current inputs (*middle*). Vertical dashed lines are spike events. The bottom graph depicts the remaining refractory period.

2.2.2 Simplified Neuron Models

The models discussed in the previous section describe individual neurons at a relatively low level of abstraction. This incurs a high computational cost, especially when considering detailed compartmental neurons with Hodgkin-Huxley-like super-threshold dynamics.

Fortunately, important electrophysiological properties of neurons can be captured by less detailed, computationally inexpensive, models (cf. Koch, 1999, Chapter 14). For example, the Izhikevich (Izhikevich, 2004) and Adaptive Exponential (Brette and Gerstner, 2005) point neuron models require only two state variables, but are capable of reproducing spike patterns observed in nature. Here, we focus on the even simpler “leaky integrate-and-fire” (LIF) model.

The leaky integrate-and-fire model. A variant of the LIF model—lacking refractoriness—dates back to Lapicque (1907). Just like the Hodgkin-Huxley model (Section 2.1.3), the LIF neuron is based on a capacitive membrane with a leak channel. The input current is integrated by the capacitor, which, at the same time, is discharged through the leak channel:

$$C_m \dot{v}(t) = g_L (E_L - v(t)) + J(t), \quad (2.13)$$

where g_L is the leak conductance, E_L is the leak reversal potential, and $J(t)$ is a (synaptic) current injected into the membrane. Unlike the Hodgkin-Huxley model, the dynamics of the system do not model spike production; “firing” is handled outside the dynamical system. Once $v(t)$ reaches a threshold v_{th} , a spike event is recorded and v is held at the *reset potential* v_{reset} for the duration of the refractory period τ_{ref} . This is depicted in Figure 2.14.

Response curve. Figure 2.15 depicts a comparison between Hodgkin-Huxley-type and LIF dynamics. The reset and threshold potential of the LIF neuron model have been tuned to match the sub-threshold membrane potentials observed in the Hodgkin-Huxley neuron. In both

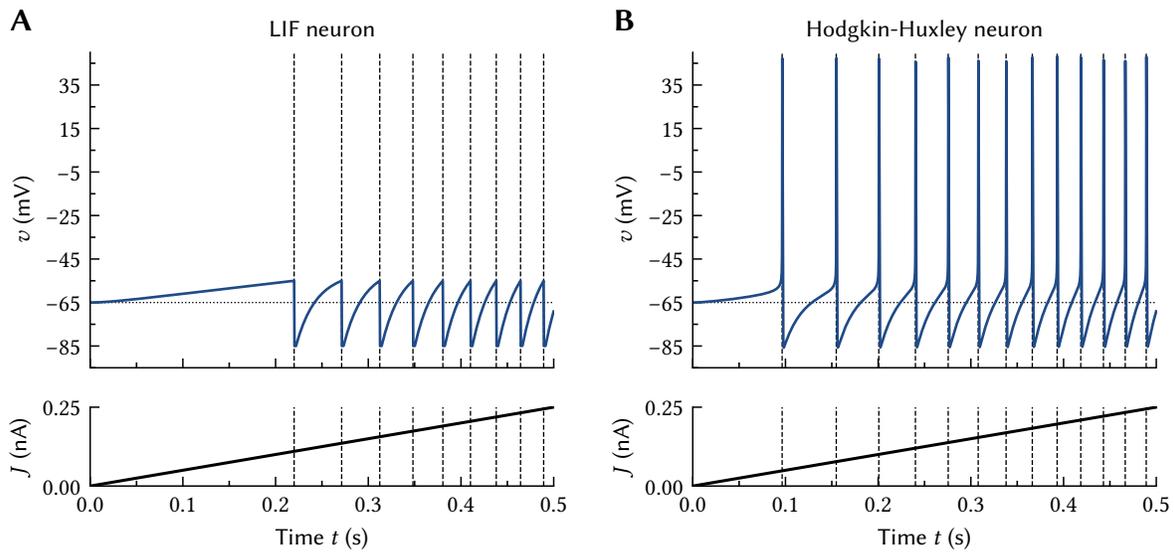


Figure 2.15: Voltage traces (*top*) for the LIF and a Hodgkin-Huxley-type neuron dynamics described by Traub and Miles (1991) given a current ramp input (*bottom*). Dashed vertical lines correspond to spike events. Dotted line is the resting potential at $E_L = -65$ mV. Both neurons have a membrane capacitance of $C_m = 200$ pF and a leak conductance of $g_L = 10$ nS. **(A)** The LIF neuron model does not account for spike production; the voltage is simply reset to $v_{\text{reset}} = -85$ mV for $\tau_{\text{ref}} = 2$ ms once the membrane potential passes the threshold at $v_{\text{th}} = -55$ mV. **(B)** Same experiment for a Hodgkin-Huxley neuron. Notably, the super-threshold dynamics of the Hodgkin-Huxley model action potentials and the first spike appears significantly earlier compared to the LIF neuron.

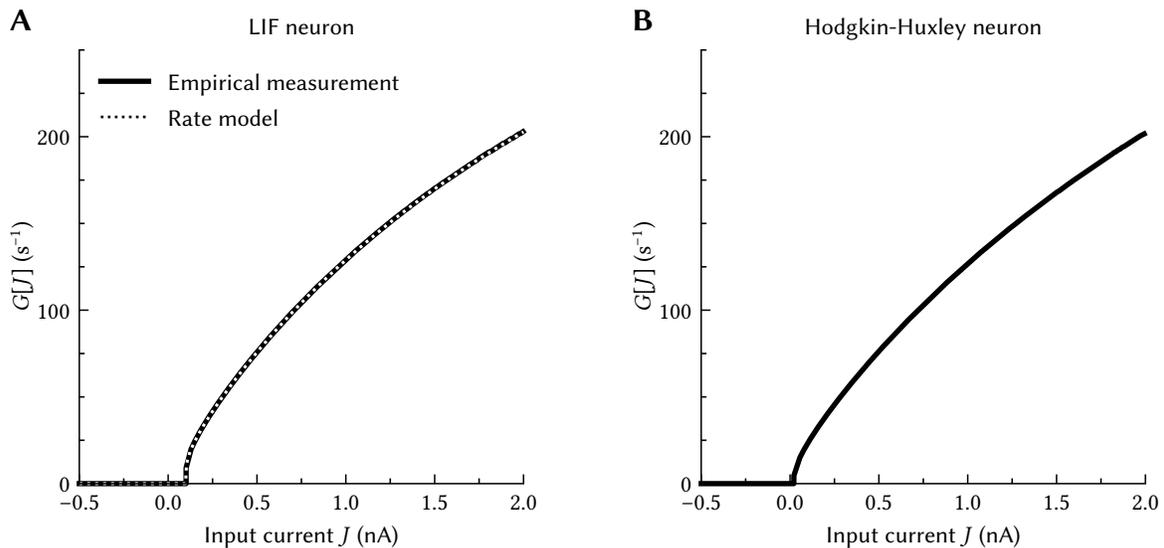


Figure 2.16: Response curves (also called IF-curves) $G[J]$ for the LIF and Hodgkin-Huxley-type neuron model from Figure 2.15. Data extracted from the inter-spike-intervals of a current ramp experiment (sweep from 0 nA to 2 nA over ten seconds at a $10 \mu\text{s}$ resolution). Although the LIF and Hodgkin-Huxley models have drastically different dynamics, their response curves are qualitatively similar.

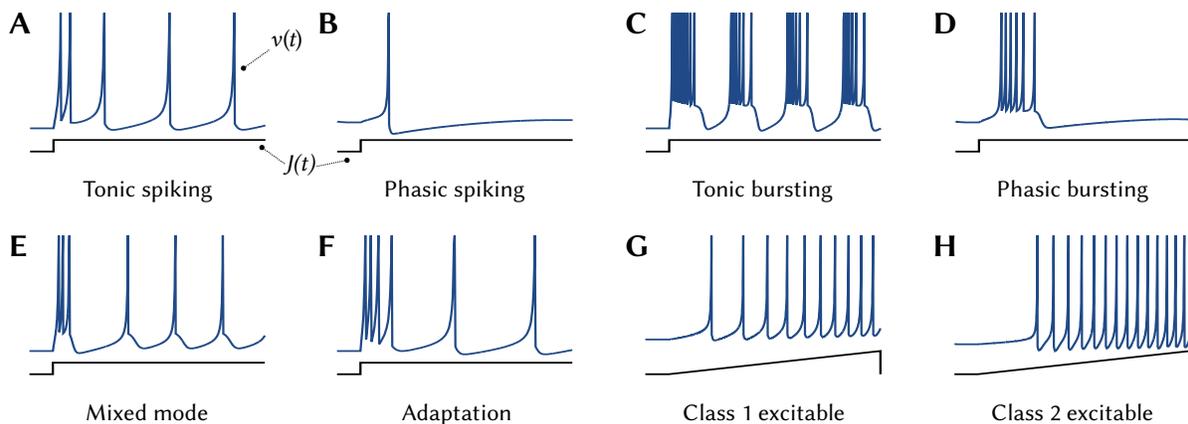


Figure 2.17: Diversity of neural dynamics for constant or ramp input currents. Each plot depicts membrane-potential traces $v(t)$ (top) of the Izhikevich neuron model (with different parameters) for an input current $J(t)$ (bottom). LIF neurons are only capable of behaviours A and G. Partially reproduced (dynamics for input pulses were skipped) with minor modifications from Izhikevich (2004). Electronic version of the figure and reproduction permissions freely available at <http://www.izhikevich.org>.

cases—apart from the obviously missing super-threshold dynamics—the inter-spike-interval decreases the larger J , leading to a higher spike rate. This suggests a characterisation of neurons in terms of their *response curve* $G[J]$ (Figure 2.16)

$$G[J] = \lim_{T \rightarrow \infty} \frac{n_{\text{spikes}}(J, T)}{T}, \quad \text{where } n_{\text{spikes}}(J, T) = \# \text{spikes for input } J \text{ over a period } T. \quad (2.14)$$

Of course, this only makes sense if we assume that the neuron has a non-zero steady-state activity for a constant superthreshold input current. This is not necessarily true; neurons can exhibit complex behaviours that violate this assumption (Izhikevich, 2004). For example, *phasic* neurons only produce a limited number of spikes in response to an input step (Figure 2.17B, D, E), while other neurons adapt to constant input currents over time, resulting in a reduction in frequency (Figure 2.17F). Still, neurons often behave *tonically* (Figure 2.17A, C, G, H), and can be reasonably well characterised in terms of a response curve.

Response curves and time-less “rate neurons” form the basis for *artificial* neural networks commonly used in machine learning. We discuss this in more detail later in this thesis. Furthermore, we revisit temporal properties of the response curve when we discuss temporal tuning curves in Chapter 4.

LIF response curve. For most neuron models, it is impossible to derive a closed-form expression for the average spike rate $G[J]$ given a constant current J ; there typically is no closed-form solution for $G[J]$. Instead, we have to rely on the numerical method suggested by eq. (2.14). That is, we simulate the neural dynamics over a sufficiently long time-period and count the number of spikes n_{spikes} .

The LIF neuron is a notable exception to this, as its subthreshold dynamics form a simple first-order linear dynamical system. Using basic techniques for solving differential equations, the membrane potential v at time t for a constant input current J is given as

$$v(t) = \left(1 - e^{-\frac{t}{\tau_m}}\right) \left(E_L + \frac{J}{g_L}\right) + e^{-\frac{t}{\tau_m}} v(0), \quad \Rightarrow \quad \lim_{t \rightarrow \infty} v(t) = E_L + \frac{J}{g_L}. \quad (2.15)$$

To compute the average spike rate, we need to know how long it takes to reach the threshold potential after a spike has been issued. To this end, we simply solve $v(t_{\text{spike}}) = v_{\text{th}}$ with $v(0) = v_{\text{reset}}$ for t_{spike} . Keeping in mind that the neuron is held at the reset potential for τ_{ref} seconds after every spike, and defining the threshold current $J_{\text{th}} = (v_{\text{th}} - E_L)g_L$, we have

$$G[J] = \begin{cases} 0 & \text{if } J \leq J_{\text{th}}, \\ \frac{1}{\tau_{\text{ref}} + t_{\text{spike}}} & \text{if } J > J_{\text{th}}, \end{cases} \quad \text{where } t_{\text{spike}} = -\tau_m \log \left(1 - \frac{(v_{\text{th}} - v_{\text{reset}})g_L}{(E_L - v_{\text{reset}})g_L + J}\right). \quad (2.16)$$

As depicted in Figure 2.16A, the predicted rate perfectly matches the empirical measurement. Notably, the LIF response curve is qualitatively similar to that of the Hodgkin-Huxley neuron (Figure 2.16B)—surprisingly so, as the dynamical systems differ wildly between the two models. This makes LIF neurons a reasonable choice for simulating large-scale neurobiological systems at small computational costs (Meunier et al., 2002).

Simplified LIF neuron. A qualitatively equivalent form of the LIF neuron is given by normalising the voltages such that $E_L = 0$ and $v_{\text{th}} = 1$, and furthermore setting $J_{\text{th}} = 1$:

$$\dot{v}(t) = -\frac{1}{\tau_m} v(t) + J(t), \quad G[J] = \begin{cases} 0 & \text{if } J \leq 1, \\ \frac{1}{\tau_{\text{ref}} - \tau_m \log\left(1 - \frac{1 - v_{\text{reset}}}{J - v_{\text{reset}}}\right)} & \text{if } J > 1. \end{cases} \quad (2.17)$$

The primary advantage of this form is the reduction in mathematical operations compared to eq. (2.13). This simplification (among other optimisations) is exploited by the neural network simulator “Nengo” to support large-scale spiking neural network simulations (Bekolay, Bergstra, et al., 2014). The downside is a reduced interpretability of the now unit-less $v(t)$ and $J(t)$.

Current-based synapse models. Another possible simplification—both in terms of computation and mathematical analysis—concerns the synapse model. In conductance-based synapses (Section 2.2.1) the current $J_{\text{syn}}(t)$ explicitly depends on the membrane potential $v(t)$. Going back to Figure 2.12B, one may notice that the post-synaptic currents can be approximated as a scaled version of the conductance (cf. Roth et al., 2009).

This suggests modelling the post-synaptic current J_{syn} as a linear superposition of filtered spike events. Let the synaptic weight w_i denote the peak post-synaptic current. Then, the first-order *current-based* synapse model is given as (analogously for higher-order filters)

$$\dot{J}_{\text{syn}}(t) = -\frac{J_{\text{syn}}(t)}{\tau} + \sum_i w_i \sum_j \delta(t_{ij} - t). \quad (2.18)$$

We compare conductance- and current-based synapse models in more detail in the next chapter.

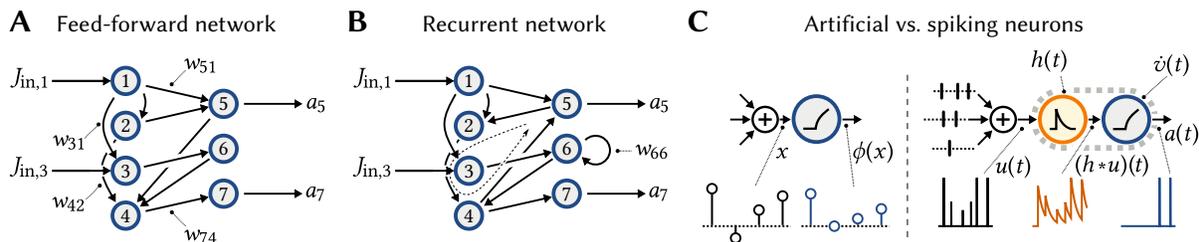


Figure 2.18: Illustration of different neural network types. Blue circles depict individual neurons. **(A)** Feed-forward neural networks can be represented as acyclic graphs. **(B)** If the network connectivity contains cycles (e.g., the 5-2-4-5 cycle), the network is called *recurrent*. Cycles can also be self-recurrences (e.g., neuron 6). **(C)** Artificial neurons (*left*) process time-discrete samples. Spiking neurons (*right*) are time-continuous; they receive a weighted input spike train, filter it, and produce an output spike train.

2.2.3 Spiking Neural Networks

We are now well-equipped to construct and simulate spiking neural networks (SNNs). In theory, we just take n spiking neurons and connect them up according to synaptic weights w_{ij} . As mentioned before, these weights determine the connection strength between a pre-neuron j and post-neuron i . Inputs are additive over multiple pre-neurons (eqs. 2.10, 2.18).¹¹

Mathematically, the *weight matrix* $\mathbf{W} \in \mathbb{R}^{n \times n}$ is the adjacency matrix of a directed connectivity graph.¹² Networks with acyclic connectivity are called *feed-forward* (Figure 2.18A); networks with cyclic connectivity are *recurrent* (Figure 2.18B). Feed-forward SNNs have finite memory. Assuming plausible synaptic filters, their impulse response decays exponentially. In contrast, and as we discuss later, recurrent networks can possess more interesting dynamics.

To actually simulate an SNN, we feed an external input (e.g., a set of currents or spike-trains) into the network and numerically integrate the coupled differential equations. There are numerous general-purpose simulator software packages that do this efficiently, e.g., NEST (Gewaltig and Diesmann, 2007), Brian (Stimberg, Brette, and Goodman, 2019), GeNN (Yavuz, Turner, and Nowotny, 2016), or Nengo (Bekolay, Bergstra, et al., 2014).

The most important question—and one that we pursue for the remainder of this thesis—is how to ultimately select \mathbf{W} . For the artificial neural networks (ANNs) used in machine learning (Figure 2.18C), the astoundingly successful answer is to use stochastic gradient descent (SGD) on a loss function E , i.e., $\dot{\mathbf{W}} = -\eta \nabla_{\mathbf{W}} E(\mathbf{a}_k, \hat{\mathbf{a}}_k)$, where η is a learning rate, \mathbf{a}_k is the desired output activity for an input sample \mathbf{x}_k , and $\hat{\mathbf{a}}_k$ is the actual output (LeCun, Bengio, and Hinton, 2015). For example, one common choice for the loss function is $E(\mathbf{a}_k, \hat{\mathbf{a}}_k) = \|\mathbf{a}_k - \hat{\mathbf{a}}_k\|^2$.

¹¹This notion of synaptic weights w_{ij} assumes that each neuron possesses a single synaptic input channel. Hence, there is only a single synaptic weight matrix. For neurons with multiple synaptic input channels we can simply use an individual weight matrix for each input channel. We discuss this in the next chapter.

¹²For convenience and computational efficiency, neurons are typically grouped into “layers”, “populations”, or “ensembles” (all terms are used synonymously). Not all layers have connections among each other. Hence, \mathbf{W} typically is a block-matrix that can be split into multiple sub-matrices.

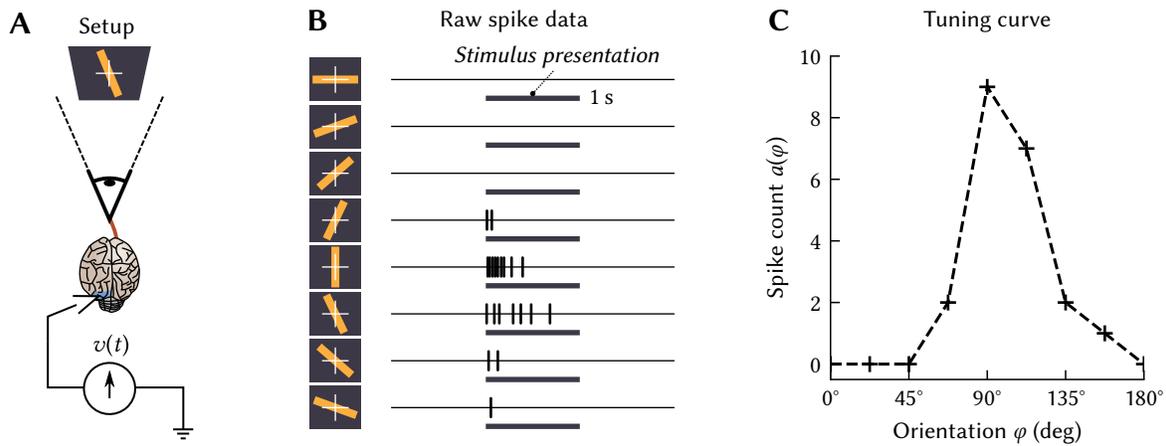


Figure 2.19: Neural tuning in the visual cortex. **(A)** Illustration of the original experiment by Hubel and Wiesel (1959). Bright rectangles are briefly presented on a dark screen to an animal while recording from a neuron in its visual cortex. **(B)** Recorded spike trains for different stimulus orientations. Horizontal lines are the stimulus presentation interval. Adapted from Hubel and Wiesel (1959, Figure 3). **(C)** The mapping between stimulus x and activity a forms a *tuning curve*. This neuron is tuned to vertical bars.

Similar gradient-descent global optimisation methods can be applied in the context of SNNs as well. Hunsberger (2018) characterises the LIF response curve $G[J]$ in a network context and then trains an ANN that uses $G[J]$ as a nonlinearity. The resulting \mathbf{W} can then be used in a spiking context. Göltz et al. (2021) optimise synaptic weights such that desired spike-timings are generated by a black-box spiking neural network simulator.

Notably, such global optimisation methods can be useful when modelling neurobiological systems. For example, there are striking similarities between the neural tuning observed in visual cortex and the learned tuning of neurons in hierarchical image classification networks (Yamins and DiCarlo, 2016). However, this is, to some degree, accidental, and not explicitly imposed. As we discussed at the beginning of this chapter, it can be desirable to build models that reliably adhere to certain constraints—such as the aforementioned tuning properties.

2.2.4 Neural Tuning and Population Codes

There are two important concepts that shed some light onto the organisation of brain networks—at least those brain networks involved in sensory processing: *tuning curves* and *receptive fields*. In turn, these ideas motivate *population codes*. We heavily rely on these codes in the context of our modelling tool, the Neural Engineering Framework (cf. Section 2.3).

Tuning curves. In a famous 1959 experiment (cf. Figure 2.19), Hubel and Wiesel present a visual stimulus in the form of a bright rectangle at different orientations φ to an experimental animal. At the same time, the activity of a neuron in the animal’s visual cortex is recorded. This characterises how the stimulus propagates through the brain up to the recording site.

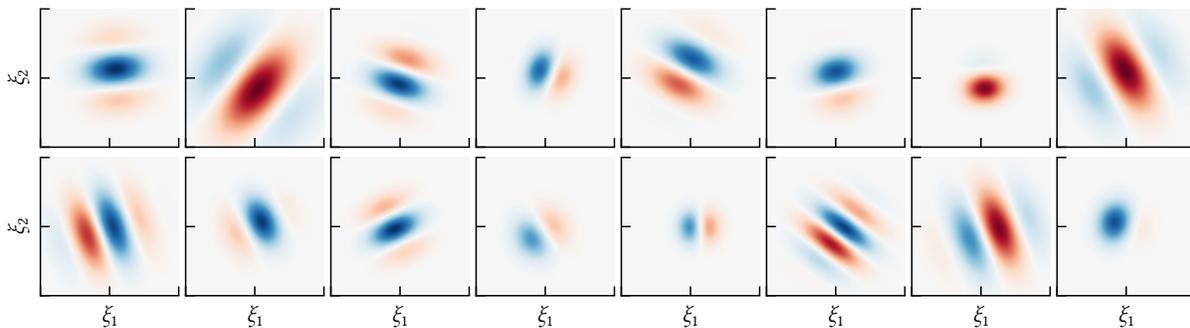


Figure 2.20: Examples of randomly generated Gabor filters. Each filter is a two-dimensional function $e(\xi_1, \xi_2)$, where (ξ_1, ξ_2) is a spatial location. Negative values (red) indicate that neural activity is reduced if a stimulus is present at that location, positive values (blue) indicate an increase in activity.

In the recording depicted in Figure 2.19, we find that the neuron is most sensitive to—or, *tuned* to—a certain orientation φ . We could say that in this specific case, vertical bars are the *preferred stimulus* of the neuron. Generally, such a mapping from a stimulus property φ onto the average neural activity $a(\varphi)$ is called a *tuning curve* (e.g., VandenBos, 2015, p. 1112).

Receptive fields. The concept of *receptive fields* is closely related to tuning curves. This is best illustrated in the context of stimuli that can be described as an intensity over a two-dimensional surface—such as vision or touch. Generally, the receptive field of a neuron is the collection of points (ξ_1, ξ_2) at which the presence of a stimulus—a point of light or mechanical pressure—triggers a neural response. However, as evidenced by Hubel and Wiesel (1962), neural receptive fields are not only characterised in terms of the locations where a stimulus *increases* the average neural activity, but also where the presence of a stimulus *decreases* activity.

Mathematically, we can describe this in terms of a function $e(\xi_1, \xi_2)$ that assigns a weight to each stimulus location (ξ_1, ξ_2) . This function e is the receptive field. *Gabor filters* are a class of mathematical functions that fit observed receptive fields in the visual cortex well (Marçelja, 1980; Field, Tolhurst, and Campbell, 1986).¹³ Examples are depicted in Figure 2.20.

Borrowing some notation from the next section, the activity of the neuron $a(x)$ can be modelled as the product between the receptive field e and the stimulus x integrated over space (where $x(\xi_1, \xi_2)$ is a function describing the stimulus intensity at a certain location)

$$a(x) = G \left[\alpha \iint e(\xi_1, \xi_2) x(\xi_1, \xi_2) d\xi_1 d\xi_2 + \beta \right] = G [\alpha \langle e, x \rangle + \beta]. \quad (2.19)$$

Here, $\langle e, x \rangle$ is the inner product between the receptive field and the stimulus. Furthermore, G is a rate approximation of the neuron model (see previous subsection), and $\alpha > 0$ and β translate the inner product into a neural input current.

¹³Gabor filters are sinusoidals scaled by a radial Gaussian envelope. Such functions are maximally localised in time and space, as pointed out in the context of Gabor’s Fourier uncertainty principle (Gabor, 1946).

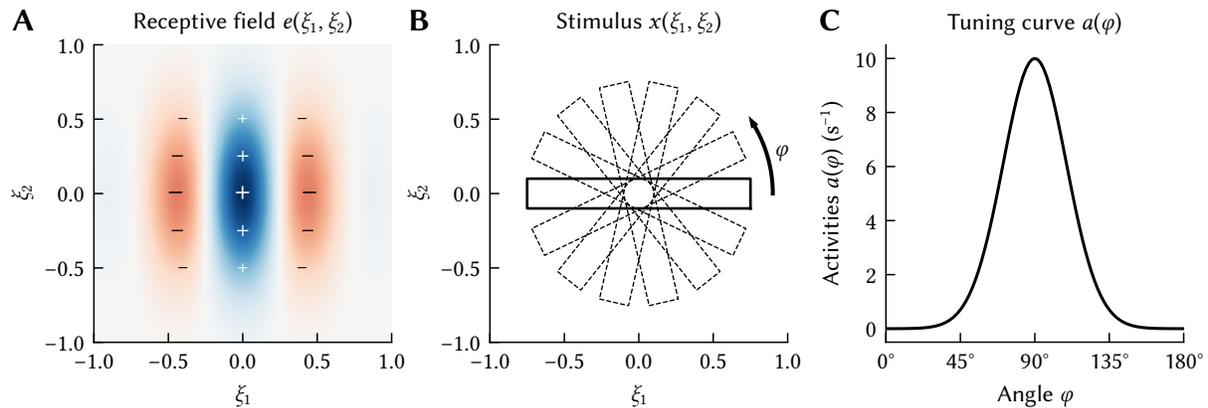


Figure 2.21: Model of the original Hubel and Wiesel experiment (cf. Figure 2.19). **(A)** The receptive field $e(\xi_1, \xi_2)$ describes whether a stimulus location (ξ_1, ξ_2) acts excitatory (blue), or inhibitory (red). **(B)** The stimulus is a light rectangle at different orientations φ ; this can be described as a function x mapping locations onto light intensity. **(C)** Tuning curve obtained according to the model in eq. (2.19).

Importantly, if the stimulus and receptive field are normalised, then $\langle e, x \rangle$ is the cosine similarity between the two functions. Correspondingly, according to the above model, the neural response is maximised if $e = x$. In other words, the receptive field is the “true” preferred stimulus of a neuron. This is consistent with modern definitions of the term “receptive field” in the neuroscience literature (cf. Troy, 2009).

Figure 2.21 illustrates the relationship between the receptive field of a neuron and one of its tuning curves. Choosing a vertically oriented Gabor filter as a receptive field, we can model the orientation tuning from the original Hubel and Wiesel experiment. Thus, in a nutshell, a tuning curve is a characterisation of the neuron in terms of a specific stimulus property (e.g., the orientation of a bar of light), whereas the receptive field models its tuning properties in general (e.g., the predicted activity for any visual stimulus).

Population codes. Although tuning curves and receptive fields are reconstructed within a network context, they only characterise individual neurons. To better understand biological neural networks, we need to consider the tuning properties of groups of neurons.

Doing this, we find that neighbouring neurons often have similar tuning properties—they are tuned to the same sensory modality, and possess similar receptive fields (Berkowitz, 2009). Again, this has been explored well in context of orientation tuning in the visual cortex (Kandel et al., 2012, Chapter 25). For example, neurons in visual cortex are organised in “cortical columns”. Each column consists of several layers of neurons. Within a single layer of the same column, neurons possess similar preferred orientations. Between columns, preferred orientations differ significantly.

This, as well as several other observations (Yuste, 2015), suggest that brain networks use *population codes*. The idea is that a single underlying quantity—such as the orientation of a

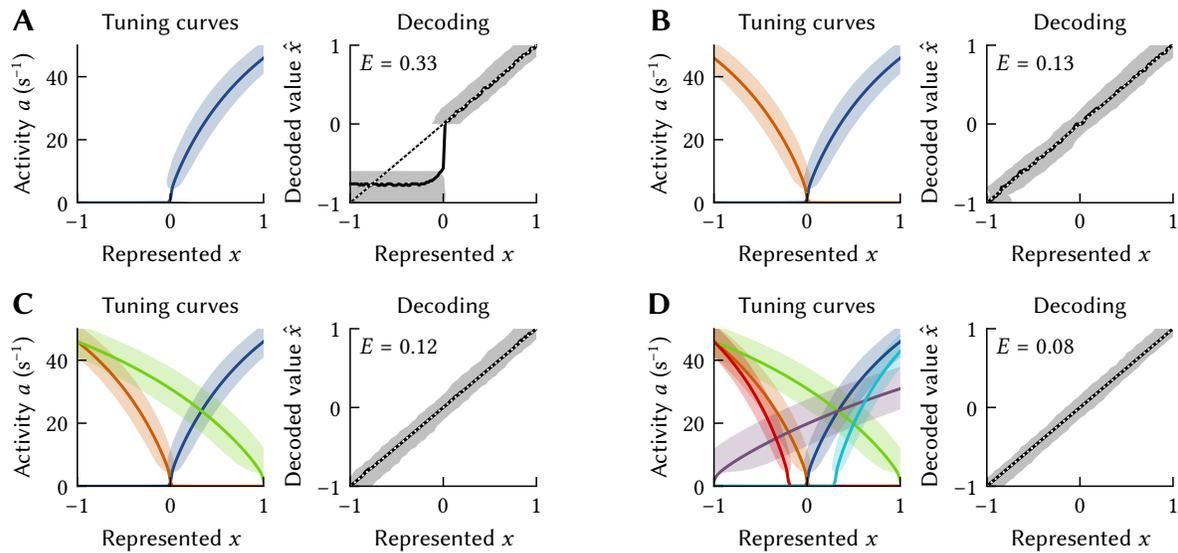


Figure 2.22: Probabilistic decoding of scalar quantities using population codes. **(A) Left:** The activity a of a single neuron represents a scalar value $x \in [-1, 1]$. The current J fed into each neuron is subject to Gaussian noise with standard deviation $\sigma = 1$ nA. Solid line corresponds to the median activity for the given represented value; shaded area corresponds to a contour line of the underlying probability density. **Right:** Trying to decode the represented value x from the activity results in significant uncertainties; negative values cannot be disambiguated. Dotted line is the optimum; solid line the median of a maximum-likelihood estimate, shaded area a contour line of the underlying probability density. Error E is the decoding RMSE. **(B, C, D)** The larger the number of neurons, the smaller the uncertainties.

visual stimulus—is represented by a group of neurons in a distributed manner. The opposite would be “single-neuron coding”, where the activity of individual, localised neurons maps onto some quantity, or, in higher-level cognition, onto specific concepts (Berkowitz, 2009).

Population codes are intriguing as a model for representation in biological networks. They allow precise inferences from imprecise or ambiguous observations. This is illustrated in Figure 2.22. For example, we can imagine that a single neuron represents a scalar value $x \in [-1, 1]$ as expressed by its tuning curve $a(x)$. Unfortunately, the firing rate a cannot be measured without error. That is, neurons produce spontaneous activity that, as far as we can tell, is not related to a stimulus. This is part due to noise inherent to neurobiological processes (cf. Eliasmith and Anderson, 2003, Section 2.2.1), and in part due to the Fourier uncertainty principle.¹⁴ Hence, when trying to *decode* the represented value x from the neural activity, there is a large degree of uncertainty. Adding neurons that are tuned to the same quantity x , but with different tuning curves to the population drastically reduces this uncertainty from an information-theoretic perspective (Ma and Pouget, 2009).

¹⁴We can either estimate the frequency (i.e., rate) of a signal with a high precision at a low temporal resolution, or estimate the frequency with a low precision at a high temporal resolution (Gabor, 1946).

2.3 The Neural Engineering Framework

As we mentioned at the beginning of this chapter, the Neural Engineering Framework (NEF; Eliasmith and Anderson, 2003) is a modelling framework for neurobiological systems. The NEF combines techniques from engineering and mathematics—particularly control and dynamical systems theory—with the neurobiological concepts discussed before. The goal is to allow researchers to map high-level descriptions of cognitive phenomena onto low-level spiking neural networks. As such, the NEF provides a synthesis of bottom-up and top-down modelling, and is an attempt at a theory of neuroscience that spans multiple levels of analysis.

To use the NEF, researchers systematically apply three “principles” that encapsulate key assumptions about neurobiological systems. These principles exploit neural nonlinearities and synaptic filters to describe how vectorial quantities can be represented, transformed, and made to follow certain dynamics in a spiking neural fabric. The resulting neural networks adhere to mechanistic constraints, such as neural connectivity, maximum firing rates, tuning properties, and synaptic time-constants. This is illustrated schematically in Figure 2.23. A software tool that automates the process of translating mathematical models into spiking neural networks using the NEF principles is part of the Nengo spiking neural network simulator package (Bekolay, Bergstra, et al., 2014).¹⁵

In this section, we first discuss applications of the NEF from the perspective of the cognitive sciences (model validation and hypothesis generation), as well as engineering and computer science (a programming model for neuromorphics). We continue with a detailed description of the theory underpinning the NEF, and conclude with a list of limitations that we address in the next chapters.

2.3.1 Model Validation and Hypothesis Generation

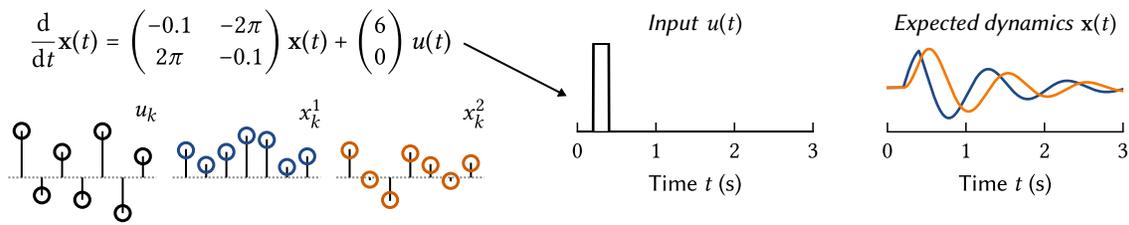
As we alluded to in the introduction of this chapter, models built using the NEF bridge multiple levels of analysis. That is, they implement high-level behaviour (given as a mathematical description) using low-level mechanisms (spiking neurons and synapses). This facilitates model validation and hypothesis generation.

Model validation. The concept of model validation generally refers to comparing a model to empirical measurements (Adams et al., 2012). In the case of NEF models, we can, of course, compare the high-level behaviour of the biological system to that of the constrained network model. The smaller the deviations, the “better” the original mathematical model.¹⁶ While such comparisons would also be possible with the high-level model alone, biological constraints

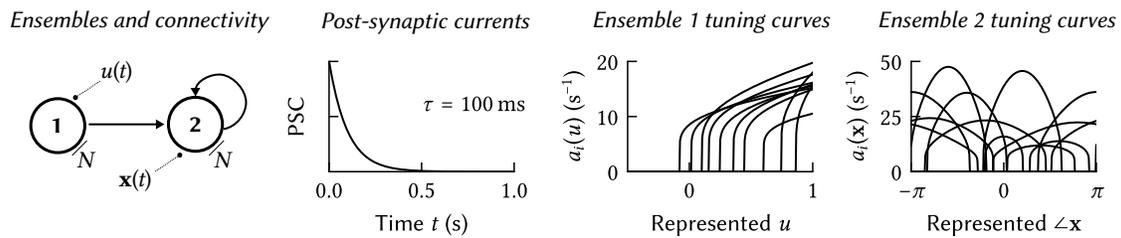
¹⁵See <https://www.nengo.ai/> for more information on Nengo. Nengo is free for non-commercial use.

¹⁶Of course, naively evaluating models in this manner does not account for overfitting. The model could inadvertently be tuned to reproduce the behaviour of interest, but not be able to generalise to different scenarios. In contrast to other machine learning approaches, overfitting in this sense is typically less of a problem with NEF models. This is due to the transparency of the training process, only few (typically hand-selected) parameters, and biological constraints, all of which can be seen as regularisation.

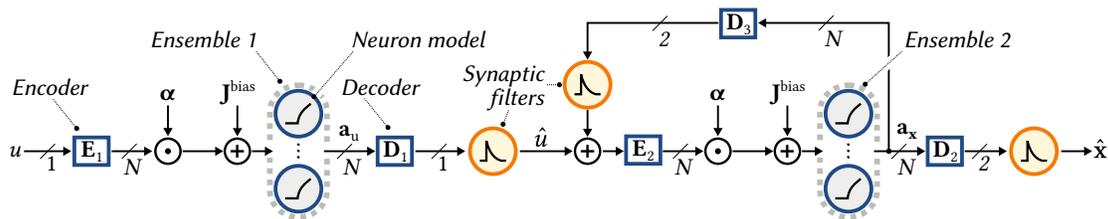
A Mathematical model or sampled data



B Constraints



C Spiking neural network



D Simulation results

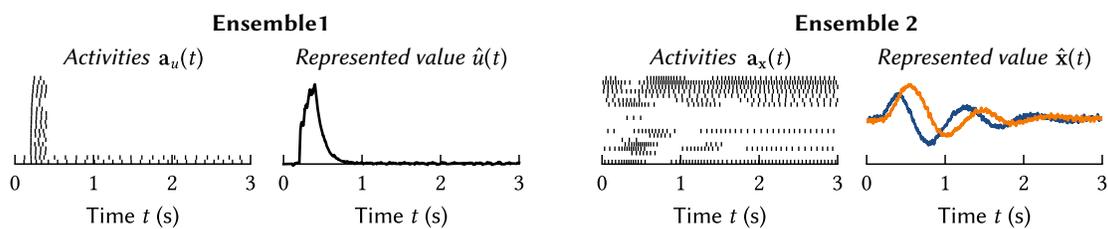


Figure 2.23: Overview of the Neural Engineering Framework (NEF). **(A, B)** Researchers provide a mathematical model or sampled data (sampled data shown here are random and only for illustration purposes), as well as a set of constraints. In this example, constraints are the overall connectivity, tuning properties and filter properties of post-synaptic currents. $\angle \mathbf{x}$ denotes the angle of $\mathbf{x} = (x_1, x_2)$, i.e., $\angle \mathbf{x} = \arctan2(x_2, x_1)$. **(C)** The NEF can be used to systematically construct a spiking neural network from the given data. This network optimally implements the mathematical model under the given constraints. **(D)** Simulating the network gives insights into how well the mathematical description can be implemented on a neural substrate. Spike raster plots with neural activity depict output spikes (short vertical lines) for each neuron (corresponding to rows) over time.

can drastically influence the high-level behaviour of the model. The NEF is thus a good litmus test for the suitability of a mathematical model to be realised on a neural substrate. This is because the NEF *optimally* implements a set of mathematical equations as an idealised spiking neural network. If the model cannot be realised on a spiking neural substrate despite these idealised conditions, it is unlikely that it describes a biological process well.

NEF models can also be compared in terms of detailed neurophysiological data, and not just high-level behaviour. Because NEF models are based on a biologically plausible neural substrate, researchers can measure neurophysiological quantities, such as spike trains, local field potentials, or blood oxygenation (Eliasmith, 2013, Chapters 5.8 & 9.4). Examples of this can be found in Stewart, Bekolay, and Eliasmith (2012), Bekolay, Laubach, and Eliasmith (2014), Duggins et al. (2017), Voelker and Eliasmith (2018), and Gosmann and Eliasmith (2021).

Hypothesis generation. There are two ways in which the NEF aids *hypothesis generation*. First, and closely related to model validation, deviations between the constrained simulation and the original mathematical model may indicate that the high-level description cannot be realised by the biological system. Hence, the NEF can be used to reject unsuitable hypotheses and to guide theories towards higher degrees of biological plausibility. Examples of cognitive theories developed in conjunction with the NEF include the Semantic Pointer Architecture (SPA; Eliasmith, 2013) and SPAUN (the “Semantic Pointer Architecture Unified Network”), a functional brain model capable of a variety of cognitive tasks (Eliasmith, Stewart, et al., 2012).

Second, NEF models can be used to predict the behaviour of a system in experimental paradigms for which no empirical data exists (yet). This prediction then forms a hypothesis that can be compared to future empirical data. Furthermore, the NEF also enables the systematic variation of observed biological parameters (e.g., time-constants or connectivity), which is often not easily possible in real biological systems. Assessing the system performance in such artificial situations may yield hypotheses as for why evolution favoured certain parameter combinations. We give an example of this in Chapter 5.

2.3.2 Applications to Neuromorphic Computing

The scientific applications discussed above were the primary motivation for the development of the NEF. However, coincidentally, the same methods are useful as a programming model for neuromorphic computers (Boahen, 2017; Voelker and Eliasmith, 2021).

Neuromorphic computers. Coined by Mead (1990), this term refers to computer systems that—to some degree—mimic information processing in the brain. This typically implies numerous simple computational units, *neurons*, connected via a flexible interconnect (Furber, 2016). Beyond these basic characteristics there is no consensus as for what exactly constitutes a “neuromorphic computer”. However, most neuromorphic hardware systems execute computations asynchronously and possess provisions for sparse event-based signalling. Akin to biology, individual computational units independently generate binary events. In contrast to artificial

neural networks, computation is inherently temporal; communication occurs sporadically and does not carry continuous values. Hence, the underlying communication infrastructure solely carries temporally sparse spike events, minimising the amount of transferred data.

Current neuromorphic computers can be roughly split into two categories, depending on the specific realisation of the computing elements. Computation either takes place in analogue model circuits, similar to the LIF circuit (Figure 2.14), or digital processors with varying degrees of programmability. Examples of systems in the former “mixed-signal” category include Braindrop (Neckar et al., 2019) and BrainScaleS (Schemmel, Brüderle, et al., 2010). Systems that fall into the latter “purely digital” category are SpiNNaker (Furber et al., 2013) and Loihi (Davies et al., 2018). See Furber (2016) for a more comprehensive list.

Applications of neuromorphic computing. The primary motivation behind neuromorphic hardware is to construct computing hardware that approaches the energy efficiency of biological brains (Mead, 1990; Boahen, 2017). Compared to conventional computer architectures, these energy savings are mostly due to the asynchronous nature of the computational units, as well as the reduction in communication bandwidth due to event-based signalling (Painkras et al., 2013). Furthermore, mixed-signal neuromorphic computers with analogue model circuits benefit from each computational unit operating close to the theoretically required minimum energy (Boahen, 2017). However, compared to purely digital systems, such analogue designs are much more challenging to build from an engineering perspective.

Being able to reliably simulate large spiking neural networks with a low energy footprint would be a major scientific breakthrough. For example, this would enable large-scale brain simulations that are infeasible with conventional supercomputers (Calimera, Macii, and Poncino, 2013). From an engineering perspective, being able to map artificial neural networks onto neuromorphic computers could greatly reduce energy costs of machine learning in the data centre and enable new mobile applications of artificial intelligence (Hunsberger and Eliasmith, 2016; Blouw, Choo, et al., 2018; Göltz et al., 2021; Blouw and Eliasmith, 2020). Due to the inherent temporal aspect of the computation, spiking neural networks and neuromorphic hardware are particularly well-suited for applications requiring real-time stream processing, such as robotic control or autonomous drones (Komer, 2015; Yan et al., 2021).

The NEF as a programming model for neuromorphic computers. While, as of writing, several neuromorphic hardware platforms are available to the wider research community, a remaining challenge is to actually program these systems. Since the computational fabric of neuromorphic computers is similar to biological neural substrates, the NEF can be used to translate mathematical descriptions of dynamical systems into network models that can be executed on the hardware system (Boahen, 2017). The given constraints can be used to reflect the capabilities of the hardware and to specify the desired precision of the computation. A variety of neuromorphic platforms have been included as an execution target in Nengo, catering to both scientific and engineering applications.

2.3.3 Principle 1: Representation

As we saw above, the Neural Engineering Framework has useful applications in both science and engineering. In the first part of this thesis, we discuss extensions to the NEF that allow us to model neurobiological circuits more accurately and to take better advantage of the computational resources found on neuromorphic computers. In preparation for these extensions, we now provide a detailed discussion of the aforementioned “three principles” at the core of the NEF.

The first two NEF principles are best explained assuming steady-state average activities, although, as mentioned before, biological neural networks inherently form dynamical systems. However, our use of average rate approximations should not be interpreted as the adoption of a “rate code” by the NEF, but, as we discuss below, is a convenience for solving the synaptic weight optimisation problem. All optimisations can be done in the spiking domain, but the computational costs are significantly higher (MacNeil and Eliasmith, 2011).

For now, our goal is to map static equations of the form $\mathbf{y} = f(\mathbf{x})$ onto rate approximations of spiking neural networks. The first NEF principle describes how neuron populations represent quantities such as \mathbf{x} and \mathbf{y} . Paraphrasing Eliasmith and Anderson (2003):

NEF Principle 1. The momentary activity $\mathbf{a} \in \mathbb{R}^n$ of a population of n neurons represents a vectorial quantity $\mathbf{x} \in \mathbb{R}^d$. The encoding process mapping \mathbf{x} onto activities \mathbf{a} is nonlinear, the decoding process is linear. That is, there exists a matrix $\mathbf{D} \in \mathbb{R}^{d \times n}$ such that $\mathbf{x} \approx \mathbf{D}\mathbf{a}(\mathbf{x})$.

There are two fundamental assumptions in this principle that deserve to be untangled. First, we assume that neuron *populations* form basic functional units and, second, that biological systems represent vectorial quantities. We already discussed the first assumption regarding populations in more detail in Section 2.2.4. To summarise, neighbouring neurons are sensitive to the same quantity and neural activities are noisy—this suggests that nervous systems use population codes. The second assumption deserves more explanation, given that vectors are abstract mathematical objects.

Vectorial representations. The assumption that populations represent vectors $\mathbf{x} \in \mathbb{R}^d$ is—to some degree—a mathematical convenience. Many mathematical objects have useful equivalent vectorial representations, including functions (Eliasmith and Anderson, 2003, Chapter 3), probability densities (Eliasmith and Anderson, 2003, Chapter 9), and symbols (via vector symbolic architectures; see Gayler, 2003, Eliasmith, Stewart, et al., 2012 and Eliasmith, 2013).

Additionally, there is strong evidence that the activities of neural populations form smooth low-dimensional manifolds in a high-dimensional activity space (Gallego et al., 2017; Stringer et al., 2019). There exists a mapping $\mathbf{a} : \mathbb{R}^d \rightarrow \mathbb{R}^n$ (with $d \ll n$) between a low-dimensional vector $\mathbf{x} \in \mathbb{R}^d$ and the average neural activities $\mathbf{a}(\mathbf{x}) \in \mathbb{R}^n$, where n is the number of neurons

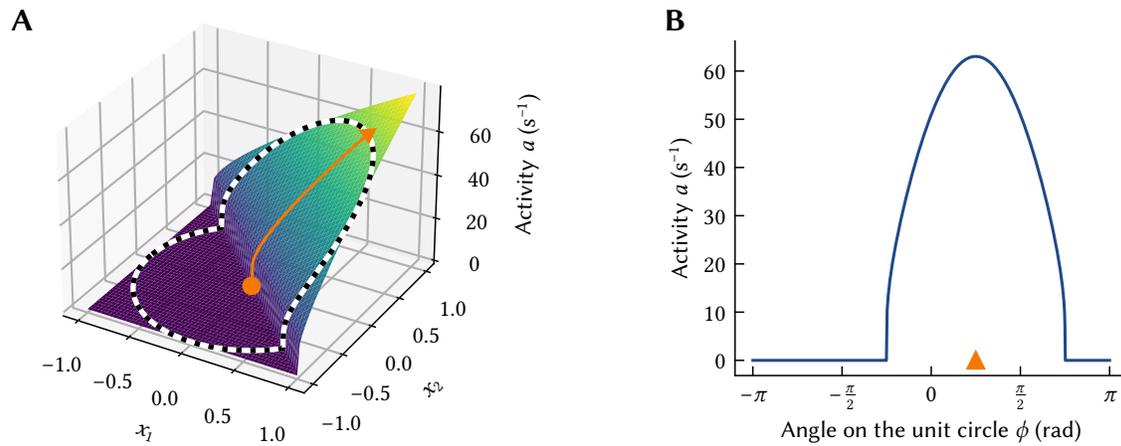


Figure 2.24: Obtaining bell-shaped tuning curves when using two-dimensional encoders. Figure adapted from Eliasmith and Anderson (2003, Figure 2.8, p. 52). **(A)** The surface plot of the tuning curve of a single neuron. The orange arrow points in the direction of the encoder \mathbf{e}_i . **(B)** The tuning curve for a represented value \mathbf{x} of constant magnitude over the angle $\angle \mathbf{x}$. The orange triangle corresponds to the direction of the encoder. The neuron is most active for represented values pointing in this direction.

in the population. If the population exhibits activities $\mathbf{a}(\mathbf{x})$ associated with a value \mathbf{x} , then the population *represents* a d -dimensional quantity \mathbf{x} .

Encoding vectorial quantities. As we have discussed in Section 2.2.4, the relationship between a stimulus \mathbf{x} and activities of an individual neuron are also referred to as a “tuning curve”. While tuning curves in neuroscience are typically defined over scalar quantities, Eliasmith and Anderson propose a parametrised encoding equation $a_i(\mathbf{x})$ that maps $\mathbf{x} \in \mathbb{R}^d$ onto the activities of the i th neuron in a population—similar to the mathematical model we presented to describe tuning curves in the Hubel and Wiesel experiment (Section 2.2.4, Figure 2.21 and eq. 2.19):

$$a_i(\mathbf{x}) = G[J_i(\langle \mathbf{x}, \mathbf{e}_i \rangle)] = G[\alpha_i \langle \mathbf{x}, \mathbf{e}_i \rangle + \beta_i], \quad \text{where } i \in \{1, \dots, N\}. \quad (2.20)$$

Depending on the interpretation, the encoding vector \mathbf{e}_i is equivalent to the “receptive field” or “preferred direction” of the neuron. For example, as depicted in Figure 2.24, a two-dimensional encoding vector can be used to construct bell-shaped tuning curves similar to those discussed in the context of orientation tuning in the visual cortex; here, the vector \mathbf{e}_i would stand for a preferred direction. Of course, \mathbf{e}_i and \mathbf{x} could also be discretised visual stimuli—in this case, \mathbf{e}_i would be a receptive field and eq. (2.20) is a discrete approximation of eq. (2.19).

The response curve $G[J]$ depends on the underlying neuron model; for example, we derived the response curve for LIF neurons in Section 2.2.2 (eq. 2.16); the current translation function $J_i(\xi)$ is typically an affine mapping parametrised by a gain α_i and a bias current β_i .

When building NEF systems, modellers choose the parameters of $a_i(\mathbf{x})$ to characterise how a stimulus \mathbf{x} *should* be reflected in the activities of individual neurons. This last emphasis on *should* is important. Modellers establish a *normative constraint*. They specify what the desired

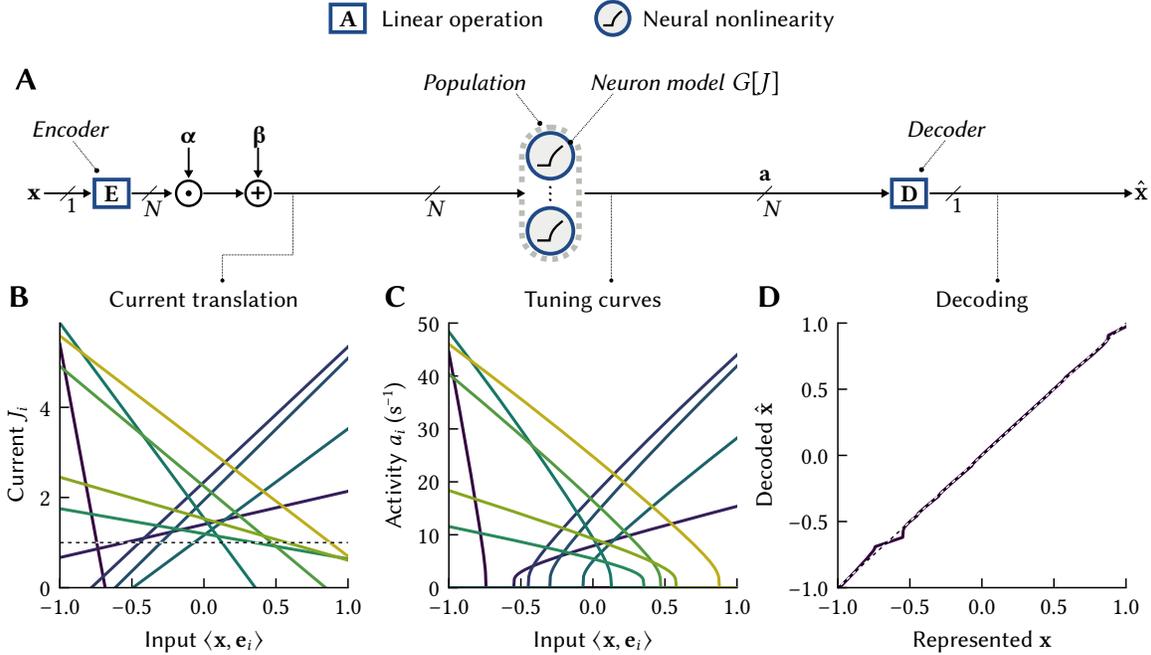


Figure 2.25: Representation in the Neural Engineering Framework. Example for a population of $N = 10$ neurons representing a one-dimensional quantity. **(A)** Schematic overview of the encoding and decoding process. **(B)** Randomly selected affine current translation functions $J_i(\xi) = \alpha_i \langle \mathbf{x}, \mathbf{e}_i \rangle + J_i^{\text{bias}}$. The dotted line corresponds to the activity threshold J_{th} . **(C)** Tuning curves for each neuron in the population after applying the somatic nonlinearity $G[J]$ (eq. 2.16). **(D)** Reconstructing the represented value from neuron activity by means of linear decoding. Dashed line corresponds to the ideal.

tuning should *optimally* be—for example, taking biological data into account. The purpose of the NEF is to ensure that this constraint is met in a network context.

Selecting encoders, gains, and biases. Of course, this raises the question of how modellers should select encoders \mathbf{e}_i , biases β_i , and gains α_i . To form a good basis for a population code, the population must exhibit diverse tuning (cf. Figure 2.22). At the same time, the tuning curves must stay within modelling constraints such as the maximally observed firing rates a_i for the range of stimuli \mathbf{x} over which we characterise the neuron population. We denote the set of represented values as a compact set \mathbb{X} with finite, non-zero volume $\text{vol}(\mathbb{X})$.

Unless more specific data are available, we can for example assume that \mathbf{x} are within a hyperball of radius r , i.e., $\mathbb{X} = r\mathbb{B}^d$. The encoding vectors \mathbf{e}_i can be sampled from the unit hypersphere \mathbb{S}^d . For monotone increasing response curves G , gain α and bias β can be computed for each neuron by sampling an “ x -intercept” value ξ_0 from the interval $[-r, r]$ and a maximum rate a_{\max} . We then solve for α, β such that the following equations hold:

$$\alpha \xi_0 + \beta = J_0 = G^{-1}[0], \quad \alpha r + \beta = J_{\max} = G^{-1}[a_{\max}], \quad \text{where } G^{-1}[a] := \max \{ J \mid G[J] = a \}.$$

We get $(r - \xi_0)\alpha = J_{\max} - J_0$, $(r - \xi_0)\beta = (rJ_0 - \xi_0 J_{\max})$. For example, for each tuning curve in

Figure 2.25, we randomly sampled a ξ_0 from a uniform distribution over $[-1, 1]$, and a maximum firing a_{\max} rate uniformly from $[10, 50]$.

Computing identity decoders. Equation (2.20) describes the average neural activities \mathbf{a} we expect when a population is representing a certain value \mathbf{x} ; that is, this equation describes how quantities \mathbf{x} are *encoded* in a neural network. Complementary to encoding is *decoding*. That is, given the activities \mathbf{a} , we would like to reconstruct the represented \mathbf{x} . This could—for example—be accomplished using Bayesian inference, as we demonstrated in Figure 2.22. If the number of neurons n in a population is large enough we can, to a very similar effect, just use a linear decoding matrix \mathbf{D} (Figure 2.25; Salinas and Abbott, 1994).

The idea is to simply multiply the activities \mathbf{a} with a matrix $\mathbf{D} \in \mathbb{R}^{d \times n}$ such that the decoded $\hat{\mathbf{x}} = \mathbf{D}\mathbf{a}(\mathbf{x})$ is approximately equal to the encoded \mathbf{x} , for any $\mathbf{x} \in \mathbb{X}$. Under the assumption that the activities \mathbf{a} are subject to independent and identically distributed (i.i.d.) zero-mean Gaussian noise with standard-deviation σ , the decoding matrix \mathbf{D} can be obtained by minimising a least-squares loss-function E :

$$E = \frac{1}{\text{vol}(\mathbb{X})} \iint_{\mathbb{X}} \|\mathbf{x} - \mathbf{D}(\mathbf{a}(\mathbf{x}) + \nu)\|_2^2 d\mathbf{x} \quad \text{where } \nu \sim \mathcal{N}(0, \sigma). \quad (2.21)$$

Of course, minimising this integral directly is infeasible. We approximate a solution numerically by drawing N samples from \mathbb{X} , denoted $\mathbf{x}_1, \dots, \mathbf{x}_N$. Arranging the samples and the corresponding population activities in matrices $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N) \in \mathbb{R}^{N \times d}$ and $\mathbf{A} = (\mathbf{a}(\mathbf{x}_1), \dots, \mathbf{a}(\mathbf{x}_N)) \in \mathbb{R}^{N \times n}$, we can phrase finding \mathbf{D} as a Tikhonov regularised least-squares optimisation problem

$$\min_{\mathbf{D}} \sum_{k=1}^N \|\mathbf{D}\mathbf{a}(\mathbf{x}_k) - \mathbf{x}_k\|_2^2 + \|\sigma\mathbf{D}\|_F^2 = \min_{\mathbf{D}} \|\mathbf{A}\mathbf{D}^T - \mathbf{X}\|_F^2 + \sigma^2 N \|\mathbf{D}\|_F^2, \quad (2.22)$$

where $\lambda = N\sigma^2$ is the regularisation factor. From a probabilistic perspective, this particular choice of λ results in the exact maximum a-posteriori solution for \mathbf{D} with the prior assumption that there is Gaussian noise with standard deviation σ on the measurements \mathbf{A} (Boyd and Vandenberghe, 2004, Chapter 6). Larger λ increase the robustness of the solution with respect to noise, but decrease the noise-free approximation error. The solution can be expressed in terms of the regularised Moore-Penrose pseudo-inverse:

$$\mathbf{D}^T = \mathbf{A}^+ \mathbf{X}, \quad \text{where } \mathbf{A}^+ = (\mathbf{A}^T \mathbf{A} + \lambda \mathbf{N}\mathbf{I})^{-1} \mathbf{A}^T. \quad (2.23)$$

As explored by Eliasmith and Anderson (2003, Chapter 2), the decoding error E tends to decrease with $\mathcal{O}(\sqrt{n})$, assuming that the tuning curves have uniform x -intercepts and random encoders \mathbf{e}_i . An example of this decoding scheme is given in Figure 2.25D.

Note that, as mentioned before, we do not take dynamics into account when computing the decoders \mathbf{D} . However, as we discuss next, the same \mathbf{D} can be used to decode represented values through time in spiking networks when defining activity $\mathbf{a}(t)$ as low-pass filtered population spike trains. We explicitly take temporal aspects of neural computation into account in Section 2.3.5 below, and when we discuss temporal tuning in Chapter 4.

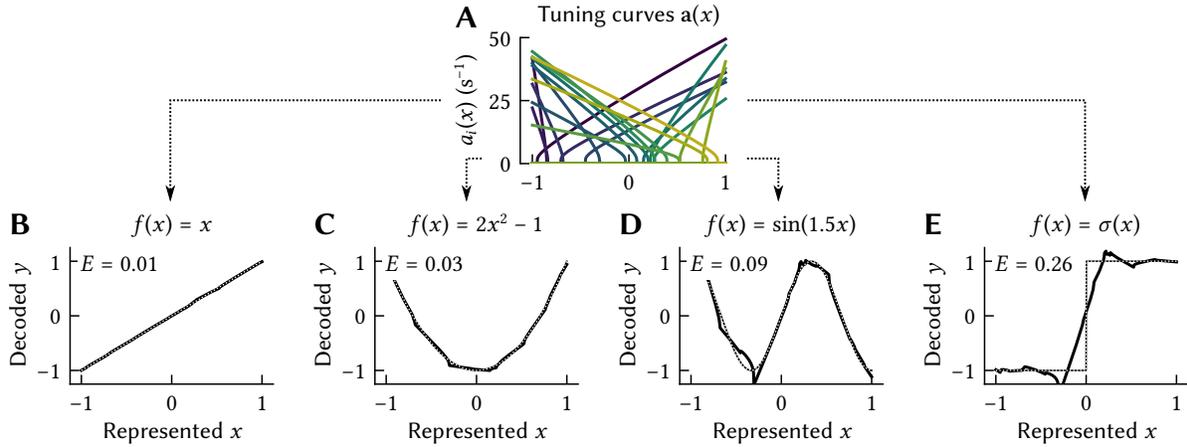


Figure 2.26: Examples of the function decoding scheme. Instead of decoding the represented value x from a neuron population, we can approximate arbitrary functions $f(x)$. **(A)** $n = 20$ tuning curves $a(x)$; same parameters as in Section 2.3.3. **(B-E)** Different linear decodings $D^f a(x)$ of the above tuning curves. Dotted line is the target function, thick line the decoded function. Inset E is the RMSE.

2.3.4 Principle 2: Transformation

The representation principle maps vectors \mathbf{x} onto average neural activities $\mathbf{a}(\mathbf{x})$. Of course, individual neuron populations seldom exist in isolation. To construct *networks* that realise the desired tuning properties, we need to determine synaptic weights w_{ij} that describe the connection strength between the j th pre- and the i th post-neuron. In addition to realising the chosen representations, the NEF puts a further modelling constraint on the synaptic weights:

NEF Principle 2. Synaptic connections between neural ensembles approximate nonlinear transformations $\mathbf{y} = f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^d$ is the value represented in the pre-population, and $\mathbf{y} \in \mathbb{R}^{d'}$ the value represented in the post-population.

That is, modellers can choose a transformation f that should be computed in a connection. We can easily approximate functions f of the value \mathbf{x} represented in a pre-population using a *function decoder* D^f . This D^f has the property that $D^f \mathbf{a}(\mathbf{x}) = \hat{\mathbf{y}} \approx \mathbf{y} = f(\mathbf{x})$. Modifying the least-squares loss from eq. (2.21), we get

$$E = \frac{1}{\text{vol}(\mathbb{X})} \iint_{\mathbb{X}} \|f(\mathbf{x}) - D(\mathbf{a}(\mathbf{x}) + \nu)\|_2^2 d\mathbf{x} \quad \text{where } \nu \sim \mathcal{N}(0, \sigma). \quad (2.24)$$

Analogously to the above, a discrete approximation of D^f for N samples $\mathbf{x}_1, \dots, \mathbf{x}_N$ is given as

$$(D^f)^T = A^+ Y, \quad \text{where } A = (\mathbf{a}(\mathbf{x}_1), \dots, \mathbf{a}(\mathbf{x}_N)) \in \mathbb{R}^{N \times n}, \quad Y = (f(\mathbf{x}_1), \dots, f(\mathbf{x}_N)) \in \mathbb{R}^{N \times d'}.$$

Examples of functions decoded from $\mathbf{a}(\mathbf{x})$ are depicted in Figure 2.26. The decoding error E depends on the number of pre-neurons n , as well as the “smoothness” of the decoded function.

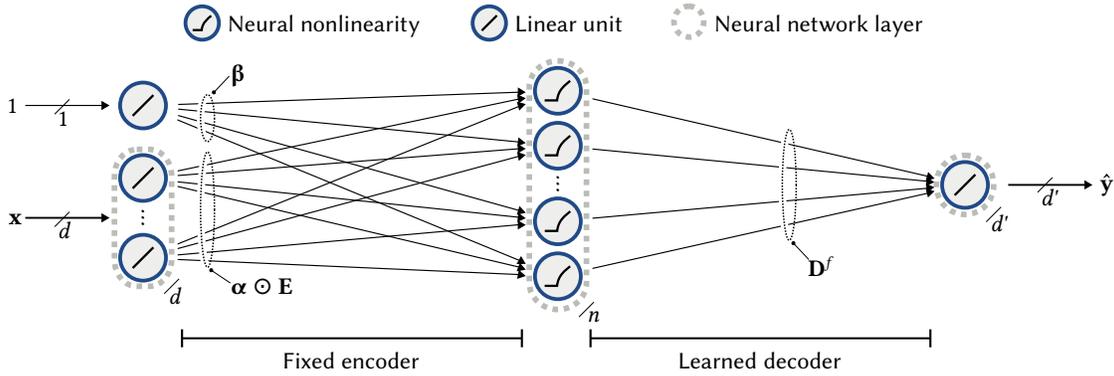


Figure 2.27: Neuron populations in the NEF form a single-hidden-layer artificial neural network. Encoders \mathbf{E} , gains α , and gains β correspond to a fixed input transformation. The function decoder \mathbf{D}^f forms a set of linear output weights. Since the input transformations are fixed, computing \mathbf{D}^f is simple.

Comparison to artificial neural networks. From the perspective of artificial neural networks, the NEF characterises neuron populations as a single-hidden-layer neural network with a fixed input transformation defined by the encoders, gains, and biases. Since the input transformation up to the neural nonlinearity—approximated by the response curve $G[J]$ —is fixed, learning the output weights—the function decoder—is a convex optimisation problem and no stochastic optimisation such as gradient descent is required (Figure 2.27).

Similar nonlinear encoding and linear decoding schemes have been explored in machine learning for decades. One early example are the radial basis function networks described by Broomhead and Lowe (1988). Going back even further, the overall idea is similar to the pattern-recognition model of granule cell activity in the cerebellum proposed by Marr (1969). From a more theoretical perspective, neuron population tuning-curves $\mathbf{a}(\mathbf{x})$ span a function space with a set of non-orthogonal basis functions. Hornik, Stinchcombe, and White (1989) show that—assuming a reasonable distribution of these basis functions—we can approximate any continuous function over \mathbb{X} to a desired degree by increasing the number of hidden units.

Computing synaptic weights. As we have seen, function decoders \mathbf{D}^f can approximate arbitrary functions f over \mathbb{X} . Of course, this is only half of what we set out to accomplish. Our goal was to find synaptic weights $\mathbf{W} \in \mathbb{R}^{m \times n}$ that connect from a pre-population of n neurons representing \mathbf{x} to a post-population of m neurons, such that the tuning properties of the target population are preserved, *and* the post-population is representing a transformed version $f(\mathbf{x})$.

If we assume that the current translation function $J_i(x)$ is an intrinsic part of each post-neuron i , encoding and decoding can be linearly combined into a weight vector \mathbf{w}_i :

$$a_i^{\text{post}}(f(\mathbf{x})) \approx a_i^{\text{post}}(\mathbf{D}^f \mathbf{a}_{\text{pre}}(\mathbf{x})) = G[J_i(\langle \mathbf{e}_i^T, \mathbf{D}^f \mathbf{a}_{\text{pre}}(\mathbf{x}) \rangle)] = G[J_i(\langle \mathbf{w}_i, \mathbf{a}_{\text{pre}}(\mathbf{x}) \rangle)]. \quad (2.25)$$

Hence, $\mathbf{W} = \mathbf{E}\mathbf{D}^f$ forms a synaptic weight matrix, where $\mathbf{E} \in \mathbb{R}^{m \times d'}$ is a matrix of post-population encoding vectors \mathbf{e}_i , and \mathbf{D}^f is the pre-population function decoder. This matrix implicitly decodes $\hat{\mathbf{y}} = f(\mathbf{x})$ from a pre-population, and re-encodes $\hat{\mathbf{y}}$ in the post-population.

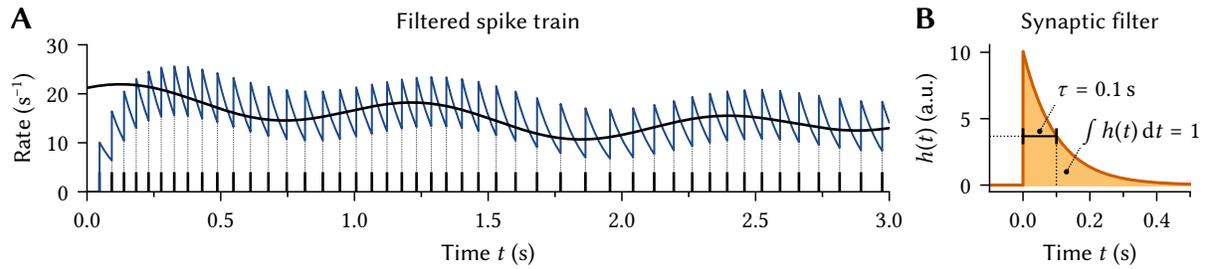


Figure 2.28: Estimating the instantaneous firing rate using a first-order exponential synaptic filter. **(A)** Blue line depicts a spike train (short black lines at the bottom) filtered by a first-order exponential low-pass filter $h(t)$ with $\tau = 0.1$ s. The black line is the “ground-truth” rate underlying the spike train (using a $\Delta\Sigma$ -modulator). Apart from a phase shift and some added noise, the low-pass filtered spike train tracks the ground truth well. **(B)** Visualisation of the synaptic filter $h(t)$. We assume that spikes are Dirac deltas and that the filter has unit DC-gain, i.e., the area under the curve is one.

A welcome side-effect of this formalisation is that the weight matrix \mathbf{W} is of low rank. This significantly reduces the amount of computation required to evaluate NEF networks. Consider a matrix-vector product of the form $\mathbf{W}\mathbf{a}$. Evaluating this requires $\mathcal{O}(mn)$ operations, where n and m are the number of neurons in the pre- and post-population. In contrast, multiplying the low-rank factorisation with an activity vector, i.e., computing $\mathbf{E}(\mathbf{D}\mathbf{a})$, requires only $\mathcal{O}(dn + d'm)$ operations. Since, typically, $d \ll n$ and $d' \ll m$, this is a linear time operation. Along with the linearity of homogeneous synaptic filters, this enables Nengo to simulate large spiking neural networks on commodity hardware (Bekolay, Bergstra, et al., 2014).

2.3.5 Principle 3: Dynamics

So far, and as mentioned several times before, we have ignored the fact that spiking neural networks possess temporal dynamics. Instead, and very similar to the rate-based artificial neural networks used in machine learning, we assumed average firing rates in the encoding and decoding equations. Fortunately, this is less of a problem as it may seem.

Synaptic dynamics (cf. Section 2.2.1) act as a filter that estimates the pre-synaptic *instantaneous firing rate* of a spike train. The instantaneous firing rate is the spike rate a_i of a neuron at a single point in time t . Of course, this is a quite paradoxical notion. As mentioned before, by the Fourier uncertainty principle, we cannot measure an event rate without averaging over time; the smaller the time-window, the smaller the frequency resolution (cf. Gabor, 1946).

Still, low-pass filters can be reasonably effective at inferring the state of a process generating binary events. This is illustrated in Figure 2.28. The low-pass filtered spike train is clearly centred around the ground-truth rate. The deviations from the ground-truth can be interpreted as noise. Conveniently, we already accounted for noise by relying on population codes, and regularising the decoding matrices \mathbf{D}^f (eq. 2.24). More information on this topic can be found in Eliasmith and Anderson (2003, Chapter 4), as well as later in Chapter 4 of this thesis.

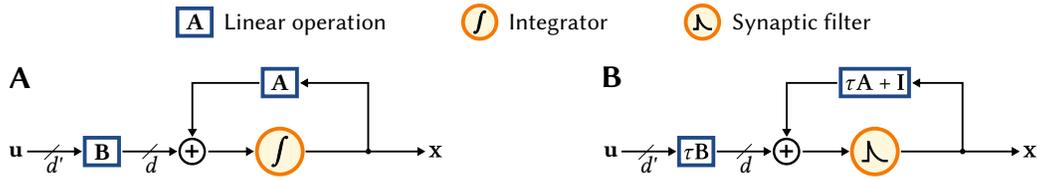


Figure 2.29: Integrator and synaptic filter realisations of an LTI system. Orange circles depict temporal convolutions. **(A)** The standard LTI system. **(B)** Using a low-pass filter instead of a perfect integrator.

Correspondingly, since synaptic filters approximate the instantaneous firing rate, the techniques discussed so far will also work well in the context of spiking neural networks. However, as discussed above, instead of merely translating mathematical descriptions into spiking neural networks, we would also like to know how resources available in biologically plausible spiking neural networks can *support* high-level function.

In this vein, the third NEF principle describes how synaptic filters can be exploited to approximate arbitrary dynamical systems. Paraphrasing Eliasmith and Anderson (2003):

NEF Principle 3. Represented values are state variables $\mathbf{x}(t)$ in a dynamical system. Recurrent connections approximate arbitrary dynamical systems of the form $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t), t)$, where $\mathbf{x}(t)$ is the represented value, $\mathbf{u}(t)$ is some external input, and t is time.

This principle is best explained considering linear time-invariant (LTI) systems of the form

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t),$$

where $\mathbf{A} \in \mathbb{R}^{d \times d}$ is the feedback matrix, and $\mathbf{B} \in \mathbb{R}^{d \times d'}$ is the input matrix. Integration yields

$$\mathbf{x}(t) = \int_0^t \dot{\mathbf{x}}(\tau) d\tau + \mathbf{x}_0 = \int_0^t \mathbf{A}\mathbf{x}(\tau) + \mathbf{B}\mathbf{u}(\tau) d\tau + \mathbf{x}_0, \quad (2.26)$$

where \mathbf{x}_0 is the initial state. Hence, and as illustrated in Figure 2.29A, if we have access to an integrator as a computational primitive, we can easily evaluate dynamical systems of this form.

Unfortunately, biological neural networks do not contain “perfect” integrators. Instead, they need to rely on “leaky” integrators, such as low-pass filters. It is easy to see why a low-pass filter is a leaky integrator. The impulse response of an integrator is a step function (i.e., zero for $t < 0$, one for $t \geq 0$). Similarly, the impulse response $h(t)$ of a first-order low-pass filter jumps to non-zero value at $t = 0$, but in contrast to the integrator, quickly decays (Figure 2.28B). The key idea of the dynamics principle is to compensate for this “forgetfulness”. That is, we change the desired dynamical system to account for leaky integration (Figure 2.29B).

Despite the name, the “leaky integrator” dynamics of the LIF neuron itself are of little use. The neuron resets its state with every output spike, losing all the integrated information. Fortunately, the low-pass filter dynamics of synapses are decoupled from the neural

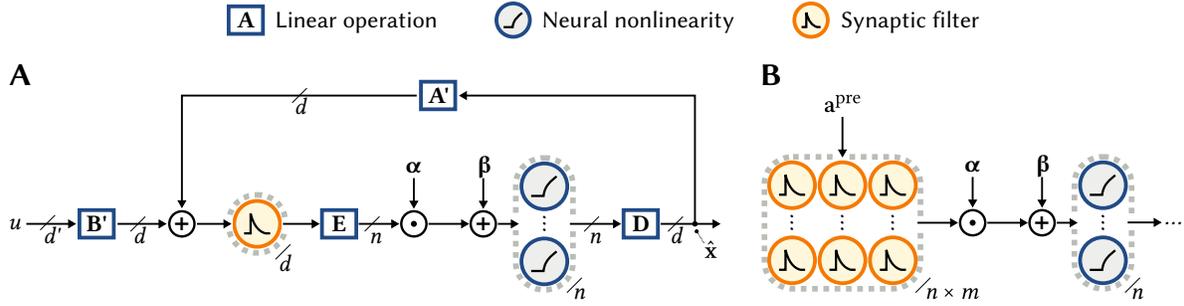


Figure 2.30: Synaptic filters and LTI systems in NEF networks. **(A)** NEF realisation of an LTI system. Homogeneous synaptic filters are modelled as d filters placed before the encoder. **(B)** Technically, each of the $n \times m$ synaptic connections between a pre- and a post-population has its own synaptic filter. When scaled appropriately, the resulting filter matrix replaces the synaptic weights; however, this has drastically higher computational costs than placing d filters ahead of the encoder.

superthreshold dynamics. Moreover, as explained by Eliasmith and Anderson (2003, Chapter 8 & Appendix F.1), it is not unreasonable to assume that the synaptic filter dominates the overall neural dynamics. In other words, dynamics mainly stem from the synaptic filter $h(t)$. We revisit this assumption in Section 2.3.6 below.

We can easily derive a system that compensates for the low-pass filter dynamics using the Laplace transform. Assume that both the feedback and the input passes through the same first-order low-pass filter with time-constant τ . Treating both the integral in eq. (2.26) and the low-pass filter as convolutions with their respective impulse response, we have

$$X(s) = \frac{1}{s} (\mathbf{A}X(s) + \mathbf{B}U(s)) \quad \text{and} \quad X(s) = \frac{1}{1 + \tau s} (\mathbf{A}'X(s) + \mathbf{B}'U(s)). \quad (2.27)$$

Rearranging the second equation and comparing coefficients to the first equation we get

$$X(s) = \frac{1}{s} \left(\frac{\mathbf{A}' - \mathbf{I}}{\tau} X(s) + \frac{\mathbf{B}'}{\tau} U(s) \right) \quad \rightsquigarrow \quad \mathbf{A}' = \tau \mathbf{A} + \mathbf{I}, \quad \mathbf{B}' = \tau \mathbf{B}. \quad (2.28)$$

For arbitrary dynamics $f(\mathbf{x}, \mathbf{u}, t)$ we similarly obtain $f'(\mathbf{x}, \mathbf{u}, t) = \tau f(\mathbf{x}, \mathbf{u}, t) + \mathbf{x}$ (Eliasmith, 2013, Appendix B.3). In other words, to compensate for low-pass filter dynamics, we scale the dynamics by τ and feed back \mathbf{x} to “remind” the system of its current state.

Using the representation and transformation principles, we can turn the block-diagram in Figure 2.29B into a spiking neural network. This is schematically depicted in Figure 2.30A. The decoded state $\hat{\mathbf{x}}$ and the input \mathbf{u} are passed through compensated LTI matrices \mathbf{A}' and \mathbf{B}' , summed, and fed into the synaptic filters. We provided an example of this in Figure 2.23.

Of course, as before, the network can, due to linearity, be expanded into biologically plausible synaptic weights and arrays of synaptic filters (Figure 2.30B). Function decoders \mathbf{D}' can be used to support arbitrary nonlinear dynamics, and not just LTI systems.

2.3.6 *Some Limitations of the Neural Engineering Framework*

To date, the Neural Engineering Framework has seen considerable use in the scientific community (cf. Appendix B.1). Nevertheless, the NEF has a series of shortcomings that limit the biological plausibility of the generated networks, the number of constraints exposed to modellers, and how well resources available on some neuromorphic hardware platforms are utilised. Resolving these issues could make the NEF more appealing to a wider group of researchers and contribute to ongoing efforts in cognitive science and neuromorphic computing.

Of course, the following list of limitations is by no means complete. To the contrary. One can quite effortlessly identify areas of neuroscience that are not reflected in the NEF at all—for example nervous system development, or networks of glial cells. However, as we implied in the beginning this chapter, we think that, as of now, there is too little consensus on these topics to account for them in a general-purpose modelling tool.

Overall, while it is tempting to incorporate as much biological detail as possible into NEF networks, this is misguided if merely done as an end in itself. Remember that our goal is to bridge low-level mechanism and high-level function. This distinguishes NEF models from “bottom-up” approaches, that develop mechanistically detailed models of brain circuits, with the hope that the final system—if truthfully describing biology—exhibits some interesting function (cf. Komer and Eliasmith, 2016). We think that, optimally, extensions to the NEF should expose new low-level detail as a “computational resource” that can be systematically exploited for high-level function. At the very least, extensions should expose additional constraints that provide modellers with an opportunity to explore the impact of individual aspects of neurobiological mechanism on high-level function (e.g., Duggins, 2017).

Limitation 1: Current-based synapses. In our discussion of Principle 2 (Section 2.3.4), we implicitly assumed that neurons use current-based synapses (cf. Section 2.2.2). As expressed in eq. (2.25), we model the current injected into the post-neuron as linearly depending on the pre-population activities. However, as we discussed in Section 2.2.1, synaptic transmission is better modelled in terms of *conductances*, and not currents. As noted by Eliasmith and Anderson (2003, Section 2.1.2, p. 35), accounting for conductances adds another layer of nonlinearity to the neuron. While such nonlinearities have been analysed in the context of the NEF by Tripp (2009, Chapter 4) as well as Bobier, Stewart, and Eliasmith (2014), this prior work does not propose a systematic method for exploiting synaptic nonlinearities for computation.

Adapting the NEF to support nonlinear conductance-based synapses would increase compatibility with neuromorphic hardware platforms emulating conductance-based neurons (e.g., BrainScaleS; Schemmel, Brüderle, et al., 2010) and allow modellers to more easily constrain models according to neurophysiological parameters, such as synaptic reversal potentials. Furthermore, individual synapse types (e.g., excitatory and inhibitory) act as independent input channels to a neuron. As we discuss in the next chapter, these input channels interact nonlinearly in the case of multi-compartment neurons, increasing the class of functions that can be approximated using the same number of neurons.

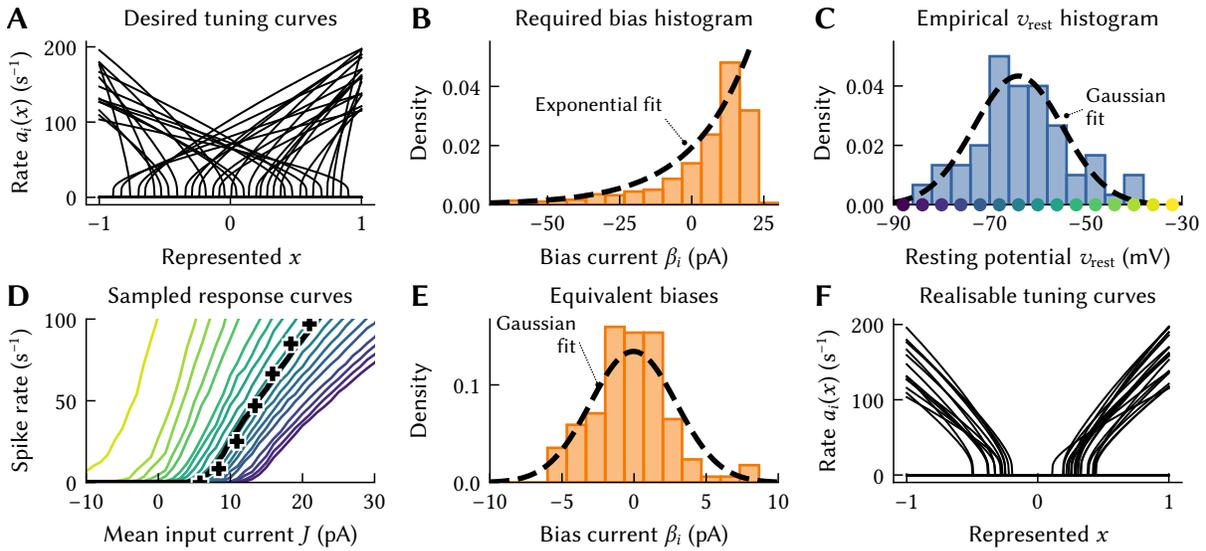


Figure 2.31: Neuron parameter variability only accounts for small bias currents. **(A, B)** Standard NEF tuning curves and corresponding bias current distribution. **(C)** Empirical data on resting potential v_{rest} variations in cerebellar granule cells from Chadderton, Margrie, and Häusser (2004, Figure 1b) with a Gaussian fit. **(D)** Noisy LIF neuron response curves for resting potentials samples from the empirical distribution. Colours indicated in **(C)**. Other LIF parameters fit to empirical data at $v_{rest} = -64$ mV (black crosses; cf. Figure 1g in Chadderton et al.). Input currents J are subject to Gaussian noise ($\sigma = 10$ pA). **(E)** Histogram of bias currents β_i having the same average effect as varying v_{rest} . **(F)** Tuning curves that can be realised using these bias currents for the target maximum rates. Unfortunately, the empirically observed parameters do not result in tuning curve distributions assumed by the NEF **(A)**.

Limitation 2: Bias currents. A minor (as we will see) limitation is the assumption that each neuron possess an intrinsic bias current β_i (eq. 2.20). This current is necessary to ensure diverse tuning. For example, some neurons are active even if no input is present (“spontaneous” or “background” activity). This can be modelled using positive bias currents.

Eliasmith and Anderson (2003, Chapter 2, p. 35) interpret the bias current as a “constant input current from the rest of the nervous system”. While reasonable, this assumption cannot be strictly true. The mean firing rate of individual populations varies significantly over time (Okun et al., 2012); it is not immediately clear how these variations would support constant β_i . Hence, an explicit translation of pre-activities into a bias current β_i using synaptic connectivity may be preferable. Depending on the pre-population tuning, this can affect high-level function.

Furthermore, and directly related to Limitation 1, pre-synaptic activity affects synaptic *conductances* and does not directly induce *currents*. This is particularly problematic since tuning curves with uniform x -intercepts tend to require large *negative* bias currents (Figures 2.31A and 2.31B). It is unclear how such currents should be generated, particularly since the inhibitory synaptic reversal potential E_I is close to the resting potential v_{rest} . Natural neuron parameter variations are insufficient to explain this discrepancy (Figures 2.31C to 2.31F).

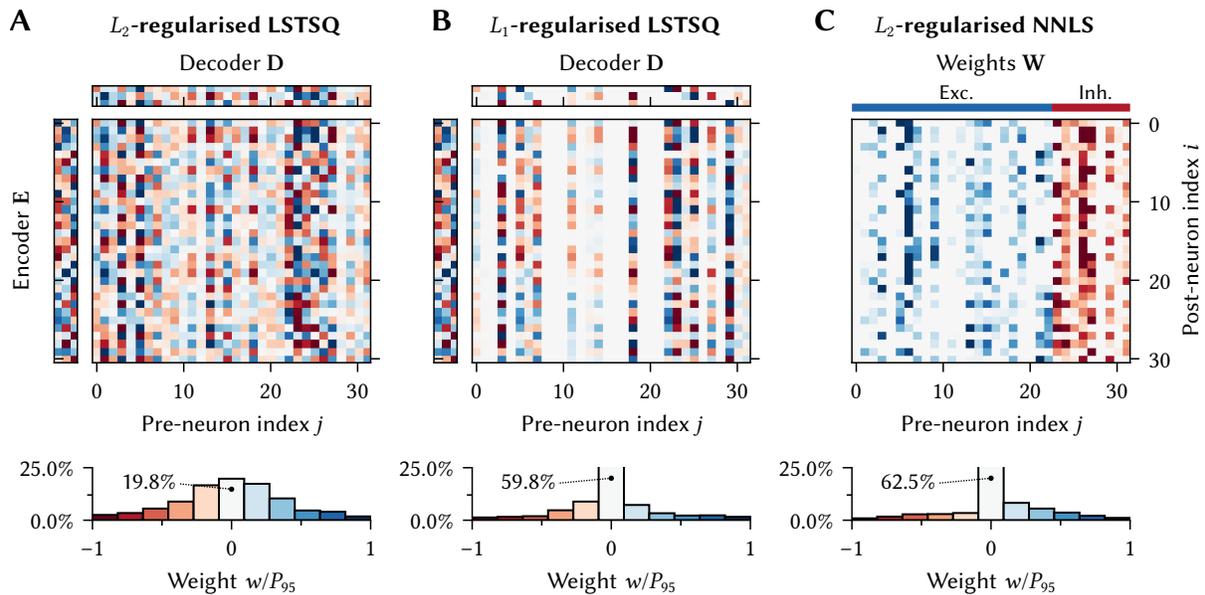


Figure 2.32: Constrained connectivity and Dale’s principle. *Top:* Weight matrix \mathbf{W} and (if applicable) factorisation into \mathbf{E} , \mathbf{D} . *Bottom:* Weight histogram relative to the 95-percentile P_{95} . Red indicates inhibitory, blue to excitatory weights. **(A)** Typical NEF weight matrices are dense. **(B)** L_1 -regularisation of decoders \mathbf{D} increases sparseness, but does so in an uncontrolled manner. Note that \mathbf{D} is (relatively speaking) sparser than \mathbf{W} . **(C)** Dale’s principle imposes an excitatory/inhibitory split on \mathbf{W} . This is difficult when factorising \mathbf{W} , but one can solve for \mathbf{W} directly using nonnegative least-squares (NNLS).

Limitation 3: Constrained connectivity and Dale’s principle. NEF weight matrices $\mathbf{W} = \mathbf{E}\mathbf{D}$ are typically dense. That is, they contain few zeros, implying all-to-all connectivity between neurons (Figure 2.32A). However, in biology, connectivity in the brain can be quite sparse. For example, as detailed by Braitenberg and Schüz (1998, Chapter 20), the likelihood of two neighbouring cortical cells to be connected is less than 90%. Similarly, connectivity between Golgi and Granule cells in the cerebellum is highly constrained (cf. Chapter 5). It may be desirable to give modellers the opportunity to account for these statistics.

A common way to enforce sparsity is to use L_1 , or “lasso”, regularisation instead of the L_2 regularisation proposed in eqs. (2.21) and (2.24) (Boyd et al., 2004, Chapter 6). However, solving for sparse decoders \mathbf{D} in this way has two issues. First, this process is uncontrolled—naïve L_1 regularisation does not take biological constraints into account. Second, for $d \gg 1$, orthogonality between the encoders and decoders determines sparsity, and not just the sparsity of \mathbf{D} (Figure 2.32B). Both issues mandate more complex regularisation terms.

Another form of constrained connectivity is to account for Dale’s principle. As we discussed before in Section 2.2.1, individual neurons typically act either exclusively excitatorily or inhibitorily. Empirical data suggest that, depending on the modeled brain region, excitatory cells outnumber inhibitory cells by a factor between two and four (Hendry and Jones, 1981; Gabbott and Somogyi, 1986). This imposes a certain structure onto the weight matrix (Figure 2.32C).

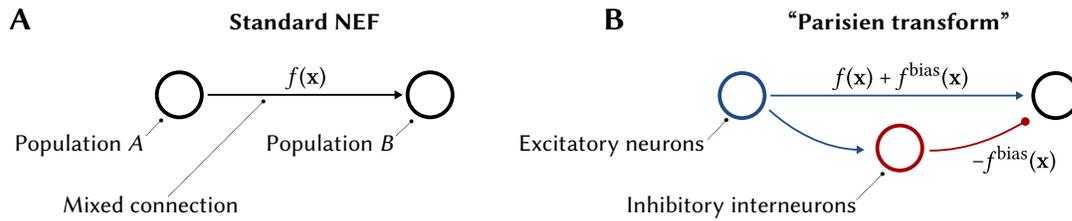


Figure 2.33: Illustration of the “Parisien transform”. This transformation accounts for Dale’s principle by splitting standard NEF connections, as depicted in (A), into an excitatory and inhibitory path (B).

Eliasmith and Anderson (2003, Section 6.4) propose a method to account for Dale’s principle later refined by Parisien, Anderson, and Eliasmith (2008). The basic idea of this “Parisien transform” is to add a specially crafted “bias function” to the function computed in the excitatory connection. This bias function ensures that all connection weights are positive. An inhibitory inter-neuron population is then added to the network and connected in parallel to the excitatory connection to subtract the superfluous current induced by the bias function (Figure 2.33).

While mathematically clever, the Parisien transform changes the network structure by adding an inhibitory interneuron population. Although interneurons are indeed common in brain networks, the connectivity patterns resulting from the Parisien transform may not be desirable. For example, as we discuss in Chapter 5, Golgi cells are recurrent inhibitory interneurons between granule cells. However, there are no excitatory recurrent connections between granule cells, as assumed by the Parisien transform. In the next chapter, we discuss an approach to accounting for Dale’s principle that is independent of the network structure.

Limitation 4: Temporal tuning and neural dynamics. The dynamics principle derived in Section 2.3.5 was based on two assumptions. First, synaptic dynamics dominate neural dynamics. Second, synaptic filters are homogeneous first-order low-pass filters in both the input and recurrent connection. While reasonable, these modelling assumptions are, as so often, only coarse approximations of biology.

The assumption that synaptic dynamics dominate the overall dynamics of a neuron is a good approximation of the behaviour of simple neuron models operating at high firing-rates. However, for smaller firing rates, the neural dynamics can have a significant impact on the overall system dynamics. For example, for maximum firing rates below about 50 s^{-1} , the sub-threshold low-pass dynamics of the LIF neuron begin to have a moderate effect on the neural dynamics. This is depicted in Figure 2.34.

While these rate-dependent effects are relatively small, more complex neuron models possess strong intrinsic dynamics. For example, the adaptive LIF (ALIF) neuron model (Camera et al., 2004) accounts for firing rate adaptation; that is, each output spike increases the current required to evoke another output spike. This can be modelled as an inhibitory current resulting from low-pass filtering the neuron’s output spike train. As demonstrated by Tripp (2009, Chapter 7), the internal dynamics of the ALIF neuron model can, under certain circumstances,

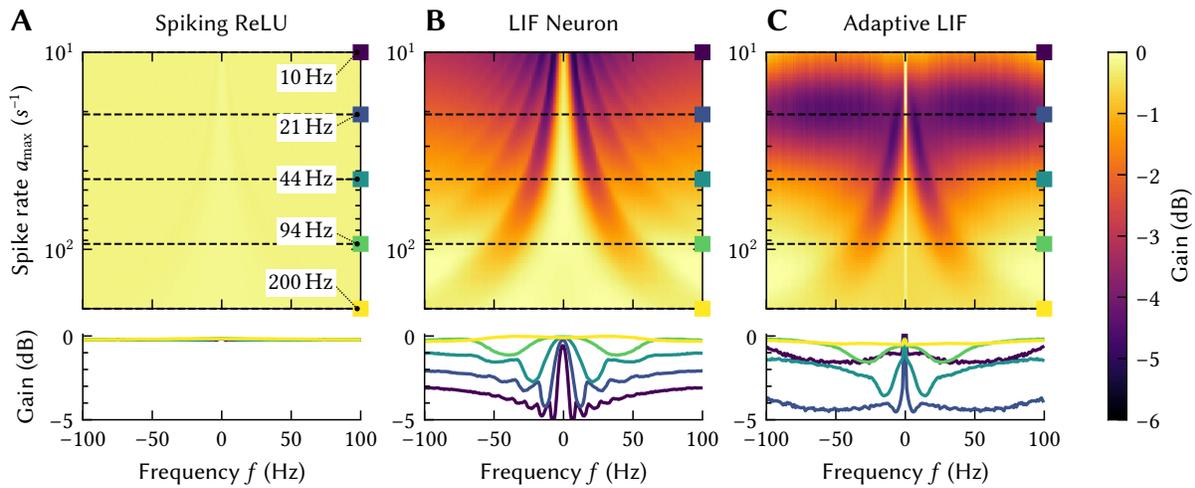


Figure 2.34: Neural dynamics for different maximum firing rates a_{\max} . A 100 s band-limited white-noise signal $u(t)$ with a band-width of 100 Hz is fed through an NEF population with uniform x -intercepts. The transfer function is determined from the decoded $\hat{u}(t)$ using the method described in Eliasmith and Anderson (2003, Section 4.3.3). We ensure a constant amount of spike noise by setting the number of neurons in the population to $100\,000/a_{\max}$. *Top:* Linear transfer function for different maximum rates a_{\max} . *Bottom:* Slice through the transfer functions at the indicated locations. **(A)** Data for a spiking rectified linear unit (ReLU), a $\Delta\Sigma$ -modulator with a rectified input current. This kind of neuron exhibits no significant dynamics of its own, independent of the firing rate. **(B)** Transfer function for the LIF neuron. For small firing rates, the neuron acts as a low-pass with a relatively moderate attenuation of about -5 dB. For firing rates of about 200 Hz, the neuron acts as a pass-through. **(C)** While similar to the LIF neuron, the ALIF acts as a very narrow low-pass filter when driven at a maximum firing rate of about 20 Hz. The location of this “trough” depends on the adaptation time-constant.

be exploited to implement arbitrary dynamics using a variant of the equations derived for the NEF dynamics principle.

The second assumption, namely that synaptic filters are homogeneous, is likely violated if a neuron possesses multiple receptor types. As we saw in Section 2.2.1, the synaptic time-constant can vary significantly between (and even within) individual receptor types. We also saw that synaptic transmission is better modelled by higher-order filters. Voelker and Eliasmith (2018) discuss methods to generalise the dynamics principle to arbitrary synaptic filters; however, these methods require access to the input signal differentials.

In Chapter 4, we suggest the concept of “temporal tuning curves” to—at least partially—account for these issues. Temporal tuning curves are a direct extension of standard NEF tuning curves, and are inspired by the concept of temporal receptive fields. In essence, we treat both synaptic and neural dynamics as temporal resources that can be harnessed to approximate a desired temporal tuning by solving a simple least-squares problem. This naturally extends the NEF and, to an extent, unifies the approaches described by Tripp and Voelker systematically.

Chapter 3

Computational Properties of Multi-Compartment LIF Neurons

It may well be that certain nerve pulse combinations will stimulate a given neuron not simply by virtue of their number but also by virtue of the spatial relations of the synapses to which they arrive. This is, one may have to face situations in which there are, say, hundreds of synapses on a single nerve cell, and the combinations of stimulations on these that are effective [...] are characterised not only by their number, but also by their coverage of certain special regions on that neuron [...], by the spatial relations of such regions to each other, and by even more complicated quantitative and geometrical relationships that might be relevant.

— JOHN VON NEUMANN
The Computer and the Brain (1958)

Material in this chapter from prior publications

Portions of this chapter (in particular Sections 3.4 and 3.6) are, in extended and edited form, adapted from Stöckel and Eliasmith (2021), originally published in *Neural Computation*. These parts are ©2021 Massachusetts of Technology and used by permission of *The MIT Press*. In turn, this publication builds upon work presented at COSYNE 2018 (Stöckel, Voelker, and Eliasmith, 2018), and an earlier technical report (Stöckel, Voelker, and Eliasmith, 2017). While these prior publications focus on two-compartment LIF neurons, we deepen the provided theoretical background and discuss n -LIF neurons in general.

Contributions in this chapter

The work in this chapter can be seen as a continuation of the research presented in Koch (1999, Chapter 5), as well as Tripp (2009, Chapter 4). In contrast to this prior work, we discuss a systematic approach to exploiting passive dendritic nonlinearities as a computational resource to approximate arbitrary functions. As far as we are aware, our mathematical description and analysis of n -LIF neurons, the idea to use quadratic programming for subthreshold relaxation, our spatial frequency-sweep analysis technique, and the presented soft trust-region optimisation scheme are novel.

As we saw in the previous chapter, neurons possess intricately detailed dendrites (cf. Figure 2.2). While the growth of these structures can to some degree be modelled by stochastic processes (e.g., Nowakowski, Hayes, and Egger, 1992), dendrites are not merely a means to establishing random connectivity. Dendritic development is guided by several extrinsic signals, including the activities of neighbouring neurons (McAllister, 2000). This suggests that dendrites are fine-tuned to fulfil a certain computational function.

Indeed, both theoretical and empirical studies show that the locations of synaptic sites within the dendrites have a significant impact on neural computation (Mel, 1994; Koch, Mo, and Softky, 2002; Polsky, Mel, and Schiller, 2004). Still, there is no widely accepted high-level theory of dendritic computation (London et al., 2005), that would, for example, be on a similar level of abstraction as the admirably successful LIF neuron.

Finding such an overarching theory of dendritic computation is difficult due to the sheer complexity of the nonlinear dynamical systems formed by active dendritic structures (Beniaguev, Segev, and London, 2021). This may sometimes lead to seemingly contradictory observations. For example, the distance between a synaptic event and the soma determines the strength of the somatic post-synaptic potential—this is a direct result of the passive cable properties of a neuron (cf. Section 2.1.4). In fact, isolated distal spikes hardly influence the somatic membrane potential (Stuart and Spruston, 1998). However, some dendrites with active Hodgkin-Huxley-like cell membranes (cf. Section 2.1.3) negate the effects of distance-dependent attenuation (Koch, Mo, et al., 2002). Coincident distal input may trigger dendritic action-potentials that in turn strongly influence the soma (Williams and Stuart, 2002).

Hypothesised dendritic function. Early theoretical studies suggest that the passive properties of the dendrites can be exploited to implement arbitrary logic. This is accomplished by mapping “and-not” expressions onto dendritic branches (Koch, Poggio, and Torre, 1983; Mel, 1994; London et al., 2005). Unfortunately, this relies on the empirically not well-supported concept of “shunting inhibition” (cf. Section 3.4.3; Holt and Koch, 1997; Abbott and Chance, 2005).

More recent investigations into the theoretical properties of dendritic trees tend to take the active properties of dendritic compartments into account. Poirazi, Brannon, and Mel (2003) argue that the dendrites of a cortical pyramidal neuron are equivalent to a two-layer network of artificial neurons. This implies that artificial models of cortical circuits require at least twice as many layers as the biological circuitry. Interestingly, this is consistent with comparisons between older deep neural networks and cerebral cortex (e.g., Güçlü and van Gerven, 2015). Taking dynamics into account, Beniaguev et al. (2021) even claim that a single cortical neuron is equivalent to a five-layer temporal convolution network (cf. Section 4.4).

Dendritic structures have also been found to play a significant role in learning. One of the most prominent examples of this are the Purkinje cells in the cerebellum, where basal input is believed to trigger synaptic plasticity (cf. Chapter 5). Similar mechanisms have been proposed as a learning mechanism in cortical pyramidal cells and suggested as a biological basis for error backpropagation (Richards and Lillicrap, 2019; Richards, Lillicrap, et al., 2019).

Goal of this chapter. Compared to the complex mechanisms discussed in many of the studies listed above, the goal of this chapter is decidedly modest. In fact, we would like to incorporate the *simplest possible* model of dendritic computation into the NEF. By “simple” we mean that our model should be as mathematically tractable as possible, while still being exploitable as a computational resource. Correspondingly, our work establishes a baseline for the computational power of mechanisms found in most biological neurons and allows modellers to meaningfully connect this low-level biological detail to high-level function.

Specifically, we do *not* include active effects such as dendritic spikes in our model. Instead, we investigate to what degree passive nonlinear interactions between dendritic compartments provide a substantial computational advantage over standard LIF neurons. As a result, we obtain a more conservative estimate of the computational power of dendritic trees compared to the two- and five-layer networks proposed by Poirazi et al. (2003) and Beniaguev et al. (2021). While scientific interest in passive dendritic effects has waned over the past two decades, our work approaches this topic from a new angle, and, importantly, produces results that are compatible with the aforementioned empirical observations regarding shunting inhibition. Furthermore, as demanded by London et al. (2005), we demonstrate that our theoretical results hold up in noisy spiking networks with relatively low firing rates and few neurons.

There are two primary reasons why we think that integrating dendritic computation into the NEF is important. First, the presence of dendritic structures suggests that individual neurons are computationally more powerful than typically assumed in the NEF. This may be misleading when using the NEF as a litmus test for exploring whether a certain high-level function could at all be implemented in a biological network (cf. Section 2.3.1). One example of this, and a recurring theme in this chapter, is the matter of computing nonnegative multiplication, also referred to as “gain modulation” (Salinas and Thier, 2000). This function can only be computed in the standard NEF if the multiplicands are represented in a common pre-population (Eliasmith and Anderson, 2003, Section 6.3). However, we know that certain circuits in the brain, such layer six in visual cortex, act as gain-control mechanisms that do not rely on common representations (Olsen et al., 2012; Bobier et al., 2014).

Second, accounting for dendritic computation may be of interest for neuromorphic computing. This is particularly true for mixed-signal systems, where individual neurons are analogue model circuits, and communication infrastructure between neurons is digital (e.g., Neckar et al., 2019; Schemmel, Brüderle, et al., 2010). Introducing dendrites could move more of the computation into the analogue domain, and thus improve the power efficiency of the system.

Prior work. There is some prior work regarding the integration of dendritic computation into the NEF. For example, Tripp (2009, Chapter 4) shows that two-compartment neurons with conductance-based synapses can in principle be used in NEF networks. However, Tripp does not investigate how these neurons could be systematically exploited to perform computation.

Bobier et al. (2014) implement a model of visual attention based on the aforementioned gain-control signals present in layer six of the visual cortex. As originally suggested by

Eliasmith and Anderson (2003, Section 6.3), Bobier et al. work around the limitations of the NEF by presupposing that pyramidal cells are capable of nonnegative multiplication. Similar techniques have been pursued in the context of the FORCE and EBN frameworks (Thalmeier et al., 2016; Alemi et al., 2018). While supported by empirical evidence, these approaches are not generalisable to systematically solving for arbitrary functions under biological constraints.

Another line of research related to ours is integrating detailed multi-compartment neuron models into NEF networks. Eliasmith, Gosmann, and Choo (2016) demonstrate that it is possible to replace portions of the SPAUN model (Eliasmith, Stewart, et al., 2012) with detailed multi-compartment neurons, while mostly retaining the performance of the model. Similarly, Duggins (2017) presents techniques for integrating detailed neurons into NEF networks. Our goal is less to demonstrate that such detailed neurons can be used in NEF models, but that accounting for this detail can be advantageous with respect to high-level function.

Structure of this chapter. In Section 3.1, we define the concept of “dendritic computation” in a theoretical function approximation context. Specifically, we treat different synaptic sites in the dendritic structure as separate “input channels”, resulting in a multivariate neural nonlinearity. We find that such multi-channel neurons are not universal function approximators, but can potentially outperform two-layer networks in real-world scenarios.

Next, in Section 3.2, we extend the NEF to support biologically plausible multi-channel neurons. To this end, we first generalise the weight-optimisation problem to act in current space (resulting in full weight matrices \mathbf{W}) instead of representational space (resulting in decoders \mathbf{D}). We furthermore discuss solving for nonnegative weights, as is required for conductance-based channels in more realistic neuron models, and introduce “subthreshold relaxation”, a method for de-emphasising subthreshold target currents and improving superthreshold accuracy.

We continue in Section 3.3 by formally defining n -LIF neurons, a family of n -compartment LIF neurons. We derive an approximate closed-form expression for the average current flowing into the somatic compartment. Further theoretical analysis of this expression yields rules according to which input channels interact divisively, multiplicatively, or linearly.

Subsequently, in Section 3.4, we apply these theoretical insights to the simplest non-trivial n -LIF neuron, the two-compartment LIF neuron. We derive a convex optimisation problem that allows us to solve for near-optimal synaptic weights, and show that we can exploit this neuron model to compute a wide range of functions at similar or lower errors than two-layer spiking neural networks.

Finally, in Section 3.5, we discuss a general weight-solving method for n -LIF neurons. While we cannot guarantee that the resulting weights are globally optimal, our method typically converges within a few iterations. We show that we can use this method to systematically solve for weights to compute functions such as XOR with a single neuron.

We close with a discussion of our results in Section 3.6. An overview of the software libraries developed to perform the experiments presented here is given in Appendix C.

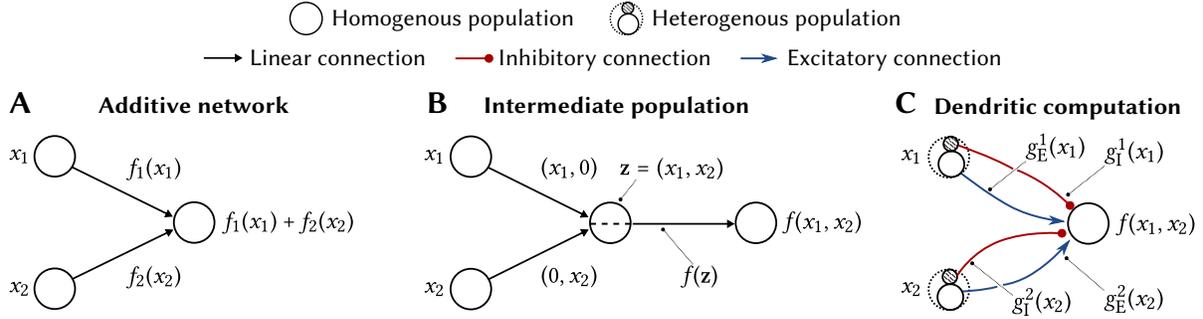


Figure 3.1: Using dendritic computation to approximate multivariate functions in the NEF. **(A)** Standard NEF networks are additive: summation in activity space corresponds to addition in representation space. **(B)** Computing nonlinear multivariate functions f generally requires all variables to be represented in an intermediate population. **(C)** The dendritic computation scheme discussed here. Two pre-populations project onto a post-population with separate excitatory and inhibitory input channels.

3.1 Theoretical Aspects of Dendritic Computation

Dendritic computation as pursued here is best explained by reviewing fundamental properties of neural networks, and exploring how multivariate functions such as $f(x_1, \dots, x_\ell)$ can be approximated in the NEF. Specifically, we compare three different network architectures (cf. Figure 3.1). *Additive networks* represent x_1, \dots, x_ℓ in independent neuron populations. These networks cannot approximate most multivariate functions. In contrast, *two-layer networks* represent all variables in a common intermediate population. These networks are universal approximators. Lastly, our notion of *dendritic computation* relies on nonlinear interaction between independent neural input channels. While not as powerful as multi-layer networks, dendritic computation can approximate a larger class of functions than additive networks.

3.1.1 Additive Multivariate Networks

As stated above, our goal is to compute multivariate functions $f(x_1, \dots, x_\ell)$ within the context of the NEF. For the sake of simplicity, assume that two pre-populations representing variables x_1, x_2 , respectively, are connected to a common post-population. To compute $f(x_1, x_2)$, we must find connection weights $\mathbf{w}_{1,i}, \mathbf{w}_{2,i}$ such that the following holds for every post-neuron i

$$a_i(f(x_1, x_2)) = G_i[\langle \mathbf{e}_i, f(x_1, x_2) \rangle] \stackrel{!}{=} G_i[\langle \mathbf{w}_{1,i}, \mathbf{a}_1^{\text{pre}}(x_1) \rangle + \langle \mathbf{w}_{2,i}, \mathbf{a}_2^{\text{pre}}(x_2) \rangle]. \quad (3.1)$$

Here, a_i is the desired post-neuron activity according to the normative tuning-curve constraint (eq. 2.20). As discussed in the context of the NEF transformation principle (cf. Section 2.3.4), we assume that the current-translation function J_i is part of the individual neuron response curve G_i , and that the currents induced by the pre-populations are summed.

This optimisation problem can be easily solved for multivariate functions that can be decomposed into a sum of two univariate functions, i.e., $f(x_1, x_2) = f_1(x_1) + f_2(x_2)$ (cf. Figure 3.1A).

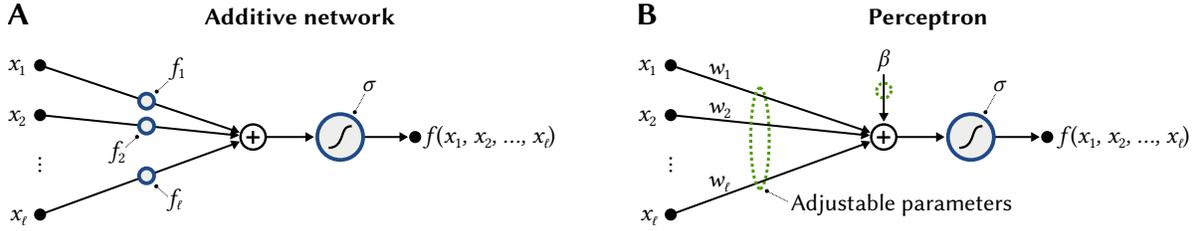


Figure 3.2: Additive networks are a generalisation of the perceptron. **(A)** An additive network is a sum of arbitrary univariate functions. **(B)** A perceptron is an additive network with functions of the form $f_i(x_i) = w_i x_i + \beta \ell^{-1}$. The weights w_i are learned such that the output approximates a desired function.

Specifically, we can find weights $w_{1,i}, w_{2,i}$ that approximate f using the encoder-decoder split of the NEF. Computing decoders $\mathbf{D}^{f_1}, \mathbf{D}^{f_2}$ and applying $(\mathbf{w}_i)^T = \mathbf{e}_i \mathbf{D}$, we have

$$G_i[\langle \mathbf{e}_i \mathbf{D}^{f_1}, \mathbf{a}_1^{\text{pre}}(x_1) \rangle + \langle \mathbf{e}_i \mathbf{D}^{f_2}, \mathbf{a}_2^{\text{pre}}(x_2) \rangle] = G_i[\langle \mathbf{e}_i, \mathbf{D}^{f_1} \mathbf{a}_1^{\text{pre}}(x_1) + \mathbf{D}^{f_2} \mathbf{a}_2^{\text{pre}}(x_2) \rangle] \approx a_i(f_1(x_1) + f_2(x_2)).$$

This equation expresses that addition in activity space (i.e., summing weighted pre-activities, eq. 3.1) is equal to addition in represented space. In other words, standard NEF networks are *additive*. Summing functions incurs no additional decoding error.

Additive networks cannot compute most multivariate functions. The only way to compute general multivariate functions $f(x_1, x_2)$ in additive networks is to *approximate* f as an additive univariate decomposition. As is expressed by the following proposition (see Appendix B.3.1 for a proof), it is impossible to compute many continuous multivariate f in this way.¹ This is true even if we can decode arbitrary univariate functions f_1, \dots, f_ℓ over the pre-variables (this is equivalent to having an infinite number of pre-neurons, see below), and we were able to freely choose a fixed nonlinearity σ (cf. Figure 3.2A).

Theorem 3.1. *Let $\ell > 1$, $\mathbb{X} \subset \mathbb{R}^\ell$ and $\mathbb{Y} \subset \mathbb{R}$ be compact sets of full dimensionality, and σ, f, f_i be continuous. For any fixed $\sigma : \mathbb{R} \rightarrow \mathbb{Y}$, there always exist $f : \mathbb{X} \rightarrow \mathbb{Y}$ such that there is no $f_1, \dots, f_\ell : \mathbb{R} \rightarrow \mathbb{R}$ with $f(x_1, \dots, x_\ell) = \sigma(f_1(x_1) + \dots + f_\ell(x_\ell))$ for all $(x_1, \dots, x_\ell) \in \mathbb{X}$.*

The Perceptron and XOR. Consider monotonic σ and affine f_i of the form $w_i x_i + \beta \ell^{-1}$. We obtain the *perceptron*, an early single-layer neural network (cf. Figure 3.2B; Rosenblatt, 1958). Minsky and Papert (1987, Chapter 2, originally published in 1969) point out that such networks cannot compute the boolean XOR function (Figure 3.3A).² Even the more general additive networks from our theorem cannot solve a *weaker* version of the XOR problem, formalised below.

¹We limit our propositions to continuous functions for the sake of simplicity. However, we conjecture that the same results hold for larger classes of functions, for example square Lebesgue integrable functions.

²Minsky et al. (1987) note that the perceptron was proved by Rosenblatt to “learn to do anything it was possible to program it to do”; this ambiguous statement endowed researchers with a surplus of optimism—especially since perceptrons sometimes learned difficult problems. Among other factors, realising that these networks could not be *programmed* to solve trivial problems such as XOR led to “the first AI winter” (e.g., Muthukrishnan et al., 2020).

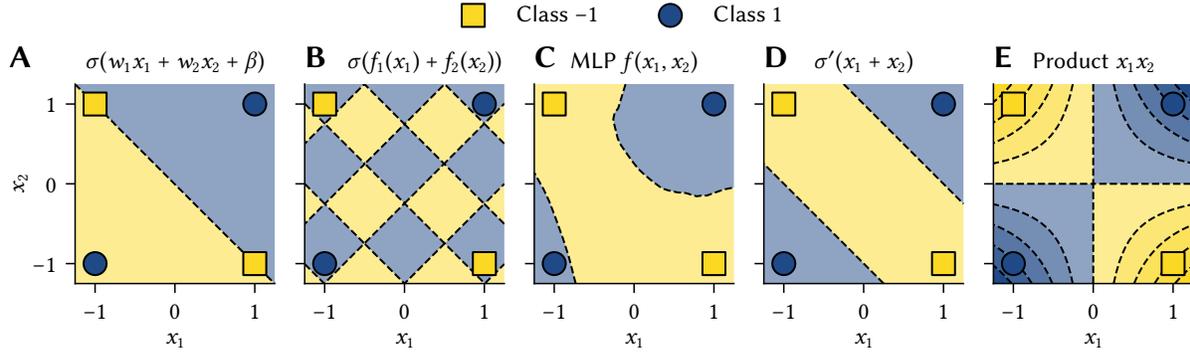


Figure 3.3: Visualisation of the XOR decision problem for different classifiers. The goal is to find classifier parameters such that the four samples are classified as depicted. The background corresponds to the sign of the monotonic function $\sigma(\xi)$. **(A)** The linear decision boundary formed by the Perceptron cannot solve the XOR problem. **(B)** This holds for any function of the form $\sigma(f_1(x_1) + f_2(x_2))$, here $f_1(x_1) = \cos(2\pi x_1)$ and $f_2(x_2) = \sin(2\pi x_2)$. **(C)** A multi-layer Perceptron (MLP) of the form $\sum_i w_i \sigma(e_i^1 x_1 + e_i^2 x_2 + \beta_i)$ can solve the problem, although the decision boundary is quite erratic. **(D)** An alternative solution using the nonlinearity $\sigma'(\xi) = \sigma(\xi^2 - 1)$. **(E)** Multiplication of two real-valued variables x_1, x_2 can be seen as a continuous form of the XOR problem. Additive networks cannot compute this function.

Definition 3.1. A function $\varphi(x_1, x_2)$ solves the weak XOR problem if there exist a_0, b_0, a_1, b_1 with

$$\left(\varphi(a_0, b_0) < \varphi(a_0, b_1) \right) \wedge \left(\varphi(a_1, b_1) < \varphi(a_0, b_1) \right) \wedge \left(\varphi(a_0, b_0) < \varphi(a_1, b_0) \right) \wedge \left(\varphi(a_1, b_1) < \varphi(a_1, b_0) \right).$$

Theorem 3.2. Let σ be monotonic. Then, an additive network of the form $\varphi(x_1, x_2) = \sigma(f_1(x_1) + f_2(x_2))$ cannot solve the weak XOR problem.

This may be surprising, given that, as depicted in Figure 3.3B, we can generate highly nonlinear classification boundaries. We provide a proof in Appendix B.3.2.

To solve the XOR problem, we can either, as discussed next, use multi-layer networks (cf. Figure 3.3C), or, alternatively make σ nonmonotonic. As depicted in Figure 3.3D, setting $\sigma(\xi) = \xi^2 - 1$ allows us to solve the XOR problem. This illustrates our goal with dendritic computation: exploit “more powerful” σ to approximate a larger class of functions.

Still, the functions that we can compute using additive networks are limited, even if we can freely choose σ . For example, we can compute $x_1 x_2$ for $(x_1, x_2) \in [\varepsilon, 1]^2$ and $0 < \varepsilon < 1$ by setting f_1 and f_2 to the logarithm and σ to the exponential. However, it is impossible to find functions that compute multiplication over all four quadrants—which can be seen as a continuous version of the XOR problem (Figure 3.3E). More precisely, allowing x_1, x_2 to be zero makes it impossible to compute multiplication in these networks (proof in Appendix B.3.3).

Theorem 3.3. There are no continuous, real-valued functions f_1, f_2, σ such that $\sigma(f_1(x_1) + f_2(x_2)) = x_1 x_2$ for all $(x_1, x_2) \in [0, 1]^2$.

Intermediate populations. As discussed by Eliasmith and Anderson (2003, Chapter 6), multivariate functions $f(x_1, x_2)$ can in general only be computed in the NEF if all variables x_1, x_2 are represented in a common population. Correspondingly, if all variables are represented in separate univariate populations, we must add an intermediate population to our network that represents a vectorial quantity $\mathbf{z} = (x_1, x_2)$ (cf. Figure 3.1B). We can construct such a representation from two univariate populations by computing the functions $f_1(x_1) = (x_1, 0)$ and $f_2(x_2) = (0, x_2)$ in the connections to the intermediate population. According to Theorem 3.4 we can then decode any multivariate function from the intermediate population.

Potential issues with intermediate populations. In theory, the number of neurons required to cover a d -dimensional space rises exponentially with d . Representing a d -dimensional quantity in an intermediate population can thus require a large number of neurons to achieve a certain decoding error for many different target functions. In practice, it is quite difficult to *a priori* judge the number of neurons required to decode specific functions f with some desired accuracy; the decoding error heavily depends on f and the encoding vectors \mathbf{e}_i .

Another problem arises when modelling neurobiological systems. There may be no indication that an intermediate population exists in a particular biological circuit, although the hypothesised high-level function is a multivariate function. An example of this would be the attention system in layer six of the cortex mentioned in the introduction, where a group of control neurons modulates another population without an intermediary (Bobier et al., 2014).

Finally, there is the issue of noise. In spiking neural networks, every intermediate neuron population introduces additional noise due to static distortion and spike noise (Eliasmith and Anderson, 2003, Section 2.2.2). We see the effects of this later.

3.1.3 Dendritic Computation

Dendritic computation is one way to partially alleviate the limitations arising from intermediate populations. The basic idea is that each neuron possesses k *input channels*. Inputs fed through these channels interact nonlinearly, modelling information processing within the dendrites.

Mathematically, the response curve describing the average neural activity is now a multivariate function $\mathcal{G}[\xi_1, \dots, \xi_k]$, where the ξ_i are linear combinations of the pre-activities (Figure 3.5A). To compute $f(x_1, \dots, x_\ell)$, the following must hold for each post-neuron i

$$a_i(f(x_1, \dots, x_\ell)) \stackrel{!}{=} \mathcal{G} \left[\langle \mathbf{w}_{1,i}^1, \mathbf{a}_1^{\text{pre}}(x_1) \rangle + \dots + \langle \mathbf{w}_{\ell,i}^1, \mathbf{a}_\ell^{\text{pre}}(x_\ell) \rangle, \dots, \right. \\ \left. \langle \mathbf{w}_{1,i}^k, \mathbf{a}_1^{\text{pre}}(x_1) \rangle + \dots + \langle \mathbf{w}_{\ell,i}^k, \mathbf{a}_\ell^{\text{pre}}(x_\ell) \rangle \right] \quad (3.3)$$

where $a_i(f(x_1, \dots, x_\ell))$ expresses the normative tuning constraint defined in eq. (2.20).

Note that we deliberately left the concept of an “input channel” open. An input channel could either refer to a different location in the dendritic tree, different synapse types (e.g., excitatory or inhibitory synapses), or even the influence of other signalling molecules. We discuss a particular neuron model family with multiple input channels in Section 3.3.

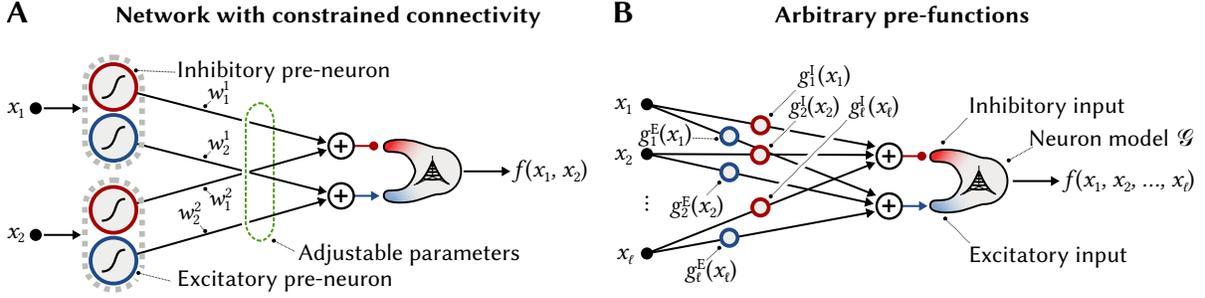


Figure 3.5: Overview of our notion of dendritic computation. **(A)** A neuron with an excitatory and inhibitory input channel. In a network context, these functions are decoded from pre-populations representing these variables. Connectivity can be constrained such that excitatory and inhibitory pre-neurons only connect to the corresponding channel. **(B)** Conceptually, each channel receives a sum of univariate functions computed over the pre-variables x_1, \dots, x_ℓ .

Mathematical analysis. More formally, using $\sigma(\xi_1, \dots, \xi_\ell)$ as an abstract nonlinearity and assuming that we can compute any univariate function g_i^j over ξ_1, \dots, ξ_ℓ , we have (Figure 3.5B)

$$\sigma(g_1^1(x_1) + \dots + g_\ell^1(x_\ell), \dots, g_1^k(x_1) + \dots + g_\ell^k(x_\ell)) = \varphi(x_1, \dots, x_\ell). \quad (3.4)$$

It is trivial to show that such networks are more powerful than additive networks. For example, let $\sigma(\xi_1, \xi_2) = \xi_1 \xi_2$. Now, setting the functions feeding into the second channel to one, i.e., $g_{1,i}^2(x_1) = \dots = g_{\ell,1}^2(x_\ell) = 1$, we obtain an additive network. Of course, in contrast to additive networks, we can use the same σ to compute products of the pre-variables. Still, dendritic computation networks are not universal approximators:

Conjecture 3.5. *Let $\ell > 1$, $\mathbb{X} \subset \mathbb{R}^\ell$ and $\mathbb{Y} \subset \mathbb{R}$ be compact sets of full dimensionality, and σ, f, g_i^j be continuous. For any fixed $\sigma : \mathbb{R}^k \rightarrow \mathbb{Y}$, there always exist $f : \mathbb{X} \rightarrow \mathbb{Y}$ such that there are no $g_1^1, \dots, g_\ell^1, \dots, g_1^k, \dots, g_\ell^k : \mathbb{R} \rightarrow \mathbb{R}$ with the property*

$$f(x_1, \dots, x_\ell) = \sigma(\xi_1, \dots, \xi_k) = \sigma(g_1^1(x_1) + \dots + g_\ell^1(x_\ell), \dots, g_1^k(x_1) + \dots + g_\ell^k(x_\ell)) \text{ for all } \mathbf{x} \in \mathbb{X}.$$

While a rigorous proof eludes us, counting the degrees of freedom (DOF) required to describe an ℓ -dimensional function f suggests that this should be true. Given o orthogonal one-dimensional basis functions, the corresponding ℓ -dimensional basis has o^ℓ basis functions. Correspondingly, o^ℓ DOF are required to describe an ℓ -dimensional f with a basis of order o . However, when using dendritic computation, we only use $k\ell$ one-dimensional functions; correspondingly, there are only $ok\ell$ DOF. For $o \rightarrow \infty$, and $\ell > 1$, $ok\ell$ grows much slower than o^ℓ . Hence, there are always functions that cannot be computed using dendritic computation.⁴

⁴Mathematically, the cardinality of the set of continuous functions f constructed from a function basis of order o is $|\mathbb{R}^{o^\ell}|$, while the cardinality of functions that can be constructed using dendritic computation is $|\mathbb{R}^{ok\ell}|$. Unintuitively, even for $o \rightarrow \infty$, it holds $|\mathbb{R}^{o^\ell}| = |\mathbb{R}^{ok\ell}| = \mathfrak{c}$, where \mathfrak{c} is the cardinality of the continuum (e.g., Jech, 2003, Chapter 4). Fortunately, this is irrelevant in practice. Assuming some baseline precision and dynamic range of the generalised Fourier coefficients, there are only finitely many parameters N that can be used to uniquely characterise each function. It clearly holds $N^{o^\ell} \gg N^{ok\ell}$ for $\ell > 1$ and $o \rightarrow \infty$.

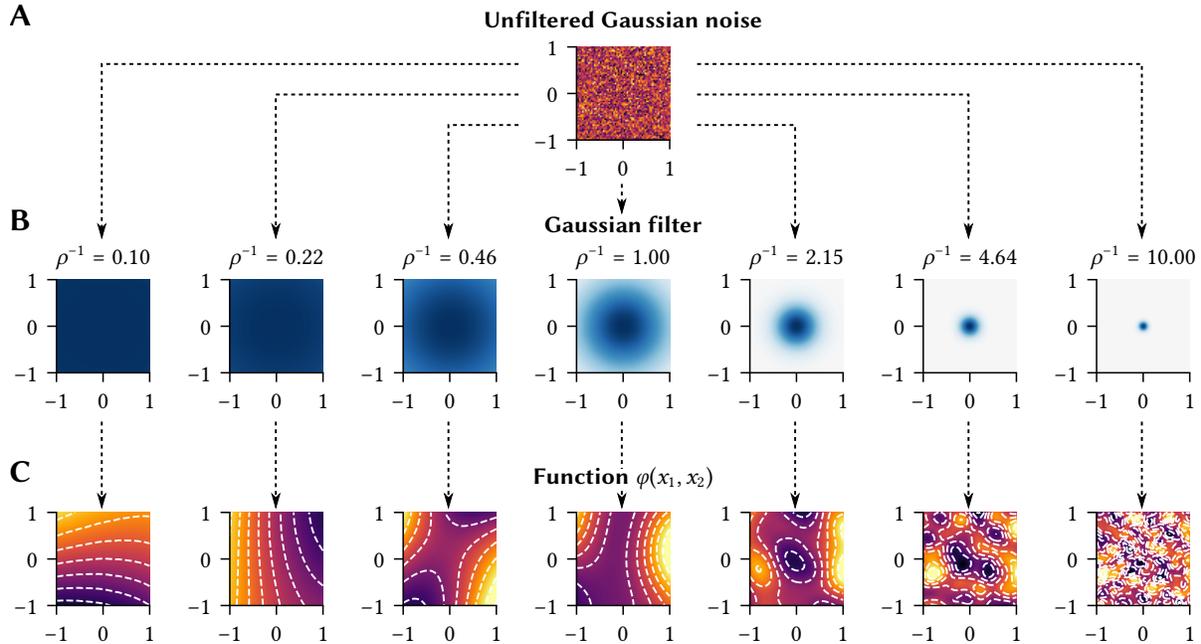


Figure 3.6: Overview of our procedure for generating random 2D functions. **(A)** We sample a 2D array from a normal distribution (only a portion of the array is depicted; the size $n' \times n'$ depends on the filter width). **(B)** The noise is filtered by convolving with a Gaussian kernel with standard-deviation ρ (for large filter widths, i.e., small ρ^{-1} , only a small portion of the filter is depicted). **(C)** The resulting functions are transformed to have mean zero (i.e., no DC component) and a standard deviation of one.

3.1.4 Numerical Exploration

The above discussion offers only limited insight into the practical implications of using dendritic computation. The goal of this section is to provide a numerical analysis of our different network types. Specifically, we characterise the “computational power” of a network by approximating two-dimensional functions $f(x_1, x_2)$ of varying complexity. More “powerful” networks should be capable of computing “more complex” $f(x_1, x_2)$ at lower errors than other networks. Since the “complexity” of a function is somewhat ill-defined, we use frequency content as a proxy. Functions with a higher spatial bandwidth contain more information (cf. Shannon, 1949) than functions only composed of lower frequencies and should thus qualify as “more complex”.

Generating random test functions. To generate sampled random $f(x_1, x_2)$, we use the scheme depicted in Figure 3.6. In a nutshell, we draw $n' \times n'$ points from a normal distribution and apply a 2D Gaussian filter with standard deviation ρ . We then extract the desired $n \times n$ samples not affected by boundary effects, and normalise the result to zero mean and unit standard deviation. As a result, small ρ^{-1} (i.e., large filters), result in almost linear functions, whereas large ρ^{-1} (small filters) result in high-frequency functions.⁵

⁵Note that this scheme generates aperiodic functions. Correspondingly, while the filtered grid of $n' \times n'$ samples is band-limited, the final $n \times n$ samples are not. High frequencies are required to represent the discontinuous

Network models. To compare the above networks types, we could construct neural networks and compare approximation errors for our randomly generated two-dimensional functions $f(x_1, x_2)$. However, as a simplification, we can replace the pre-populations by the first o Legendre polynomials $P_i(x)$.⁶ This is possible because the principal components of neural populations resemble Legendre polynomials (Eliasmith and Anderson, 2003, Chapter 7). Furthermore, using a function basis of order o connects back to our previous theoretical discussion of dendritic computation in terms of degrees of freedom. The number of basis functions that can be decoded well from a pre-population depends on the number of neurons in that population. We explore this in more detail below.

Replacing the pre-population in the additive network with o basis functions, and under the assumption that $\sigma(\xi) = \xi$ (there is no additional nonlinearity), we get

$$f_{\text{add}}(x_1, x_2) = g_1(x_1) + g_2(x_2) = \sum_{i=0}^o w_i^1 P_i(x_1) + \sum_{i=0}^o w_i^2 P_i(x_2), \quad (3.5)$$

where w_i^1, w_i^2 (for $2o$ degrees-of-freedom; DOF) can be computed using least squares. For our analysis of dendritic computation, we choose $\sigma(\xi_1, \xi_2) = \xi_1 \xi_2$ as a nonlinearity σ and obtain

$$\begin{aligned} f_{\text{den}}(x_1, x_2) &= \sigma(g_1^1(x_1) + g_2^1(x_2), g_1^2(x_1) + g_2^2(x_2)) \\ &= \left(\sum_{i=0}^o w_i^1 P_i(x_1) + \sum_{i=0}^o w_i^2 P_i(x_2) \right) \left(\sum_{i=0}^o w_i^3 P_i(x_1) + \sum_{i=0}^o w_i^4 P_i(x_2) \right). \end{aligned} \quad (3.6)$$

Solving for $w_i^1, w_i^2, w_i^3, w_i^4$ (for a total of $4o$ DOF) is a non-convex optimisation problem that tends to possess numerous local minima. We use the BFGS quasi-Newton method (Nocedal and Wright, 2006, Chapter 8) with ten random initialisations and pick the best solution.

Finally, we emulate the universal approximator nature of multi-layer networks by constructing an additive network based on two-dimensional basis functions $P_{ij}(x_1, x_2) = P_i(x_1)P_j(x_2)$:

$$f_{\text{mlp}}(x_1, x_2) = \sum_{i=0}^o \sum_{j=0}^o w_{ij} P_{ij}(x_1, x_2) = \sum_{i=0}^o \sum_{j=0}^o w_{ij} P_i(x_1) P_j(x_2). \quad (3.7)$$

Again, the weights w_{ij} (for a total of o^2 DOF) can be optimally determined using least-squares.

Note that if we had chosen $\sigma(\xi) = \xi^2$ in Equation (3.5), then $f_{\text{add}}, f_{\text{den}}$ and f_{mlp} would all be composed of the same o^2 product-terms. Hence, the difference between these networks ultimately lies in the degrees of freedom; additive networks and dendritic computation are fundamentally constrained in how these product-terms can be combined. In particular, we can expect f_{den} to compare favourably well to f_{mlp} if $4o \geq o^2$, that is $o \leq 4$. This is the case if only the first few basis functions can be decoded well from the pre-population.

boundary—a result of applying a box-window to our data (cf. the “windowing theorem”; e.g., Oppenheim and Schaffer, 2009, Section 2.9.7). Hence, there is no trivial way to sample these functions directly in the frequency domain, and to make use of the fast inverse Fourier transform. Still, this scheme can be implemented efficiently by separating the Gaussian filters into horizontal and vertical components (e.g., Bolon, 2006).

⁶We discuss Legendre polynomials in Section 4.2.2. Note that we use a discretised version of the Legendre basis, “Discrete Legendre Orthogonal Polynomials” (DLOPs; Neuman and Schonbach, 1974; Stöckel, 2021b).

3.1. Theoretical Aspects of Dendritic Computation

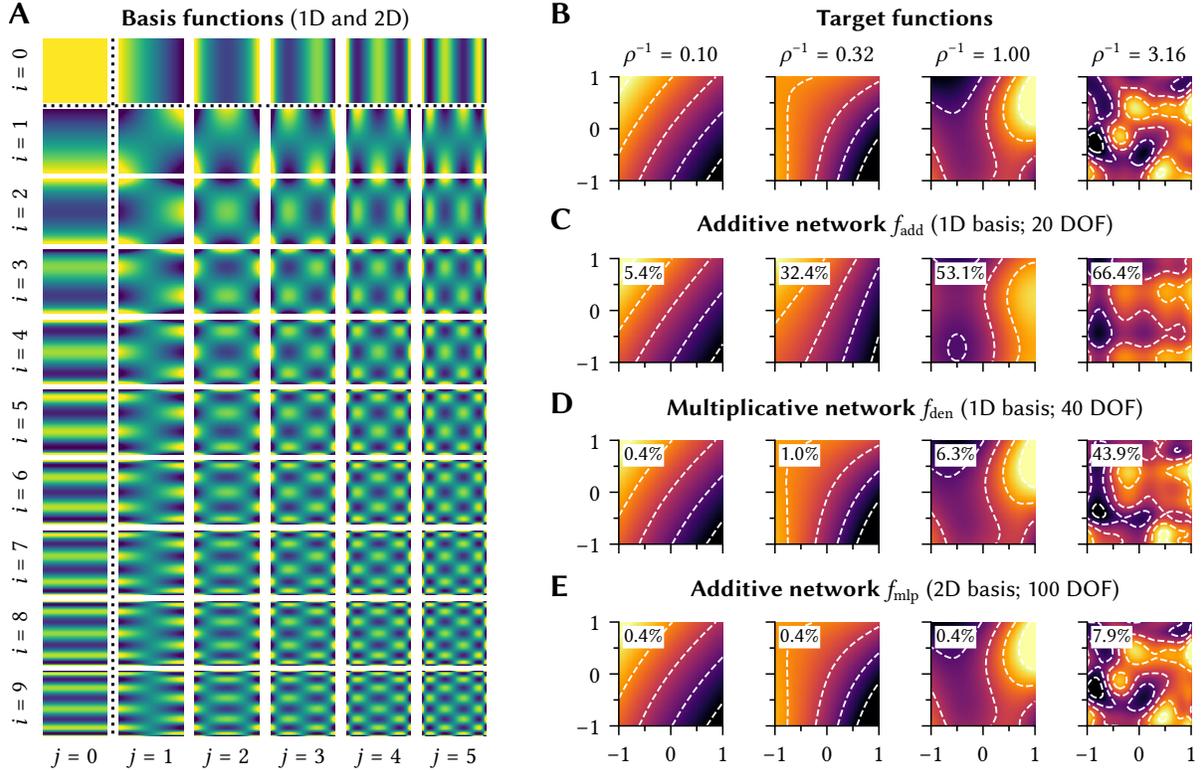


Figure 3.7: Function approximation using different network setups. **(A)** Grid of the first 10×7 2D Legendre basis functions $\tilde{P}_{ij}(x_1, x_2)$. The 1D basis functions depending only on x_1 and x_2 are to the left/top of the dashed lines. **(B)** Randomly generated target functions of different complexity ρ^{-1} . **(C, D, E)** Approximations of the target functions using the different networks discussed in the text. Inset error values are the normalised RMSE in percent.

Experiment: Decoding random functions. We compare the three network types by varying the spatial low-pass filter size ρ^{-1} and the basis order o . Figure 3.7 provides an overview of the overall procedure. The one- and two-dimensional Legendre polynomials are depicted in Figure 3.7A, randomly generated target functions are depicted in Figure 3.7B. These functions are reconstructed using the function approximators defined above (Figures 3.7C to 3.7E).

As expected, the network type has a large influence on the reconstruction error. While the additive network f_{add} only fares well for almost linear functions, f_{mlp} reaches reasonable approximation errors even for complex target functions. The dendritic network f_{den} sits somewhere in between; f_{den} can be used to decode the target function for $\rho^{-1} = 1$ at a reasonable 6% NRMSE, while failing to reconstruct the target function for $\rho^{-1} = 3.16$ at a 44% error.

Figure 3.8 depicts the results of a more systematic experiment. For a fixed basis function count (Figure 3.8A), the error achieved with an additive network f_{add} increases almost linearly with ρ^{-1} on a log-log plot, while both f_{den} and f_{mlp} possess a sublinear plateau.

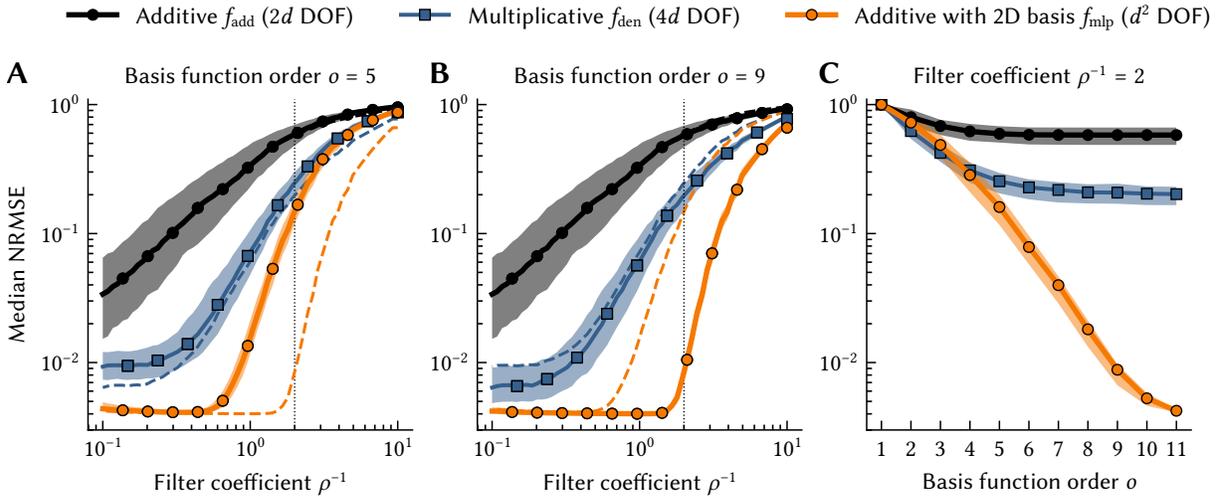


Figure 3.8: Computing functions with different spatial frequencies using different network types. Each line is the median NRMSE when approximating with respect to the RMS of the target function over $N = 1000$ randomly generated functions with spatial filter coefficient ρ^{-1} using d Legendre basis functions (cf. Figure 3.6). Shaded areas correspond to the 25/75-percentile. **(A, B)** Median error for different filter coefficients ρ^{-1} for fixed basis function counts d . Dashed lines correspond to the median depicted in the neighbouring diagram. Vertical dotted line is at $\rho^{-1} = 2$. **(C)** Median error over different basis function counts and for a fixed filter coefficient $\rho^{-1} = 2$. Increasing the basis function count only substantially decreases the error in the case of an additive network with 2D bases.

Increasing the basis order o (Figure 3.8B) has only a small effect on f_{add} and f_{den} . We only observe a small decrease in error for f_{den} below $\rho^{-1} < 0.5$, whereas there is virtually no change in error for f_{add} . Conversely, in the context of f_{mlp} , the entire error curve is shifted to the right when increasing o . In other words, increasing the basis function order o truly allows f_{mlp} to compute more complex functions, whereas f_{add} and f_{den} are fundamentally limited in the maximum complexity of the functions that can be computed in this manner.

This is even more clearly visible in Figure 3.8C, where we keep the spatial filter coefficient ρ^{-1} fixed, and sweep over the basis order o instead. Increasing o consistently decreases the error for f_{mlp} down to the base-line error induced by regularisation, while the errors for f_{add} and f_{den} plateau much sooner. Interestingly, the point at which f_{den} and f_{mlp} diverge is at $o = 4$, which is exactly as we predicted above.

Experiment: Determining the number of decodable basis functions d . An intrinsic assumption of the above experiment (particularly in f_{mlp} , eq. 3.7) is that the number of decodable basis functions d is given as $d = o^\ell$, where o is the order of the underlying basis, and ℓ is the number of dimensions represented by the pre-population. However, in practice, this is almost never the case—increasing the dimensionality of a neuron population only has a small impact on the absolute number of decodable basis functions.

3.1. Theoretical Aspects of Dendritic Computation

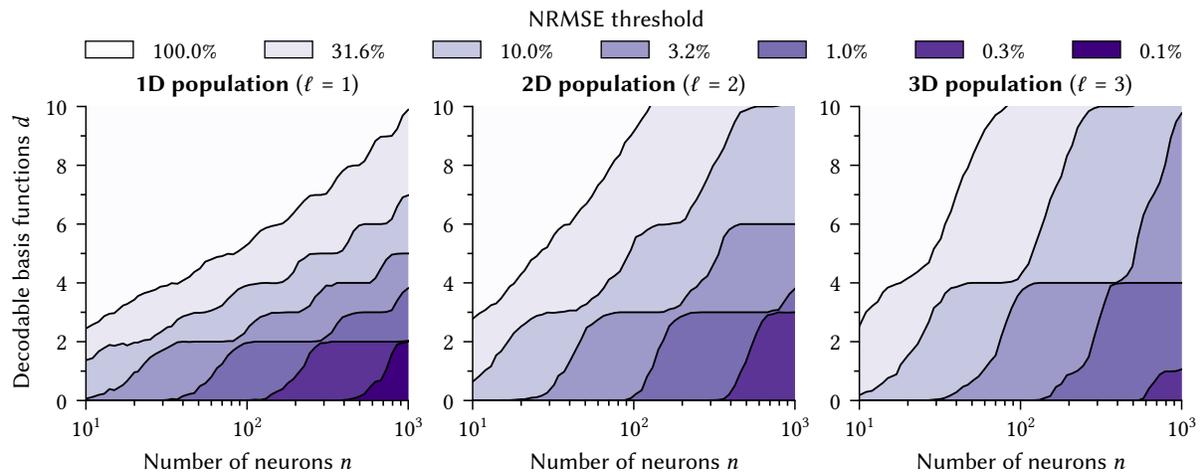


Figure 3.9: Number of decodable basis functions over the number of neurons. Each line is the mean number of Legendre polynomials that can be decoded with an error smaller than the indicated threshold from a ReLU neuron population of size n (mean is over 100 trials). The three columns correspond to a one-, two- and three-dimensional population. The total number of basis functions tested for each data point is 20^ℓ , where ℓ is the dimensionality of the population. The number of decodable basis functions is not drastically larger for higher-dimensional neuron populations.

This is depicted in Figure 3.9, where we vary the number of neurons in the pre-population and count the number of ℓ -dimensional Legendre basis functions that can be decoded with an NRMSE below some threshold. Given a 3.2% error threshold and 1000 neurons, we are able to decode $d = 5$ orthogonal basis functions from the one-dimensional population, $d = 6$ from the two- and $d = 10$ functions three-dimensional population. While this is a substantial increase in the number of decodable functions, it is nowhere near the exponential increase that we would expect from increasing the order of the underlying basis.

This suggests that, in some scenarios, dendritic computation schemes could outperform multi-layer networks. Let d and d' be the number of basis functions that can be decoded from a one- and ℓ -dimensional population, respectively. Dendritic computation has more DOF if $k\ell d > d'$, and could thus approximate a larger class of functions below some error threshold.

This analysis should be taken with a grain of salt, since we do not take redundant DOF into account and rely on a questionable binary notion of a basis function being “decodable”. Still, as we will see in Sections 3.4 and 3.5, dendritic computation can indeed outperform multi-layer networks in practice.

Conclusion. Our numerical experiments confirm that, under optimal circumstances, dendritic computation schemes can only compute some functions better than purely additive networks. This is independent of the order of the function basis induced by the pre-populations. Our analysis also suggests that dendritic computation may be a viable alternative if only few multivariate basis functions can be decoded well from an intermediate population.

3.2 Extending the Neural Engineering Framework

Up to this point, we have formally defined dendritic computation and discussed its theoretical benefits. The goal of this section is to open avenues toward systematically integrating multi-compartment neuron models with dendritic trees into NEF networks using the formalisms discussed above.

Naïvely, incorporating neurons with multiple nonlinear input channels into the NEF is merely a matter of solving for \mathbf{w} such that eq. (3.3) holds. Phrasing this as an optimisation problem, we must minimise the difference between the desired average activity according to the normative tuning-curve constraint $a_i(\mathbf{x})$ and according to our multivariate response curve \mathcal{G} . Using a least-squares loss (and omitting the regularisation term), we have

$$E = \frac{1}{\text{vol}(\mathbb{X})} \int_{\mathbb{X}} \left(a_i(\varphi(x_1, \dots, x_\ell)) - \mathcal{G}[\langle \mathbf{w}_{1,i}^1, \mathbf{a}_1^{\text{pre}}(x_1) \rangle + \dots + \langle \mathbf{w}_{\ell,i}^1, \mathbf{a}_\ell^{\text{pre}}(x_\ell) \rangle, \dots, \langle \mathbf{w}_{1,i}^k, \mathbf{a}_1^{\text{pre}}(x_1) \rangle + \dots + \langle \mathbf{w}_{\ell,i}^k, \mathbf{a}_\ell^{\text{pre}}(x_\ell) \rangle] \right)^2 dx_1 \dots dx_\ell. \quad (3.8)$$

One way to minimise this loss-function would be to use stochastic gradient descent. In fact, this can be a viable strategy—and may be the only option for many detailed neuron models.

Still, we would like to suggest a more systematic approach that, in some cases, reduces the task of finding weights to a convex quadratic program. This is more in line with the “standard” NEF, where connection weights are computed by solving a convex least-squares problem.

To arrive at a point where we can integrate complex neuron models more seamlessly, we first need to address two of the limitations of the NEF discussed in Section 2.3.6. Specifically, we discuss how to eliminate the bias currents and to account for Dale’s principle. Furthermore, we present a modified version of Equation (3.8) that splits the multivariate response curve \mathcal{G} into a multivariate input-dependent nonlinearity H and a univariate response curve G . We avoid decoding subthreshold currents using a technique we call “subthreshold relaxation”.

3.2.1 Decoding the Current-Translation Function

So far we assumed that the current translation function $J_i(\xi)$ is an intrinsic part of the neuron model. Our typical choice of $J_i(\xi) = \alpha_i \xi + \beta_i$ introduces a bias current β_i into each neuron. As we elaborated in Section 2.3.6, this is slightly implausible from a biological perspective.

Tripp and Eliasmith (2007) demonstrate that it is possible to robustly solve for synaptic weights that approximate arbitrary post-synaptic current functions. We use this insight to directly approximate the target current $J_i(\langle \mathbf{e}_i, \mathbf{x} \rangle)$; as a side effect, we implicitly solve for the bias. Again, assuming that the post-synaptic current is linear in the pre-population activities, we must find a weight vector \mathbf{w}_i such that the following regularised loss is minimised

$$E = \frac{1}{\text{vol}(\mathbb{X})} \int_{\mathbb{X}} \left(J_i(\langle \mathbf{e}_i, \varphi(\mathbf{x}) \rangle) - \langle \mathbf{w}_i, \mathbf{a}^{\text{pre}}(\mathbf{x}) \rangle \right)^2 d\mathbf{x} + \lambda \|\mathbf{w}_i\|_2^2. \quad (3.9)$$

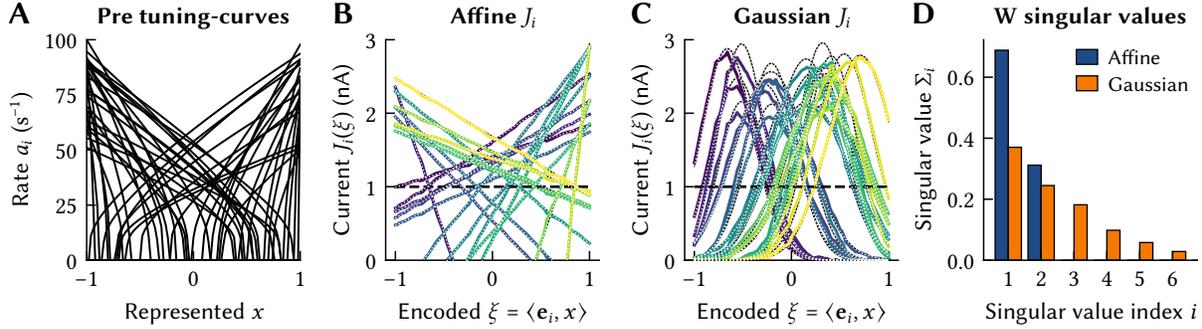


Figure 3.10: Decoding for currents instead of represented values. **(A)** Tuning curves of a pre-population with 100 LIF neurons (only 50 tuning curves are shown). **(B)** Decoding affine current translation functions J_i (dotted lines are the target). The function φ being computed in representation space is the identity function. Dashed line corresponds to the threshold current $J_{\text{th}} = 1$ nA. **(C)** Same as **(B)** but for Gaussian current-translation functions J_i . Such functions can be used to produce localised tuning curves. **(D)** First six singular values of the two weight matrices \mathbf{W} from **(B, C)**. Singular values are normalised by dividing by their sum, resulting in the relative contribution of each singular value to the decoding. For affine J_i , \mathbf{W} is of rank $d + 1$ (here $d = 1$); Gaussian J_i result in full-rank \mathbf{W} .

As before, this equation can be discretised, brought into canonical least squares form, and solved using the regularised Moore-Penrose pseudo inverse (cf. eqn. 2.22 and 2.23):

$$\mathbf{W} = \mathbf{A}^+ \mathbf{J}, \quad \text{where} \quad \mathbf{A}^+ = (\mathbf{A}^T \mathbf{A} + \lambda \mathbf{N} \mathbf{I})^{-1} \mathbf{A}^T.$$

Here, N is the number of samples, $\mathbf{W} \in \mathbb{R}^{m \times n}$ is the connection weight matrix, $\mathbf{J} \in \mathbb{R}^{N \times m}$ is a matrix of target currents, and $\mathbf{A} \in \mathbb{R}^{N \times n}$ is the matrix of pre-activities.

Importantly, we no longer solve for weights directly in the domain of represented values \mathbf{x} ; this is in contrast to solving for decoders \mathbf{D} according to eq. (2.22). Similarly, we do not solve for target activities a_i , as was suggested by our naïve loss function in eq. (3.8). Side-stepping the neural nonlinearity G enables a simple least-squares solution. Furthermore, this optimisation scheme supports arbitrary current-translation functions J_i , providing modellers with a greater flexibility over the tuning curve constraint (cf. Figures 3.10A to 3.10C).

Low-rank factorisation of \mathbf{W} . Solving the optimisation problem in eq. (3.9) directly results in a weight matrix \mathbf{W} instead the low-rank factorisation $\mathbf{W} = \mathbf{E} \mathbf{D}^\varphi$. As discussed in Section 2.3.4, this factorisation was useful, since it enables $\mathbf{W} \mathbf{a}$ to be computed in $\mathcal{O}(n)$ instead of $\mathcal{O}(n^2)$.

Fortunately, at least in the case of the affine current-translation function $\alpha_i \langle \mathbf{e}_i, \varphi(\mathbf{x}) \rangle + \beta_i$, we still obtain a factorisable matrix (cf. Figure 3.10D). The resulting \mathbf{W} is merely of rank $d + 1$, where d is the dimensionality of the post-population. Specifically, the weight matrix can be expressed as a sum of the low-rank factorisation $\mathbf{E} \mathbf{D}^\varphi$ and the outer product of the biases $\boldsymbol{\beta} \in \mathbb{R}^{m \times 1}$ with a decoding vector $\mathbf{D}^1 \in \mathbb{R}^{1 \times n}$. This “bias-decoder” decodes the constant “one” from the pre-population. In other words, it simply holds $\mathbf{W} = \mathbf{E} \mathbf{D}^\varphi + \boldsymbol{\beta} \mathbf{D}^1$ (cf. Stöckel, Voelker, and Eliasmith, 2017; Duggins, 2017, Chapter 4).

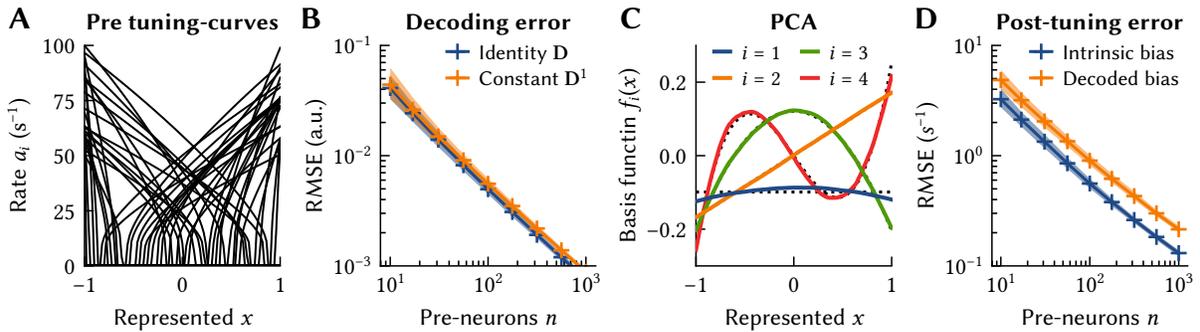


Figure 3.11: Bias decoding and post-population tuning curve accuracy. **(A)** Pre-population tuning-curves for $n = 50$ LIF neurons. **(B)** Error for decoding the identity function compared to decoding a constant (regularisation factor $\sigma = 10$). The error for decoding a constant is minimally larger than that for decoding the identity function. **(C)** The first four principal components of the tuning curves (for $n = 1000$). The principal components resemble the Legendre polynomials, an orthogonal function basis (dotted lines). **(D)** RMSE between the desired post-population tuning and the actually achieved tuning. Decoding the bias approximately doubles the error compared to intrinsic bias currents.

Impact of decoding bias currents on network function. Generally speaking, decoding biases increases the error between the actual and desired post-population tuning. The magnitude of this error depends on the pre- and post-population. The former determine how well a constant offset can be decoded, the latter determine the magnitude of the required bias currents.

In the case of “standard” NEF tuning with uniform x -intercepts and random encoders (cf. Figure 3.11A), constant functions can, counter-intuitively, only be decoded with a slightly higher error than the identity function (error is about 15% higher; cf. Figure 3.11B). This becomes apparent when considering the principal components of the pre-population tuning curves. As, for example, discussed in Eliasmith and Anderson (2003, Chapter 7), the principal component analysis (PCA) can be seen as “uncovering” the best orthogonal basis that linearly generates the tuning-curves. In turn, the first principal components characterise the functions that can be decoded well from a population. As illustrated in Figure 3.11C, the principal components f_i of the “standard” NEF tuning curves resemble the Legendre polynomials (cf. Section 4.2.2 for a definition). While the second principal component f_2 is linear, just like the corresponding Legendre polynomial, f_1 differs significantly from the constant first Legendre polynomial. Decoding constants is hence “more difficult” than decoding the identity function.

In our example, and as depicted in Figure 3.11D, decoding J_i doubles the RMSE between the desired and actual post-population tuning. However, as we will see in the next subsection, there are circumstances where the absence of an intrinsic bias improves the network performance.

Accounting for multiple pre-populations. As we discussed in Section 3.1.1, a welcome side effect of intrinsic current-translation is that standard NEF networks are additive. Summing the activities $\mathbf{a}_1^{\text{pre}}, \dots, \mathbf{a}_\ell^{\text{pre}}$ from multiple pre-populations is equivalent to summing the decoded $f_1(\mathbf{x}_1), \dots, f_\ell(\mathbf{x}_\ell)$. This is no longer the case when minimising the current-based loss in eq. (3.9).

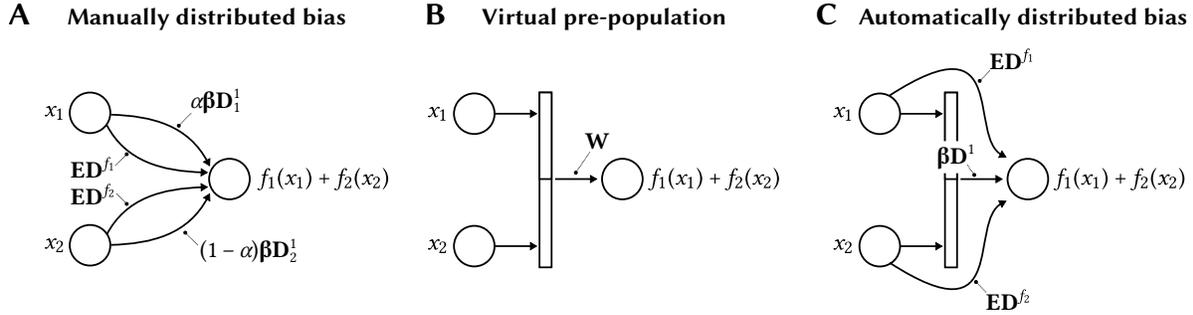


Figure 3.12: Accounting for multiple pre-populations when decoding the current-translation function. **(A)** Biases can be manually distributed between pre-populations by scaling the bias decoders D_1^1 and D_2^1 by α and $(1 - \alpha)$, respectively. **(B)** The general solution is to solve for weights \mathbf{W} assuming stacked pre-population activities. The two pre-populations form a “virtual” pre-population. **(C)** A combination of the two approaches, where only the bias D^1 is decoded from the pre-populations in this way.

In the case of the affine J_i , each of the ℓ connections decodes the bias current, effectively multiplying the bias by ℓ . Of course, we can only decode a fraction of the bias from each pre-population (Figure 3.12A). Alternatively, we can combine the pre-populations into a “virtual pre-population” and let the optimisation process take care of distributing the responsibility for providing the bias between all pre-neurons (Figure 3.12B).

More precisely, we explicitly solve for weights that result in $f_1(\mathbf{x}_1) + \dots + f_\ell(\mathbf{x}_\ell)$ to be represented in the post-population. For two populations, and skipping regularisation, we have

$$E = \frac{1}{\text{vol}(\mathbb{X})^2} \iint_{\mathbb{X}} (J_i(\langle \mathbf{e}_i, f_1(\mathbf{x}_1) + f_2(\mathbf{x}_2) \rangle) - \langle \mathbf{w}_{1,i}, \mathbf{a}_1^{\text{pre}}(\mathbf{x}_1) \rangle - \langle \mathbf{w}_{2,i}, \mathbf{a}_2^{\text{pre}}(\mathbf{x}_2) \rangle)^2 d\mathbf{x}_1 d\mathbf{x}_2. \quad (3.10)$$

We can bring this problem into a canonical form by stacking the pre-activities and weights (see below for an example). However, note that we now need to sample a much higher-dimensional space. This is worrisome if we attempt to decode f_i that must be finely sampled to obtain a good decoding. Again, under the assumption that J_i is affine, it is possible to expand the above integral and to solve for a—relatively easy to decode—population-spanning bias decoder D^1 independent of the (untouched) function decoders D^{f_1} and D^{f_2} (Figure 3.12C).

3.2.2 Nonnegative Weights and Dale’s Principle

Biological neurons tend to follow Dale’s principle—they act either excitatorily or inhibitorily (see Section 2.2.1 for more detail). As discussed in Section 2.3.6, we ignored this in our weight-solving procedures. Least-squares assigns arbitrary algebraic signs to the individual weights, typically with an even split between positive and negative (cf. Figure 2.32). Such weights are not compatible with conductance-based synapses or, more generally, multi-compartment neurons, where modellers may connect pre-neurons to specific post-neuron channels. The corresponding connection weights describe nonnegative quantities, such as the number of vesicles released from the pre-synapse, or the ion-channel density (Roth et al., 2009).

Solving for weights using nonnegative least squares. Solving for individual synaptic weights in current space suggests a simple procedure to account for nonnegativity. Assume that each population is arbitrarily split into a group of excitatory and inhibitory neurons. The somatic input current of post-neuron i in response to pre-synaptic activity is $\langle \mathbf{w}_i^+, \mathbf{a}^+(\mathbf{x}) \rangle - \langle \mathbf{w}_i^-, \mathbf{a}^-(\mathbf{x}) \rangle$; here, \mathbf{w}_i^+ , \mathbf{w}_i^- are nonnegative excitatory and inhibitory weight vectors and $\mathbf{a}^+(\mathbf{x})$, $\mathbf{a}^-(\mathbf{x})$ are the activities of the excitatory and inhibitory neurons in the pre-population. Combining this current term with eq. (3.9) yields the following optimisation problem for each post-neuron i

$$\min_{\mathbf{w}_i^+, \mathbf{w}_i^-} \frac{1}{\text{vol}(\mathbb{X})} \int_{\mathbb{X}} (J_i(\langle \mathbf{e}_i, \varphi(\mathbf{x}) \rangle) - \langle \mathbf{w}_i^+, \mathbf{a}^+(\mathbf{x}) \rangle + \langle \mathbf{w}_i^-, \mathbf{a}^-(\mathbf{x}) \rangle)^2 d\mathbf{x} + \sigma^2 \|\mathbf{w}_i^+\|_2^2 + \sigma^2 \|\mathbf{w}_i^-\|_2^2 \quad (3.11)$$

subject to $\mathbf{w}_i^+, \mathbf{w}_i^- \geq 0$.

To obtain a canonical least-squares form, let N be the number of samples, n^+ , n^- be the number of excitatory and inhibitory pre-neurons,⁷ and \mathbf{A} be the sampled, stacked, and signed pre-activities $(\mathbf{A}^+, -\mathbf{A}^-) \in \mathbb{R}^{N \times (n^+ + n^-)}$. Additionally, let \mathbf{W} be the stacked weight matrices $(\mathbf{W}^+, \mathbf{W}^-) \in \mathbb{R}^{(n^+ + n^-) \times m}$, and $\mathbf{J} \in \mathbb{R}^{N \times m}$ be a matrix of sampled target currents. We have

$$\|(\mathbf{A}^T \mathbf{A} + N\sigma^2 \mathbf{I})\mathbf{W} - \mathbf{A}^T \mathbf{J}\|_2^2 \quad \text{subject to } \mathbf{W} \geq 0. \quad (3.12)$$

This is a standard nonnegative least-squares (NNLS) problem that can be solved in polynomial time (Lawson and Hanson, 1995, Chapter 23). An overview of efficient algorithms to solve this kind of problem is given in Chen and Plemmons (2009).⁸ Of course, similarly to eq. (3.10), the optimisation problem can be extended to take multiple pre-populations into account.

Impact of nonnegative weights on network function. The degree to which separating populations into excitatory and inhibitory sub-populations impacts network function once again depends on the pre- and post-population tuning, as well as the φ that we would like to compute.

We explore this in the network depicted in Figure 3.13. Decoding errors are small over a wide range of ratios between the excitatory and inhibitory pre-neurons. However, without additional precautions (see below) information cannot be transmitted over purely inhibitory connections. In contrast, purely excitatory connections can work reasonably well, at least when computing the identity function $\varphi(x) = x$ and when decoding the bias from the pre-population. We further reduce this error below, using “subthreshold relaxation”.

Note that purely excitatory connections do not work for the selected post-tuning in the presence of intrinsic bias currents. The excitatory input cannot counter positive β_i for neurons with negative x -intercepts, making it impossible to reach firing rates smaller than $a_i(\beta_i)$.⁹

⁷It does not necessarily hold that $n = n^+ + n^-$; neurons can *technically* be marked as both excitatory and inhibitory. Specifically, the special case $n = n^+ = n^-$ reduces the NNLS problem to standard least squares.

⁸Most linear algebra software packages bundle a solver for nonnegative least-squares; for example `scipy.optimize.nnls` in SciPy or `lsqnonneg` in Matlab. Alternatively, a general quadratic programming (QP) solver can be used; this is what we do in our library *libnlif* that we discuss in Appendix C.1.

⁹This phenomenon is also described in the documentation for the Nnls solver in Nengo. Our optimisation procedure differs from that in Nengo in that we account for excitatory and inhibitory pre-neurons and that we can choose to decode the current translation function, eliminating the post-population tuning restrictions.

3.2. Extending the Neural Engineering Framework

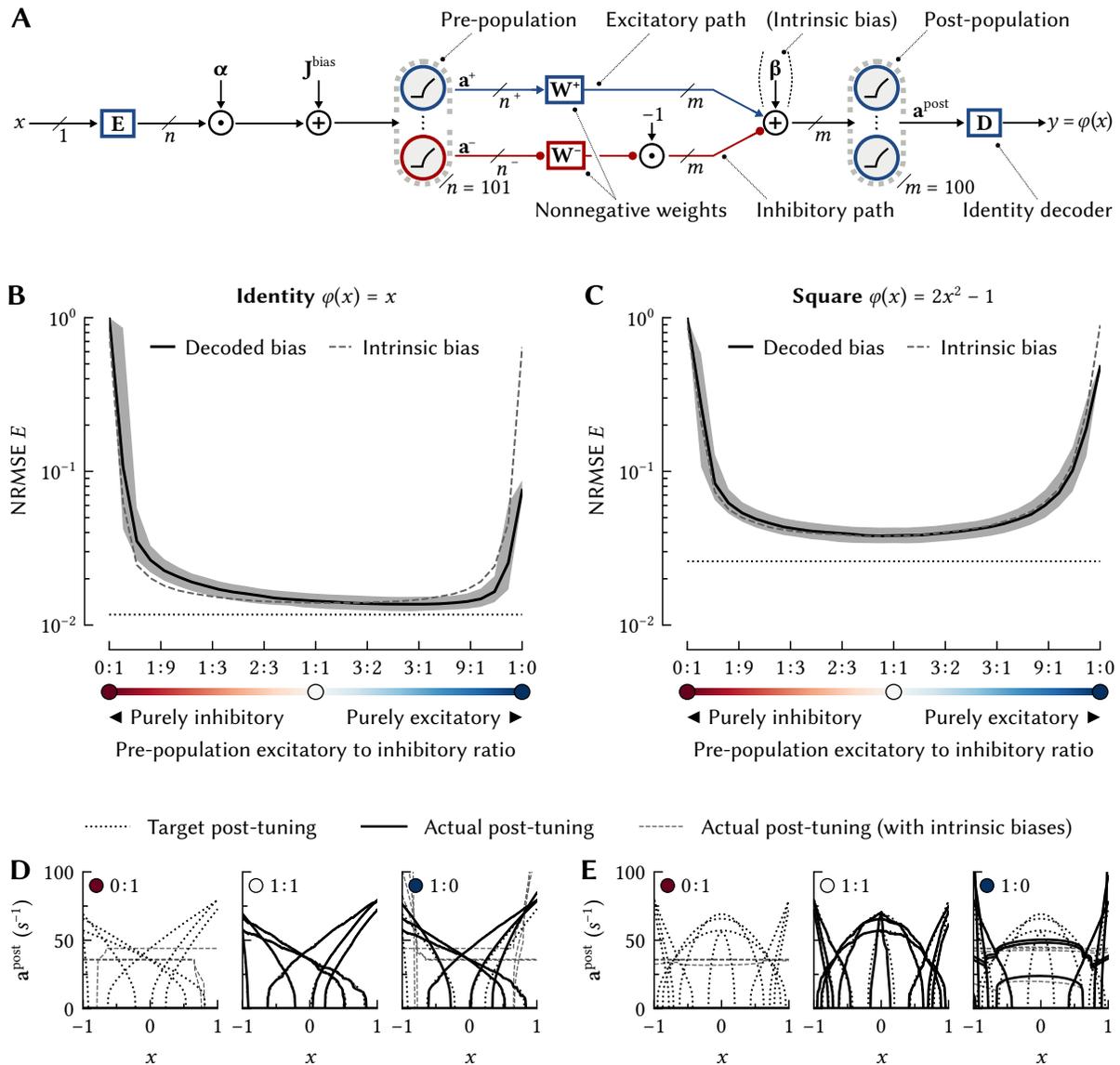


Figure 3.13: Impact of the ratio between excitatory to inhibitory neurons on network function. **(A)** A variable x is represented in a pre-population. This population is randomly split into n^+ excitatory and n^- inhibitory neurons. Nonnegative weights W^+ , W^- are optimised according to eq. (3.12). An identity decoder D is used to decode the value represented in the post-population. **(B, C)** Median normalised RMSE (relative to the RMS of $\varphi(x)$) between the decoded value and the desired value $\varphi(x) = x$ (B) or $\varphi(x) = 2x^2 - 1$ (C) for different ratios $n^+ : n^-$ over 1000 runs ($n = m = 100$, $\sigma = 10$, maximum rates between 50 and 100). The shaded area depicts the 25/75 percentiles. The dashed line depicts results for the intrinsic biases. The horizontal dotted line is the least-squares baseline. Except for extreme $n^+ : n^-$, the network works well over a large range of ratios. Intrinsic biases are detrimental in purely excitatory networks. **(D, E)** Examples of desired versus actual post-population tuning at different excitatory to inhibitory pre-neuron count ratios. A reasonably good post-population tuning can be obtained for purely excitatory pre-populations and a decoded (non-intrinsic) bias.

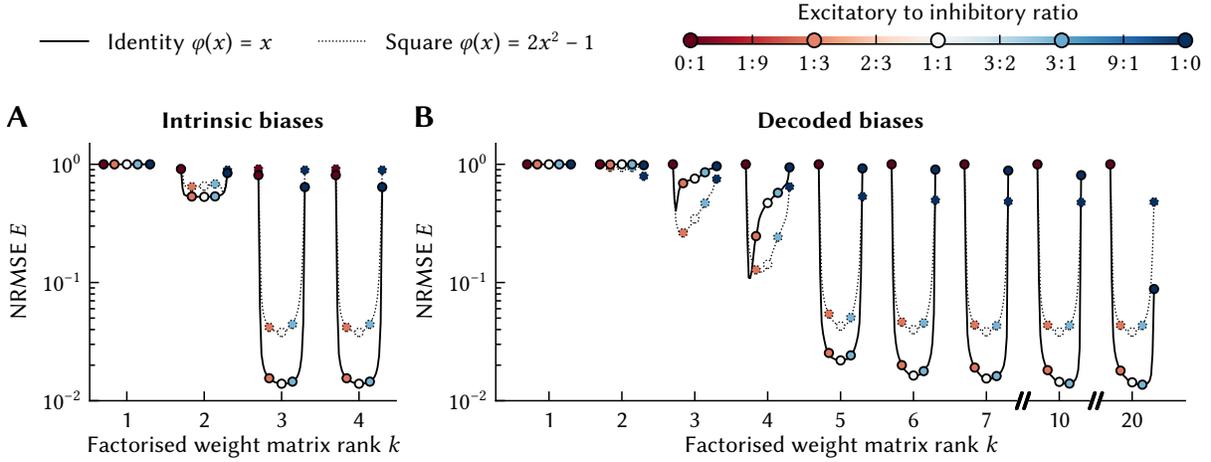


Figure 3.14: Rank-reduced factorisation of nonnegative weight matrices. Same experiment as in Figure 3.13, but for independently factorised and rank-reduced excitatory and inhibitory weight matrices \mathbf{W}^+ , \mathbf{W}^- (see text). Each curve corresponds to a sweep over the excitatory to inhibitory ratio (cf. figs. 3.13B and 3.13C). **(A)** With intrinsic biases, the nonnegativity increases the effective rank of the weight matrices by one. **(B)** When decoding the biases, factorisations with higher ranks are required, particularly for purely excitatory connections (likely due to higher sparsity, see fig. 3.15C).

Factorisability of nonnegative weight matrices. As we discussed in the previous subsection, directly solving for weights \mathbf{W} forfeits the computationally efficient low-rank factorisation $\mathbf{W} = \mathbf{E}\mathbf{D}^\varphi$. At least for affine J_i it is possible to work around this using the bias decoder \mathbf{D}^1 .

Such simple workarounds are no longer possible for nonnegative \mathbf{W}^+ , \mathbf{W}^- . Still, we can construct low-rank approximations of \mathbf{W}^+ and \mathbf{W}^- using their singular value decomposition. Generally, a rank- k factorisation of a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ can be obtained according to

$$\mathbf{M}_{(k)} = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^T, \quad \text{where } \mathbf{U}^T \Sigma \mathbf{V} = \mathbf{M} \text{ and } \Sigma = \text{diag}(\sigma_1, \dots, \sigma_{\min\{m,n\}}) \quad (3.13)$$

with $\sigma_1 \geq \dots \geq \sigma_{\min\{m,n\}}$

According to the Perron-Frobenius theorem and generalisations thereof for square matrices (Avin et al., 2013), $\mathbf{M}_{(1)}$ is nonnegative if \mathbf{M} is nonnegative (at least for practically relevant classes of \mathbf{M}). Since σ_1 is the dominating singular value, and the σ_k tend to decay quickly in magnitude, the higher-rank factorisations $\mathbf{M}_{(k)}$ will mostly be nonnegative. Still, nonnegativity of $\mathbf{M}_{(k)}$ is not guaranteed. When factorising \mathbf{W}^+ and \mathbf{W}^- in this manner, we hence suggest ensuring that the decoded currents $\mathbf{J} \in \mathbb{R}^m$ injected into the post-neurons are nonnegative, i.e.,

$$\mathbf{J} = \max(0, \mathbf{W}_{(k)}^+ \mathbf{a}^+) - \max(0, \mathbf{W}_{(k)}^- \mathbf{a}^-).$$

We explore this factorisation in Figure 3.14. Typically, a relatively small $k \ll \min\{m, n\}$ suffices to obtain low decoding errors in the post-population.

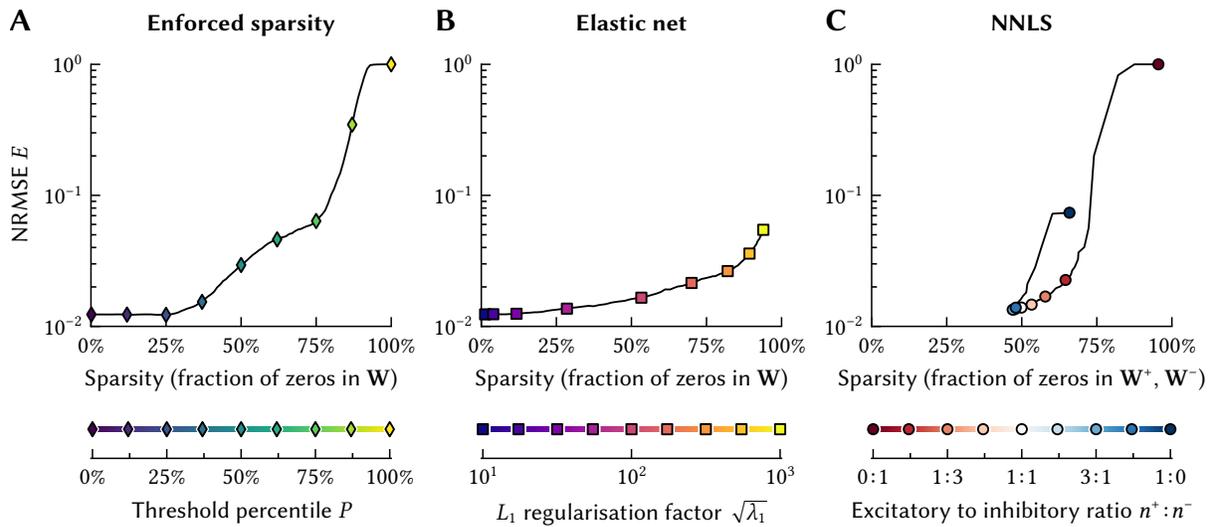


Figure 3.15: Comparison of weight optimisation schemes in terms of sparsity. Curves (*top*) depict sparsity and decoding errors over different hyperparameters (*bottom*). Data are for decoding the bias current and $\varphi(x) = x$ and are the median over 1000 trials. Network, parameters, and error measure are the same as in Figure 3.13. Weights with a magnitude below 10^{-6} are counted as zero. **(A)** Solving for weights \mathbf{W} according to eq. (3.9) with enforced sparsity. Weights \mathbf{W} with a magnitude below the P th percentile are set to zero; the weights are re-solved. Sparsity up to 25% has no impact on the error; errors increase drastically for sparsities over 75%. **(B)** Encouraging sparsity using an L_1 term (in addition to L_2 regularisation) results in lower errors compared to the enforced sparsity. **(C)** Solving for nonnegative \mathbf{W}^+ , \mathbf{W}^- using eq. (3.12). For a large range of excitatory to inhibitory ratios this results in a sparsity of about 50%, with errors similar to L_1 regularisation for the same sparsity.

Sparsity of nonnegative weight matrices. The weight matrices returned by the nonnegative least-squares solver tend to be sparse. This is, for example, quite apparent in our motivational illustration from the last chapter (Figure 2.32). Interestingly, the sparsity of NNLS solutions is not just an artefact of our particular problem domain. Slawski and Hein (2013) show that, under certain circumstances, the nonnegativity constraint induces an implicit L_1 regularisation term $\lambda_1 \|\mathbf{w}_i\|_1$. This kind of regularisation, also referred to as “lasso”, is a standard method for obtaining sparse solutions (Bishop, 2006, Section 3.1.4). If combined with our original L_2 regularisation term $\sigma^2 \|\mathbf{w}_i\|_2^2$, the resulting optimisation problem is also called “elastic net” (Zou and Hastie, 2005). The L_2 regularisation factor σ^2 accounts for Gaussian noise and ensures that the problem is non-singular, while the L_1 regularisation factor λ_1 encourages sparsity.

Indeed, as we explore in Figure 3.15, the NNLS solution has sparsity of about 50%. Notably, this is the case over a wide range of excitatory to inhibitory ratios; sparsity is *not* just a result of the solver requiring certain pre-neurons to be excitatory or inhibitory, and there being a 50% chance that this pre-condition is met. The performance of the NNLS solver is comparable to an “elastic net” version of our current-based weight solving problem from eq. (3.9). Errors are substantially smaller than what we obtain by naïvely enforcing sparsity.

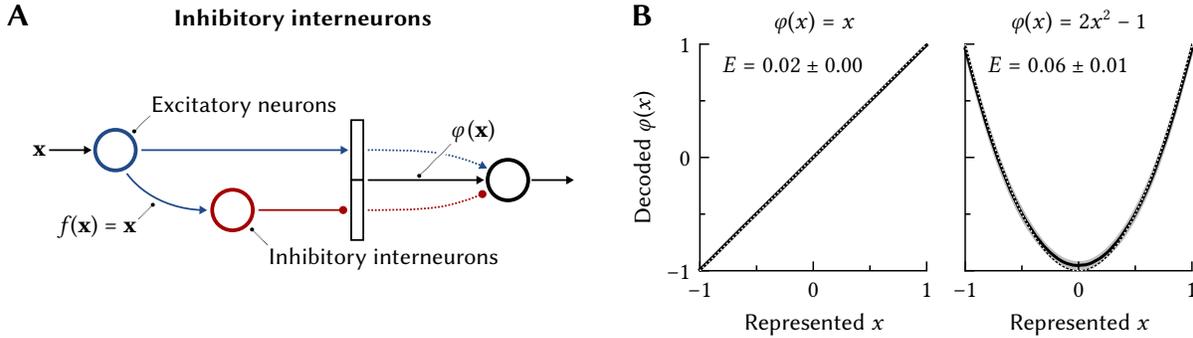


Figure 3.16: Inhibitory interneurons. **(A)** To establish interneuron populations, we compute the identity function $f(x) = x$ in the purely excitatory projection onto the interneurons. The desired $\varphi(x)$ is then decoded from a virtual population encompassing both the inhibitory interneurons, and the excitatory pre-neurons. **(B)** Using this scheme, we can compute linear and nonlinear functions $\varphi(x)$. Data for 100 neurons per population with maximum firing rates between 50 and 100 Hz; weight matrices are determined by solving eq. (3.11) with $\sigma = 10$. The dotted line is the target $\varphi(x)$, black line is the median decoded value over 1000 trials, shaded grey areas correspond to the 10th and 90th percentiles. Errors E are the mean NRMSE with standard deviation.

Inhibitory interneurons. Connectivity patterns in biology do not suggest an arbitrary split of neural ensembles into excitatory and inhibitory neurons. Instead, as we explained in Section 2.3.6, excitatory signals are often mediated through interneurons that provide local inhibition (e.g., Kandel et al., 2012, Chapter 2). Parisien et al. (2008) suggest a way to construct NEF networks with such inhibitory interneurons.

The techniques we discussed above can similarly be used to construct networks with inhibitory interneurons, albeit in a much simpler manner. Recall that we can compute the identity function over purely excitatory connections (cf. Figure 3.13). We can hence represent \mathbf{x} in the interneurons using excitatory connection weights (cf. Figure 3.16A). With this connection in place, the pre- and interneurons can be thought of as forming a “virtual pre-population” representing the same quantity \mathbf{x} . Using eq. (3.11) we can solve for excitatory weights \mathbf{w}_i^+ originating from the pre-population, and inhibitory weights \mathbf{w}_i^- originating from the interneurons, that project onto the post-population while approximating a function $\varphi(\mathbf{x})$.

Although we decode from multiple pre-populations, we do not need to resort to an optimisation problem such as eq. (3.10), where we decoded additive functions from multiple pre-populations. This is possible because both pre-populations represent the same value. Hence, we do not require a double integral, and we can compute nonlinear functions over \mathbf{x} .

As depicted in Figure 3.16B, we can use this technique to approximate linear and nonlinear functions $\varphi(\mathbf{x})$; errors mostly stem from the pre- to interneuron connection. Crucially, in contrast to the “Parisien transform”, we did not take any special precautions regarding the interneuron tuning curve distributions. All populations use the “standard” tuning with uniform x -intercepts. This is possible because we solve for weights directly in current space.

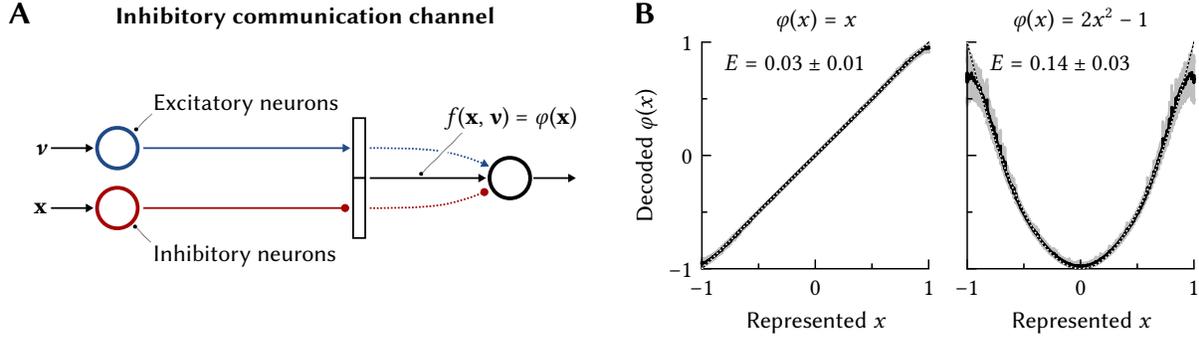


Figure 3.17: Inhibitory communication channels. **(A)** Inhibitory neurons can be used to form a communication channel, as long as there is some excitatory population that can provide a bias. **(B)** Experiment demonstrating the use of this network setup as a communication channel. While the inhibitory connection is a good communication channel, nonlinear functions can only be computed with larger errors. All points are sampled from $(x, v) \in [-1, 1]^2$; the v -dimension is not depicted. See Figure 3.16B for the network parameters and description of the depicted quantities.

Inhibitory communication channels. Curiously, using the same techniques, we can also construct purely inhibitory communication channels—at least under the assumption that there is some separate excitatory pre-population that can be used as a bias source. The lack of such an excitatory population is exactly what prevented us from computing functions across inhibitory connections in Figure 3.13.

Given an inhibitory population representing \mathbf{x} , and another population representing some unrelated \mathbf{v} (cf. Figure 3.17A). Similar to eq. (3.10), we minimise (without regularisation)

$$\min_{\mathbf{w}_i^+, \mathbf{w}_i^-} \frac{1}{\text{vol}(\mathbb{X})^2} \iint_{\mathbb{X}} (J_i(\langle \mathbf{e}_i, \varphi(\mathbf{x}) \rangle) - \langle \mathbf{w}_i^-, -\mathbf{a}^-(\mathbf{x}) \rangle - \langle \mathbf{w}_i^+, \mathbf{a}^+(\mathbf{v}) \rangle)^2 d\mathbf{x} d\mathbf{v} \quad (3.14)$$

subject to $\mathbf{w}_i^+, \mathbf{w}_i^- > 0$. Crucially, we ignore \mathbf{v} in our target function; we solely use the pre-population to provide background activity but ignore its represented value.

Results of an experiment exploring this technique are depicted in Figure 3.17B. While computing the identity function—i.e., constructing a pure communication channel—works well, nonlinear functions can only be decoded with a considerable error, just as with purely excitatory channels (cf. Figure 3.13). This is due to the standard NEF pre-population tuning curves not providing a good basis for nonnegative decoding of many nonmonotonic functions. In principle, it should be possible to have pre-population tuning such that any nonnegative target current function can be nonnegatively decoded with an arbitrarily small error. An example would be the Gaussian tuning curves similar to those depicted in Figure 3.11C.

Inhibitory networks similar to what we discussed here are explored in some more detail by Tripp and Eliasmith (2016). However, as with the Parisien transform, this prior work relies on specific tuning of the inhibitory neurons, as well as intrinsic biases.

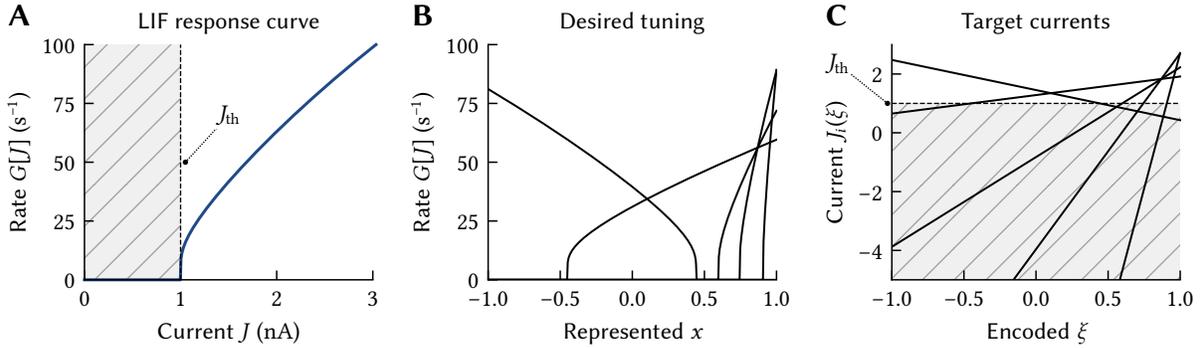


Figure 3.18: Illustration of the goal of subthreshold relaxation. **(A)** Most neurons act as rectifiers. Input currents below J_{th} (grey) are mapped onto zero. **(B)** Five randomly generated tuning curves with uniform x -intercepts and maximum firing rates between 50 and 100 spikes per second. **(C)** Using affine current-translation functions $J_i(\xi)$, these tuning curves are generated by comparably large negative currents. However, the magnitude of the currents below J_{th} (grey) has no effect on the output rate; in fact any current below J_{th} has the same effect on the firing rate of the neuron.

3.2.3 Subthreshold Relaxation

Most biological model neurons act as rectifiers. That is, input currents below a certain, usually positive, threshold J_{th} do not result in any output activity. In the case of our simplified LIF neuron model (cf. Section 2.2.2), the threshold current J_{th} is 1 nA (Figure 3.18A). We did not take this into account in our above current-space optimisation schemes.

To the contrary, our current-based loss functions in eqs. (3.9) and (3.11) aim at *precisely* evoking certain post-synaptic currents J_{tar} . Notably, the negative currents required by the affine current translation function $J_i(\xi)$ can be larger in magnitude than the positive currents (Figures 3.18B and 3.18C). This has not been an issue in NEF networks with intrinsic current-translation, as $J_i(\xi)$ takes care of appropriately scaling and offsetting the input currents.

However, in the context of our current-space optimisation schemes, a regularised least-squares optimisation problem can “prioritise” solving for exact (but irrelevant) subthreshold currents over solving for the (relevant) superthreshold currents. This can lead to an increase in the superthreshold current-decoding error, and thus also increase the representation error in the post-population. Additionally, as we will see below, dendritic nonlinearities impose asymptotic maximum and minimum post-synaptic currents. Trying to solve for large negative currents may thus result in large connection weights.

One way to work around this issue is to clamp target currents J_{tar} that are substantially below the threshold to some constant. For example, in the case of our simplified LIF neuron, we could clamp all negative target currents to a current of zero. This way, the solver does not have to generate the aforementioned large negative subthreshold currents. Still, this “clamping” approach requires the weight solver to *precisely* solve for the desired target current although *any* subthreshold current would do.

Conceptually, it would be better to “relax” the requirement to solve for J_{tar} as precisely as possible for subthreshold currents. Instead, we could merely demand that subthreshold target currents are decoded as subthreshold currents, regardless of the magnitude. If this constraint is violated, i.e., if the target current J_{tar} is below the threshold and the decoded current J_{dec} is above the threshold J_{th} , we measure the distance to the threshold, and not the distance to J_{tar} as an error. We formalise this as a superthreshold error function \mathcal{E} ,

$$\mathcal{E}(J_{\text{tar}}, J_{\text{dec}}) = \begin{cases} 0 & \text{if } J_{\text{tar}} < J_{\text{th}} \text{ and } J_{\text{dec}} < J_{\text{th}}, \\ J_{\text{dec}} - J_{\text{th}} & \text{if } J_{\text{tar}} < J_{\text{th}} \text{ and } J_{\text{dec}} > J_{\text{th}}, \\ J_{\text{dec}} - J_{\text{tar}} & \text{if } J_{\text{tar}} \geq J_{\text{th}}, \end{cases} \quad (3.15)$$

and define a new current-space optimisation problem akin to eq. (3.11)

$$\min_{\mathbf{w}_i^+, \mathbf{w}_i^-} \frac{1}{\text{vol}(\mathbb{X})} \int_{\mathbb{X}} \mathcal{E}(J_i(\langle \mathbf{e}_i, \varphi(\mathbf{x}) \rangle), \langle \mathbf{w}_i^+, \mathbf{a}^+(\mathbf{x}) \rangle + \langle \mathbf{w}_i^-, -\mathbf{a}^-(\mathbf{x}) \rangle)^2 d\mathbf{x} + \sigma^2 \|\mathbf{w}_i^+\|_2^2 + \sigma^2 \|\mathbf{w}_i^-\|_2^2. \quad (3.16)$$

Subthreshold relaxation as a quadratic program. It is not immediately clear how to solve this optimisation problem. While, it is always possible to resort to gradient descent, eq. (3.16) can be solved more efficiently by rewriting the loss function in terms of a convex quadratic program (QP). QPs are a generalisation of least-squares and defined as follows:

Definition 3.2 (Quadratic Program). *A quadratic program (QP) is an optimisation problem of the form (adapted in slightly simplified form from Boyd et al., 2004, Section 4.4)*

$$\begin{aligned} & \text{minimize} && \boldsymbol{\omega}^T \mathbf{P} \boldsymbol{\omega} + \mathbf{q}^T \boldsymbol{\omega} \\ & \text{subject to} && \mathbf{G} \boldsymbol{\omega} \leq \mathbf{h}, \end{aligned}$$

where $\boldsymbol{\omega} \in \mathbb{R}^n$, $\mathbf{P} \in \mathbb{R}^{n \times n}$, $\mathbf{q} \in \mathbb{R}^n$, $\mathbf{G} \in \mathbb{R}^{\ell \times n}$, $\mathbf{h} \in \mathbb{R}^\ell$. Here, n is the number of variables and ℓ is the number of inequality constraints. If $\boldsymbol{\omega}^T \mathbf{P} \boldsymbol{\omega}$ is a convex function (i.e., \mathbf{P} is positive definite) and the constraints $\mathbf{G} \boldsymbol{\omega} \leq \mathbf{h}$ form a convex polytope, then the QP is convex.

Convex quadratic programs can be solved in polynomial time (Kozlov, Tarasov, and Khachiyan, 1980). There are free and open-source software libraries, such as “cvxopt” (Vandenberghe, 2010) and “OSQP” (Stellato et al., 2020) that solve such problems efficiently. In our experiments we mostly rely on OSQP.¹⁰

To transform a discretised version of eq. (3.16) into a quadratic program we split the sample points \mathbf{x}_k according to whether they evoke super- or subthreshold currents. Specifically, we arrange the pre-activities \mathbf{A} and target currents \mathbf{J} as follows:

$$\mathbf{A}_{\text{sup}} = (\mathbf{A}_{\text{sup}}^+, -\mathbf{A}_{\text{sup}}^-) \in \mathbb{R}^{N_{\text{sup}} \times (n^+ + n^-)}, \quad \mathbf{J}_{\text{sup}} \in \mathbb{R}^{N_{\text{sup}}}, \quad \mathbf{A}_{\text{sub}} = (\mathbf{A}_{\text{sub}}^+, -\mathbf{A}_{\text{sub}}^-) \in \mathbb{R}^{N_{\text{sub}} \times (n^+ + n^-)}.$$

¹⁰Some experiments were conducted before OSQP was published; we used cvxopt in those experiments. We did not observe any discernible difference in the solutions produced by the two libraries, but as a C library using more modern algorithms, OSQP is substantially faster than the older Python library cvxopt.

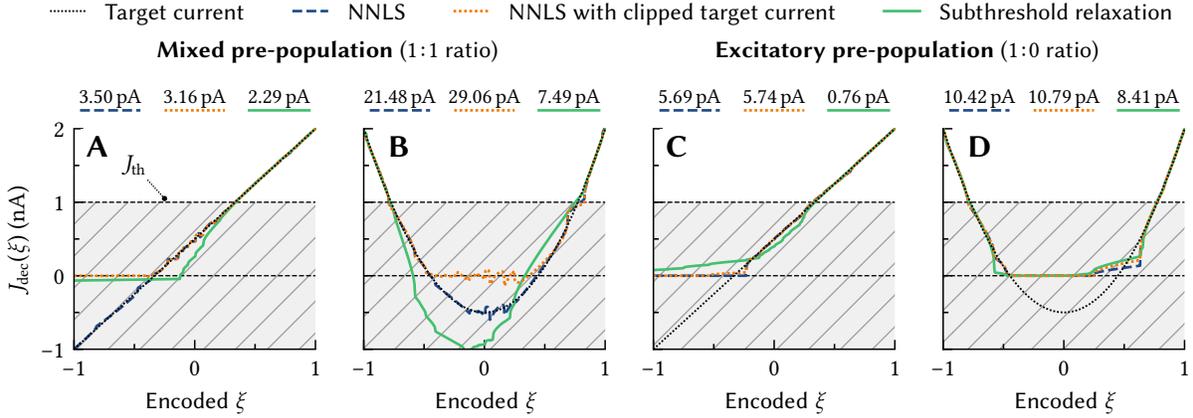


Figure 3.19: Current decoding with and without subthreshold relaxation in a setting with low regularisation ($\sigma = 0.31$) and few pre-neurons ($n = 50$). Values above each plot are the RMS of the superthreshold error \mathcal{E} (eq. 3.15). Subthreshold relaxation substantially reduces the decoding error.

Here, $N_{\text{sup}}, N_{\text{sub}}$ with $N = N_{\text{sup}} + N_{\text{sub}}$ are the number of samples with a super- and subthreshold target currents, and, as before, n^+ and n^- correspond to the number of excitatory and inhibitory pre-neurons. Using these matrices, eq. (3.16) can be expressed as a QP by letting

$$\mathbf{P} = \begin{pmatrix} (\mathbf{A}_{\text{sup}})^T \mathbf{A}_{\text{sup}} + N\sigma^2 \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}, \quad \mathbf{q} = \begin{pmatrix} (\mathbf{A}_{\text{sup}})^T \mathbf{J}_{\text{sup}} \\ \mathbf{0} \end{pmatrix}, \quad \mathbf{G} = \begin{pmatrix} \mathbf{A}_{\text{sub}} & \mathbf{I} \\ -\mathbf{I} & \mathbf{0} \end{pmatrix}, \quad \mathbf{h} = \begin{pmatrix} \mathbf{J}_{\text{th}} \\ \mathbf{0} \end{pmatrix}. \quad (3.17)$$

The number of variables is $n = n^+ + n^- + N_{\text{sub}}$, and the number of inequality constraints is $\ell = N_{\text{sub}} + n^+ + n^-$. The parameter vector $\boldsymbol{\omega} \in \mathbb{R}^n$ can be split into the excitatory and inhibitory weights $\mathbf{w}_i^+ \in \mathbb{R}^{n^+}$, $\mathbf{w}_i^- \in \mathbb{R}^{n^-}$, as well as discardable slack variables $\mathbf{s}_i \in \mathbb{R}^{N_{\text{sub}}}$.

The rationale behind eq. (3.17) is as follows. The first line of \mathbf{P} and \mathbf{q} is the standard regularised least-squares problem obtained by expanding the superthreshold portion of eq. (3.16) (cf. Boyd et al., 2004, Section 4.4). The first column and row of \mathbf{G} and \mathbf{h} add an inequality constraint that ensures that samples with subthreshold currents are decoded as subthreshold currents. Violations of this constraint, i.e., case two of eq. (3.15), are enabled by the slack variables \mathbf{s}_i (second column of \mathbf{P} and \mathbf{G}). These slack variables correspond to the error $J_{\text{th}} - J_{\text{tar}}$, which is penalised accordingly in \mathbf{P} . Finally, nonnegativity of the weights is ensured by the second row of \mathbf{G} and \mathbf{h} . One can easily show that this QP is convex.

Example: Individual post-neuron. In Figure 3.19 we decode the post-synaptic currents for a single post neuron using NNLS, NNLS with clamped target currents, and subthreshold relaxation. At least in this example, subthreshold relaxation substantially reduces the superthreshold decoding error. In contrast, clipping the target currents has only a limited positive effect. Notably, and particularly pronounced in the case of purely excitatory pre-neurons, the solution obtained with subthreshold relaxation is supported by more pre-neurons. This is visible in Figure 3.19C, where subthreshold relaxation decodes a non-zero current for negative ξ , resulting in a smaller regularisation error (i.e., weight RMS of 0.8×10^{-3} vs. 2×10^{-3}).

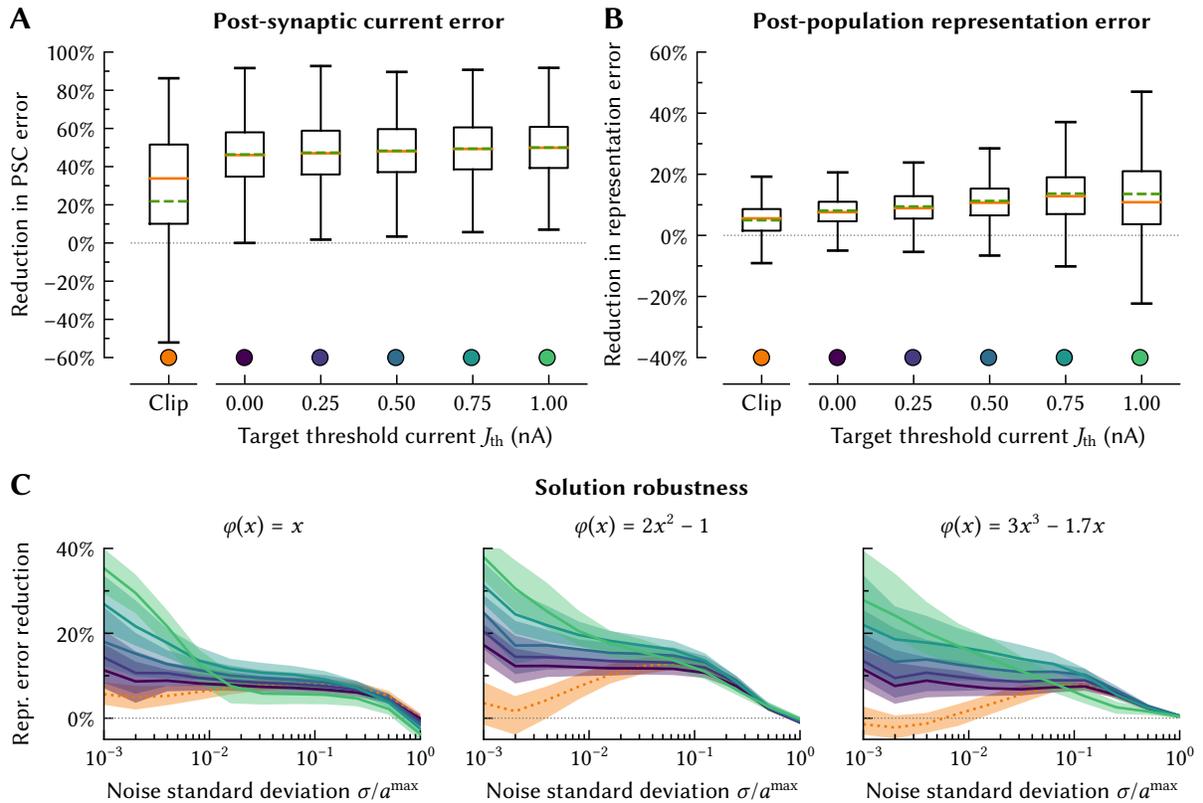


Figure 3.20: Reduction in decoding error achieved with subthreshold relaxation compared to standard NNLS and clipping the target current. **(A)** Reduction in the decoded post-synaptic currents error relative to NNLS for different thresholds in the error measure \mathcal{E} . Each box plot over three functions, 100 random networks, 11 noise magnitudes, and 9 random samplings of the noise ($N = 29\,700$); whiskers are the extrema, boxes the quartiles, orange line is the median, green dashed line the mean. **(B)** Same as (A), but for the post-population decoding error. **(C)** Same data as above, but over different pre-population noise magnitudes and for different functions $\varphi(x)$ computed in the pre to post connection. Coloured lines show the median error (see (A, B) for a legend); shaded areas are the 25% and 75% quartiles.

Systematic experiment. One issue with subthreshold relaxation is that not solving for strongly negative currents can result in a narrower separation boundary between the threshold and the decoded current. Hence, noise on the pre-activities is more likely to result in positive post-activity. However, this may be compensated for by the smaller regularisation error, and hence higher robustness to noise as observed in the previous example.

Figure 3.20 depicts the results of a more systematic experiment. We compute three polynomials in the connection between one hundred pre and post LIF rate neurons (1:1 excitatory to inhibitory ratio) with varying degree of Gaussian noise added to the pre-activities. We select the regularisation factor σ to minimise the error for the given amount of noise on a separate training set. We furthermore vary the threshold J_{th} assumed in eq. (3.15), to investigate the effect of moving J_{th} away from the true threshold.

As visible in Figure 3.20A, subthreshold relaxation reduces the current decoding error by about 50% (median), largely independent of the assumed J_{th} . Clamping the currents leads to a reduction in error with a median of about 35%, but, overall, the reduction in error is less consistent. Improvements to the representation accuracy (Figure 3.20B) are less drastic, with the largest improvement for $J_{\text{th}} = 0.75$ nA with a median reduction in error of about 13%. This confirms our suspicion that setting J_{th} to the true threshold can be slightly detrimental.

This is further confirmed by the experiment in Figure 3.20C where we plot the post-population representation error over different pre-activity noise magnitudes. For small amounts of noise subthreshold relaxation with $J_{\text{th}} = 1$ nA can lead to large improvements in the representation error; however, for larger noise magnitudes the overall benefit of subthreshold relaxation is smaller, with slightly reduced J_{th} performing the best.

Keep in mind that LIF *rate* neurons are a worst-case scenario with respect to sensitivity to pre-activity noise injected near the threshold. LIF rate neurons possess a very steep activity onset, where small changes in current lead to large fluctuations in activity. This tends to be less of an issue with transient noise in spiking neural networks, which results in a smoother response curve (see our discussion below, as well as Hunsberger and Eliasmith, 2015). We can thus expect that, in practice, the impact of subthreshold relaxation is somewhere between the values obtained for the current decoding and representation error.

3.2.4 Extension Toward Dendritic Nonlinearities

Up to this point we assumed current-based synapses. As previously discussed in Section 3.1, the defining property of current-based synapses is that the somatic current J is linear in the synaptic weights \mathbf{w} and the pre-synaptic activities \mathbf{a} , that is $a_i = G[J] = G[\langle \mathbf{w}, \mathbf{a} \rangle]$. In contrast, we described neurons with nonlinear synapses as follows

$$a_i = \mathcal{G}[g_i^1, \dots, g_i^k] = \mathcal{G}[\langle \mathbf{w}_i^1, \mathbf{a}_1 \rangle, \dots, \langle \mathbf{w}_i^k, \mathbf{a}_k \rangle]. \quad (3.18)$$

Here, k is the number of input channels, and the vector $\mathbf{g}_i = (g_i^1, \dots, g_i^k)$ describes some abstract “channel state”. Again, we assume that, on average, each channel state g_i^j is linear in the weights and the activities \mathbf{a}_j of the neurons connecting to the j th channel. However, we do not make any assumption regarding the effect of g_i^j on the somatic current J ; more fundamentally, we do not assume that there exists an easily identifiable somatic current at all.

The lack of an identifiable somatic current makes it more challenging to integrate such neurons into the NEF. The optimisation problems we discussed in this section relied on the current translation function $J_i(\xi)$ to enforce the normative tuning constraint $a_i(\mathbf{x})$. Of course, as mentioned above, we could resort to gradient descent to optimise eq. (3.8). However, our current-based optimisation schemes work well in that they allow us to quickly solve for globally optimal weights. As we will see, it is still possible to perform global current-space optimisation for some multi-channel neurons. Additionally, we can use the same ideas to iteratively solve for locally optimal weights in more complex multi-compartment neurons.

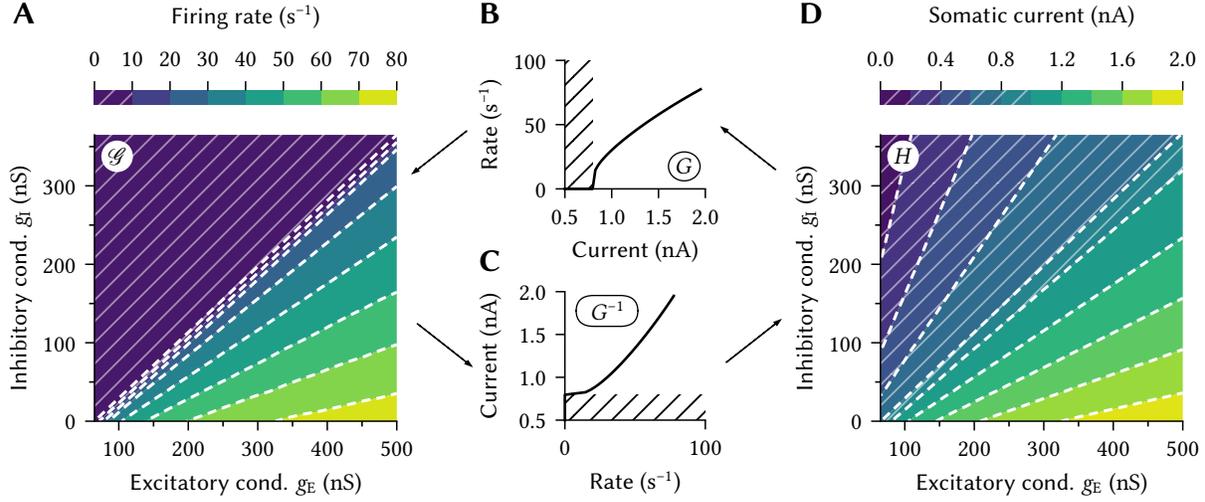


Figure 3.21: Neural response curve decomposition. **(A)** Illustration of the multivariate neuron response curve $\mathcal{G}(g_E, g_I)$ for a two-compartment LIF neuron with excitatory and inhibitory conductance-based channels. **(B, C)** The chosen somatic nonlinearity G and its inverse G^{-1} . **(D)** corresponding input-dependent nonlinearity H . The neuron does not fire in the hatched regions, that is, G^{-1} is ill-defined.

To this end, the crucial idea is to mathematically reintroduce a “virtual” somatic current J by decomposing \mathcal{G} into the standard somatic nonlinearity G and a dendritic nonlinearity H .

Definition 3.3 (Dendritic Nonlinearity). *Given a neural response curve G , the dendritic nonlinearity $H(g_i)$ of a multi-channel neuron with response curve \mathcal{G} maps the input channel state $g_i = (g_i^1, \dots, g_i^k)$ onto an average, time-independent somatic current J such that*

$$H(g_i^1, \dots, g_i^k) = J \Leftrightarrow G[J] = \mathcal{G}[g_i^1, \dots, g_i^k] \Leftrightarrow \mathcal{G}[g_i^1, \dots, g_i^k] = G[H(g_i^1, \dots, g_i^k)]. \quad (3.19)$$

Optimally, G is chosen such that H is as simple as possible. For example, if the neuron model is an extension to a LIF neuron, G can be the standard LIF response curve. In this case, H translates the input into an “LIF-equivalent current”. In the trivial case of the above current-based LIF neurons with excitatory and inhibitory inputs, we would obtain $H(J_E, J_I) = J_E - J_I$.

We use the dendritic nonlinearity H to define a new current-space loss function

$$E = \frac{1}{\text{vol}(\mathbb{X})} \int_{\mathbb{X}} \mathcal{E}(J_i(\langle \mathbf{e}_i, f(\mathbf{x}_k) \rangle), H(\langle \mathbf{w}_i^1, \mathbf{a}^k \rangle, \dots, \langle \mathbf{w}_i^k, \mathbf{a}^k \rangle))^2 dx + \sigma^2 \sum_{j=1}^{\ell} \|\mathbf{w}_i^j\|_2^2, \quad (3.20)$$

where \mathcal{E} is the superthreshold error defined in eq. (3.15), \mathbf{a}^j are the pre-activities, and connection weights \mathbf{w}_i may be—depending on the input channel type—nonnegative.

We next show that it is possible to derive H , or at least a suitable surrogate model, in closed form for multi-compartment LIF neurons with conductance-based synapses. If H cannot be derived in closed form, we can sample \mathcal{G} over varying synaptic states and, as depicted in Figure 3.21, compute H indirectly by applying an inverse mapping G^{-1} to the recorded data.

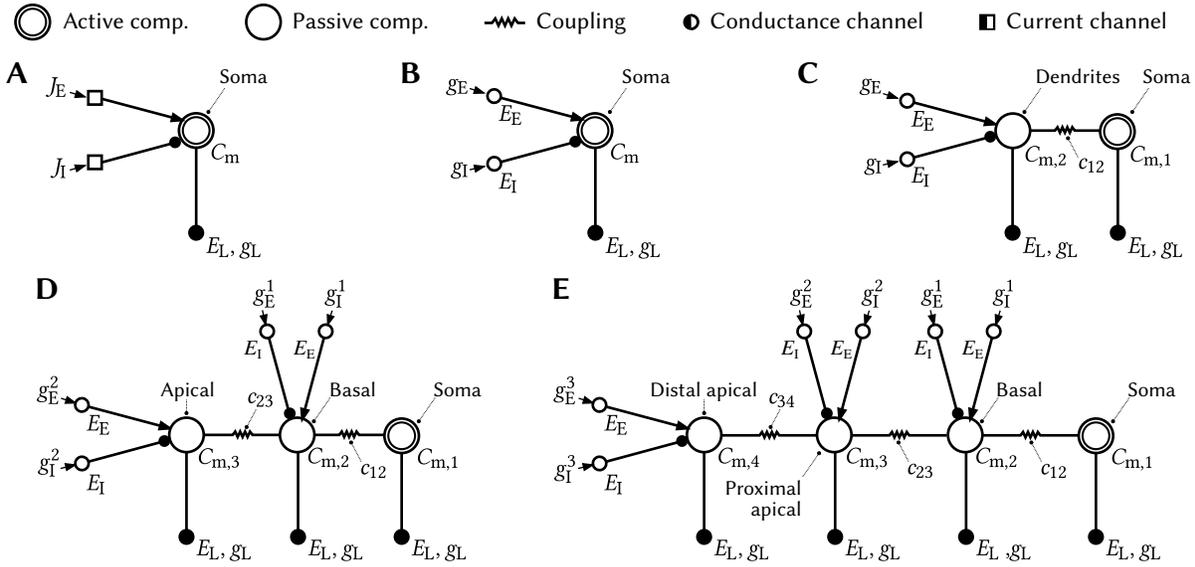


Figure 3.22: “Ball-and-stick” illustration of multi-compartment LIF neurons. Large circles correspond to compartments, small circles and rectangles to conductance- and current-based channels. Filled symbols indicate passive channels. **(A)** Standard LIF neuron with current-based inputs. **(B)** LIF neuron with conductance-based input channels. **(C)** Two-compartment neuron with a separate dendritic compartment. **(D, E)** Three- and four-compartment LIF neurons with apical and basal dendrites.

3.3 A Family of Multi-Compartment LIF Neurons

The dendritic nonlinearity H introduced in the last section maps some channel state \mathbf{g} onto an average somatic current. While this mapping can always be established numerically, expressing H in closed form has the potential to simplify the optimisation problem in eq. (3.20).

In this section, we discuss a family of multi-compartment LIF neurons that we refer to as “ n -LIF”. These neurons are based on the multi-compartment neurons that we reviewed in Section 2.1.4, and, as we discussed in the introduction of this chapter, were constructed with mathematical tractability in mind, rather than biological detail. While it is not possible to derive an exact dendritic nonlinearity H for these neurons, we derive a closed-form “surrogate” model of H that can be fit to numerical data.

3.3.1 Mathematical Description of n -LIF Neurons

We define an n -LIF neuron as a connected graph of resistively coupled capacitive compartments. Each neuron possesses exactly one active compartment with standard LIF dynamics, while the other compartments represent a passive dendritic tree. Each compartment may hold any number of passive current- and conductance-based channels. Channels are either constant (representing bias currents and leak channels), or receive external input (representing synapses). Examples of such neurons are depicted in Figure 3.22 using a “ball-and-stick” representation.

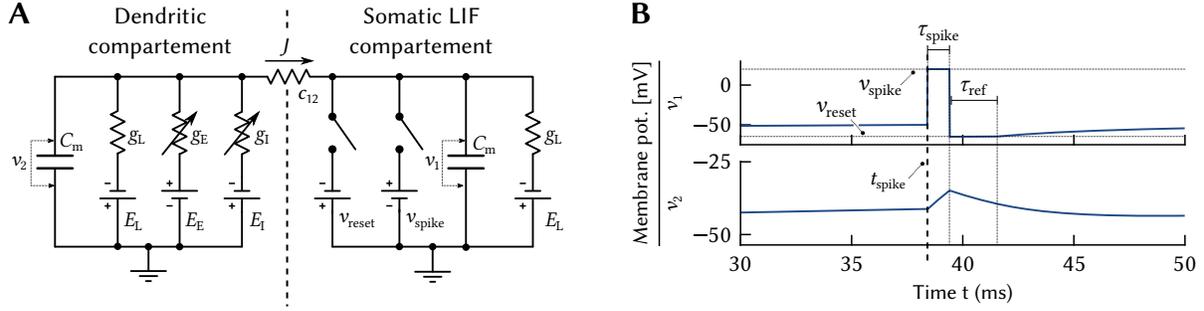


Figure 3.23: Circuit and membrane potential trace of a two-compartment neuron. **(A)** Circuit diagram corresponding to the model in Figure 3.22C. **(B)** Membrane potential traces for both compartments for a small constant excitatory input. Notice how the explicit spike model in the somatic compartment (*top*) influences the membrane potential of the dendritic compartment (*bottom*).

As a point of reference, the expanded equivalent circuit diagram of the two-compartment neuron with conductance-based synapses (Figure 3.22C) is depicted in Figure 3.23A. This particular model has originally been described by Vu, Lee, and Krasne (1993) and was subsequently discussed by Koch (1999) and Capaday and Van Vreeswijk (2006). We discuss this model in more detail in the next section, and for now focus on n -LIF neurons in general.

Suprathreshold dynamics. In contrast to the standard LIF model (cf. Section 2.2.2), the active n -LIF compartment possess an explicit spike model. As pointed out by Capaday et al. (2006), this is important in multi-compartment models, since the spike potential causes a substantial current to backpropagate into the dendritic compartments (cf. Figure 3.23B).

More precisely, we model the suprathreshold dynamics as follows. Whenever the membrane potential v surpasses the threshold v_{th} , v is clamped to a “spike potential” v_{spike} for a period τ_{spike} and subsequently forced to v_{reset} for the refractory period τ_{ref} . Unless specified otherwise, we use $\tau_{spike} = 1$ ms and $\tau_{ref} = 2$ ms in our models.

Subthreshold dynamics. The dynamics of the i th compartment are

$$C_{m,i} \frac{d}{dt} v_i(t) = \sum_{k=1}^{M_i} g_k^i(t) (E_k^i - v_i(t)) + \sum_{k=1}^{N_i} J_k^i(t) + \sum_{j=1}^n (v_j(t) - v_i(t)) c_{ij}, \quad (3.21)$$

where $C_{m,i}$ is the membrane capacitance of the compartment, M_i and N_i are the number of conductance- and current-based channels, respectively, $g_k^i(t)$ is the momentary conductance of the k th conductance-channel with reversal potential E_k^i , and $J_k^i(t)$ is the current injected into the current-based channel j . Finally, c_{ij} is the coupling conductance between the i th and the j th compartment. The adjacency matrix of coupling conductances C must be symmetric, and the connectivity graph encoded by C must have exactly one connected component (cf. our discussion in Section 2.1.4). Some conductances $g_k^i(t)$ and currents $J_k^i(t)$ are constant, such as the static leak channels and bias currents.

Table 3.1: Matrix representations of the first four neuron models in Figure 3.22. See text for details.

Model	\mathbf{a}'	\mathbf{A}'	\mathbf{b}'	\mathbf{B}'	\mathbf{L}
(A)	$[g_L]$	$[0 \ 0]$	$[g_L E_L]$	$[1 \ -1]$	$[0]$
(B)	$[g_L]$	$[1 \ 1]$	$[g_L E_L]$	$[E_E \ E_I]$	$[0]$
(C)	$\begin{bmatrix} g_L \\ g_L \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} g_L E_L \\ g_L E_L \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ E_E & E_I \end{bmatrix}$	$\begin{bmatrix} c_{12} & -c_{12} \\ -c_{12} & c_{12} \end{bmatrix}$
(D)	$\begin{bmatrix} g_L \\ g_L \\ g_L \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} g_L E_L \\ g_L E_L \\ g_L E_L \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ E_E & E_I & 0 & 0 \\ 0 & 0 & E_E & E_I \end{bmatrix}$	$\begin{bmatrix} c_{12} & -c_{12} & 0 \\ -c_{12} & c_{12} + c_{23} & -c_{23} \\ 0 & -c_{23} & c_{23} \end{bmatrix}$

Canonical matrix form. We can rearrange eq. (3.21) to *resemble* a canonical LTI system.¹¹ Let k be the number of non-static input channels, and $\mathbf{g} \in \mathbb{R}^k$ a vector representing all input channel states, that is, all non-static conductances g_{ij} and currents J_{ij} . This channel state is generally linear in the synaptic weights and the pre-activities (cf. Section 3.2.4). We have

$$\frac{d}{dt} \mathbf{C}_m \circ \mathbf{v}(t) = \mathbf{A}[\mathbf{g}(t)] \mathbf{v}(t) + \mathbf{b}[\mathbf{g}(t)] = -[\mathbf{L} + \text{diag}(\mathbf{a}' + \mathbf{A}'\mathbf{g}(t))] \mathbf{v}(t) + [\mathbf{b}' + \mathbf{B}'\mathbf{g}(t)]. \quad (3.22)$$

Here, $\mathbf{C}_m \in \mathbb{R}^n$ is a vector of membrane capacitances, “ \circ ” denotes elementwise multiplication, $\mathbf{A}[\mathbf{g}(t)] \in \mathbb{R}^{n \times n}$ is the “voltage feedback matrix”, and $\mathbf{b}[\mathbf{g}(t)] \in \mathbb{R}^n$ describes the input to the system. We further decompose \mathbf{A} , \mathbf{b} into input-independent and input-dependent terms. The Laplacian $\mathbf{L} \in \mathbb{R}^{n \times n}$ and the vectors $\mathbf{a}' \in \mathbb{R}^n$, $\mathbf{b}' \in \mathbb{R}^n$ describe the input-independent portions of the system, whereas the matrices $\mathbf{A}' \in \mathbb{R}^{n \times k}$ and $\mathbf{B}' \in \mathbb{R}^{n \times k}$ describe its input-dependent parts.

More specifically, the Laplacian \mathbf{L} is the difference between the weighted degree matrix and the adjacency matrix, and, in our case, is given as $\mathbf{L} = \text{diag}(\mathbf{C}\mathbf{I}) - \mathbf{C}$.¹² The vector \mathbf{a}' consists of sums of static channel conductances for each channel (such as the leak conductance) and \mathbf{b}' contains the sums of the static channel conductances multiplied by their reversal potential. The matrix \mathbf{A}' contains one-entries for input-variables influencing a conductance-based channel in the corresponding compartment. \mathbf{B}' contains a “one” for each variable current-based channel in the corresponding compartment, and the reversal potential for each variable conductance-based channel in a compartment. Examples of these matrices for the first four n -LIF neuron models depicted in Figure 3.22 are given in Table 3.1.

¹¹In general, eq. (3.22) is *not* a linear dynamical system. However, there are two conditions under which this system is linear: either $\mathbf{g}(t)$ is constant, or there are no product terms between $\mathbf{g}(t)$ and $\mathbf{v}(t)$. This is the case if the system has no conductance-based input channels and \mathbf{A}' is zero. Note that the constant offset \mathbf{b}' in eq. (3.22) does not make the dynamical system nonlinear, but merely mandates one auxiliary dimension.

¹²Indeed, the graph Laplacian is often motivated in textbooks with electrical networks similar to the one discussed here; in this context, it is also referred to as the “Kirchhoff matrix” (e.g., Bollobás, 1998, Chapter 2).

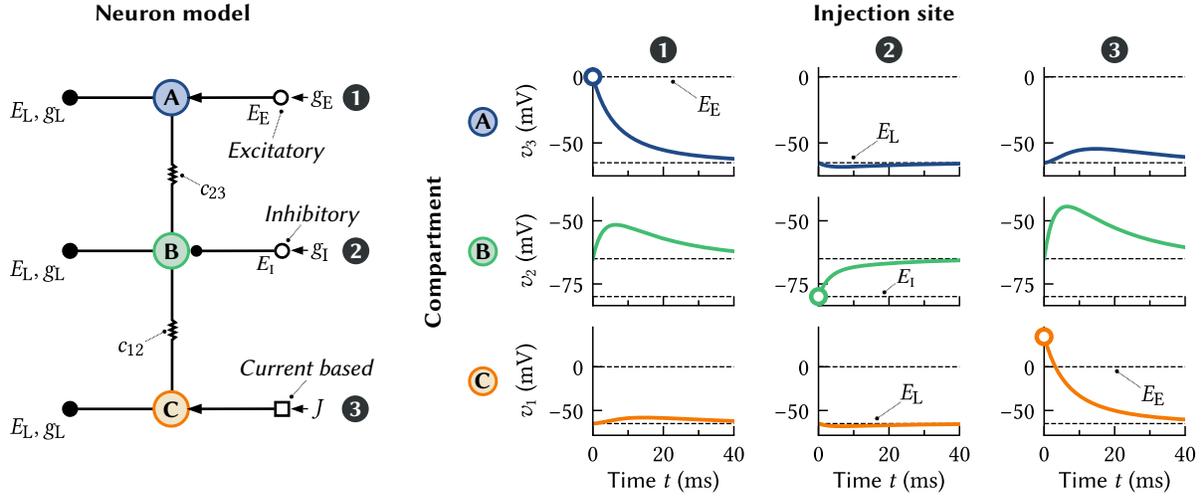


Figure 3.24: Impulse response of the n -LIF neuron. Injecting a pulse into a nonlinear conductance channel (injection sites 1, 2) at most charges the corresponding compartment up to the channel reversal potential. In contrast, current inputs (injection site 3) can charge the membrane to arbitrary potentials. The impulse is filtered and dampened while it is travelling through a neuron.

3.3.2 Analysis of the Subthreshold n -LIF Dynamics

The dynamics of the n -LIF system are quite benign. The system is generally stable and does not oscillate; in other words, the eigenvalues of $\mathbf{A}[\mathbf{g}] \text{diag}(\mathbf{C}_m)^{-1}$ are real-valued and strictly negative (e.g., Strogatz, 1994, Chapter 5). More formally (see Appendix A.1.5 for a proof):

Theorem 3.6. *Consider an n -LIF neuron with at least one static conductance-based channel and any input vector \mathbf{g} with nonnegative entries for conductance-based input channels. In this case, the feedback matrix $\mathbf{A}[\mathbf{g}] \text{diag}(\mathbf{C}_m)^{-1}$ is negative definite for any constant input vector \mathbf{g} .*

This is quite intuitive from a physical perspective. The n -LIF neuron is a resistor-capacitor network. When tying at least one point of the network to a voltage source, the potential over the capacitors converges to an equilibrium \mathbf{v}^{eq} . For constant \mathbf{g} the dynamics are

$$\mathbf{v}(t) = \mathbf{v}^{\text{eq}} + \exp(\mathbf{A}[\mathbf{g}] \text{diag}(\mathbf{C}_m)^{-1} t) (\mathbf{v}(0) - \mathbf{v}^{\text{eq}}), \quad \text{where } \mathbf{v}^{\text{eq}} = -\mathbf{A}[\mathbf{g}]^{-1} \mathbf{b}[\mathbf{g}], \quad (3.23)$$

Crucially, $\mathbf{A}[\mathbf{g}]$ is guaranteed to be invertible under the conditions listed in Theorem 3.6. Furthermore, note that \mathbf{v}^{eq} does not depend on \mathbf{C}_m ; both $\mathbf{A}[\mathbf{g}]$ and $\mathbf{b}[\mathbf{g}]$ are divided by \mathbf{C}_m (cf. eq. 3.22):

$$(\text{diag}(\mathbf{C}_m)^{-1} \mathbf{A}[\mathbf{g}])^{-1} (\text{diag}(\mathbf{C}_m)^{-1} \mathbf{b}[\mathbf{g}]) = \mathbf{A}[\mathbf{g}]^{-1} \text{diag}(\mathbf{C}_m) \text{diag}(\mathbf{C}_m)^{-1} \mathbf{b}[\mathbf{g}] = \mathbf{A}[\mathbf{g}]^{-1} \mathbf{b}[\mathbf{g}].$$

The impulse response of the system is depicted in Figure 3.24. Each compartment acts as a first-order low-pass filter; the response in compartment i to an impulse in compartment j thus resembles a low-pass filter of order $d(i, j)$, the distance between nodes i, j .

3.3.3 Deriving a Surrogate Model of the Dendritic Nonlinearity H

As defined above (cf. Definition 3.3), the dendritic nonlinearity H maps a synaptic state \mathbf{g} onto the synaptic current J such that $G[J] = \mathcal{G}[\mathbf{g}]$, where $G[J]$ is a standard single-channel response curve and $\mathcal{G}[\mathbf{g}]$ is the response curve of the multi-channel neuron. If, for example, $G[J]$ is the LIF response curve, then H maps \mathbf{g} into an “LIF-equivalent” current.

Trivial cases aside, it is generally not possible to provide H in closed form for n -LIF neurons. This is due to the nonlinear interaction between the membrane potential and conductance-based input channels, as well as the influence of the nonlinear superthreshold dynamics on the dendritic compartments. Still, we can derive a parametrised “surrogate model” that approximates H well and that we can fit to empirical data.

Subthreshold current. As a first step towards this model, consider the case where the neuron model is purely in its subthreshold regime. Without loss of generality, assume that the compartment with index $i = 1$ is the somatic compartment. Furthermore, assume that this compartment possesses a leak channel with conductance g_L and reversal potential E_L . The current $H(\mathbf{g}; t)$ flowing into the somatic compartment at time t for constant \mathbf{g} is the differential of v_1 divided by the membrane capacitance, that is

$$H(\mathbf{g}; t) = (\mathbf{A}[\mathbf{g}]\mathbf{v}(t) + \mathbf{b}[\mathbf{g}])_1. \quad (3.24)$$

To obtain a LIF-equivalent current, we must further subtract the leak current $g_L(E_L - v_1(t))$; this current is already accounted for in the LIF response curve $G[J]$.¹³

Average superthreshold somatic potential. For the purpose of building networks of spiking neurons, we are primarily interested in the superthreshold regime. Unfortunately, as mentioned above, the nonlinear superthreshold dynamics are notoriously difficult to analyse.

We work around this by exploiting the fact that n -LIF neurons are tonically spiking (cf. Section 2.2.2). That is, the somatic compartment oscillates between the reset and threshold potential with a fixed frequency (cf. Figure 2.17). We may thus assume that the somatic membrane potential is effectively clamped to some value \bar{v} between reset and threshold potential; we discuss this in more detail in Stöckel, Voelker, et al. (2017).

As is depicted in Figure 3.25, this assumption is reasonable for a wide range of output rates, as long as we ignore the spike and refractory period. This latter simplification is justified, since due to clamping, the current flowing into the somatic compartment during these periods has no direct influence on the output rate of the neuron. Of course, in multi-compartment models, there is the smaller, indirect effect of the somatic membrane potential influencing the state of the dendritic compartments, but we ignore this for now.

¹³Of course, H depends on the specific choice of $G[J]$. For example, if we used the response curve for a non-leaky IF-neuron instead, we would not have to subtract the leak current. More specifically, in the next section, we discuss using a rectified linear unit instead of the LIF response curve. Also note that, in practice, these details are not too critical. As we discuss below, we propose using a parametrised version of H that is fit to numerical measurements of J ; this process naturally compensates for missing offsets.

3.3. A Family of Multi-Compartment LIF Neurons

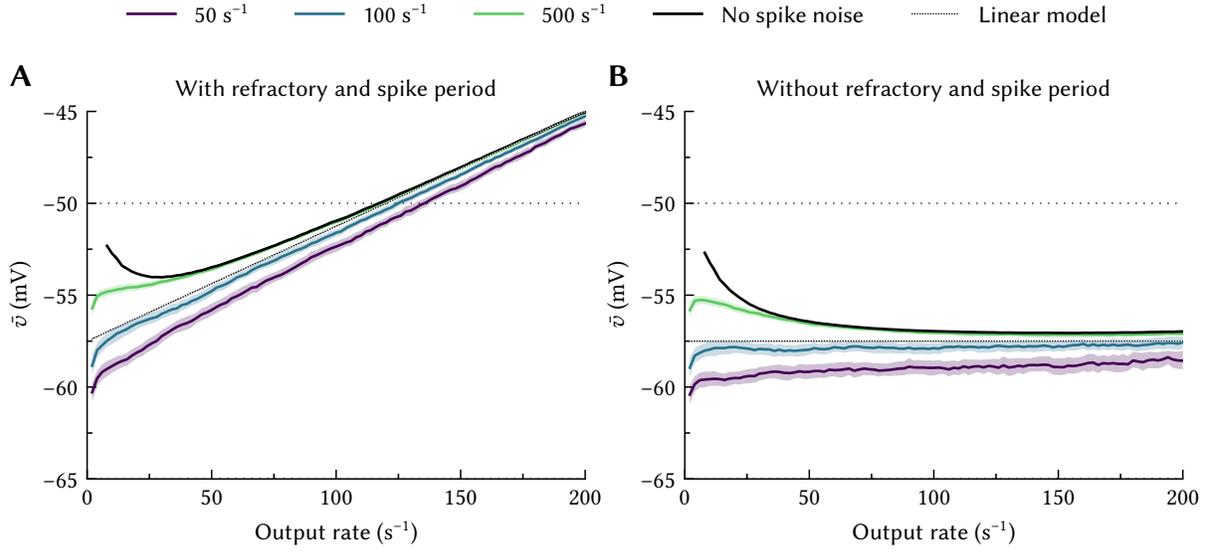


Figure 3.25: Mean somatic membrane potential \bar{v} over the output rate for a current-based LIF neuron. Lines correspond to the average potential for different pre-synaptic spike rates. Mean post-synaptic currents are fixed in individual trials, so the spike rate purely corresponds to the amount of noise. Data over 128 random Poisson spike-trains per 1000 individual mean post-synaptic currents. Synaptic filter time-constant is 5 ms. Shaded areas correspond to 25/75% percentiles, lines to the mean. **(A)** Average membrane potential including the refractory and spike period. **(B)** Average membrane potential excluding the refractory and spike period. Dotted line is a linear model that takes the relative length of the spike and refractory phase into account.

Table 3.2: Reduced matrix representations of the first four multi-compartment neuron models in Figure 3.22. The somatic compartment is disconnected from the remaining neuron model. Connections to the somatic compartment are replaced by a static conductance-based channel with reversal potential \bar{v} . The voltage difference between \bar{v} and the equilibrium potential of the new model is proportional to the current flowing into the somatic compartment.

Model	$\tilde{\mathbf{a}}'$	$\tilde{\mathbf{A}}'$	$\tilde{\mathbf{b}}'$	$\tilde{\mathbf{B}}'$	$\tilde{\mathbf{L}}$	$\tilde{\mathbf{c}}$
(A)	$[1]$	$[0 \ 0]$	$[g_L(E_L - \bar{v}) + \bar{v}]$	$[1 \ -1]$	$[0]$	$[1]$
(B)	$[1]$	$[0 \ 0]$	$[g_L(E_L - \bar{v}) + \bar{v}]$	$[E_E - \bar{v} \ E_I - \bar{v}]$	$[0]$	$[1]$
(C)	$\begin{bmatrix} 1 \\ g_L + c_{12} \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} g_L(E_L - \bar{v}) + \bar{v} \\ g_L E_L + c_{12} \bar{v} \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ E_E & E_I \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ c_{12} \end{bmatrix}$
(D)	$\begin{bmatrix} 1 \\ g_L + c_{12} \\ g_L \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} g_L(E_L - \bar{v}) + \bar{v} \\ g_L E_L + c_{12} \bar{v} \\ g_L E_L \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ E_E & E_I & 0 & 0 \\ 0 & 0 & E_E & E_I \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & c_{23} & -c_{23} \\ 0 & -c_{23} & c_{23} \end{bmatrix}$	$\begin{bmatrix} 1 \\ c_{12} \\ 0 \end{bmatrix}$

Average somatic current. To estimate the average current flowing into the somatic compartment, we replace the system from eq. (3.22) with a reduced system with vectors and matrices $\tilde{\mathbf{a}}'$, $\tilde{\mathbf{b}}'$, $\tilde{\mathbf{A}}'$, $\tilde{\mathbf{B}}'$, and $\tilde{\mathbf{L}}'$. These matrices describe a dynamical system of the form

$$\frac{d}{dt} \tilde{\mathbf{C}}_m \circ \tilde{\mathbf{v}}(t) = \tilde{\mathbf{A}}[\mathbf{g}(t)] \tilde{\mathbf{v}}(t) + \tilde{\mathbf{b}}[\mathbf{g}(t)] = -[\tilde{\mathbf{L}} + \text{diag}(\tilde{\mathbf{a}}' + \tilde{\mathbf{A}}' \mathbf{g}(t))] \tilde{\mathbf{v}}(t) + [\tilde{\mathbf{b}}' + \tilde{\mathbf{B}}' \mathbf{g}(t)]. \quad (3.25)$$

For constant \mathbf{g} , this system is linear and, analogously to what we have discussed in Section 3.3.2, converges to an equilibrium state $\tilde{\mathbf{v}}^{\text{eq}}$:

$$\tilde{\mathbf{v}}(t) = \tilde{\mathbf{v}}^{\text{eq}} + \exp(\tilde{\mathbf{A}}[\mathbf{g}] \text{diag}(\tilde{\mathbf{C}}_m)^{-1} t) (\tilde{\mathbf{v}}(0) - \tilde{\mathbf{v}}^{\text{eq}}), \quad \text{where} \quad \tilde{\mathbf{v}}^{\text{eq}} = -\tilde{\mathbf{A}}[\mathbf{g}]^{-1} \tilde{\mathbf{b}}[\mathbf{g}]. \quad (3.26)$$

This reduced system is constructed such that, for non-somatic compartments, the equilibrium potential \tilde{v}^{eq} converges to the voltage that we would obtain if the somatic compartment were clamped to \bar{v} . In the somatic compartment itself, the voltage difference $\tilde{v}_1^{\text{eq}} - \bar{v}$ is proportional to the current flowing into the compartment. Given a vector $\tilde{\mathbf{c}}$ of somatic coupling conductances with $\tilde{c}_1 = 1$, the average current $H(\mathbf{g})$ flowing into the soma is then modelled as

$$H(\mathbf{g}) = \lim_{T \rightarrow \infty} \frac{1}{T} \int_0^T H(\mathbf{g}; t) dt \approx \sum_{i=1}^n \tilde{c}_i (\tilde{v}_i^{\text{eq}} - \bar{v}). \quad (3.27)$$

In practice, we obtain the reduced system by setting the first column and row of $\tilde{\mathbf{L}}$ to zero and replacing connections to the somatic compartment with static conductance-based channels. In the somatic compartment, all conductance-based channels are replaced by current-based channels weighted by the difference between \bar{v} and the channel reversal potential (cf. Stöckel, Voelker, et al., 2017); furthermore, we let $(\tilde{\mathbf{a}}')_1 = 1$ and add \bar{v} to $(\tilde{\mathbf{b}}')_1$. We provide examples of reduced systems in Table 3.2.

As a word of warning, note that the reduced system as described here is conceptually useful, but notoriously ill-conditioned. We address this in Appendix A.2 by rescaling and offsetting the system such that $\tilde{\mathbf{A}}[\mathbf{g}]^{-1} \tilde{\mathbf{b}}[\mathbf{g}]$ directly represents currents instead of voltages.

Model parameters. Equation (3.27) is the result of major simplifications and as such unlikely to be accurate. Specifically, we assumed that the somatic compartment is effectively clamped to a constant potential \bar{v} , that \mathbf{g} is constant, and that spike generation has no effect on the dendritic compartments. These assumptions are readily violated in practice. The average somatic potential \bar{v} depends on the pre-synaptic noise-level (cf. Figure 3.25), the input \mathbf{g} is seldom constant, and spike generation affects the dendritic membrane potentials (cf. Figure 3.23B).

Thus, equation (3.27) is better interpreted as a “template” for the overall mathematical shape of the dendritic nonlinearity. That is, to at least partially compensate for the imprecisions in our derivation, we declare $\tilde{\mathbf{a}}'$, $\tilde{\mathbf{b}}'$, as well as the non-zero entries in $\tilde{\mathbf{A}}'$, $\tilde{\mathbf{B}}'$ to be free parameters. Keeping the graph Laplacian $\tilde{\mathbf{L}}$ and the zeros in $\tilde{\mathbf{A}}'$, $\tilde{\mathbf{B}}'$ fixed implies that we assume that the connectivity graph accurately describes the electrical network of the neuron.

The model parameters can be initialised with the original model and fit to direct numerical measurements of the somatic current J , or, alternatively, currents reconstructed from the neural activity, i.e., $J = G^{-1}[\mathcal{G}(\mathbf{g})]$. The calibration samples should be obtained from a setting that resembles the network context in which the neuron is used. For example, the pre-synaptic noise level should match what the neuron would be exposed to in the network.

We discuss methods for determining the model parameters, and test the quality of H in the following sections. However, before we do so, we explicitly derive H for a few special cases and analyse this dendritic nonlinearity model from a more theoretical perspective.

3.3.4 Some Worked Examples

The above framework is well-suited for algorithmically deriving the dendritic nonlinearity model H for arbitrary n -LIF neurons. Unfortunately, it may be less intuitive when manually analysing these models. Hence, we find it useful to at least provide the expanded dendritic nonlinearity H for the first four models depicted in Figure 3.22.

Single-compartment LIF neuron with current-based input. Expanding eq. (3.27) for the model depicted in Figure 3.22A using Table 3.2 yields

$$H(J_E, J_I) = J_E - J_I + g_L(E_L - \bar{v}).$$

This function is visualised in Figure 3.26A. To obtain the LIF-equivalent current we subtract the leak current, as mentioned above. Using the fully parameterised version of the equations, and renaming the parameters for better readability, we obtain the following affine model:

$$H(J_E, J_I) = \tilde{b}'_1 + \tilde{B}'_{1,1}J_E + \tilde{B}'_{1,2}J_I = b_0 + b_1J_E + b_2J_I.$$

Single-compartment LIF neuron with conductance-based input. For Figure 3.22B we obtain

$$H(g_E, g_I) = g_E(E_E - \bar{v}) + g_I(E_I - \bar{v}) + g_L(E_L - \bar{v}).$$

This function is illustrated in Figure 3.26B. The fully parameterised version of the model is

$$H(g_E, g_I) = \tilde{b}'_1 + \tilde{B}'_{1,1}g_E + \tilde{B}'_{1,2}g_I = b_0 + b_1g_E + b_2g_I.$$

This is the equivalent to the current-based neuron model.

Crucially, this is not just an artefact of our modelling framework, but indeed captures the behaviour of spiking neuron simulations well. We discuss this in more detail in Stöckel, Voelker, et al. (2017). Independent experiments by Kiselev, Ivanov, and Ivanov (2020) further support this observation. Hence, single-compartment LIF neurons with conductance-based synapses are rather uninteresting from a computational perspective.

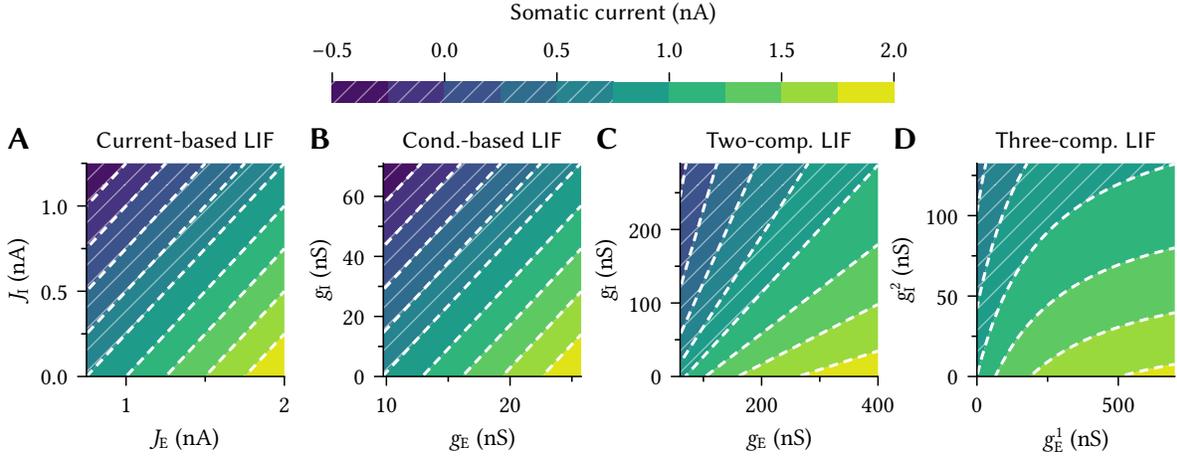


Figure 3.26: Dendritic nonlinearity models H for the n -LIF neurons depicted in Figure 3.22. Hatched regions correspond to subthreshold currents. Limits were chosen such that the spike onset is approximately on the diagonal of each plot. Parameters shared between all models: $g_L = 50$ nS, $C_m = 1$ nF, $E_L = -65$ mV, $E_E = 20$ mV, $E_I = -75$ mV, $\bar{v} = -57.5$ mV. **(A, B)** Single-compartment neurons with current- and conductance-based synapses. Apart from scaling, the two models are equivalent. **(C)** Two-compartment neuron with $c_{12} = 30$ nS. The contour lines are still straight lines but no longer parallel due to shunting. **(D)** Slice through the response curve of a three-compartment LIF neuron with $c_{12} = 40$ nS, $c_{23} = 100$ nS. The inputs $g_E^2 = 95$ nS and $g_I^1 = 20$ nS are kept constant. In contrast to the two-compartment neuron, the contour-lines are curved.

Two-compartment LIF neuron. For the model depicted in Figure 3.22C we have

$$H(g_E, g_I) = c_{12} \left(\frac{\bar{v}c_{12} + E_L g_L + E_E g_E + E_I g_I}{c_{12} + g_L + g_E + g_I} - \bar{v} \right) + g_L(E_L - \bar{v}). \quad (3.28)$$

Interestingly, in the limit of increasing the coupling conductance c_{12} to infinity we obtain

$$\lim_{c_{12} \rightarrow \infty} H(g_E, g_I) = E_E g_E + E_I g_I + 2g_L(E_L - \bar{v}) - \bar{v}(g_E + g_I),$$

that is, the two-compartment model reduces to an affine function if the two compartments are tightly coupled. The fully parameterised version of the model is given as

$$H(g_E, g_I) = c_{12} \left(\frac{\tilde{b}'_2 + \tilde{B}'_{2,1}g_E + \tilde{B}'_{2,2}g_I}{\tilde{a}'_2 + \tilde{A}'_{2,1}g_E + \tilde{A}'_{2,2}g_I} - \bar{v} \right) + b'_1.$$

An example of H is depicted in Figure 3.26C. Taking into account that the inhibitory channel should reduce the total current, we can rewrite this as a rational function and identify the maximum excitatory and inhibitory currents that can be generated by this model

$$H(g_E, g_I) = \frac{b_0 + b_1 g_E - b_2 g_I}{a_0 + a_1 g_E + a_2 g_I}, \quad \lim_{g_E \rightarrow \infty} H(g_E, g_I) = \frac{b_1}{a_1}, \quad \lim_{g_I \rightarrow \infty} H(g_E, g_I) = -\frac{b_2}{a_2}, \quad (3.29)$$

where $a_0 > 0$ and the other $a_1, a_2, b_0, b_1, b_2 \geq 0$.

Three-compartment LIF neuron. For models with more than two compartments, the dendritic nonlinearity model becomes rather unwieldy. For the neuron model depicted in Figure 3.22D, the dendritic nonlinearity $H(g_E^1, g_I^1, g_E^2, g_I^2)$ is equal to:

$$\begin{aligned}
 H = c_{12} \left(\right. & \left[(E_L g_E^1 + E_L g_I^1 + E_E g_E^2 + E_I g_I^2 + \bar{v} c_{12} + 2E_L c_{23}) g_L \right. \\
 & \left. + (E_E g_E^1 + E_I g_I^1 + E_E g_E^2 + E_I g_I^2 + \bar{v} c_{12}) c_{23} + (g_E^1 + g_I^1)(E_E g_E^2 + E_I g_I^2 + c_{12} \bar{v}) + g_L^2 E_L \right] \\
 & \left[(g_E^1 + g_I^1 + g_E^2 + g_I^2 + c_{12} + 2c_{23}) g_L \right. \\
 & \left. + (g_E^1 + g_I^1 + g_E^2 + g_I^2 + c_{12}) c_{23} + (g_E^1 + g_I^1)(g_E^2 + g_I^2 + c_{12}) + g_L^2 \right]^{-1} - \bar{v} \left. \right) + g_L (E_L - \bar{v}). \quad (3.30)
 \end{aligned}$$

A slice of this function is depicted in Figure 3.26D. Since it might not be immediately apparent, note that for $c_{23} \rightarrow 0$ the fraction in the above equation reduces to

$$\frac{(g_L + g_E^1 + g_I^1)(E_E g_E^2 + E_I g_I^2 + E_L g_L + c_{12} \bar{v})}{(g_L + g_E^1 + g_I^1)(g_L + c_{12} + g_E^2 + g_I^2)} = \frac{E_E g_E^2 + E_I g_I^2 + E_L g_L + c_{12} \bar{v}}{g_L + c_{12} + g_E^2 + g_I^2},$$

that is, the neuron reduces to the two-compartment neuron. The same is true for $c_{23} \rightarrow \infty$.

There is little room for simplifying eq. (3.30) further. However, notice that the numerator and denominator only contain products-terms between the input channels of different compartments. We can thus write H as

$$H(g_E^1, g_I^1, g_E^2, g_I^2) = \frac{b_0 + b_1 g_E^2 + b_2 g_I^2 + b_3 g_E^1 + b_4 g_I^1 + b_5 g_E^1 g_E^2 + b_6 g_I^1 g_E^2 + b_7 g_E^1 g_I^2 + b_8 g_I^1 g_I^2}{a_0 + a_1 g_E^2 + a_2 g_I^2 + a_3 g_E^1 + a_4 g_I^1 + a_5 g_E^1 g_E^2 + a_6 g_I^1 g_E^2 + a_7 g_E^1 g_I^2 + a_8 g_I^1 g_I^2}, \quad (3.31)$$

where $a_0 > 0$ and $a_1, \dots, a_8 \geq 0$. As we discuss below, the observation that the n -LIF dendritic nonlinearity only possesses product-terms between different compartments holds in general.

3.3.5 Theoretical Properties of the n -LIF Nonlinearity

So far, it is still unclear in how far n -LIF neurons provide a computational advantage over standard LIF neurons. As we discussed in Section 3.1, there must be some nonlinear interaction between different input channels for dendritic computation to offer any advantage.

Analysing the dendritic nonlinearity H sheds some light on this. To summarise, there is no nonlinear interaction between current-based inputs, inputs targeting the somatic compartment, or between inputs targeting different “branches” of the neuron. Furthermore, nonlinear interaction between inputs targeting the same compartment is limited to shunting. Conductance-based channels between different compartments of the same “branch” of a neuron interact multiplicatively. We phrase this more formally in the following theorem (see Appendix A.1.6 for a proof and Figure 3.27 for an illustration) and continue with a more thorough discussion of the individual interaction types between input channels.

Definition 3.4 (Branch). *A branch of an n -LIF neuron is a set of compartments that form a connected component in the n -LIF connectivity graph even if the somatic compartment is removed from the neuron.*

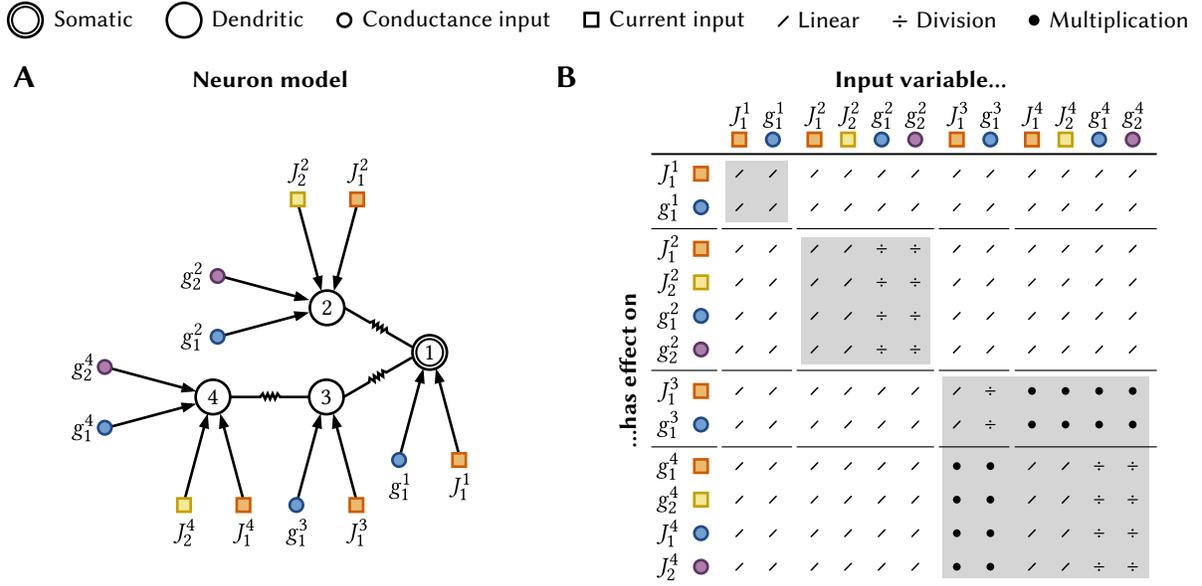


Figure 3.27: Effects of the relative location of input channels on the dendritic nonlinearity H . **(A)** Dendritic tree used in our example. The second compartment is only connected through the soma to the third and fourth compartment. **(B)** Table illustrating how pairs of input channels in **(A)** interact according to Theorem 3.7. Grey rectangles correspond to branches with nonlinear interaction. The symbol “/” indicates that H only contains linear combinations of terms containing each variable; “÷” indicates that the variable in this column acts divisively on the other variable in the numerator; “•” indicates that H contains product terms with these two variables.

Theorem 3.7. Consider an n -LIF neuron with ℓ branches, where each compartment is connected to k unique conductance- and k unique current-based input channels. We denote these inputs as g_j^i and J_j^i , where i and j are the compartment and input channel indices, respectively. Furthermore, arrange the compartment indices such that $i_{m-1} + 1, \dots, i_m$ belong to the same branch, where m with $1 \leq m \leq \ell$ is the branch index. Then, the somatic current model H has the following form

$$H_0(g_1^1, \dots, g_k^1, J_1^1, \dots, J_k^1) + \frac{H_1^B(g_1^2, \dots, g_k^2, J_1^2, \dots, J_k^2)}{H_1^A(g_1^2, \dots, g_k^2)} + \dots + \frac{H_\ell^B(g_1^{i_{\ell-1}+1}, \dots, g_k^{i_\ell}, J_1^{i_{\ell-1}+1}, \dots, J_k^{i_\ell})}{H_\ell^A(g_1^{i_{\ell-1}+1}, \dots, g_k^{i_\ell})},$$

where **(A)** H_0 is an affine function over all inputs injected into the somatic compartment, **(B)** the H_m^A are nonnegative affine functions of product terms between conductance-based inputs belonging to different dendritic compartments, and **(C)** the H_m^B are affine functions of product terms between conductance-based inputs belonging to different dendritic compartments, and at most one dendritic current-based input per product term.

Affine terms. The “computationally weakest” n -LIF neurons are those where the somatic current is merely an affine function over the input channels. As follows from Theorem 3.7, this is the case for n -LIF neurons that only possess a somatic compartment, or, more generally,

any n -LIF neuron where input is solely fed into the somatic compartment. The same holds for n -LIF neurons that only possess current-based input channels.

Shunting. Theorem 3.7 similarly dictates that nonlinear interaction between the input channels of an individual dendritic compartment is limited to “shunting” (cf. Koch, 1999, Section 1.5). That is, conductance-based input channels act “divisively” on the other input channels in that compartment; they contribute to the denominator of one of the fractions in Theorem 3.7.

Mathematically, this kind of nonlinear interaction is apparent in the dendritic nonlinearity for the two-compartment neuron (eq. 3.29); the excitatory and inhibitory conductances g_E and g_I act divisively on H . While the magnitude of this effect depends on the ratio between the input and the coupling- and leak-conductances, shunting in a single compartment cannot be used to compute “interesting” functions such as XOR (proof in Appendix A.1.7):

Theorem 3.8. *Let $g_E(x_1, x_2) = g_E^1(x_1) + g_E^2(x_2)$ and $g_I(x_1, x_2) = g_I^1(x_1) + g_I^2(x_2)$ be nonnegative functions. Furthermore, let $a_0 > 0$ and $a_1, a_2, b_0, b_1, b_2 \geq 0$. The two-compartment LIF nonlinearity*

$$\varphi(x_1, x_2) = H(g_E(x_1, x_2), g_I(x_1, x_2)) = \frac{b_0 + b_1 g_E(x_1, x_2) - b_2 g_I(x_1, x_2)}{a_0 + a_1 g_E(x_1, x_2) + a_2 g_I(x_1, x_2)}$$

cannot be used to solve the weak XOR problem (see Definition 3.1).

Still, and as we demonstrate in the next section, a single layer of two-compartment neurons can compute a wide range of functions with a substantially smaller error than a single layer, or, in some cases, even two layers of standard LIF neurons.

Product terms. Input channels targeting different dendritic compartments interact multiplicatively (cf. eq. 3.30). However, exploiting the product terms to compute multiplication over all four quadrants is difficult for multiple reasons. First, conductance-based inputs are nonnegative, and can thus only cover one quadrant. While current-based channels can provide positive and negative input, there is at most one current-based input per product term.

Second, the multiplicative terms are similar in magnitude to the linear terms—for example, all terms in the numerator in eq. (3.30) are the product of two conductances and one voltage, whereas all terms in the denominator are the product of two conductances. This implies that, to compute a function such as XOR, we must find synaptic weights that compensate for the linear terms, while, at the same time, computing the desired nonlinear function.

Lastly, there is a tradeoff between the coupling conductances c_{ij} and the maximum somatic current that can be produced by a compartment. This current is proportional to the product of the intermediate coupling conductances, making it more difficult for distal compartments to influence the soma. Countering this by choosing larger c_{ij} results in more linear H .

Still, as we demonstrate in Section 3.5, it is indeed possible to solve the weak XOR problem using a single three-compartment neuron. In particular, we discuss an iterative weight optimisation scheme that in practice quickly converges to locally optimal weights.

3.4 Networks of Two-Compartment LIF Neurons

Up to this point, we discussed the theoretical advantages of dendritic computation, extended the Neural Engineering Framework to support more complex connectivity constraints and neuron types, and introduced and analysed a family of multi-compartment LIF neurons with passive dendritic trees that we called “ n -LIF” neurons.

The goal of this section is to systematically incorporate the simplest non-trivial n -LIF neuron into NEF networks—namely the two-compartment LIF neuron depicted in Figure 3.22C. This represents the smallest possible step toward exploiting dendritic nonlinearities, and as such is a good test for our overall methodology.

We proceed as follows. First, we discuss a method for estimating the parameters of our surrogate dendritic nonlinearity model H using nonnegative least-squares and test the quality of the estimated parameters in various scenarios. A similar approach can be used to derive a quadratic program for the synaptic weights that takes nonnegative connectivity and subthreshold relaxation into account. We use this approach to evaluate the computational advantage of the two-compartment LIF nonlinearity in an optimal scenario without dynamics, akin to our experiment in Section 3.1.4. Finally, we demonstrate that this theoretical advantage persists in a dynamic spiking neural network context.

3.4.1 Estimating Model Parameters

We presented the parametrised dendritic nonlinearity surrogate model for two-compartment LIF neurons in eq. (3.29). While the model parameters $a_0, a_1, a_2, b_0, b_1, b_2$ can be estimated using eq. (3.28), it is better to fit the parameters to data from numerical simulations.

One way to generate these data is to simulate the spiking neuron for different input conductances $g_{E,k}, g_{I,k}$ over a period of time T and to compute the average spike rate $a_k = \mathcal{G}(g_{E,k}, g_{I,k})$. As discussed in Section 3.2.4, we then obtain the somatic current J_k by applying the inverse of the chosen one-dimensional response curve G^{-1} to a_k . Importantly, when doing this, samples with small a_k should be ignored: G^{-1} is not well-defined for zero rates, and our dendritic nonlinearity H was derived for superthreshold dynamics.

Optimisation problem. Given the superthreshold samples $J_k, g_{E,k}, g_{I,k}$ we now have the following least-squares loss function over the parameters a_i, b_i :

$$E = \sum_{k=1}^N (J_k - H(g_{E,k}, g_{I,k}))^2 = \sum_{k=1}^N \left(J_k - \frac{b_0 + b_1 g_{E,k} - b_2 g_{E,k}}{a_0 + a_1 g_{E,k} + a_2 g_{I,k}} \right)^2, \quad \text{subject to } a_0 > 0, \quad (3.32)$$

and $a_1, a_2, b_0, b_1, b_2 \geq 0$.

Note that this optimisation problem has one superfluous degree of freedom. Setting $b_1 = 1$ tends to be a numerically stable normalisation. In this case, all parameters are expressed relative to the effect of the excitatory channel on the input current.¹⁴

¹⁴As we mentioned in a footnote above, this is a result of the scale-invariance of the individual rows of $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{b}}$. In general, a parameterised n -LIF neuron has $n - 1$ superfluous degrees of freedom in its parameters.

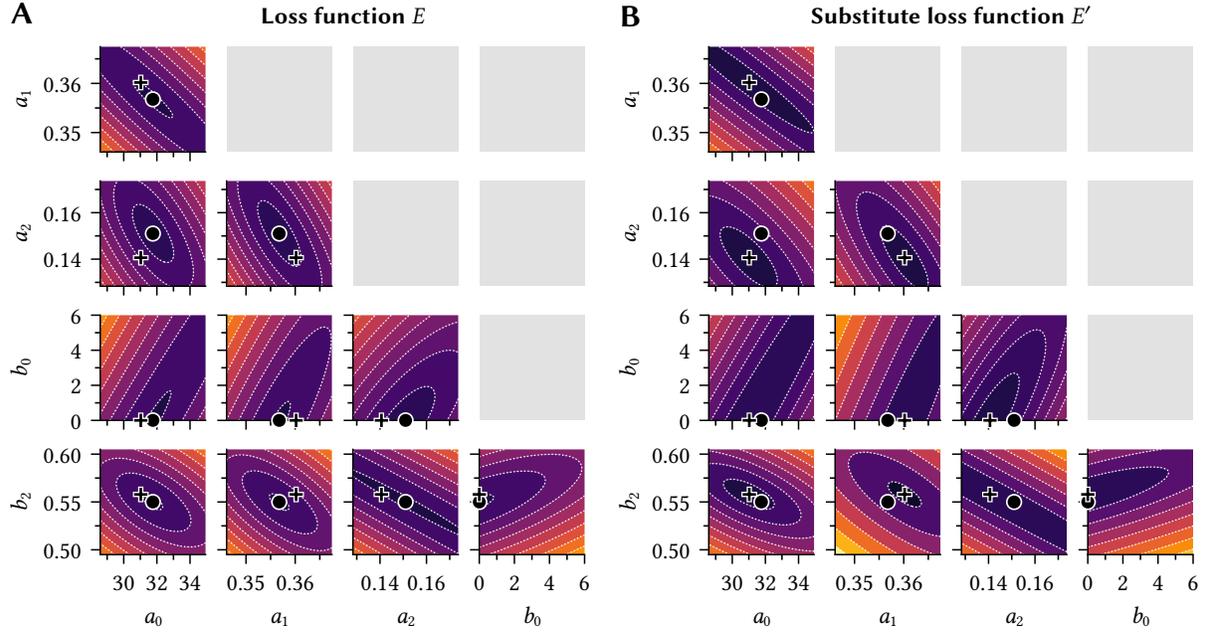


Figure 3.28: Comparison between the actual loss function (eq. 3.32; **A**) and the substitute loss function (eq. 3.33; **B**). Each plot depicts a slice of the error function E or E' over two parameters. Black crosses correspond to the optimal solution with respect to the substitute loss function E' , black circles correspond to the closest local optimum in the actual loss function. The underlying data is sampled from a two-compartment neuron with $E_E = 20$ mV, $E_I = -75$ mV, $E_L = -65$ mV, $g_L = 50$ nS, $c_{12} = 30$ nS for $N = 318$ superthreshold samples of g_E, g_I over $[0$ nS, 500 nS]. The RMSE current error for the optimised weights is $\sqrt{E/N} = 15.22$ pA when using the actual loss function for optimisation, and $\sqrt{E/N} = 15.67$ pA when using the substitute loss function for optimisation (3% increase). The parameter b_1 is set to one.

Unfortunately, eq. (3.32) is neither in *linear* least-squares form, nor convex. In theory, this complicates solving for model parameters (Rockafellar, 1993). Fortunately, we can largely work around this by minimising a convex substitute loss function (cf. Figure 3.28). Optimally, we would like the following equality to hold for each sample k

$$J_k = H(g_{E,k}, g_{I,k}) = \frac{b_0 + b_1 g_{E,k} - b_2 g_{I,k}}{a_0 + a_1 g_{E,k} + a_2 g_{I,k}} \iff 0 = J_k(a_0 + a_1 g_{E,k} + a_2 g_{I,k}) - b_0 - b_1 g_{E,k} + b_2 g_{I,k}.$$

Notably, the two equations are equivalent because the denominator is strictly non-zero. Phrasing this as a loss function for multiple samples yields

$$E' = \sum_{k=1}^N (J_k(a_0 + a_1 g_{E,k} + a_2 g_{I,k}) - b_0 - b_1 g_{E,k} + b_2 g_{I,k})^2, \quad (3.33)$$

subject to the same constraints as above. Arranging the samples in vectors $\mathbf{J}, \mathbf{g}_E, \mathbf{g}_I \in \mathbb{R}^N$, we can bring this into a canonical matrix form (where “ \circ ” is elementwise multiplication):

$$E' = \|\mathbf{A}\boldsymbol{\omega} - \mathbf{b}\|_2^2, \quad \text{where } \mathbf{A} = (\mathbf{J}, \mathbf{J} \circ \mathbf{g}_E, \mathbf{J} \circ \mathbf{g}_I, -\mathbf{1}, \mathbf{g}_I), \quad \mathbf{b} = \mathbf{g}_E, \quad \boldsymbol{\omega} = (a_0, a_1, a_2, b_0, b_2). \quad (3.34)$$

Crucially, and as is illustrated in Figure 3.28, the solutions obtained by minimising eqs. (3.32) and (3.33) are similar, but not equivalent. While both optimisation problems minimise the error for each sample, multiplication with the denominator in eq. (3.33) causes samples to be dynamically re-weighted. The impact of this is fairly low in practice, and the global minimum of the substitute loss seems to be close to a minimum in the actual loss function. Optimising rational function parameters by multiplying with the denominator is a common technique that has, for example, been proposed in the context of fitting transfer functions (Levy, 1959).

Parameter refinement. If an exact solution in terms of the closest local optimum of eq. (3.32) is desired, the parameters can easily be refined using gradient descent. Alternatively, and as originally suggested by Sanathanan and Koerner (1963) in the context of fitting transfer functions, we can refine the solution by iterative reweighting of eq. (3.34). Each sample is weighted by the inverse of its denominator from the previous iteration, i.e.,

$$E' = \|\text{diag}(\mathbf{w})^{-1} \mathbf{A}\boldsymbol{\omega} - \text{diag}(\mathbf{w})^{-1} \mathbf{b}\|_2^2, \quad \text{where } \mathbf{w} = a_0 \mathbf{1} + a_1 \mathbf{g}_E + a_2 \mathbf{g}_I. \quad (3.35)$$

This scheme converges to a (not necessarily globally optimal) fixed point. In our application this seems to be the case after two additional iterations. A more thorough review of this “Sanathanan-Koerner iteration” is given in Hokanson and Magruder (2018, Section 2.2) and Pintelon et al. (1994). For the sake of simplicity, and because we do not observe a significant reduction in errors, we chose not to rely on this parameter refinement in our experiments.

3.4.2 Experiment 1: Precision of the Model Parameter Estimation

The following experiment tests to what degree the surrogate model $G[H(g_E, g_I)]$ can be used to accurately predict the spike rate $\mathcal{G}(g_E, g_I)$ of a simulated spiking two-compartment LIF neuron—both before and after fitting the model parameters to empirical data. We conduct this experiment in an artificial scenario with constant conductances g_E, g_I , and in a simulated network context with artificial temporal spike noise superimposed onto the inputs.

Experiment 1.1: Constant conductances. We analyse three two-compartment LIF neurons with coupling conductances $c_{12} = 50 \text{ nS}, 100 \text{ nS},$ and 200 nS ; all other parameters are listed in Table B.1. We measure the output spike rate for constant input conductances g_E, g_I on a 100×100 grid. For each grid-point, the neuron’s dynamical system (cf. Section 3.3.1) is simulated for $T = 1 \text{ s}$. We then compute the steady-state firing rate by taking the inverse of the median inter-spike-interval. The conductance range has been selected such that the maximum rate is 100 s^{-1} , and the spike onset coincides with the diagonal of the g_E - g_I -rate contour plot.

We compare these numerical simulation results to both the theoretical somatic current prediction according to eq. (3.28), and the parameterised dendritic nonlinearity with the parameters fitted to the measured data. Parameter optimisation according to eq. (3.34) is based on a training-set of 200 conductance pairs sampled with uniform probability from the conductance-range. The final prediction error is computed over all 10 000 grid points.

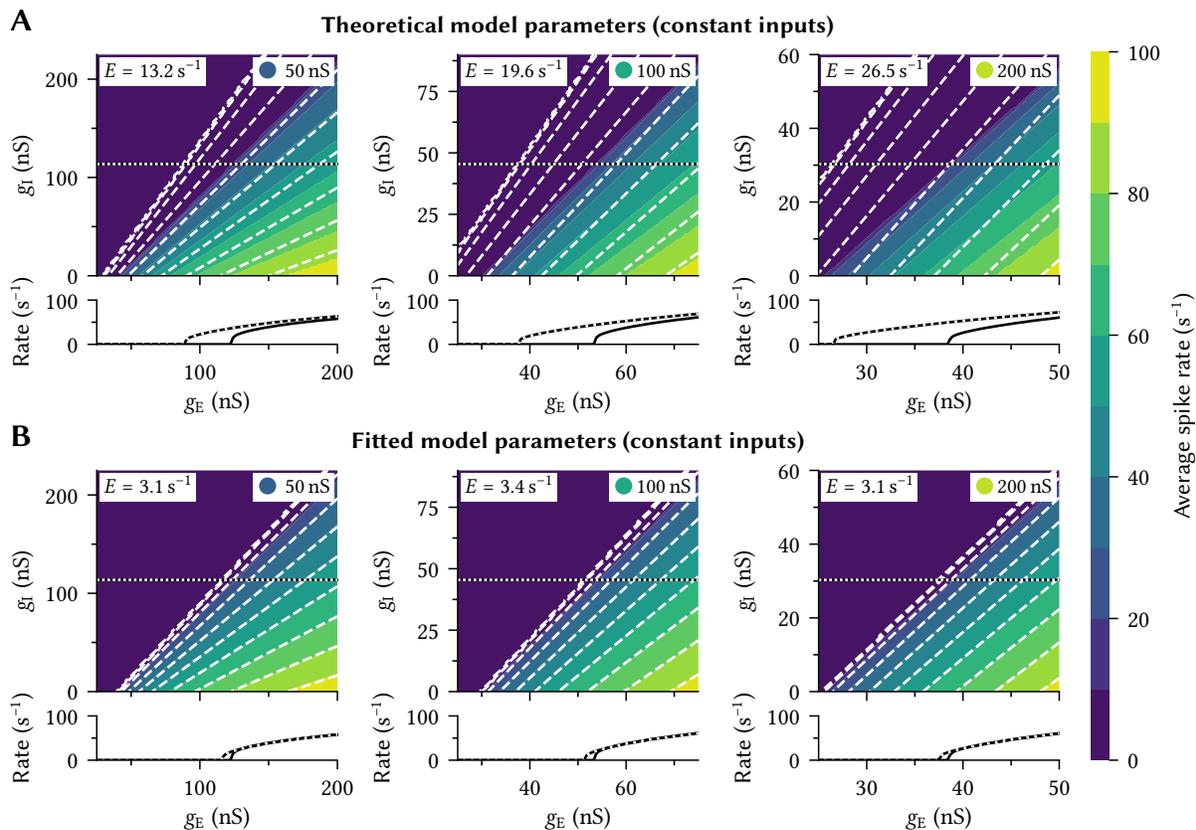


Figure 3.29: Two-compartment nonlinearity model for constant input. Contour plots depict measured average spike rates $\mathcal{G}(g_E, g_I)$. Dashed lines are the model prediction $G[H(g_E, g_I)]$. Dotted lines indicate the cross-section location. E denotes the spike-rate RMSE. Columns correspond to different coupling conductances c_{12} . The model fits the simulation well, with minor deviations near the spike onset.

Results. The results for this experiment are depicted in Figure 3.29; the fitted parameters can be found in Table B.2. Unsurprisingly, when using the theoretical parameter estimate from eq. (3.28), there is a substantial discrepancy between the model prediction and the simulation, especially for large c_{12} (Figure 3.29A). This discrepancy is reduced after fitting the model parameters (Figure 3.29B). The model prediction fits the empirical data well for output spike rates greater than 25 s^{-1} . However, it fails to predict the spike onset correctly, placing it too early with respect to increasing g_E . As we predicted in Section 3.3.5, the linearity of H increases as c_{12} is increased, i.e., the contour lines become more “parallel” for larger c_{12} .

Discussion. Overall, the model predicts spike rates well once the parameters have been fitted. The mismatch between measured rates and the theoretical, unfitted model for large c_{12} is likely due to the more pronounced effect of the somatic superthreshold dynamics on the dendritic compartment. Failure to predict the spike onset is likely due to our model not capturing low firing-rate regimes of the neuron well.

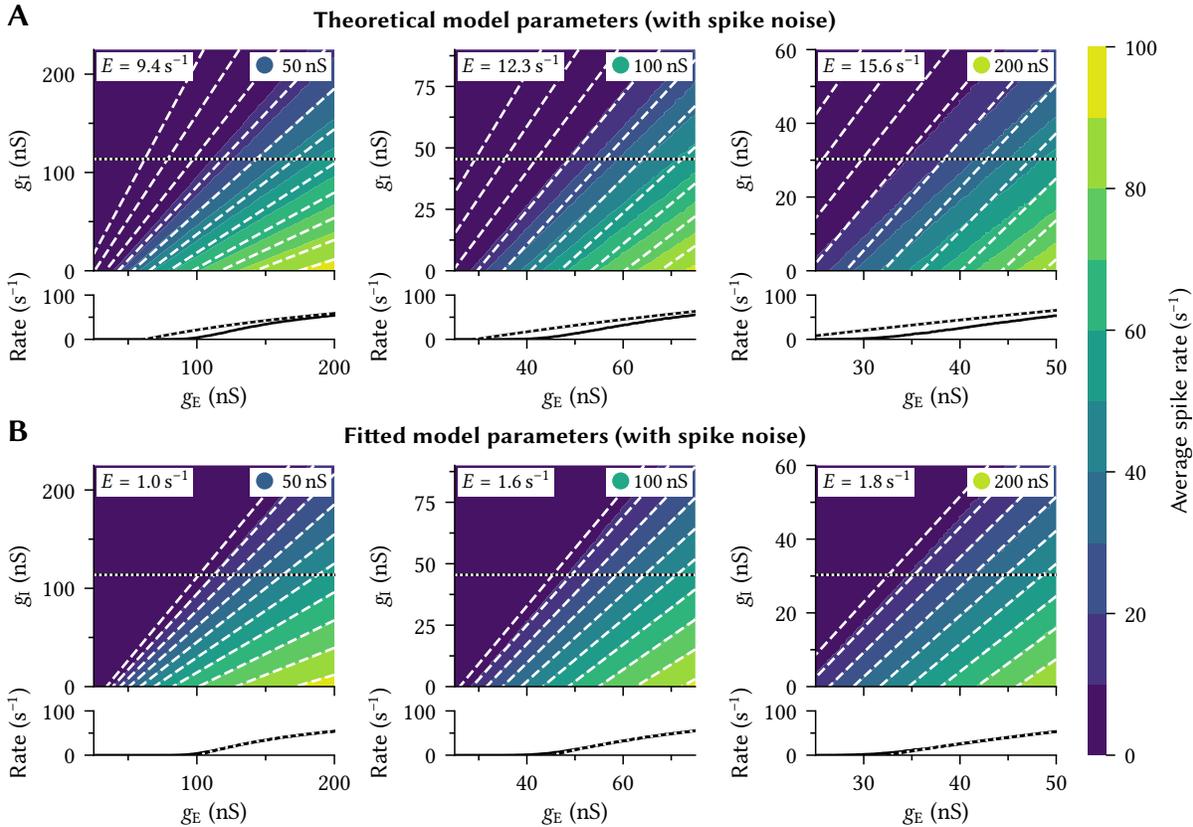


Figure 3.30: Two-compartment nonlinearity model for noisy input with combined excitatory Poisson spike rates of $1/\lambda = 4500 \text{ s}^{-1}$ for excitatory synapses and $1/\lambda = 1800 \text{ s}^{-1}$ for inhibitory synapses. See Figure 3.29 for a complete legend. The model fits the noisy data better than the noise-free data.

Experiment 1.2: Conductances with artificial temporal spike noise. In a network context, g_E and g_I are not constant, but, as we discussed in Section 2.2.2, a weighted sum of low-pass filtered spike trains. This results in a considerable amount of “spike noise” in the conductance inputs. In this experiment, we generate artificial filtered spike trains using spike times from Poisson sources (simulation period $T = 100 \text{ s}$; rates fitted to data from Experiment 3; see below). Synaptic weights are simulated by uniformly sampling scaling factors for each spike event. The resulting signals are scaled such that the average conductances are equal to g_E , g_I . We furthermore replace $G[J]$ with a ReLU, that is $G[J] = \max\{0, \alpha J + \beta\}$. We know from preliminary experiments that the hard LIF spike-onset is not present in noisy environments—this is not captured well by the standard LIF response curve. While we could use a “soft” version of the LIF response curve that takes this into account (cf. Capocelli and Ricciardi, 1971; Hunsberger, Scott, and Eliasmith, 2014; Kreutz-Delgado, 2015), a ReLU appears to work well.

Results. The results of this experiment are depicted in Figure 3.30. Final fitted parameters are provided in Table B.2. Surprisingly, our model fits the noisy spike rates better than non-noisy data. Still, the overall trends from the previous experiment are still visible. When using the

theoretical parameter estimates, larger c_{12} result in higher overall errors and fitting the model parameters greatly reduces the error to values of about 1 s^{-1} to 2 s^{-1} . While the sharp spike onset is no longer present, the model does not capture the subtle sigmoid shape of the response curve near the spike onset that is particularly pronounced for larger g_C .

Discussion. To summarise, our results indicate that, after fitting the model parameters, the surrogate model H can indeed be used to predict the neural response curve $\mathcal{G}(g_E, g_I)$ with a relatively high accuracy in both tested scenarios. It seems reasonable to choose a different one-dimensional neural response curve $G[J]$ when taking noise in the input into account.

3.4.3 Solving for Synaptic Weights

We demonstrated that the surrogate model $H(g_E, g_I)$ can be used to quite accurately predict the firing rates of individual two-compartment neurons. Now, to construct NEF networks, we must find weights $\mathbf{w}_i^E, \mathbf{w}_i^I$ that fulfil the normative tuning-curve constraint from eq. (3.3).

Optimisation problem. Given a function $f(\mathbf{x})$ that we would like to compute, we optimally minimise the following least-squares loss over $\mathbf{w}_i^E, \mathbf{w}_i^I$ for each post-neuron i

$$E = \frac{1}{\mathbb{X}} \int_{\mathbb{X}} \mathcal{E} \left(J_i(f(\mathbf{x})), \frac{b_0 + b_1 \langle \mathbf{w}_i^E, \mathbf{a}(\mathbf{x}) \rangle - b_2 \langle \mathbf{w}_i^I, \mathbf{a}(\mathbf{x}) \rangle}{a_0 + a_1 \langle \mathbf{w}_i^E, \mathbf{a}(\mathbf{x}) \rangle + a_2 \langle \mathbf{w}_i^I, \mathbf{a}(\mathbf{x}) \rangle} \right)^2 d\mathbf{x} + \sigma^2 \|\mathbf{w}_i^E\|_2^2 + \sigma^2 \|\mathbf{w}_i^I\|_2^2, \quad (3.36)$$

subject to $\mathbf{w}_i^E, \mathbf{w}_i^I \geq 0$, and where \mathcal{E} is the superthreshold error function (eq. 3.15), $\mathbf{a}(\mathbf{x})$ are the stacked pre-activities of all pre-populations, and $J_i(\mathbf{x})$ is the current-translation function.

Just like the model parameter optimisation problem from the previous subsection, eq. (3.36) is not convex. Again, we work around this by multiplying with the denominator inside the square to obtain a substitute loss function:

$$E' = \int_{\mathbb{X}} \mathcal{E} \left(J_i(f(\mathbf{x})) (a_0 + a_1 \langle \mathbf{w}_i^E, \mathbf{a}(\mathbf{x}) \rangle + a_2 \langle \mathbf{w}_i^I, \mathbf{a}(\mathbf{x}) \rangle), \frac{b_0 + b_1 \langle \mathbf{w}_i^E, \mathbf{a}(\mathbf{x}) \rangle - b_2 \langle \mathbf{w}_i^I, \mathbf{a}(\mathbf{x}) \rangle}{a_0 + a_1 \langle \mathbf{w}_i^E, \mathbf{a}(\mathbf{x}) \rangle + a_2 \langle \mathbf{w}_i^I, \mathbf{a}(\mathbf{x}) \rangle} \right)^2 d\mathbf{x} + \sigma^2 \|\mathbf{w}_i^E\|_2^2 + \sigma^2 \|\mathbf{w}_i^I\|_2^2, \quad (3.37)$$

subject to $\mathbf{w}_i^E, \mathbf{w}_i^I \geq 0$. A sampled version of this problem can be phrased as a quadratic program. To see this, let $\mathbf{A} \in \mathbb{R}^{N \times n}$ be a matrix of pre-activities and $\mathbf{J}_i \in \mathbb{R}^N$ be a vector of target currents. Assume that we only have superthreshold samples, that is $\mathcal{E}(J_{\text{tar}}, J_{\text{dec}}) = J_{\text{tar}} - J_{\text{dec}}$. We have:

$$\begin{aligned} E' &\propto \|\mathbf{J}_i \circ (a_0 + a_1 \mathbf{A} \mathbf{w}_i^E + a_2 \mathbf{A} \mathbf{w}_i^I) - (b_0 + b_1 \mathbf{A} \mathbf{w}_i^E - b_2 \mathbf{A} \mathbf{w}_i^I)\|_2^2 + \sigma^2 \|\mathbf{w}_i^E\|_2^2 + \sigma^2 \|\mathbf{w}_i^I\|_2^2 \\ &= \|(a_1 \text{diag}(\mathbf{J}_i) \mathbf{A} - b_1 \mathbf{A}) \mathbf{w}_i^E + (a_2 \text{diag}(\mathbf{J}_i) \mathbf{A} + b_2 \mathbf{A}) \mathbf{w}_i^I + (a_0 - b_0) \mathbf{J}_i\|_2^2 + \sigma^2 \|\mathbf{w}_i^E\|_2^2 + \sigma^2 \|\mathbf{w}_i^I\|_2^2 \\ &= \|\mathbf{A}' \mathbf{w}_i + \mathbf{J}'\|_2^2 + \sigma^2 \|\mathbf{w}_i\|_2^2. \end{aligned} \quad (3.38)$$

To account for subthreshold relaxation, we simply split \mathbf{A}' and \mathbf{J}' into super- and subthreshold samples as discussed in Section 3.2.3 and use the QP defined in eq. (3.17).

Again, we could use the Sanathanan-Koerner iteration (cf. eq. 3.35) to refine the solution. However, in our experience, and as with the parameter optimisation problem, the solution to the convex problem tends to be close to a local optimum in the original rational loss function.

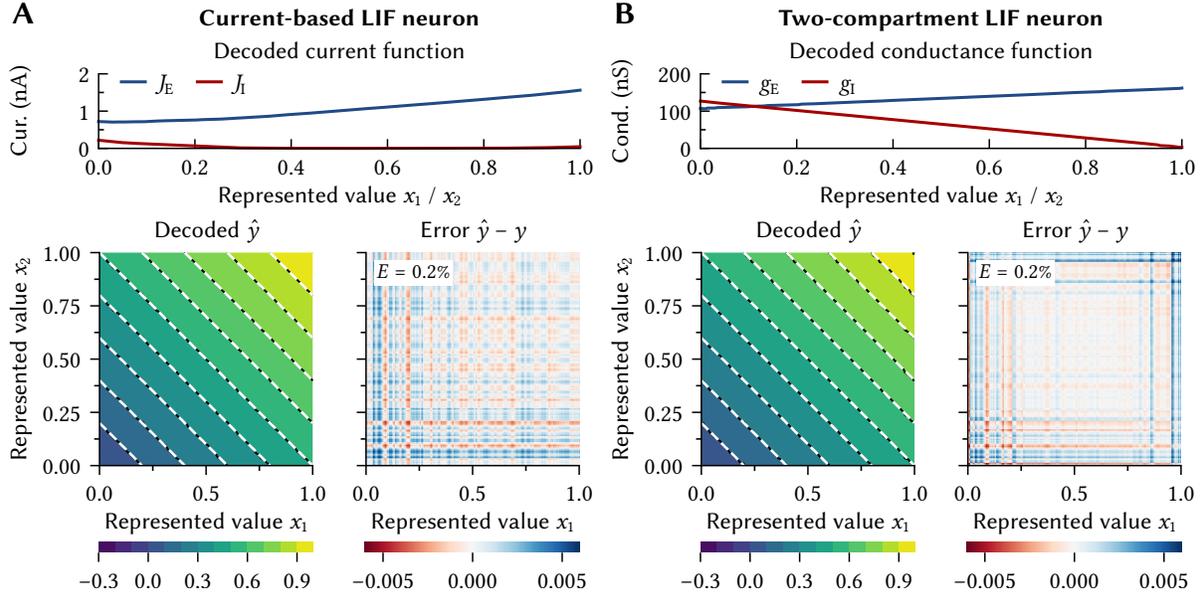


Figure 3.31: Computing addition in **(A)** single- and **(B)** two-compartment neurons. See text for a detailed description. *Top:* Decoded current and conductance functions. *Left:* Decoded represented value. Black contour lines and coloured backdrop correspond to the decoded values \hat{y} ; dashed white lines to the target function y . *Right:* Decoding error $\hat{y} - y$. Depicted error value E is the NRMSE. Single- and two-compartment neurons are equally suitable for computing linear functions.

Examples: Addition and gain modulation. Although we analyse two-compartment neurons more thoroughly in the next two sections, we would first like to test our weight optimisation scheme in the context of two specific functions f : addition and nonnegative multiplication.

Specifically, we use the two-compartment LIF neuron with $c_{12} = 50$ nS (parameter set (ii) from Table B.1) and the network setup discussed in Section 3.1.3. That is, two pre-populations with 200 neurons each project onto a single post-neuron; the pre-populations represent x_1 , x_2 , respectively, while the post-neuron represents $\hat{y} \approx f(x_1, x_2)$.¹⁵ As depicted in Figure 3.1C, the synaptic weights implicitly define a set of conductance functions. Due to commutativity of addition and multiplication, the excitatory and inhibitory functions decoded from each pre-population optimally are the same; we have $g_E(x) = g_E^1(x) = g_E^2(x)$ and $g_I(x) = g_I^1(x) = g_I^2(x)$. We emulate current-based LIF neurons by setting $a_0 = b_1 = 1$, $b_2 = -1$, and $a_1 = a_2 = b_0 = 0$ in the nonlinearity model H , that is, for these parameters we have $H(J_E, J_I) = J_E - J_I$.

Results for computing addition, that is $f(x_1, x_2) = \frac{1}{2}(x_1 + x_2)$ over $(x_1, x_2) \in [0, 1]^2$, are depicted in Figure 3.31. As is expected, we can compute this function with a low NRMSE with the current-based nonlinearity. Perhaps unintuitively, and in spite of the dendritic nonlinearity H , we achieve similarly low errors using the two-compartment LIF neuron.

¹⁵While an individual neuron is not sufficient for reconstructing the represented value \hat{y} via linear decoding, we can infer \hat{y} by inverting the current translation function; that is $\hat{y} = J^{-1}[H(g_E(x_1) + g_E(x_2), g_I(x_1) + g_I(x_2)))]$. Our post-neuron has a positive encoder, an x -intercept of zero, and a maximum rate of 100 s^{-1} .

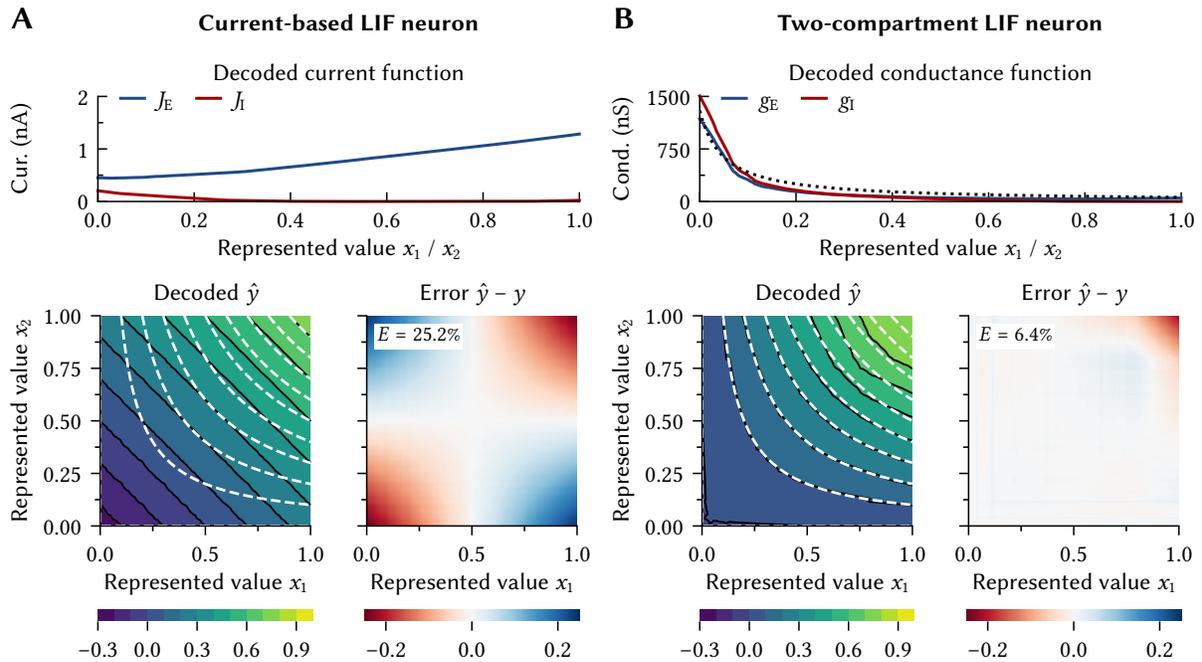


Figure 3.32: Computing multiplication in (A) single- and (B) two-compartment neurons. See text for a detailed description and Figure 3.31 for a legend. Nonnegative multiplication can be reasonably approximated using two-compartment neurons. *Top:* Dotted line in (B) is a hyperbolic fit.

The way in which the weight solver accomplishes this becomes apparent when considering the decoded current functions g_E and g_I in Figure 3.31B. Both g_E and g_I are affine functions with opposing slopes. Correspondingly, the denominator $a_0 + a_1 g_E + a_2 g_I$ stays approximately constant, while the numerator $b_0 + b_1 g_E - b_2 g_I$ generates the desired target currents.

Results for computing nonnegative multiplication, that is $f(x_1, x_2) = x_1 x_2$ over $(x_1, x_2) \in [0, 1]^2$, are depicted in Figure 3.32. Using current-based LIF neurons we can, at best, approximate multiplication with a linear function. This results in an NRMSE of about 25%. In contrast, using the two-compartment LIF nonlinearity results in an error of about 6%, with the largest discrepancies being in regions where both x_1 and x_2 are close to one.

Importantly, and as is depicted in Figure 3.33B, two-compartment LIF neurons cannot approximate nonnegative multiplication with an arbitrarily small error; more precisely, the error cannot be reduced past 6% for a single post-neuron. This is in contrast to addition, where we can reach arbitrarily small errors by increasing the number of pre-neurons (cf. Figure 3.33A). It is possible to reach smaller (but not arbitrarily small) errors with multiple post neurons dividing up the represented space (we reach down to 4%; see E_{model} in Table B.4).

Curiously, looking at the conductance functions g_E and g_I , *both* excitation and inhibition increase substantially for small x_1 or x_2 , similar to a shifted and scaled hyperbola $(\beta + \alpha x)^{-1}$ (black dotted line in Figure 3.32B). This may be counter-intuitive, given that inhibition alone

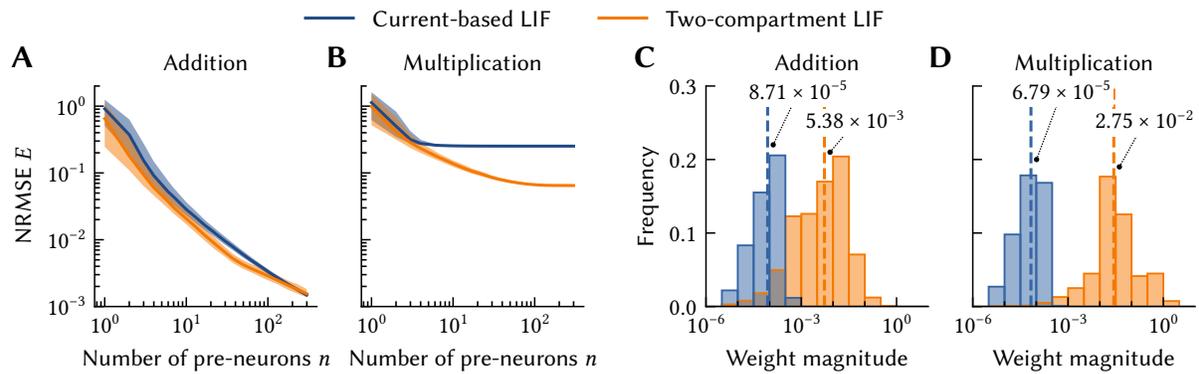


Figure 3.33: Error and weight statistics for computing addition and nonnegative multiplication. (A, B) Depicted is the median over 1000 experiments, shaded areas the 25/75-percentiles. While errors monotonically decrease with more pre-neurons in the addition task (A), errors quickly plateau when computing multiplication (B). (C, D) Comparison between the weight magnitude frequencies for the two tasks and neuron types (for $n = 300$ pre-neurons). Weights are normalised such that a value of one corresponds to 1 nA or 1 nS, respectively. Dashed line and depicted values are the median; frequencies include zero-weights. Weights below 10^{-6} are counted as zero. Two-compartment neurons require larger weight magnitudes. The spread of the magnitudes changes depending on the computed function.

should suffice to shut off the target neuron.¹⁶ However, when increasing both excitation and inhibition, this common-mode increase mostly cancels out in the numerator (due to the inhibitory conductance being subtracted), but increases the magnitude of the denominator; this amplifies the divisive effect of the input.

Notably, as we mentioned in the introduction of this chapter, nonnegative multiplication (or, alternatively, nonnegative division) is also referred to as “gain modulation” in the neuroscience literature (Salinas and Thier, 2000). The observation that both excitation and inhibition must be increased to multiplicatively (or divisively) reduce the gain of a neuron is consistent with *in vitro* experiments (Chance et al., 2002).

An earlier hypothesised mechanism for gain modulation is *shunting inhibition*. Here, the idea is that an inhibitory channel with a reversal potential close to the resting potential acts divisively on the average input current. This effect could in theory be used to implement multiplication (Koch and Poggio, 1992); however, in practice, increasing inhibitory conductances alone predominantly has a linear effect on the spike rate (since $a_2 \ll b_2$), or requires implausibly high conductance values (Holt et al., 1997; Abbott and Chance, 2005).

Even when exploiting the common-mode increase of excitation and inhibition, generating the large input conductances for small x_1 , x_2 results in relative large synaptic weights. This is depicted in Figures 3.33C and 3.33D. Compared to computing addition, the median synaptic weight increases by a factor of five.

¹⁶Remember that these results are for a single post-neuron with positive encoder. That is, to represent smaller values, the neuron must receive a smaller input current.

3.4.4 Experiment 2: Theoretical Analysis of Two-Compartment LIF Neurons

Given the encouraging results from the previous subsection, we would now like to analyse the theoretical advantage of the two-compartment LIF model in a more systematic manner. In particular, our goal is to characterise the “computational power” of different network and neuron types using the methodology from our basis-function experiment in Section 3.1.4.

We abbreviate the two-compartment LIF nonlinearity with conductance-based synapses as H_{cond} , and the standard current-based LIF neuron as H_{cur} . In addition, we implicitly assume that H_{cond} accurately describes the somatic current flowing into the two-compartment neuron. This is supported by our earlier results from Experiment 1 (Section 3.4.2). As mentioned before H_{cur} , is simply defined as $H_{\text{cur}}(J_E, J_I) = J_E - J_I$.

Methods. Just as in Section 3.1.4, we would like to measure how well different networks can approximate functions of increasing complexity ρ^{-1} . We sample random 2D target-current functions J_ρ over $(x_1, x_2) \in [-1, 1]^2$ with an RMS of 0.5 nA on a 63×63 grid using the technique illustrated in Figure 3.6. We then measure how well J_ρ can be approximated for a single post-neuron given the static tuning of the networks depicted in Figure 3.1. The pre-populations consist of 100 neurons each and project both excitatorily and inhibitorily onto the post-neuron; the intermediate population in the two-layer network (Figure 3.1B) consists of 200 neurons. In each trial, the pre-population tuning is randomly generated.

All synaptic weights are computed by solving the QP in eq. (3.38) for 256 randomly selected training samples, with and without subthreshold relaxation (the relaxation threshold is set to 0 nA). The regularisation parameters were selected independently for each setup (cf. Appendix B.2.1). The final error E_{model} is the NRMSE over all grid points with Gaussian noise (standard deviation $10^{-2} a_{\text{max}}$) added to the pre-activities to test for generalisation.

Results. Overall, the results depicted in Figure 3.34 are similar to those from our theoretical basis-function experiment Section 3.3.5 (cf. Figure 3.8). In particular, the error curve for the two-compartment neuron qualitatively resembles the multiplicative nonlinearity, although errors tend to be higher for H_{cond} : the inflection point of the sigmoid is at $\rho^{-1} \approx 0.7$ for the two-compartment LIF nonlinearity, and was at $\rho^{-1} \approx 1$ for the multiplicative nonlinearity.

Similarly to the additive network in the basis-function experiment (cf. the dotted line labelled “additive baseline”), the median error for H_{cur} initially increases almost linearly on a log-log plot, starting from a 2.5% error for low-frequency functions to an error of about 50% for functions with ρ^{-1} greater than one. Subthreshold relaxation reduces this error by up to 50%.

The error for the conductance-based two-compartment LIF nonlinearity initially increases sub-linearly on a log-log plot, starting at median errors of about 0.8% for low-frequency functions. The maximum reduction in the median error compared to the standard-LIF neuron is about 65%. Errors are competitive with the two-layer network for $\rho^{-1} < 0.5$. The error converges to the results for the standard LIF neuron for $\rho^{-1} > 1$. Compared to standard LIF neurons, the advantage of subthreshold relaxation is, on average, not as pronounced.

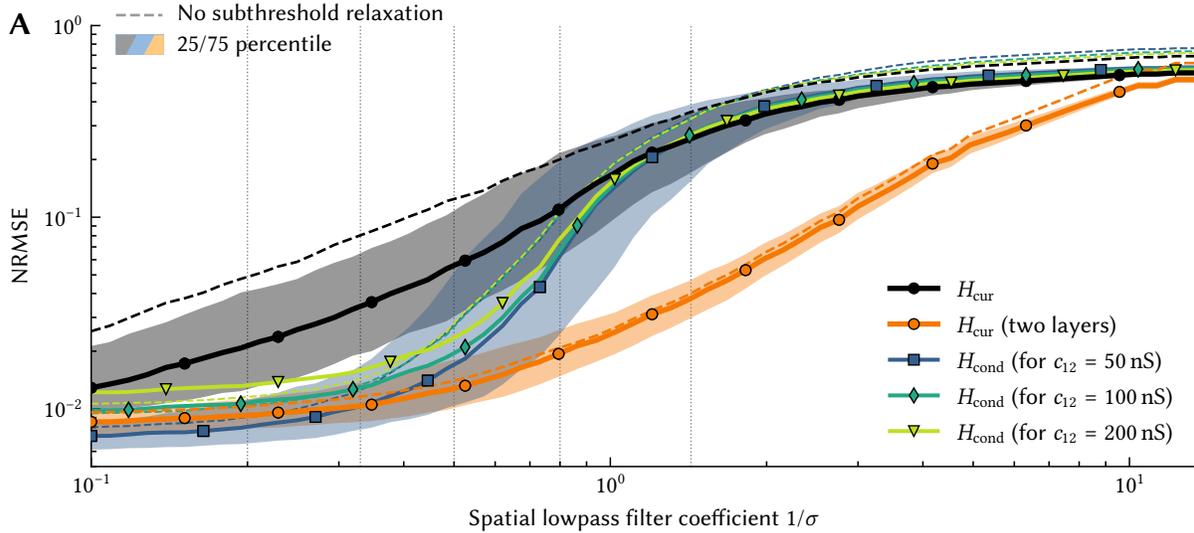


Figure 3.34: Decoding error for random multivariate current functions. NRMSE between random, two-dimensional current functions and the decoded approximation using different input-dependent nonlinearities; errors are based on the superthreshold error function \mathcal{E} (eq. 3.16). The low-pass filter coefficient ρ^{-1} is a proxy for the spatial frequency content in the target function (cf. Section 3.1.4). All points correspond to the median over 100 trials. Dashed lines show results for not taking subthreshold relaxation into account when solving for weights. Dotted lines are theoretical predictions from Figure 3.8B. The black lines show the results for a linear, current-based H_{cur} ; blue/green lines show the results for the two-compartment conductance-based model H_{cond} with parameters given in Table B.2 (without noise). Orange lines correspond a current-based network with two-dimensional pre-neuron tuning (i.e., a two-layer neural network). Shaded areas correspond to the 25/75 percentile for the current-based models and the conductance-based model with $c_{12} = 50$ nS.

Discussion. The large errors for H_{cur} and H_{cond} in regimes with $\rho^{-1} > 0.5$ are likely due to both functions not being able to solve the XOR problem (cf. Appendix A.1.7). Functions with $\rho^{-1} \geq 0.5$ may possess multiple maxima/minima over $[-1, 1]^2$ (cf. Figure 3.6), akin to XOR. Trying to approximate these functions using H_{cur} or H_{cond} thus leads to a large error.

This observation explains the difference between the multiplicative nonlinearity from the basis-function experiment (see the “multiplicative baseline” in the above figure) and the two-compartment LIF neuron—the multiplicative nonlinearity *can* be used to solve the XOR problem. Evidently, networks capable of approximating XOR-like functions are, using our terminology from Section 3.1, more “powerful” than those limited to conductance-based shunting.

To summarise, this experiment demonstrates that the two-compartment LIF dendritic nonlinearity H_{cond} substantially reduces the approximation error for J_ρ with $\rho^{-1} < 1$ compared to an additive network with nonlinearity H_{cur} . The two-compartment neuron is competitive with a two-layer network for $\rho^{-1} < 0.5$ and reaches similar errors as the multiplicative nonlinearity.

3.4.5 Experiment 3: Dendritic Computation in Spiking Neural Networks

Our results for Experiment 1 suggest that we can use the two-compartment LIF nonlinearity H_{cond} to predict the average current flowing into the somatic compartment of a two-compartment neuron. Furthermore, Experiment 2 shows that we can use H_{cond} to approximate a relatively large class of functions well. In our final experiment, we validate whether these results still hold true in a spiking neural network context.

While we use the same basic network setup as in Experiment 2, we now decode the represented value from a population of 100 neurons through time. Optimally, this decoded value should be $f(x_1, x_2)$, where the input signals $x_1(t), x_2(t) \in [-1, 1]$ are represented by the pre-populations. Additionally, we now adhere to Dale’s principle. Neurons are randomly marked as either excitatory or inhibitory, with a 30% probability of a neuron being inhibitory.

We model fast excitatory synapses as an exponential low-pass (cf. Section 2.2.2) with a time-constant of 5 ms as observed in glutamatergic pyramidal neurons with AMPA receptor (Jonas, Major, and Sakmann, 1993). The inhibitory pathway is modeled with a 10 ms time-constant as observed in inhibitory interneurons with GABA_A receptors (Gupta, Wang, and Markram, 2000). All remaining neuron parameters can be found in Table B.1.

In each trial, we simulate the network over 10 s at a time-resolution of 100 μs . Inputs $x_1(t)$ and $x_2(t)$ are moving along a fourth-order space-filling Hilbert curve covering $[-1, 1]^2$ (Hilbert, 1891). The activities of the output population are decoded and filtered with a first-order exponential low-pass at $\tau = 100$ ms. To determine the approximation error, we pass the target $f(x_1(t), x_2(t))$ through exactly the same filter chain. Our final error E_{net} is the NRMSE between the decoded output and the filtered target signal over time. Figure 3.35 depicts a single trial.

Again, all synaptic weights are computed by solving the QP in eq. (3.38). We set the relaxation threshold for subthreshold relaxation to 75% of J_{th} , as suggested by our experiments in Section 3.2.3. The regularisation factor σ has been selected independently for each parameter set such that E_{net} is minimised when computing nonnegative multiplication (cf. Appendix B.2.1).

Experiment 3.1: Random bandlimited functions. We first test our setup with the bandlimited functions f_ρ used in Experiment 2. Results are depicted in Figure 3.36. Qualitatively, the results are similar to what we saw before, although the minimum errors are substantially higher. The reduction in error between the current- and conductance-based models is not quite as large as suggested by the previous experiment, with a maximum reduction (in terms of the median) of only 40% (instead of 65% before). While subthreshold relaxation mostly increases the performance of the current-based model in the previous experiment, the improvement in error is now clearly visible for the conductance-based model as well (especially for $c_{12} = 50$ nS).

Notably, the minimum median approximation error of the two-layer network is about 10%, whereas the single-layer current- and conductance-based models reach minimum errors of about 6% and 4%, respectively. The two-layer network clearly surpasses the performance of the two-compartment LIF single-layer network for $\rho^{-1} > 0.6$.

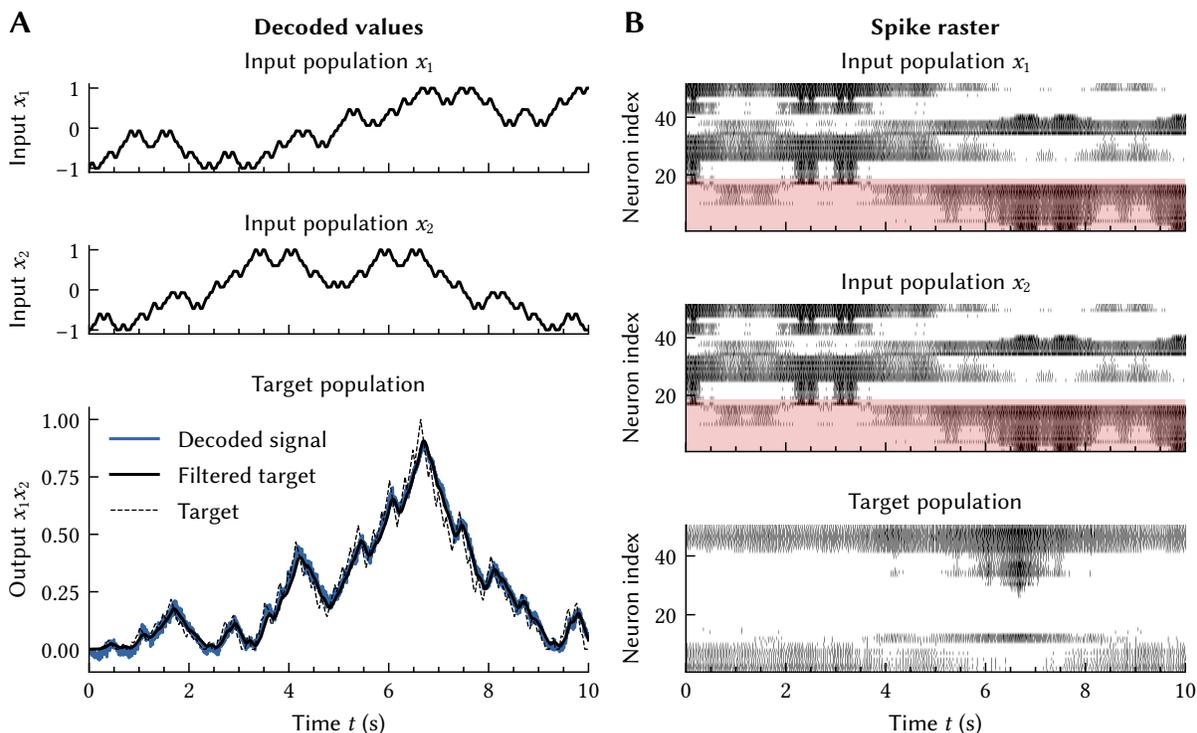


Figure 3.35: Computing nonnegative multiplication in a spiking neural network using two-compartment LIF neurons. **(A)** *Top two plots:* inputs $x_1(t)$ and $x_2(t)$ as represented by the two pre-populations. The input is a fourth order 2D Hilbert curve. *Bottom:* mathematical target $f(x_1(t), x_2(t)) = (x_1(t) + 1)(x_2(t) + 1)/4$, filtered target function, as well as the decoded target population output. **(B)** Spike raster plots corresponding to the spiking activity of each of the populations (only half of the neurons are depicted). Red shaded background corresponds to inhibitory neurons in the pre-populations, all other neurons are excitatory.

Discussion. The elevated error levels are likely due to the spike noise that is now superimposed onto all input signals. Notably, this noise, as well as the added dynamics, do not affect the two-compartment neuron over-proportionally, as one might expect due to the simplifications made during the derivation of the dendritic nonlinearity model (cf. Section 3.3.3). To the contrary, the largest effect of the additional noise is on the two-layer network.

We think that the larger errors for the two-layer network are caused by the representation of a two-dimensional quantity being noisier than the representation of the individual scalars in the two pre-populations. This is in line with our observation from Section 3.1.4. Given a maximum error, the number of basis functions decodable from a multi-dimensional population is not much larger than the decodable function count for a one-dimensional population.

To solve this issue, we would optimally have to square the number of neurons used in the intermediate layer. In our case, we would have to use $100^2 = 10\,000$ neurons instead of 200, which would not really be comparable to the single-layer setups.

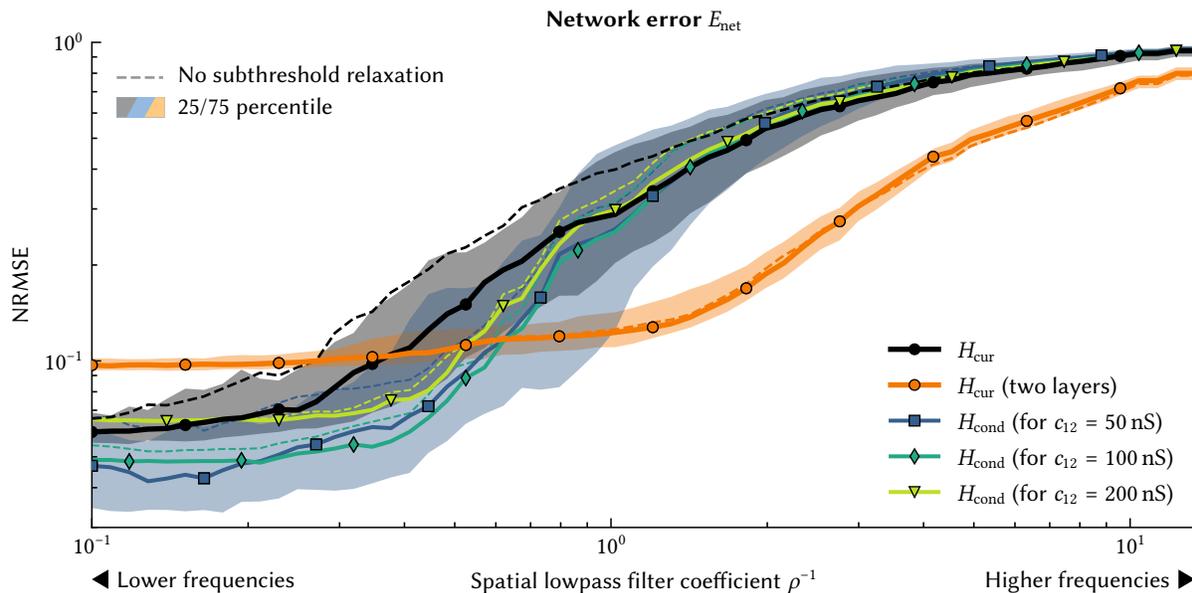


Figure 3.36: Median error for computing random bandlimited functions in a spiking feed-forward network over 100 trials. Measured NRMSE is the difference between the represented $\hat{y}(t)$ and the expected $f_\rho(x_1(t), x_2(t))$ relative to the standard deviation of f_ρ . See Figure 3.34 for a detailed description.

Experiment 3.2: Benchmark functions. While the random functions in the above experiments are useful to systematically characterise the individual setups, it is hard to tell from these data alone what the impact of using two-compartment neurons could be in practice. To this end, we selected eight benchmark functions $f(x_1, x_2)$ typically found in models of neurobiological systems and repeated the experiment. See Table B.3 for a detailed list.

Note that we rescale the input $(x_1, x_2) \in [-1, 1]^2$ to map onto the domain $[0, 1]^2$. Similarly, we rescale the output of each function to fully cover the codomain $[-1, 1]$. This ensures that we fully utilise the range of values that can be represented by the neuron populations, while restricting functions such as multiplication to a single quadrant.

Results. A summary of the results over 256 trials per function and setup is given in Table 3.3. More detailed results can be found in Table B.4. The two-compartment model with a coupling conductance of $g_C = 50$ nS achieves the smallest error E_{net} across all target functions.

Using the dendritic nonlinearity model parameters derived under noise (cf. Section 3.4.2) is beneficial when computing multiplicative functions and the maximum. For these target functions, the synaptic connection matrix tends to be sparser, increasing individual parameter weights and thus input noise (cf. Figures 3.33C and 3.33D).

The minimum error for the two-layer network is about 9% even for simple functions, matching the observation we made in the random function experiment above. The current-based neuron is only competitive with the two-compartment neuron for computing addition. Subthreshold relaxation decreases the error by up to 40%.

Table 3.3: Spiking neural network approximation errors for function approximations on $[0, 1]^2$. Error values correspond to the NRMSE and are measured as the difference between the output decoded from the target population and the desired output for a ten second sweep across a 4th order 2D Hilbert curve over the input space. Results are the mean and standard deviation over 256 trials. The best result for a target function is set in bold; darker background colours indicate a worse ranking of the result in the corresponding row. Columns labelled “standard” refer to the default, single layer network setup, “two layers” refers to the two-layer setup, and “noisy” to the single layer network setup with model parameters derived under noise (see Experiment 1.2). Additional tables can be found in Appendix B.2.

Target	Experiment setup						
	LIF			Two comp. $c_{12} = 50$ nS		Two comp. $c_{12} = 100$ nS	
	standard	standard [†]	two layers [†]	standard [†]	noisy [†]	standard [†]	noisy [†]
$x_1 + x_2$	5.7 ± 0.2%	5.8 ± 0.3%	9.9 ± 0.4%	4.3 ± 0.3%	8.2 ± 0.9%	4.7 ± 0.3%	11.2 ± 0.9%
$x_1 \times x_2$	25.8 ± 1.2%	15.5 ± 1.2%	9.2 ± 0.3%	4.4 ± 0.6%	4.7 ± 0.4%	5.4 ± 0.6%	6.9 ± 0.4%
$\sqrt{x_1 \times x_2}$	24.9 ± 1.8%	14.7 ± 1.7%	12.6 ± 0.9%	7.1 ± 0.9%	6.0 ± 0.7%	7.7 ± 1.0%	7.6 ± 0.6%
$(x_1 \times x_2)^2$	23.1 ± 0.7%	16.2 ± 0.9%	9.3 ± 0.5%	5.1 ± 0.9%	3.9 ± 0.6%	7.2 ± 1.2%	8.8 ± 1.0%
$x_1/(1 + x_2)$	15.0 ± 0.7%	10.1 ± 0.8%	9.2 ± 0.6%	4.5 ± 0.4%	11.4 ± 1.7%	6.1 ± 0.6%	14.8 ± 1.8%
$\ (x_1, x_2)\ $	18.6 ± 0.6%	10.8 ± 0.9%	9.7 ± 0.4%	5.3 ± 0.5%	8.2 ± 0.9%	6.0 ± 0.6%	12.1 ± 0.9%
$\text{atan}(x_1, x_2)$	17.8 ± 1.0%	12.4 ± 1.0%	12.5 ± 0.8%	7.2 ± 1.8%	7.5 ± 1.5%	8.3 ± 1.8%	8.1 ± 1.0%
$\max(x_1, x_2)$	34.5 ± 1.1%	22.6 ± 1.4%	10.0 ± 0.4%	12.9 ± 1.5%	9.9 ± 1.3%	14.9 ± 1.8%	16.3 ± 0.9%

[†]With subthreshold relaxation

Discussion. An effect that could contribute to the superior performance of the two-compartment neurons is the low-pass filter property of the dendritic compartment. This filter reduces high-frequency spike noise and thus may positively impact E_{net} . We control for this effect in an experiment described in Appendix B.2.1, where we add an optimal low-pass filter to each network setup. Results are shown in Table B.6. A matched pre-filter reduces the error in all setups by only 1%-2% (absolute), which indicates that the low-pass filter dynamics of the dendritic compartment cannot be the primary source for the reduction in error.

To summarise our experiments, we demonstrated in three stages (validation of the nonlinearity model H_{cond} for a single neuron, purley mathematical properties of H_{cond} , and, finally, performance on a network-level) that we are able to successfully incorporate two-compartment LIF neurons, an admittedly simple model of shunting in dendritic trees, into functional modeling frameworks. Instead of reducing the accuracy of our networks, the added detail can be systematically leveraged for computation.

Our experiments also suggest that—at least in a biologically plausible setting, i.e., using spiking neurons—this type of computation may result in a higher accuracy compared to two-layer architectures that suffer from an increase in the amount of spike-induced temporal noise due to the additional neuron layer.

3.5 Weight-Optimisation for Arbitrary Dendritic Trees

As demonstrated in the previous section, we are now able to integrate two-compartment LIF neurons—the simplest non-trivial n -LIF neuron—into NEF networks. Although this model only features *divisive*, but not *multiplicative* interaction between input channels (cf. Section 3.3.5), we observed a substantial computational advantage over standard LIF neurons.

The goal of this section is to discuss techniques for integrating arbitrary n -LIF neurons into spiking neural networks. This is more difficult than working with two-compartment LIF neurons, especially when it comes to finding model parameters and synaptic weights. Remember that, in the context of the two-compartment LIF neuron, we were able to optimise a convex loss function, while finding that, in our application, the global minimum of the convex loss was close to the optimum of the non-convex target loss function (cf. Section 3.4.1).

Similar “one-shot” optimisation schemes are unsuccessful for n -LIF neurons with more than two compartments. Of course, we could still approximate the parameter and weight optimisation problems as convex quadratic programs (QPs). However, the optimum of these QPs will not be a good solution for the original problem. Instead, we must iteratively refine our solution to reach a local optimum in the target loss function. Iterative algorithms based on quadratic programs for solving non-convex optimisation problems are also referred to as “sequential quadratic programs” (SQPs; e.g., Nocedal et al., 2006, Chapter 18).

Unfortunately, there are few guarantees with respect to the quality of the solution obtained with such iterative methods. Special cases such as the rational functions encountered in the two-compartment LIF neuron aside, finding the global optimum of non-convex functions is generally “hopeless” in that this problem is NP-hard (e.g., Sun, 2016).

Still, and as we demonstrate below, we are able to exploit the structure of the surrogate nonlinearity model H to construct an SQP that outperforms quasi-Newton methods such as L-BFGS in terms of speed of convergence, while being highly competitive in the final test error. We continue by using our optimisation techniques to repeat the experiments pertaining the theoretical power of n -LIF neurons and the actual performance in spiking neural networks.

3.5.1 Solving for n -LIF Surrogate Model Parameters

Similarly to what we discussed in the context of integrating two-compartment neurons into the NEF, our first step toward using arbitrary n -LIF neurons is to calibrate our surrogate nonlinearity model H to ground-truth somatic current measurements J_ℓ given input $\mathbf{g}_\ell \in \mathbb{R}^k$.

Recall from Section 3.3.3—specifically eqs. (3.26) and (3.27)—that the surrogate dendritic nonlinearity model $H(\mathbf{g})$ for an arbitrary n -LIF neuron is described in terms of the reduced system matrices $\tilde{\mathbf{A}}[\mathbf{g}]$, $\tilde{\mathbf{b}}[\mathbf{g}]$, which in turn can be decomposed into matrices $\tilde{\mathbf{L}}$, $\tilde{\mathbf{a}}'$, $\tilde{\mathbf{A}}'$, $\tilde{\mathbf{b}}'$, $\tilde{\mathbf{B}}'$, $\tilde{\mathbf{c}}$.

$$H(\mathbf{g}) \approx \sum_{i=1}^n \tilde{c}_i (\tilde{v}_i^{\text{eq}}(\mathbf{g}) - \bar{v}), \quad \text{where } \tilde{\mathbf{v}}^{\text{eq}}(\mathbf{g}) = -\tilde{\mathbf{A}}[\mathbf{g}]^{-1} \tilde{\mathbf{b}}[\mathbf{g}] = [\tilde{\mathbf{L}} + \text{diag}(\tilde{\mathbf{a}}' + \tilde{\mathbf{A}}' \mathbf{g})]^{-1} [\tilde{\mathbf{b}}' + \tilde{\mathbf{B}}' \mathbf{g}].$$

In the following, we implicitly assume that this system has been preconditioned as described in Appendix A.2. This implies that $\bar{v} = 0$ and $\tilde{c}_i \in \{0, 1\}$. In addition to simplifying our equations, preconditioning is crucial for guaranteeing that our optimisation algorithms work correctly.

As we discussed in Section 3.3.1, the vectors $\tilde{\mathbf{a}}'$ and $\tilde{\mathbf{b}}'$, as well as the entries of $\tilde{\mathbf{A}}'$ and $\tilde{\mathbf{B}}'$ associated with an input channel should be seen as free parameters that must be calibrated to ground-truth data. Specifically, we would like to minimise the following loss over $\tilde{\mathbf{a}}'$, $\tilde{\mathbf{A}}'$, $\tilde{\mathbf{b}}'$, $\tilde{\mathbf{B}}'$:

$$E = \sum_{k=1}^N \left(\langle \tilde{\mathbf{v}}^{\text{eq}}[\mathbf{g}_\ell], \tilde{\mathbf{c}} \rangle - J_\ell \right)^2 = \sum_{k=1}^N \left(\langle (\tilde{\mathbf{L}} + \text{diag}(\tilde{\mathbf{a}}' + \tilde{\mathbf{A}}' \mathbf{g}_\ell))^{-1} (\tilde{\mathbf{b}}' + \tilde{\mathbf{B}}' \mathbf{g}_\ell), \tilde{\mathbf{c}} \rangle - J_\ell \right)^2. \quad (3.39)$$

Minimising this equation is challenging because of the highly nonlinear matrix inverse $\tilde{\mathbf{A}}[\mathbf{g}]^{-1}$. Below, we discuss two approaches for minimising this loss function: gradient-based methods including stochastic gradient descent and quasi-Newton methods, as well as a soft trust-region based nonnegative sequential quadratic program (SQP).

Gradient-based optimisation. The most straightforward way to approach most optimisation problems is via stochastic gradient descent (SGD; e.g., Bishop, 2006 Section 3.1.3; Goodfellow et al., 2016, Section 5.9). To perform stochastic gradient descent, we must compute the partial derivative of eq. (3.39) with respect to the parameters that we are optimising. The same gradients can also be used in quasi-Newton methods such as BFGS (Nocedal et al., 2006, Chapter 2) and the more memory-efficient L-BFGS (Nocedal et al., 2006, Section 9.1).

To compute these gradients, we make use of the fact that the matrix differential of the matrix inverse \mathbf{X}^{-1} is the Kronecker product (denoted as “ \otimes ”) of the inverse matrices, resulting in a tensor of order four (see Lütkepohl, 1997, Section 10.6, eq. 1, p. 198):

$$\frac{\partial \mathbf{X}^{-1}}{\partial \mathbf{X}} = (\mathbf{X}^T)^{-1} \otimes \mathbf{X}^{-1} \quad \Leftrightarrow \quad \left(\frac{\partial \mathbf{X}^{-1}}{\partial (\mathbf{X})_{rs}} \right)_{ij} = ((\mathbf{X}^T)^{-1})_{ir} (\mathbf{X}^{-1})_{sj}. \quad (3.40)$$

Using this equation, we obtain the following partial derivatives of eq. (3.39):

$$\frac{\partial E}{\partial (\tilde{\mathbf{a}}')_r} = -2 \sum_{\ell=1}^N \left(H(\mathbf{g}_\ell) - J_\ell \right) \left(\sum_{i=1}^n \sum_{j=1}^n (\tilde{\mathbf{A}}[\mathbf{g}_\ell]^{-1})_{ir} (\tilde{\mathbf{A}}[\mathbf{g}_\ell]^{-1})_{rj} (\tilde{\mathbf{b}}[\mathbf{g}_\ell])_j (\tilde{\mathbf{c}})_i \right), \quad (3.41)$$

$$\frac{\partial E}{\partial (\tilde{\mathbf{A}}')_{rs}} = -2 \sum_{\ell=1}^N \left(H(\mathbf{g}_\ell) - J_\ell \right) \left(\sum_{i=1}^n \sum_{j=1}^n (\tilde{\mathbf{A}}[\mathbf{g}_\ell]^{-1})_{ir} (\tilde{\mathbf{A}}[\mathbf{g}_\ell]^{-1})_{rj} (\mathbf{g}_\ell)_s (\tilde{\mathbf{b}}[\mathbf{g}_\ell])_j (\tilde{\mathbf{c}})_i \right), \quad (3.42)$$

$$\frac{\partial E}{\partial (\tilde{\mathbf{b}}')_r} = -2 \sum_{\ell=1}^N \left(H(\mathbf{g}_\ell) - J_\ell \right) \left(\sum_{i=1}^n (\tilde{\mathbf{A}}[\mathbf{g}_\ell]^{-1})_{ir} (\tilde{\mathbf{c}})_i \right), \quad (3.43)$$

$$\frac{\partial E}{\partial (\tilde{\mathbf{B}}')_{rs}} = -2 \sum_{\ell=1}^N \left(H(\mathbf{g}_\ell) - J_\ell \right) \left(\sum_{i=1}^n (\tilde{\mathbf{A}}[\mathbf{g}_\ell]^{-1})_{ir} (\mathbf{g}_\ell)_s (\tilde{\mathbf{c}})_i \right). \quad (3.44)$$

To ensure that $\tilde{\mathbf{A}}[\mathbf{g}]$ is nonsingular, $\tilde{\mathbf{a}}'$ and $\tilde{\mathbf{A}}'$ must be nonnegative (see Section 3.3.2). Hence, when performing gradient descent, these parameters must be clipped to zero after every update step. When using quasi-Newton methods, an algorithm that supports bounds must be employed—an example of such an algorithm is L-BFGS-B (Byrd et al., 1995).

Soft trust-region based optimisation. As mentioned above, optimising eq. (3.39) is particularly difficult because of the matrix inverse $\tilde{\mathbf{A}}[\mathbf{g}_\ell]^{-1}$. Luckily, we can eliminate the matrix inverse—albeit in exchange for vastly increasing the dimensionality of our optimisation problem.

The basic idea is to split the loss function into two separate optimisation problems with N additional auxiliary variables $\mathbf{v}_\ell \in \mathbb{R}^n$. Let $\boldsymbol{\theta}$ summarise all parameters that we would like to optimise. Then, we first need to find \mathbf{v}_ℓ such that $(\langle \mathbf{v}_\ell, \tilde{\mathbf{c}} \rangle - J_\ell)^2$ is minimised, and second, find $\boldsymbol{\theta}$ such that $\tilde{\mathbf{A}}[\mathbf{g}_\ell; \boldsymbol{\theta}]^{-1} \tilde{\mathbf{b}}[\mathbf{g}_\ell; \boldsymbol{\theta}] = \mathbf{v}_\ell$. This idea is roughly expressed in the following two-part loss function over both the auxiliary variables \mathbf{v}_ℓ and our desired parameters $\boldsymbol{\theta}$:

$$E = \alpha_1 E_1 + \alpha_2 E_2, \quad E_1 = \sum_{\ell=1}^N (\langle \mathbf{v}_\ell, \tilde{\mathbf{c}} \rangle - J_\ell)^2, \quad E_2 = \sum_{\ell=1}^N \|\tilde{\mathbf{A}}[\mathbf{g}_\ell; \boldsymbol{\theta}]^{-1} \tilde{\mathbf{b}}[\mathbf{g}_\ell; \boldsymbol{\theta}] - \mathbf{v}_\ell\|_2^2, \quad (3.45)$$

where the hyperparameters α_1, α_2 determine the “importance” of each optimisation problem. Importantly, splitting the loss function in this way is technically incorrect! This is because the equality constraint $\tilde{\mathbf{A}}[\mathbf{g}_\ell; \boldsymbol{\theta}] \mathbf{v}_\ell = \tilde{\mathbf{b}}[\mathbf{g}_\ell; \boldsymbol{\theta}]$ is no longer strictly enforced. We could for example enforce strict equality using Lagrange multipliers (e.g., Boyd et al., 2004, Section 5.1); however, the linearisation step below would once more ruin this constraint.

Similar to our “trick” in Section 3.4.1, where we multiplied with the denominator of a rational function, we can now multiply with $-\tilde{\mathbf{A}}[\mathbf{g}_\ell; \boldsymbol{\theta}]$ inside the L_2 -norm to eliminate the inverse. While this does not change the optimal solution for each individual sample ℓ , doing so implicitly reweights the system, just as we discussed above.¹⁷ Blissfully ignoring this, we obtain

$$\begin{aligned} E_2' &= \sum_{\ell=1}^N \|\tilde{\mathbf{A}}[\mathbf{g}_\ell; \boldsymbol{\theta}] \mathbf{v}_\ell + \tilde{\mathbf{b}}[\mathbf{g}_\ell; \boldsymbol{\theta}]\|_2^2 = \sum_{\ell=1}^N \|(\tilde{\mathbf{L}} + \text{diag}(\tilde{\mathbf{a}}'[\boldsymbol{\theta}] + \tilde{\mathbf{A}}'[\boldsymbol{\theta}]\mathbf{g}_\ell))\mathbf{v}_\ell - (\tilde{\mathbf{b}}'[\boldsymbol{\theta}] + \tilde{\mathbf{B}}'[\boldsymbol{\theta}]\mathbf{g}_\ell)\|_2^2 \\ &= \sum_{\ell=1}^N \|\tilde{\mathbf{L}}\mathbf{v}_\ell - \tilde{\mathbf{b}}'[\boldsymbol{\theta}] - \tilde{\mathbf{B}}'[\boldsymbol{\theta}]\mathbf{g}_\ell + \tilde{\mathbf{a}}'[\boldsymbol{\theta}] \circ \mathbf{v}_\ell + \tilde{\mathbf{A}}'[\boldsymbol{\theta}]\mathbf{g}_\ell \circ \mathbf{v}_\ell\|_2^2, \end{aligned}$$

where “ \circ ” denotes elementwise multiplication. Obviously, this is not quite a convex loss function—note the product terms including both $\boldsymbol{\theta}$ and \mathbf{v}_ℓ . We work around this by performing a first-order Taylor expansion. Specifically, note that for elementwise multiplication we have

$$\mathbf{x} \circ \mathbf{y} \approx \mathbf{x}_0 \circ \mathbf{y}_0 + (\mathbf{x} - \mathbf{x}_0) \circ \mathbf{y}_0 + \mathbf{x}_0 \circ (\mathbf{y} - \mathbf{y}_0) = \mathbf{x} \circ \mathbf{y}_0 + \mathbf{x}_0 \circ \mathbf{y} - \mathbf{x}_0 \circ \mathbf{y}_0. \quad (3.46)$$

This is guaranteed to be a good approximation close to the expansion points \mathbf{x}_0 and \mathbf{y}_0 .

¹⁷We experimented with applying a Sanathanan-Koerner-inspired correction (cf. eq. 3.35) by weighting each sample proportionally to the inverse of the determinant of $\tilde{\mathbf{A}}[\mathbf{g}_\ell; \boldsymbol{\theta}]$ (i.e., the denominator of $-\tilde{\mathbf{A}}[\mathbf{g}_\ell; \boldsymbol{\theta}]^{-1} \tilde{\mathbf{b}}[\mathbf{g}_\ell; \boldsymbol{\theta}]$). However, we once again found that this has no appreciable positive effect on the quality of the obtained results.

Now, to linearise eq. (3.46), we choose $\boldsymbol{\theta}_0$ and $\mathbf{v}_{\ell,0}$ as expansion points; $\boldsymbol{\theta}_0$ could be the initial estimate from Table 3.2, and $\mathbf{v}_{\ell,0}$ the corresponding equilibrium potentials. Performing a Taylor expansion of the element-wise multiplications inside the L_2 -norm, we obtain

$$E_2'' = \sum_{\ell=1}^N \left\| \tilde{\mathbf{L}}\mathbf{v}_\ell - \tilde{\mathbf{b}}'[\boldsymbol{\theta}] - \tilde{\mathbf{B}}'[\boldsymbol{\theta}]\mathbf{g}_\ell + \tilde{\mathbf{a}}'[\boldsymbol{\theta}] \circ \mathbf{v}_{\ell,0} + \tilde{\mathbf{a}}'[\boldsymbol{\theta}_0] \circ \mathbf{v}_\ell - \tilde{\mathbf{a}}'[\boldsymbol{\theta}_0] \circ \mathbf{v}_{\ell,0} \right. \\ \left. + \tilde{\mathbf{A}}'[\boldsymbol{\theta}]\mathbf{g}_\ell \circ \mathbf{v}_{\ell,0} + \tilde{\mathbf{A}}'[\boldsymbol{\theta}_0]\mathbf{g}_\ell \circ \mathbf{v}_\ell - \tilde{\mathbf{A}}'[\boldsymbol{\theta}_0]\mathbf{g}_\ell \circ \mathbf{v}_{\ell,0} \right\|_2^2. \quad (3.47)$$

Our modified loss $\alpha_1 E_1 + \alpha_2 E_2''$ is now in linear least-squares form and can be expressed as a QP. However, we must also account for the Taylor expansion only yielding a good approximation in a small region surrounding the expansion points. Optimally, we would like to find a solution $\boldsymbol{\omega} = (\boldsymbol{\theta}, \mathbf{v}_1, \dots, \mathbf{v}_\ell)$ to our optimisation problem that is within the boundaries of an ellipsoid described by $\|\boldsymbol{\omega} - \boldsymbol{\omega}_0\|_2^2 < r^2$. This is a common concept in the numerical optimisation literature and referred to as a *trust-region* (Nocedal et al., 2006, Chapter 4).

While it is possible to augment quadratic program solvers to include a trust-region constraint (Nocedal et al., 2006, Chapter 18), this is not supported by our off-the-shelf QP solver of choice, OSQP (Stellato et al., 2020). We thus rely on a theoretically less sound, but more pragmatic solution—we simply add a third L_2 term E_3 to our loss function. This term penalises $\boldsymbol{\omega}$ moving away from the expansion point $\boldsymbol{\omega}_0$, establishing a “soft” trust-region:

$$E_3 = N\|\boldsymbol{\theta} - \boldsymbol{\theta}_0\|_2^2 + \sum_{k=1}^N \|\mathbf{v}_k - \mathbf{v}_{k,0}\|_2^2. \quad (3.48)$$

Crucially, and in contrast to canonical trust-region methods, the soft trust-region by design shifts the location of the global optimum. This may lead to undesired convergence behaviour when using the soft trust-region as part of a sequential quadratic program—that is, solving the QP and using the solution as an expansion point for a next iteration, as we discuss below.

This undesired convergence behaviour is apparent when interpreting the soft trust-region as a way to emulate gradient descent, as is depicted in Figure 3.37. Just as with naïve gradient descent, the final convergence to the local minimum is extremely slow, since we do not use curvature information that would be available in the Hessian (Goodfellow et al., 2016, Section 4.3.1). However, in the context of our parameter optimisation problem, this interpretation should not be taken too literally. Our soft trust-region algorithm does *not* perform gradient descent; we do not linearise according to the local curvature $\nabla_{\boldsymbol{\theta}} E$, but within a higher-dimensional space taking prior knowledge about our model and curvature information into account.

To summarise, our final loss function E' with two additional regularisation terms is

$$E' = \alpha_1 E_1 + \alpha_2 E_2'' + \alpha_3 E_3 + \lambda_1 N\|\boldsymbol{\theta}\|_2^2 + \lambda_2 \sum_{\ell=1}^N \|\mathbf{v}_\ell\|_2^2. \quad (3.49)$$

Note that the regularisation factor λ_1 , as well as the first term of E_3 are scaled by N . This is because all other expressions are sums with N elements; without the scaling factor, the impact of these terms would go to zero as the number of samples N increases.

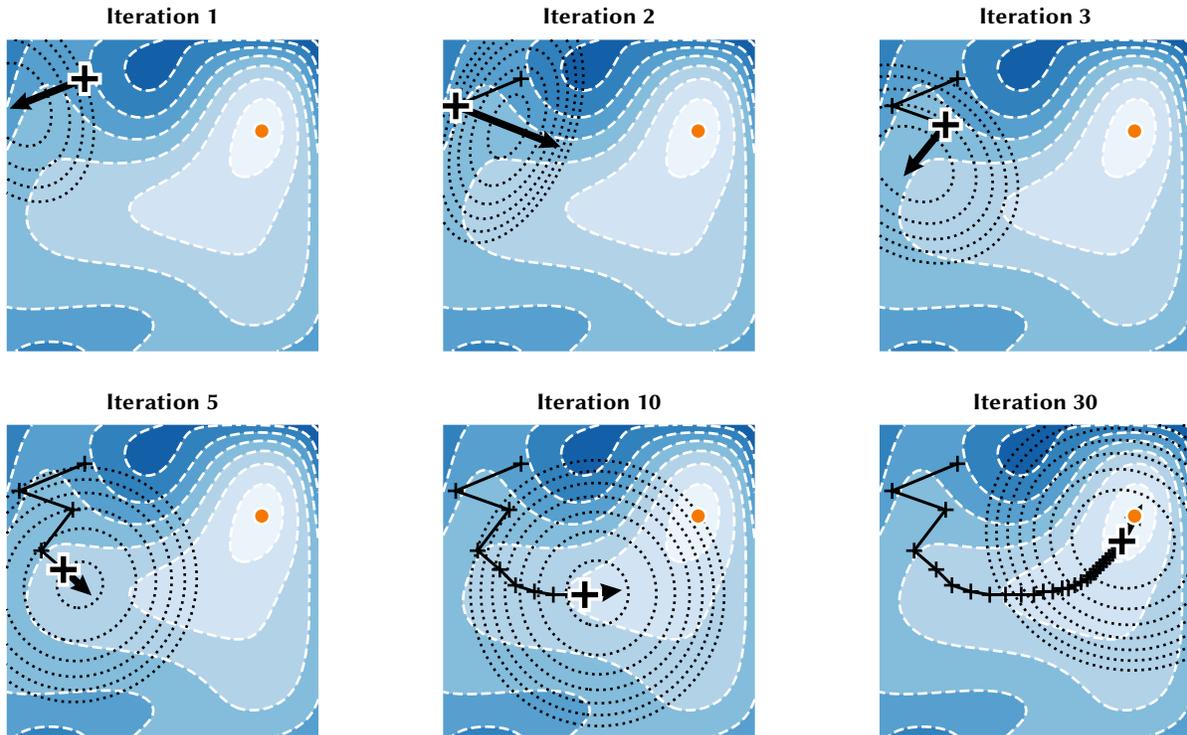


Figure 3.37: Illustration of gradient descent using a soft trust-region. In this example, we approximate a loss function E (coloured contours in the background; darker colours correspond to larger values) using a first-order Taylor approximation at \mathbf{x}_0 (black cross) with a soft trust-region term, i.e., $E' = (\langle \mathbf{x} - \mathbf{x}_0, \nabla E \rangle - y_{\min})^2 + \|\mathbf{x} - \mathbf{x}_0\|_2^2$ (dotted black contour lines). Repeatedly minimising E' over \mathbf{x} results in a gradient-descent-like algorithm. Note that convergence to the local minimum (orange circle) tends to be slow towards the end. Keep in mind that our soft trust-region optimisation algorithm is not purely based on a linear approximation of the loss function as is assumed for illustration purposes here.

With some patience, this problem, including nonnegativity constraints for $\tilde{\mathbf{a}}'$ and $\tilde{\mathbf{A}}'$, can be converted into a quadratic program with sparse \mathbf{P} and \mathbf{G} (cf. Definition 3.2).¹⁸ Alternatively, this problem can be solved using an NNLS solver, where $\tilde{\mathbf{b}}'$ and $\tilde{\mathbf{B}}'$ are allowed to be negative.

As mentioned several times above, to find a local optimum of our original loss function, eq. (3.49) must be treated as a sequential quadratic program. That is, the solution $\boldsymbol{\theta}$ from one iteration becomes the expansion point $\boldsymbol{\theta}_0$ for the next iteration. Importantly, the auxiliary variables $\mathbf{v}_{\ell,0} = \tilde{\mathbf{A}}[\mathbf{g}_\ell; \boldsymbol{\theta}_0]^{-1} \tilde{\mathbf{b}}[\boldsymbol{\theta}_0]$ must be recomputed before each iteration. We furthermore suggest increasing α_3 over time. In the limit, this trivially ensures convergence of the algorithm. The final algorithm is listed in Algorithm 3.1.

¹⁸Formally stating the quadratic program is rather tedious. The parameter matrices and vectors $\tilde{\mathbf{A}}'[\mathbf{g}_\ell; \boldsymbol{\theta}]$, $\tilde{\mathbf{a}}'[\mathbf{g}_\ell; \boldsymbol{\theta}]$, $\tilde{\mathbf{B}}'[\mathbf{g}_\ell; \boldsymbol{\theta}]$, $\tilde{\mathbf{b}}'[\mathbf{g}_\ell; \boldsymbol{\theta}]$ must be split into constant sub-expressions and linear terms dependant on $\boldsymbol{\theta}$. Readers interested in these technicalities are kindly directed at the *libnlif* source code (`nlif_solver_parameters.cpp`).

Algorithm 3.1: Soft trust-region parameter optimisation for n -LIF neurons. The hyperparameter selection is generally not critical; good starting values are $\alpha_1 = \alpha_2 = 1$, $\alpha_3 = 10^{-5}$, $\lambda_1 = \lambda_2 = 10^{-9}$. For $n_{\text{iter}} \geq 100$, the decay constant γ should be set to $\gamma \approx 0.99$, for $n_{\text{iter}} = 10$, $\gamma \approx 0.9$.

```

1  function softTrustRegionParameters( $\theta, \mathbf{g}_1, \dots, \mathbf{g}_N, J_1, \dots, J_N, \alpha_1, \alpha_2, \alpha_3, \lambda_1, \lambda_2, \gamma, n_{\text{epochs}}$ )
2  |   forall  $i \in \{1, \dots, n_{\text{epochs}}\}$  do
3  |   |    $\theta_0 \leftarrow \theta$ 
4  |   |   forall  $\ell \in \{1, \dots, N\}$  do
5  |   |   |    $\mathbf{v}_{\ell,0} \leftarrow -\tilde{\mathbf{A}}[\mathbf{g}_\ell; \theta]^{-1} \tilde{\mathbf{b}}[\mathbf{g}_\ell; \theta]$ 
6  |   |   |    $\theta \leftarrow \arg_{\theta} \min_{\theta, \mathbf{v}_1, \dots, \mathbf{v}_N} \left( \alpha_1 E_1(\mathbf{v}_1, \dots, \mathbf{v}_N, J_1, \dots, J_N) + \alpha_2 E_2''(\theta, \mathbf{v}_1, \dots, \mathbf{v}_N; \theta_0, \mathbf{v}_{1,0}, \dots, \mathbf{v}_{N,0}) \right.$ 
7  |   |   |    $\quad \left. + \alpha_3 \gamma^{-i} E_3(\theta, \mathbf{v}_1, \dots, \mathbf{v}_N; \theta_0, \mathbf{v}_{1,0}, \dots, \mathbf{v}_{N,0}) + \lambda_1 N \|\theta\|_2^2 + \lambda_2 \sum_{\ell=1}^N \|\mathbf{v}_\ell\|_2^2 \right)$ 
8  |   |   return  $\theta$ 

```

3.5.2 Assessing Optimiser Performance and Model Quality

We now perform a series of experiments to ensure that our soft trust-region approach is indeed working as intended; in particular, we analyse how our own optimisation scheme compares to general-purpose optimisers, and that what degree we are able to model multi-compartment neurons well using our techniques.

We consider n -LIF neurons with two, three, and four compartments; all as depicted in Figures 3.22C to 3.22E. Since, as per our discussion in Section 3.3.5, n -LIF neurons with the same number of compartments and input configuration in a single branch possess the same mathematical structure, our results should readily generalise to other configurations as well. To ensure that more distal compartments can influence the soma, we use larger coupling-conductances in neurons with more than two compartments. Specifically, we use $c_{12} = 50$ nS in the two-compartment LIF neuron, and $c_{12} = 100$ nS, $c_{23} = 200$ nS and $c_{34} = 500$ nS in the three- and four-compartment neurons. All other parameters are as specified in Table B.2.

Optimiser comparison. To compare different optimisation methods, we sample $N = 100$ super-threshold training and test samples \mathbf{g}_ℓ from $[0 \mu\text{S}, 1 \mu\text{S}]^k$. We determine the ground-truth somatic currents J_ℓ by simulating the neuron for one second, measuring the spike rate $a_\ell = \mathcal{G}(\mathbf{g}_\ell)$, and applying the inverse response curve G^{-1} as described in Section 3.4.2.

To test whether the algorithms converge to a solution even when starting from sub-optimal points in the parameter space, we compare two different initial parameter sets. First, the theoretical parameter estimate (e.g., Table 3.2), and second, uniformly sampled parameters θ_0 .

We use the hyperparameters listed in Algorithm 3.1 for our QP. From our experience, the specific values of these hyperparameters are not especially critical. The most important parameter is the soft trust-region weight $\alpha_3 = 10^{-5}$; for this parameter values between 10^{-4} and 10^{-6} seem to work well. As a stochastic gradient descent method, we use the ‘‘Adam’’ optimiser. Adam dynamically adapts the learning rate η of each parameter based on the first

and second gradient moments (i.e., mean and variance; Kingma and Ba, 2015). We use a batch size of $N_{\text{batch}} = 10$ and $\alpha = 0.05$ as a base learning-rate. This last parameter was hand-tuned for fastest possible convergence without instabilities. Finally, as a quasi-Newton method, we choose L-BFGS-B, a popular low-memory approximation of BFGS with support for bounded variables (Byrd et al., 1995). In particular, we use the L-BFGS-B reference implementation with default parameters, i.e., the rank of the Hessian approximation is $m = 10$.

Results. Loss curve statistics are depicted in Figure 3.38.¹⁹ When using the theoretical parameter estimate as an initialisation (cf. Figure 3.38A), all optimisers converge to similar errors. For the three- and four-compartment neuron models, our method reaches this seemingly optimal solution in two iterations, L-BFGS-B in about 30, and Adam after about 100 iterations. The optimal loss for the two-compartment neuron model is considerably lower, and convergence to this point is consistently slower among all tested methods.

When using randomly initialised parameters (cf. Figure 3.38B), our method requires about five iterations to reach the same minimum as in the previous experiment. L-BFGS-B converges to the optimal solution within 200 iterations. Adam fails to converge to a good solution in the case of the two- and three-compartment neurons even after 400 epochs.

Discussion. Our hand-tailored optimisation method outperforms general-purpose methods. This is likely because we solve a less nonlinear optimisation problem in a higher-dimensional space. Compared to quasi-Newton methods, our QP possesses an explicit sparse Hessian \mathbf{P} (cf. Definition 3.2), whereas L-BFGS-B must estimate the Hessian over several iterations.

The previously mentioned issues with our method—incorrectly splitting the loss function into two parts, haphazardly multiplying with $\tilde{\mathbf{A}}[\mathbf{g}_\ell; \boldsymbol{\theta}]$ inside the L_2 norm, and potentially bad convergence behaviour due to using a soft trust-region—seem to have no effect on the quality of the solution. In fact, the two gradient-based optimisation methods tend to converge to the same solution (in terms of the final NRMSE) despite them not relying on these simplifications.

Rate approximation errors. To gauge how well our calibrated surrogate dendritic nonlinearity model predicts somatic currents, we repeat the rate approximation experiment from Section 3.4.2. We simulate the individual neurons for one second with constant input \mathbf{g}_ℓ sampled from $[0 \mu\text{S}, 1 \mu\text{S}]^k$ and record the average spike rates. Out of $N = 1000$ samples, those with a spike rate above 12.5 s^{-1} are used to train our model, final parameters are given in Table B.7.²⁰ After calibration, we compute the true spike rates along several axis-aligned slices through the input space, and compare these data to the model prediction $G[H(\mathbf{g})]$.

¹⁹We count one iteration of our soft trust-region algorithm and L-BFGS-B as a single epoch, although, internally, both algorithms iteratively solve a convex optimisation problem in each iteration. The wall-clock time taken for each epoch is roughly the same for each algorithm. At $N = 100$ about 10 ms for our trust-region algorithm, 30 ms for L-BFGS-B, and 20 ms for Adam. All times scale about linearly with the number of samples N . However, these numbers are not very meaningful, since all three implementations are optimised to different degrees.

²⁰Our optimisation library *libnlf* optionally accounts for subthreshold relaxation in eq. (3.39) (using the same techniques as in Section 3.2.3). While this can slightly improve the optimised model, the improvements are minuscule compared to the pragmatic method of just discarding subthreshold samples.

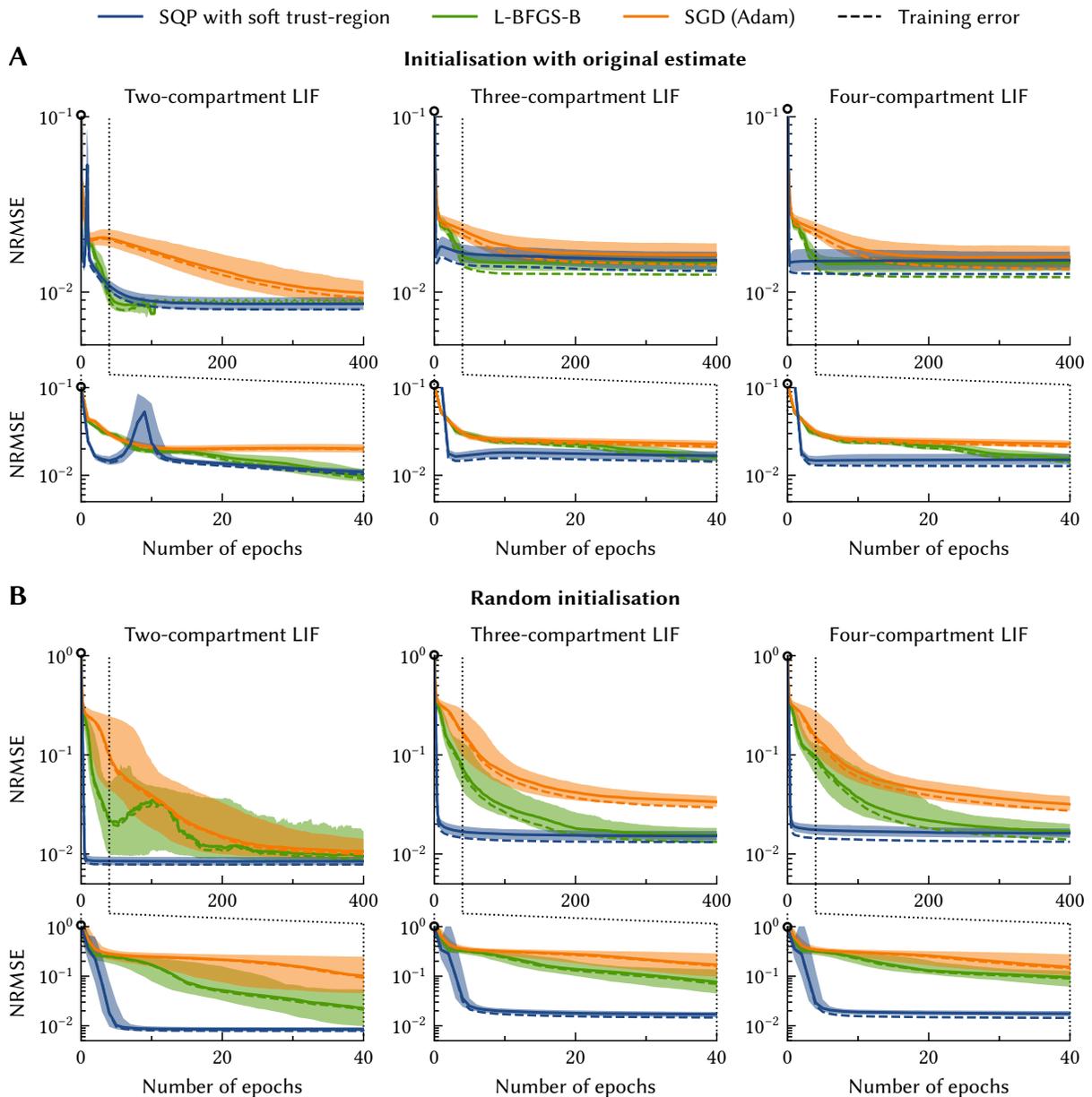


Figure 3.38: Comparison between stochastic gradient descent, the quasi-Newton method L-BFGS-B, and our soft trust-region based parameter estimation for three different neuron models (see text). Depicted is the NRMSE for the estimated somatic currents. Solid lines are the median validation errors over 1000 trials as described in the main text. Dashed lines depict the training error. Shaded areas correspond to the 25% and 75% percentiles. **(A)** Optimisation starting from the original parameter estimate θ as given in Table 3.2. Note that the data for L-BFGS-B ends after convergence to the local optimum. **(B)** Same for random initial parameters θ_0 . In both **(A)** and **(B)**, the soft trust-region algorithm on converges to a good solution after a few iterations. SGD and L-BFGS-B reach similar errors, but require more iterations and are less robust to noisy initialisation.

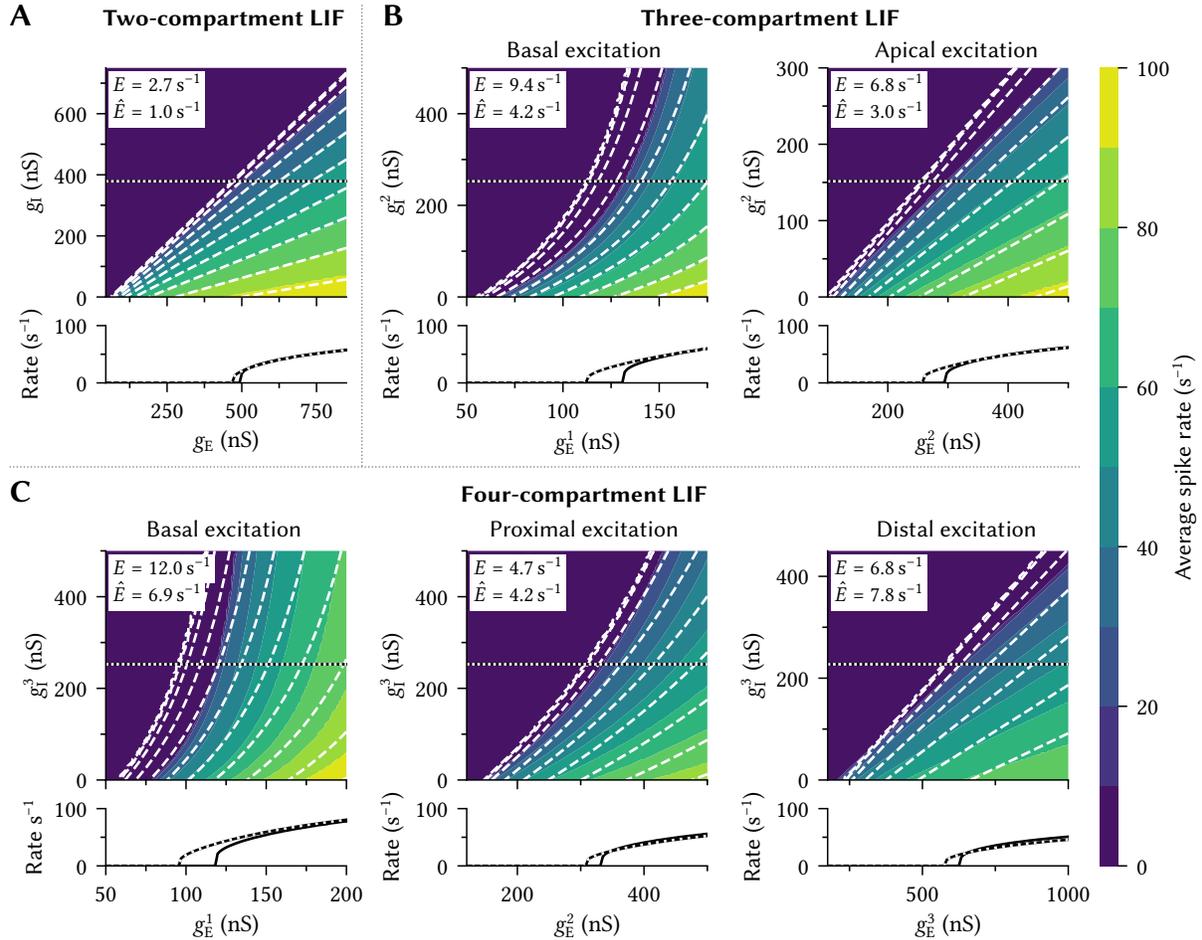


Figure 3.39: Calibrated n -LIF dendritic nonlinearity model for constant input. Coloured contour plots depict ground-truth average spike rates $\mathcal{G}(\mathbf{g})$. Dashed white lines are the model prediction $G[H(\mathbf{g})]$. Dotted lines indicate the cross-section location. E denotes the spike-rate RMSE over the entire grid, \hat{E} the error for super-threshold ground-truth samples (i.e., those with a rate above 12.5 s^{-1}).

Results. The estimated and ground-truth spike rates are depicted in Figure 3.39. Across different neuron types, the prediction matches the ground-truth data quite well—at least in the superthreshold regimes (the superthreshold error is denoted as \hat{E} in the figure). However, our model consistently fails to predict the spike-onset correctly, resulting in relatively larger overall RMSEs E . Still, calibration reduces errors by a factor of about two to four (cf. Figure B.5).

Discussion. Our surrogate model correctly predicts the overall shape of the response curve $\mathcal{G}(\mathbf{g})$. As we already discussed in the context of the two-compartment neuron, failure to predict the spike onset is a fundamental weakness of our approach; apparently this issue is even more pronounced when switching to more complex neuron models. It is unclear how exactly this issue could be fixed without further complicating the surrogate model; one option could be to optimise the one-dimensional response curve G .

3.5.3 Solving for n -LIF Synaptic Weights

In order to integrate n -LIF neurons into spiking neural networks, we must of course solve for synaptic weights. That is, given a set of pre-activities \mathbf{a}_ℓ and a set of target currents J_ℓ , we would like to find weights w_{ij} such that the somatic current predicted by our model is close to the desired target. In principle, this is very similar to estimating parameters for the somatic current model. We can thus mostly reuse the techniques we derived in Section 3.5.1.

Just as we discussed in Definition 3.3, we assume that each of the k input channels g_i is a linear combination of the pre-neuron activities \mathbf{a}_i influencing this channel, as well as the synaptic weights \mathbf{w}_i . For the sake of simplicity, we simply assume that $\mathbf{g} = \mathbf{W}\mathbf{a}$, where $\mathbf{a} \in \mathbb{R}^m$ is the vector of all pre-activities, and $\mathbf{W} \in \mathbb{R}^{k \times m}$ is a nonnegative sparse connection matrix.²¹ Entries in \mathbf{W} corresponding to invalid connections between pre- and post-neurons must be forced to zero when performing weight optimisation.

Let \mathcal{E} be the superthreshold error function (eq. 3.15), and \mathbf{L} , $\tilde{\mathbf{c}}$, $\tilde{\mathbf{a}}'$, $\tilde{\mathbf{A}}'$, $\tilde{\mathbf{b}}'$, $\tilde{\mathbf{B}}'$ describe the reduced and conditioned system matrices describing the neuron. Given N samples $(\mathbf{a}_\ell, J_\ell)$, we must minimise the following loss function with respect to the non-zero entries of \mathbf{W} :

$$\begin{aligned} E &= \sum_{\ell=1}^N \mathcal{E}(\langle \tilde{\mathbf{v}}^{\text{eq}}[\mathbf{W}\mathbf{a}_\ell], \tilde{\mathbf{c}} \rangle, J_\ell)^2 + \lambda N \|\mathbf{W}\|_{\text{F}}^2 \\ &= \sum_{\ell=1}^N \mathcal{E}(\langle (\tilde{\mathbf{L}} + \text{diag}(\tilde{\mathbf{a}}' + \tilde{\mathbf{A}}'\mathbf{W}\mathbf{a}_\ell))^{-1} (\tilde{\mathbf{b}}' + \tilde{\mathbf{B}}'\mathbf{W}\mathbf{a}_\ell), \tilde{\mathbf{c}} \rangle, J_\ell)^2 + \lambda N \|\mathbf{W}\|_{\text{F}}^2. \end{aligned} \quad (3.50)$$

Just as in Section 3.5.1, we explore two different approaches to solving this problem. First, using gradient-based methods, and second, by constructing a sequential quadratic program.

Weight gradient. To determine the loss function gradient, we again use the matrix inverse differential tensor (cf. eq. 3.40). Patiently applying the chain and product rule, the gradient of the loss function eq. (3.50) with respect to \mathbf{W} is (without the regularisation gradient $2\lambda N\mathbf{W}$):

$$\begin{aligned} \frac{\partial E}{\partial (\mathbf{W})_{rs}} &= -2 \sum_{\ell=1}^N \mathcal{E}(H(\mathbf{W}\mathbf{a}_\ell), J_\ell) \left(\sum_{i=1}^n \sum_{j=1}^n \left[(\tilde{\mathbf{A}}[\mathbf{W}\mathbf{a}_\ell]^{-1})_{ij} (\tilde{\mathbf{B}}')_{jr} (\mathbf{a}_\ell)_s (\tilde{\mathbf{c}})_i \right. \right. \\ &\quad \left. \left. + \sum_{k=1}^n (\tilde{\mathbf{A}}[\mathbf{W}\mathbf{a}_\ell]^{-1})_{ik} (\tilde{\mathbf{A}}[\mathbf{W}\mathbf{a}_\ell]^{-1})_{kj} (\tilde{\mathbf{A}}')_{kr} (\mathbf{a}_\ell)_s (\tilde{\mathbf{b}}[\mathbf{W}\mathbf{a}_\ell])_j (\tilde{\mathbf{c}})_i \right] \right). \end{aligned} \quad (3.51)$$

Note that the superthreshold error function $\mathcal{E}(J_{\text{dec}}, J_{\text{tar}})$ is a piecewise function with three separate cases; luckily, the above equation handles all three at the same time. In case $J_{\text{tar}} < J_{\text{th}}$ and $J_{\text{dec}} < J_{\text{th}}$, the output of \mathcal{E} is zero, and thus the gradient is zero as well. In the other two cases, \mathcal{E} is equal to J_{dec} with some offset. This does not affect the gradient.

²¹Of course, inputs to current-based channels can technically be negative; however, this is best remedied by providing both an excitatory and an inhibitory input channel, as we suggested in Figure 3.22A.

To enforce nonnegativity of \mathbf{W} , and thus nonsingularity of $\tilde{\mathbf{A}}[\mathbf{W}\mathbf{a}_\ell]$, weights should be clipped to zero after each gradient descent step. In the case of quasi-Newton methods, a bounded algorithm such as L-BFGS-B must be used.

Soft trust-region optimisation. We follow the same overall procedure as in Section 3.5.1 to derive a soft trust-region based SQP. Let $\boldsymbol{\theta}$ denote the non-zero weights in \mathbf{W} . Introducing N auxiliary variables $\mathbf{v}_\ell \in \mathbb{R}^n$ we obtain the following two-part optimisation problem:

$$E_1 = \sum_{\ell=1}^N \mathcal{E}(\langle \mathbf{v}_\ell, \tilde{\mathbf{c}} \rangle, J_\ell)^2, \quad E_2 = \sum_{\ell=1}^N \left\| (\tilde{\mathbf{L}} + \text{diag}(\tilde{\mathbf{a}}' + \tilde{\mathbf{A}}' \mathbf{W}[\boldsymbol{\theta}]\mathbf{a}_\ell)) \mathbf{v}_\ell - (\tilde{\mathbf{b}}' + \tilde{\mathbf{B}}' \mathbf{W}[\boldsymbol{\theta}]\mathbf{a}_\ell) \right\|_2^2.$$

Curiously, there is only one product term containing both $\boldsymbol{\theta}$ and \mathbf{v}_ℓ . This system is thus “more linear” than the parameter optimisation problem. Linearising E_2 at $(\boldsymbol{\theta}_0, \mathbf{v}_{1,0}, \dots, \mathbf{v}_{N,0})$ we get

$$E_2' = \sum_{\ell=1}^N \left\| \tilde{\mathbf{L}} \mathbf{v}_\ell - \tilde{\mathbf{b}}' - \tilde{\mathbf{B}}' \mathbf{W}[\boldsymbol{\theta}]\mathbf{a}_\ell + \tilde{\mathbf{a}}' \circ \mathbf{v}_\ell + \tilde{\mathbf{A}}' \mathbf{W}[\boldsymbol{\theta}]\mathbf{a}_\ell \circ \mathbf{v}_{\ell,0} + \tilde{\mathbf{A}}' \mathbf{W}[\boldsymbol{\theta}_0]\mathbf{a}_\ell \circ \mathbf{v}_\ell - \tilde{\mathbf{A}}' \mathbf{W}[\boldsymbol{\theta}_0]\mathbf{a}_\ell \circ \mathbf{v}_{\ell,0} \right\|_2^2.$$

This equation is in linear least-squares form over $\boldsymbol{\theta}$ and $\mathbf{v}_1, \dots, \mathbf{v}_N$. Combining this with a soft trust-region term E_3 , as well as regularisation factors λ_1, λ_2 , the final loss function E is

$$E = \alpha_1 E_1 + \alpha_2 E_2' + \alpha_3 E_3 + \lambda_1 N \|\boldsymbol{\theta}\|_2^2 + \lambda_2 \sum_{\ell=1}^N \|\mathbf{v}_\ell\|_2^2, \quad E_3 = N \|\boldsymbol{\theta} - \boldsymbol{\theta}_0\|_2^2 + \sum_{\ell=1}^N \|\mathbf{v}_\ell - \mathbf{v}_{\ell,0}\|. \quad (3.52)$$

This loss function, including a nonnegativity constraint on $\boldsymbol{\theta}$ and subthreshold relaxation can be expressed as a QP using the techniques discussed in Section 3.2.3.²² We obtain a sequential QP by minimising E and using the resulting $\boldsymbol{\theta}$ as $\boldsymbol{\theta}_0$ in a subsequent iteration (cf. Algorithm 3.2).

Unlike the parameter optimisation problem (Section 3.5.1), and just like neural networks in general (cf. Sutskever et al., 2013; He et al., 2015), this SQP is quite sensitive to the initial $\boldsymbol{\theta}$. In our experience, uniformly sampling $\boldsymbol{\theta}_0$ from $[0 \mu\text{S}, 1 \mu\text{S} m^{-1} a_{\max}^{-1}]$, where m is the number of pre-neurons and a_{\max} the maximum pre-population firing rate, works well in practice.

Alternatively, we suggest the fully deterministic initialisation scheme given in Algorithm 3.2. To find $\boldsymbol{\theta}_0$, we solve an auxiliary optimisation problem where we minimise eq. (3.52) for zero target currents $J_\ell = 0$ with zero initial weights. In other words, we solve for the input weights that “balance” the neuron to have zero somatic current for the given pre-activities \mathbf{a}_ℓ . The resulting $\boldsymbol{\theta}$ is then used as initial weights $\boldsymbol{\theta}_0$ for the main problem. Perhaps counter-intuitively, this $\boldsymbol{\theta}$ is non-zero—at least for multi-compartment neurons. This is because active input is required to cancel the non-somatic leak-currents.

²²While mathematically trivial, rearranging Equation (3.52) into a quadratic program including subthreshold relaxation results in a quite unwieldy expression. Please refer to the source code of *libnlif*, in particular `nlif_solver_weights.cpp`, for a software-implementation of the quadratic program.

Algorithm 3.2: Soft trust-region synaptic weight optimisation for n -LIF neurons and deterministic weight initialisation. Reasonable choices for the hyperparameters are $\alpha_1 = \alpha_2 = 1$, $\alpha_3 = 10^{-3}$, $\lambda_1 = 10^{-3}$, $\lambda'_1 = 10^{-1}$, $\lambda_2 = 10^{-6}$, $\gamma = 0.95$, $n_{\text{epochs}} = 50$.

```

1  function softTrustRegionWeights( $\mathbf{a}_1, \dots, \mathbf{a}_N, J_1, \dots, J_N, \alpha_1, \alpha_2, \alpha_3, \lambda_1, \lambda'_1, \lambda_2, \gamma, n_{\text{epochs}}$ )
2  |    $\boldsymbol{\theta} \leftarrow \arg_{\boldsymbol{\theta}, \mathbf{v}_1, \dots, \mathbf{v}_N} \left( \alpha_1 E_1(\mathbf{v}_1, \dots, \mathbf{v}_N, 0, \dots, 0) + \alpha_2 E'_2(\boldsymbol{\theta}, \mathbf{v}_1, \dots, \mathbf{v}_N; \mathbf{0}, \mathbf{v}_{1,0}, \dots, \mathbf{v}_{N,0}) \right.$ 
3  |   |    $\left. + \alpha_3 E_3(\boldsymbol{\theta}, \mathbf{v}_1, \dots, \mathbf{v}_N; \mathbf{0}, \mathbf{v}_{1,0}, \dots, \mathbf{v}_{N,0}) + \lambda'_1 N \|\boldsymbol{\theta}\|_2^2 + \lambda_2 \sum_{\ell=1}^N \|\mathbf{v}_{\ell}\|_2^2 \right)$ 
4  |   forall  $i \in \{1, \dots, n_{\text{epochs}}\}$  do
5  |   |    $\boldsymbol{\theta}_0 \leftarrow \boldsymbol{\theta}$ 
6  |   |   forall  $\ell \in \{1, \dots, N\}$  do
7  |   |   |    $\mathbf{v}_{\ell,0} \leftarrow -\tilde{\mathbf{A}}[\mathbf{W}[\boldsymbol{\theta}]\mathbf{a}_{\ell}]^{-1} \tilde{\mathbf{b}}[\mathbf{W}[\boldsymbol{\theta}]\mathbf{a}_{\ell}]$ 
8  |   |   |    $\boldsymbol{\theta} \leftarrow \arg_{\boldsymbol{\theta}, \mathbf{v}_1, \dots, \mathbf{v}_N} \left( \alpha_1 E_1(\mathbf{v}_1, \dots, \mathbf{v}_N, J_1, \dots, J_N) + \alpha_2 E'_2(\boldsymbol{\theta}, \mathbf{v}_1, \dots, \mathbf{v}_N; \boldsymbol{\theta}_0, \mathbf{v}_{1,0}, \dots, \mathbf{v}_{N,0}) \right.$ 
9  |   |   |    $\left. + \gamma^{-i} \alpha_3 E_3(\boldsymbol{\theta}, \mathbf{v}_1, \dots, \mathbf{v}_N; \boldsymbol{\theta}_0, \mathbf{v}_{1,0}, \dots, \mathbf{v}_{N,0}) + \lambda_1 N \|\boldsymbol{\theta}\|_2^2 + \lambda_2 \sum_{\ell=1}^N \|\mathbf{v}_{\ell}\|_2^2 \right)$ 
10  |   return  $\boldsymbol{\theta}$ 

```

Example: Four-quadrant multiplication. Figure 3.40 illustrates that the above method can indeed be used to solve for synaptic weights that approximate interesting somatic current functions. Specifically, three- and four-compartment neurons can approximate four-quadrant multiplication well, that is $x_1 x_2$ with $(x_1, x_2) \in [-1, 1]^2$. As a corollary, and in contrast to one- and two-compartment neurons with conductance-based synapses, these neurons can thus be used to solve the weak XOR problem (cf. Section 3.1 and Appendix A.1.7).²³

To obtain the results depicted in Figure 3.40, we use the soft trust-region optimiser with the parameters given in Algorithm 3.2. As dendritic nonlinearities we use the optimised parameters from Section 3.5.2 (see Table B.7). The variables x_1 and x_2 are represented in separate pre-populations with 100 neurons each, and each pre-neuron is connected to all target channels (cf. Figures 3.1C and 3.5B). Errors are based on 256 training samples (on a uniform 16×16 grid) and 10 000 test samples (on a 100×100 grid).

Notably, the NRMSE between the currents predicted by the surrogate dendritic nonlinearity model and our target function is between 4% and 6%. Of course, and as we explore in Section 3.5.5, this does not necessarily translate into final rate errors in a network context. We expect significantly larger errors due to the imprecisions in the surrogate model (cf. Section 3.5.2), and the more pronounced dynamics of multi-compartment neurons (cf. Section 3.3.2). Additionally, we use a relatively small regularisation factor of $\lambda_1 = 10^{-3}$ to achieve these results. Hence, while it is possible to compute multiplication, we expect noise to negatively impact the performance of a network relying on this kind of dendritic computation.

²³We technically compute $J(x_1, x_2) = \frac{1}{2}(x_1 x_2 + 1) \in [0, 1]$, that is a shifted and scaled version of four-quadrant multiplication—the conductance-based neuron types analysed here cannot compute. In practice, this affine transformation corresponds to approximating a current-translation function within an NEF ensemble (cf. Section 3.2.1).

3.5. Weight-Optimisation for Arbitrary Dendritic Trees

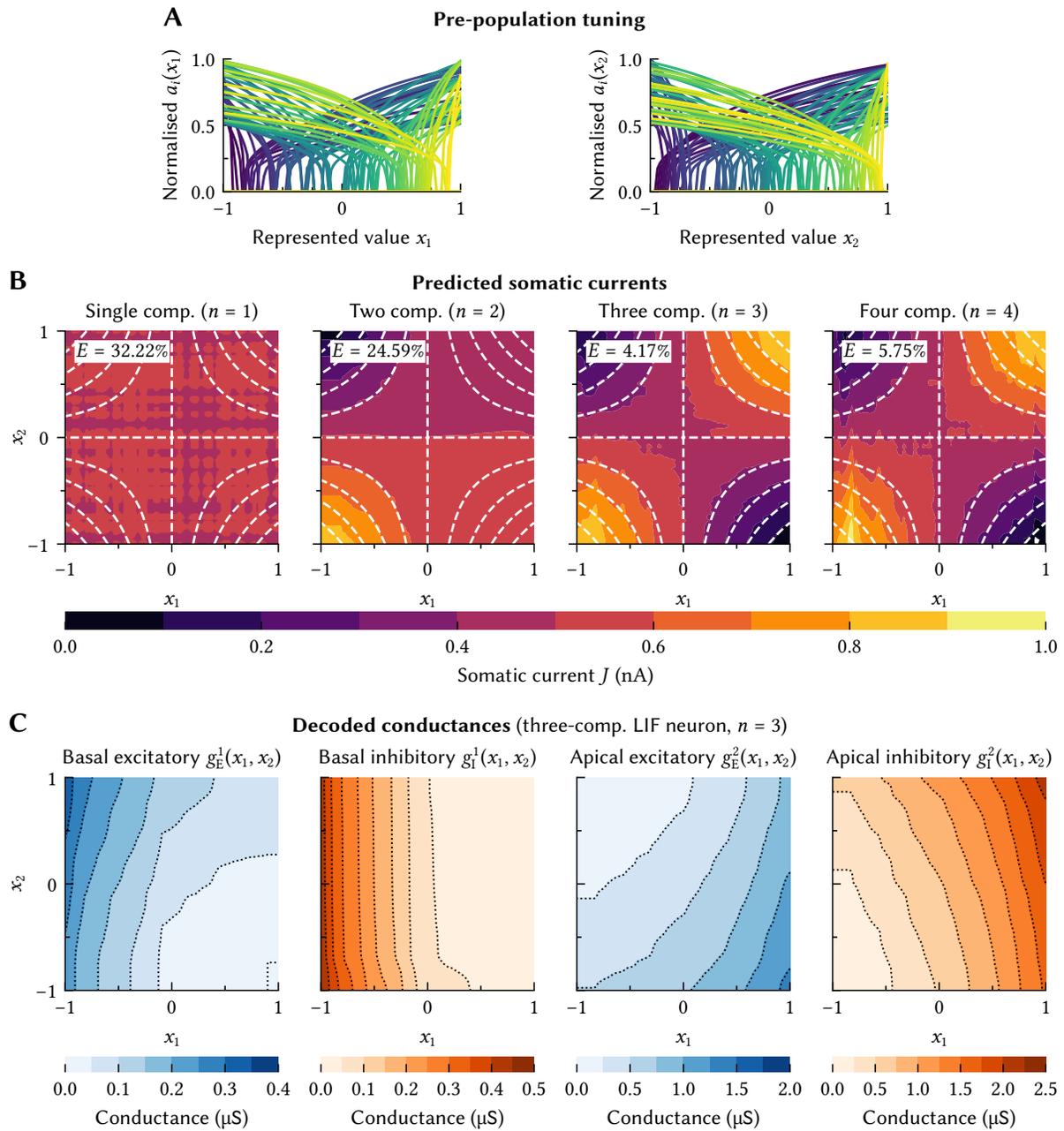


Figure 3.40: Approximating four-quadrant multiplication in different n -LIF neurons. **(A)** Tuning curves of two one-dimensional pre-populations with 100 neurons each. **(B)** Predicted somatic currents (coloured contours) after solving for synaptic weights approximating $J(x_1, x_2) = \frac{1}{2}(x_1 x_2 + 1)$ over $[-1, 1]^2$ (i.e., a scaled and shifted version of four-quadrant multiplication; dashed). Columns correspond to n -LIF neurons with one to four compartments. Error values E are the NRMSE. **(C)** Decoded conductance functions for the three-compartment neuron (cf. Figure 3.22D). Nonlinear interaction between input channels results in the final somatic currents approximating four-quadrant multiplication.

Weight optimisation benchmark. A systematic comparison between different synaptic weight optimisers is depicted in Figure 3.41. Loss curves are based on the same setup as in the previous example (cf. Figure 3.40); that is, two populations representing x_1, x_2 with 100 neurons each are connected to a single post-neuron. The post neuron possesses one of the dendritic nonlinearities from Section 3.5.2 (cf. Table B.7). We try to find synaptic weights \mathbf{W} that, taking the nonlinearity model into account, induce a target somatic current $J(x_1, x_2)$.

Specifically, we compare approximating a linear current function and the four-quadrant multiplication described above. Optimisation is based on 256 training samples, test errors on 10 000 samples. For our algorithm, we use the parameters given in Algorithm 3.2; for L-BFGS-B we use the default parameters ($m = 10$). For SGD with Adam we use the largest possible learning rate without major instabilities of $\alpha = 10^{-5}$, and a batch size of $N_{\text{batch}} = 10$. Note that we use a larger regularisation factor λ for the linear function; this increases the training error by several orders of magnitude, but, surprisingly, has little to no effect on the test error.²⁴

Results. For the linear function (Figure 3.41A), our SQP converges within five epochs. BFGS converges within 15 epochs, but to a larger NRMSE. In the case of the three- and four-compartment neuron, SGD converges to similar errors as our SQP within 50 epochs, but is more unstable.

Four-quadrant multiplication (Figure 3.41B) can only be approximated well in the three- and four-compartment neurons. Both our SQP and L-BFGS-B converge to similar losses within 25 and 100 epochs, respectively.²⁵ SGD with Adam outperforms the other optimisers in that it achieves about 30% smaller errors after 1000 epochs, yet, again, is slightly unstable.

Discussion. All presented optimisation methods are viable options for computing synaptic weights. SGD with Adam is by far the simplest method and often reaches the smallest errors. However, due to instabilities, SGD is only a viable option if some kind of early stopping with tracking of a validation error is implemented (e.g., Goodfellow et al., 2016, Section 7.1).

Given the vastly different implementations of the individual optimisers, it is hard to judge their performance in terms of wall-clock time. We cautiously claim that there is no appreciable difference between the approaches. At two seconds per epoch on a single processor, our SQP is three times slower than L-BFGS-B and SGD, but requires dramatically fewer epochs. However, SGD could be sped up significantly by optimising the gradient computation (eq. 3.51) and parallelising across samples, e.g., using TensorFlow (Abadi et al., 2016).

One issue with our SQP method is that the worst-case run-time per epoch is in $\mathcal{O}(N^3)$. This is because each internal QP solver iteration solves a linear system in the auxiliary variables $\mathbf{v}_1, \dots, \mathbf{v}_N$. In practice, we only observe a slight superlinear time-complexity, likely because the matrices defining the linear system (\mathbf{P} and \mathbf{G} ; cf. Definition 3.2) are extremely sparse.

²⁴The primary reason for doing this was to make sure that the training errors fit onto the diagram in Figure 3.41.

²⁵In a previous version of this experiment (not depicted) with fewer training samples (100 instead of 256), L-BFGS-B slightly outperformed both SGD and our SQP. It is unclear why the increase in training samples leads to a worse performance; intuitively, more samples should lead to a better estimate of the Hessian and not require a larger approximation rank m . In contrast, both our SQP and SGD benefited more training samples being available.

3.5. Weight-Optimisation for Arbitrary Dendritic Trees

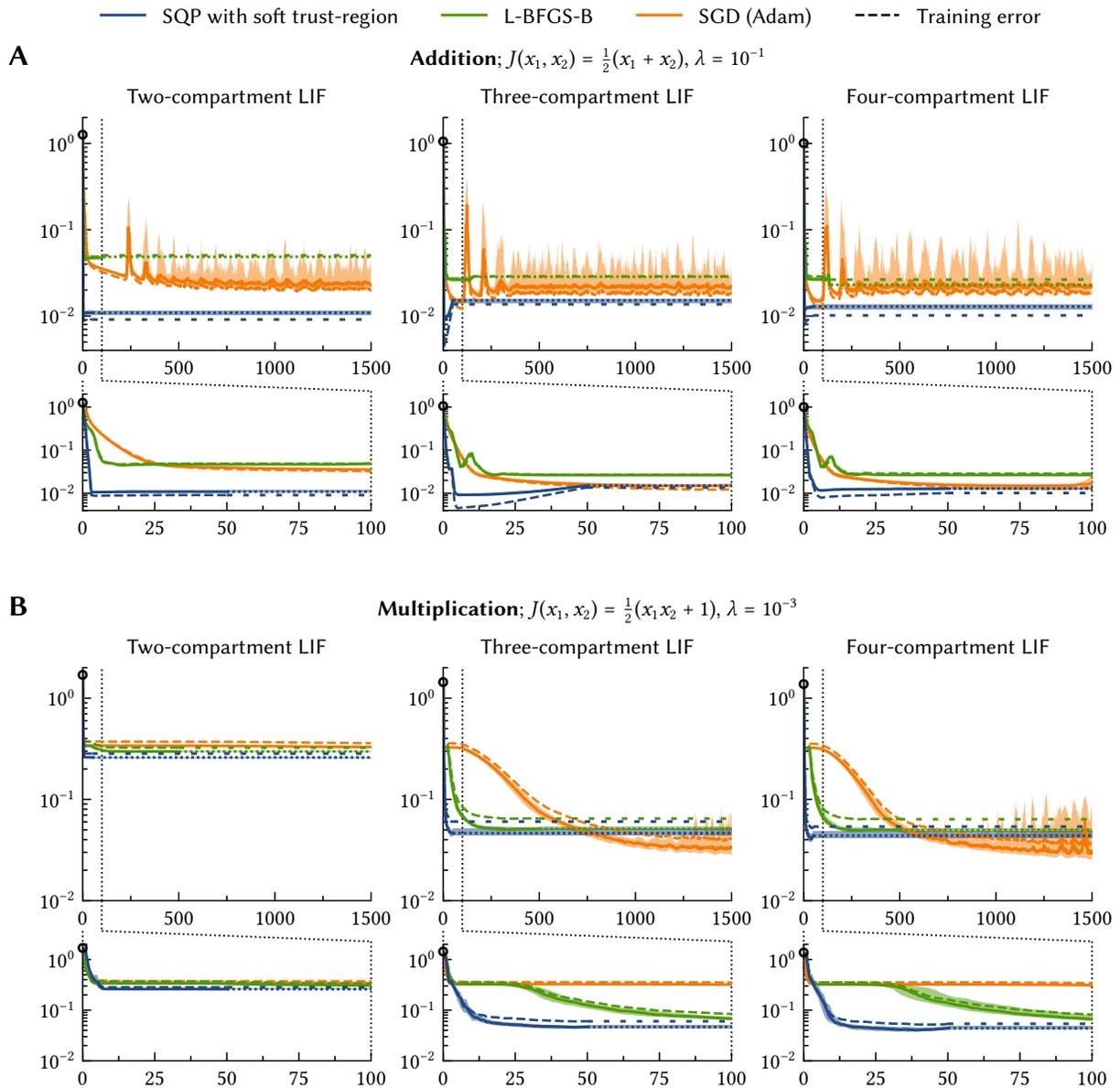


Figure 3.41: Comparing different optimisers performing synaptic weight optimisation. Solid lines are the median test error over 100 trials, shaded areas the 25 and 75 percentiles. The SQP algorithm is executed for 50 epochs, L-BFGS-B for at most 500 epochs or until convergence. The last value is continued as a dotted line. **(A)** Solving for a simple linear function. Note that we use a larger $\lambda = 10^{-1}$ to reduce overfitting. Our SQP algorithm converges to a good solution in about five iterations and typically outperforms L-BFGS-B and SGD. SGD is prone to oscillations. **(B)** Computing multiplication as illustrated in Figure 3.40. The two-compartment neuron cannot approximate four-quadrant multiplication well. Our SQP algorithm converges to a good solution after about 25 epochs and outperforms L-BFGS-B. SGD ultimately reaches solutions with slightly smaller errors, but produces a larger variance.

3.5.4 Computational Properties of the n -LIF Dendritic Nonlinearity Model

Our theoretical analysis from Section 3.3.5 suggests that adding more compartments and input-channels to n -LIF neurons should increase their computational power. Given that we are now able to fit n -LIF model parameters and to compute synaptic weights, the goal of this and the next section is to assess how far this theoretical advantage manifests itself in practice.

As with our analysis of two-compartment neurons, we proceed in two stages. First, we perform a static error analysis, where we sweep over two-dimensional current functions of varying complexity and test in how far these can be approximated using the surrogate dendritic nonlinearity model H (cf. Section 3.4.4). Of course, this assumes that H accurately describes somatic currents, which, as we saw in Section 3.5.2, is not always the case. In the next subsection, we compute network errors in a dynamic spiking neural network context.

Methods. We analyse the one- to four-compartment models with conductance-based synapses as fitted in Section 3.5.2 (cf. Table B.7). In each individual trial, we use the same setup as in the four-quadrant multiplication example. However, instead of multiplication, we approximate random two-dimensional current functions $J_\rho(x_1, x_2)$ with bandwidth ρ^{-1} (cf. Section 3.1.4). As in the two-compartment experiment, J_ρ is scaled to an RMS of 0.5 nA.

We sample 256 training samples from a 100×100 grid; the test error is computed over all 10 000 points with normal-distributed noise (standard deviation of $10^{-2} a_{\max}$) added to the pre-activities to test generalisation. We assume $J_{\text{th}} = 0$ nA for subthreshold relaxation, and use 30 epochs of our soft trust-region based SQP for weight optimisation. In contrast to Section 3.4.4, we use the same regularisation factor of $\lambda = 10^{-3}$ for all setups.

Results. Results are depicted in Figure 3.42. Notably, the performance of the three- and four-compartment neuron is not significantly different beyond $\rho^{-1} > 0.4$. Furthermore, the difference between the two- and three-compartment neuron is relatively subtle, with the largest difference at $\rho^{-1} \approx 1.5$ and a median reduction in error of 38%. Still, the three- and four-compartment neurons perform more similarly, if not slightly better than the multiplicative baseline established in Section 3.1.4. The two-compartment neuron is slightly worse than the multiplicative baseline, and features a much higher variance than the three- and four-compartment neurons.²⁶

Discussion. The computational advantage of three- and four-compartment neurons is not as clear-cut as is suggested by our theoretical analysis. While three- and four-compartment neurons can clearly compute multiplicative and not just merely divisive terms, this only results in a small average improvement in performance for random functions.

²⁶Note that the results for the one- and two-compartment neurons differ from those in Figure 3.34. The errors for the one-compartment are larger than those for LIF neuron in the previous experiment, while the errors for the two-compartment neuron are smaller. These discrepancies stem from using different regularisation factors, and, to some degree, from different neuron parameters. However, the results are qualitatively the same and, most importantly, results *within* each experiment are consistent.

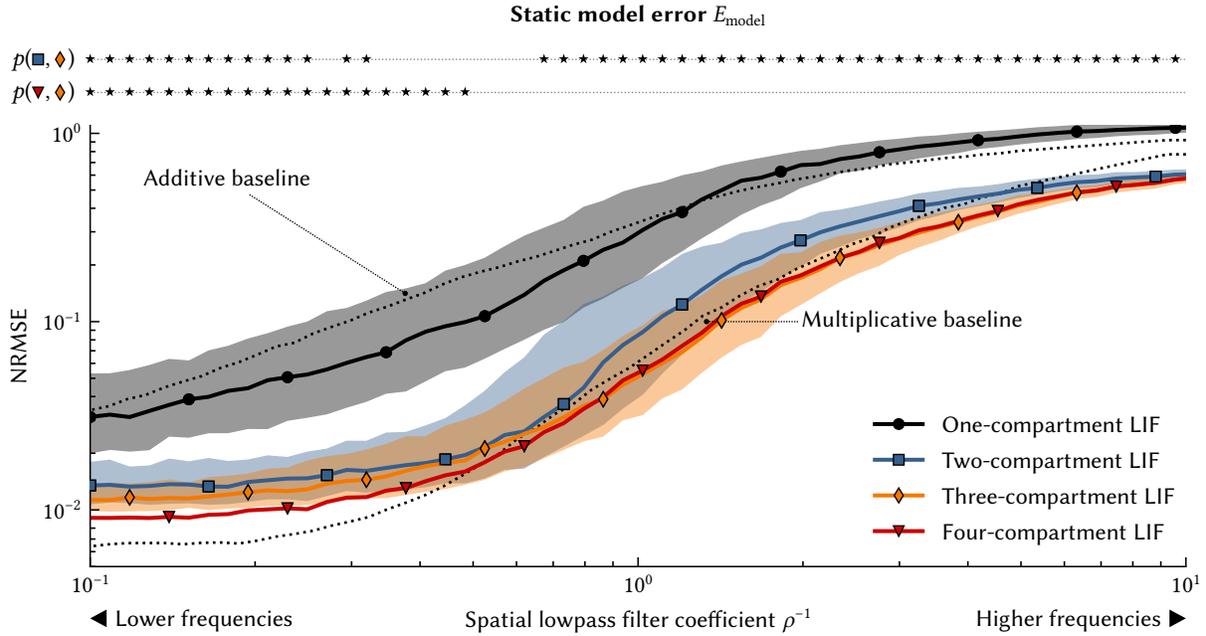


Figure 3.42: Multi-compartment current decoding error for random target functions. Same experiment as in Figure 3.34. *Bottom:* Static model error E_{model} for random current functions of “complexity” ρ^{-1} and different n -LIF dendritic nonlinearity models. The depicted NRMSE is based on the superthreshold error \mathcal{E} (cf. eq. 3.16). Lines are the median over 500 trials; shaded areas correspond to the 25th and 75th percentiles. Dotted lines are theoretical predictions from Figure 3.8B. *Top:* Rows labelled $p(\cdot, \cdot)$ depict the results of a per-point Kolmogorov–Smirnov test. Stars indicate p -values below 0.1%; that is, the two error distributions for the respective ρ^{-1} are not identical with a $p > 99.9\%$ probability. The three- and four-compartment neurons are not significantly different for $\rho^{-1} > 0.4$.

Since all pre-neurons connect to all input channels, the four-compartment neuron could, as per Theorem 3.7, compute cubic terms (i.e., x_1^3 and x_2^3). As is evident from the limited improvement in approximation errors, this does not seem to be helpful in practice.

This is likely due to the distal compartment requiring large coupling conductances. While tighter coupling is necessary to influence somatic currents, this also makes the neuron more linear, reducing its computational power (cf. Section 3.3.4). It might be possible to find more balanced configurations of the three- and four-compartment neurons that offer a larger computational benefit, but we were not able to do so in our experiments.²⁷

Another caveat with this experiment is that the weight solver is not guaranteed to converge to an optimal solution. In fact, we saw in the previous subsection that SGD sometimes converges to smaller errors than our SQP. Correspondingly, three- and four-compartment neurons may be more powerful than suggested here; a more thorough investigation is required in the future.

²⁷Judging from some preliminary experiments, an alternative configuration of the four-compartment neuron, where the third and fourth compartment are both connected to the second compartment, behaves similarly.

3.5.5 n -LIF Neurons in a Spiking Network Context

The previous experiment suggested that three- and four-compartment neurons indeed possess a small computational advantage over two-compartment neurons. The goal of this subsection is to quantify in how far this observation still holds in a spiking neural network context. To this end, we repeat the benchmark function experiment from Section 3.4.5.²⁸

One particular focus of our experiments is to test whether three-compartment neurons can approximate four-quadrant multiplication well, as is predicted by our theoretical considerations in Section 3.3.5. Indeed, as we discuss below, we find that three- and four-compartment neurons are better than two-compartment neurons at computing four-quadrant multiplication; however, they are clearly outperformed by two-layer *networks* in this regard, even when creating more favourable conditions for the multi-compartment neurons.

Experiment with standard parameters. Our experimental setup is largely identical to that discussed in Section 3.4.5. Apart from the standard LIF neurons with current-based synapses and a two-layer LIF reference network, we analyse the dendritic computation setup (Figure 3.1C) with the two-, three-, and four-compartment neurons analysed in Section 3.5.2 (cf. Table B.7). We account for Dale’s principle by marking 30% of the pre-neurons as inhibitory.

Where applicable, we approximate our benchmark functions f over the domain $[-1, 1]^2$. As a point of comparison to the previous experiment, we also provide some results for $(x_1, x_2) \in [0, 1]^2$.²⁹ Data are collected over a ten-second simulation during which the inputs $x_1(t)$, $x_2(t)$ follow a fourth order Hilbert curve. We report E_{net} , the NRMSE between the decoded network output over time and a reference signal passed through the same synaptic filters.

To solve for synaptic weights, we use our trust-region SQP solver; this includes all neuron types and network configurations. Since the time-to-convergence of our weight solver depends on the neuron type, we execute our algorithm for one epoch for LIF neurons, ten epochs for two-compartment LIF neurons, and thirty epochs for three- and four-compartment neurons. Hyperparameters are as in Algorithm 3.2; specifically, the regularisation factor is $\lambda = 10^{-3}$.

Results. As listed in the upper portion of Table 3.4 under “standard parameters”, the two-compartment and LIF neuron results are qualitatively similar to those from the previous experiment. Discrepancies stem from different regularisation factors and neuron parameters.

Notably, dendritic computation with multi-compartment neurons outperforms the two-layer network in most experiments. However, among our n -LIF neurons, baseline errors tend to increase with the number of compartments n . That is, errors for the three-compartment neuron are slightly higher than those for the two-compartment neuron, and so on.

²⁸Due to the high run-time requirements of our prototype-stage weight-solving routines, we decided not to repeat the network frequency sweep experiment (cf. Figure 3.36).

²⁹Remember that we transformed the functions in Section 3.4.5 such that their domain and co-domain were $[0, 1]^2$ and $[0, 1]$, respectively, while the quantities represented in the network were over $[-1, 1]^2$ and $[-1, 1]$. We apply the same transformation in this experiment when indicating that we use the domain $[0, 1]^2$.

3.5. Weight-Optimisation for Arbitrary Dendritic Trees

Table 3.4: Function approximation errors E_{net} for spiking networks using various n -LIF neurons. Results are the mean and standard deviation over 100 trials. The best result for a target function is set in bold; darker background colours indicate a worse ranking of the result. Functions with the domain $[0, 1]^2$ are exactly as in Section 3.4.5. All experiments are with subthreshold relaxation. Static decoding errors E_{model} and minimum network errors E_{net} are listed in Table B.8. See Table B.3 for the functions.

Function	Domain	Neuron				
		LIF		n -LIF		
		standard	two layers	$n = 2$	$n = 3$	$n = 4$
Standard parameters ($\lambda = 10^{-3}$; $\xi_0 \in [-0.95, 0.95]$; with Dale’s principle, $p_{\text{inh}} = 30\%$)						
$x_1 + x_2$	$[-1, 1]^2$	4.1 ± 0.2%	9.3 ± 0.5%	3.6 ± 0.2%	5.4 ± 0.6%	7.3 ± 1.2%
$x_1/(1 + x_2)$	$[0, 1]^2$	9.0 ± 0.7%	9.6 ± 0.8%	4.6 ± 0.5%	8.0 ± 1.9%	10.4 ± 2.7%
$\sqrt{x_1 \times x_2}$	$[0, 1]^2$	14.7 ± 1.4%	12.0 ± 1.0%	5.2 ± 0.6%	6.6 ± 1.0%	8.8 ± 1.1%
$x_1 \times x_2$	$[0, 1]^2$	17.4 ± 1.2%	8.5 ± 0.7%	4.3 ± 0.5%	5.1 ± 0.7%	7.4 ± 0.8%
$x_1 \times x_2$	$[-1, 1]^2$	102.7 ± 1.7%	13.1 ± 1.3%	79.6 ± 2.9%	38.1 ± 4.4%	36.3 ± 4.3%
$(x_1 \times x_2)^2$	$[-1, 1]^2$	17.4 ± 1.4%	19.6 ± 1.6%	18.0 ± 1.1%	13.4 ± 1.8%	13.4 ± 1.7%
$\ (x_1, x_2)\ $	$[-1, 1]^2$	10.9 ± 0.7%	14.4 ± 0.9%	7.1 ± 0.6%	7.3 ± 0.7%	11.4 ± 1.5%
$\text{atan}(x_1, x_2)$	$[-1, 1]^2$	39.9 ± 2.3%	36.9 ± 2.2%	23.9 ± 4.0%	23.9 ± 3.3%	24.5 ± 3.1%
$\max(x_1, x_2)$	$[-1, 1]^2$	24.5 ± 1.5%	9.4 ± 0.8%	8.7 ± 1.1%	8.2 ± 1.2%	8.9 ± 1.3%
Adapted parameters ($\lambda = 10^{-6}$; $\xi_0 \in [-0.95, 0]$; no Dale’s principle)						
$x_1 + x_2$	$[-1, 1]^2$	4.4 ± 0.2%	7.7 ± 0.4%	5.0 ± 0.4%	23.1 ± 1.2%	25.3 ± 1.2%
$x_1 \times x_2$	$[0, 1]^2$	23.4 ± 1.0%	10.2 ± 0.7%	5.4 ± 0.6%	11.0 ± 0.7%	11.3 ± 0.9%
$x_1 \times x_2$	$[-1, 1]^2$	103.4 ± 2.1%	11.4 ± 1.5%	71.3 ± 4.8%	20.0 ± 5.3%	18.1 ± 4.1%

A notable exception to this is four-quadrant multiplication. Here, the three- and four-compartment neurons clearly outperform the LIF and two-compartment LIF neurons. The standard LIF neuron reaches an NRMSE of 100%—the network outputs a constant zero, which indeed is the best linear approximation of multiplication over all four quadrants. Similarly, since the two-compartment LIF neuron cannot compute XOR, the error is at 79%. In contrast, the three- and four-compartment neurons reach errors of about 36%. Still, the two-layer network clearly outperforms all other setups with a 13% error.

Discussion. It is again worth pointing out that, for most functions, the errors achieved by the three- and four-compartment neuron are only *slightly* larger than those for the two-compartment neuron. While it may seem as if these neurons are thus less “powerful”, this is not the case.³⁰ As we saw in Section 3.5.2, and as follows from generally *lower* static decoding

³⁰The three- and four-compartment neuron are structural “supersets” of the two-compartment neuron. Correspondingly, the three- and four-compartment neurons *cannot* be less powerful than the two-compartment neuron (assuming that the coupling conductance c_{12} is the same, which is not the case in this specific experiment).

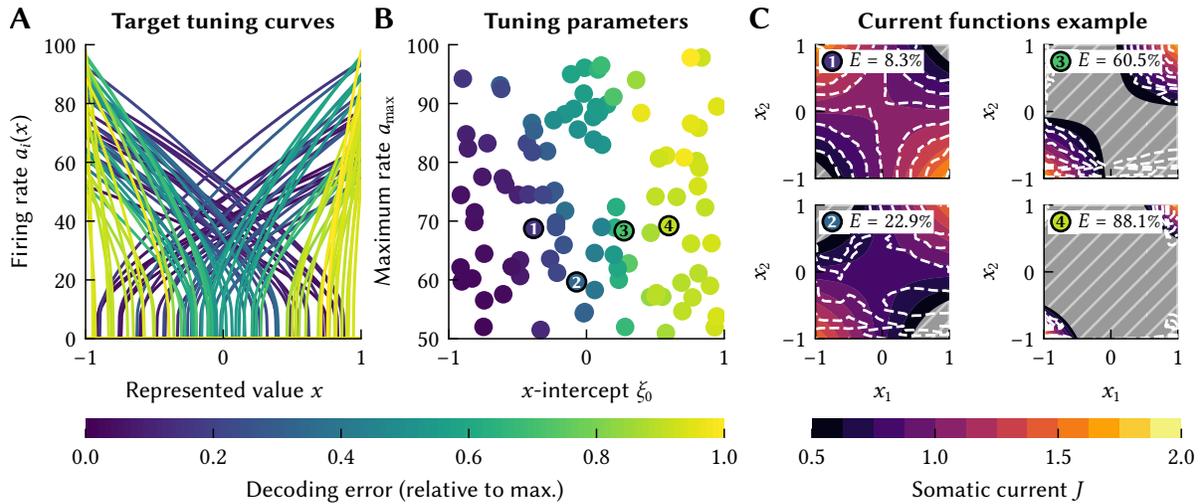


Figure 3.43: Positive intercepts induce large current decoding errors when computing four-quadrant multiplication. **(A)** Target population tuning curves coloured by the relative static decoding error. **(B)** Same data, but in terms of the x -intercepts and the maximum firing rate. Positive intercepts induce the largest errors. **(C)** Visualisation of the current function for the tuning curves highlighted in **(B)**. Coloured background corresponds to the target currents, dashed contour lines to the decoded somatic current, grey background corresponds to subthreshold currents. Errors E are the NRMSE.

errors (Table B.8), the increase in error compared to two-compartment neurons largely stems from the surrogate dendritic nonlinearity model being less accurate.

A more charitable interpretation is thus that the three- and four-compartment neurons reach errors similar to the two-compartment neuron for most functions, *despite* the surrogate model being inaccurate. At the same time, they can reach much smaller errors for complex functions such as four-quadrant multiplication. Hence, in principle, these neurons can approximate more functions well than two-compartment neurons; the methods discussed in this chapter just do not yet allow us to utilise these neurons to their fullest.

Improving four-quadrant multiplication performance. We approximated a four-quadrant multiplication current function with NRMSE below 5% in Section 3.5.3. This seems at odds with the observed error surpassing 30% in a network context, both in terms of the dynamic network error E_{net} (Table 3.4) and the static decoding error E_{model} (Table B.8).

The reason why we observe much higher errors in a network context is illustrated in Figure 3.43. The current functions $J_i(x_1, x_2) = \alpha_i \langle e_i, x_1 x_2 \rangle + \beta_i$ induced by target population tuning curves with negative x -intercepts ξ_0 (i.e., $\beta_i > 0$) can indeed be approximated well—this is what we saw in Figure 3.40B. However, the weight solver does not find good solutions for the current functions corresponding to tuning curves with positive intercepts.

It is unclear why this is the case. Potential culprits are the increased steepness of the current function, our random sampling not being dense enough to cover the relevant portions

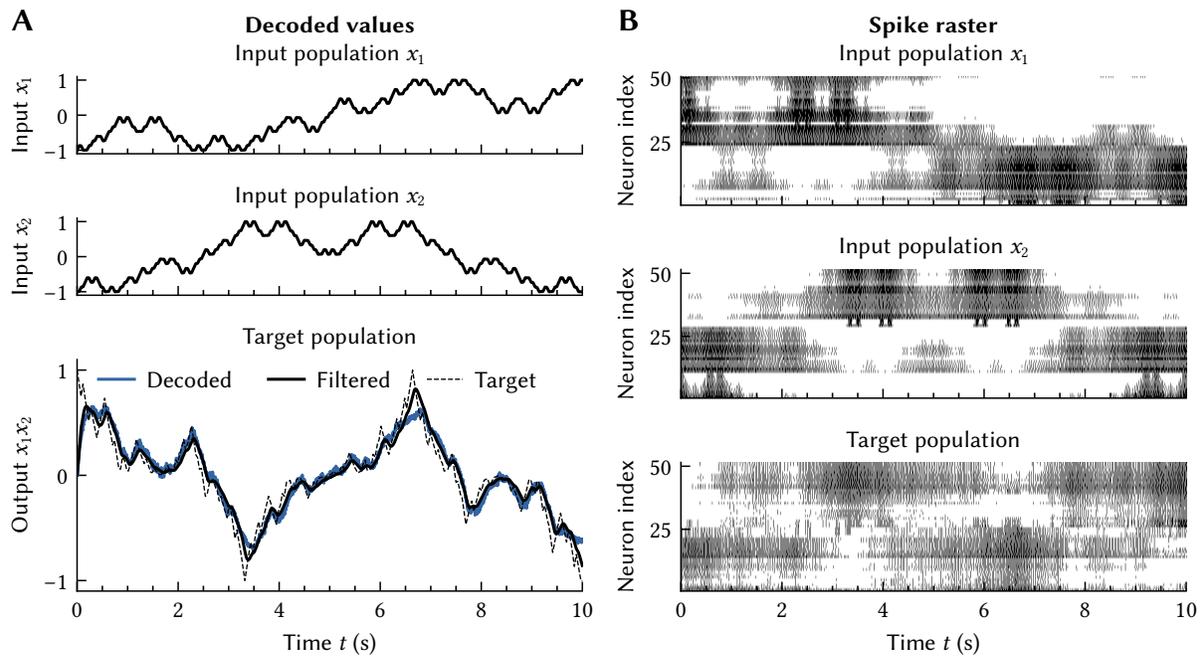


Figure 3.44: Computing four-quadrant multiplication in a spiking neural network using three-compartment LIF neurons. **(A)** *Top two plots:* Inputs $x_1(t)$ and $x_2(t)$ represented by the pre-populations. *Bottom:* Mathematical target $x_1(t)x_2(t)$, filtered target function, as well as the decoded target population output. This particular trial reaches a dynamic network error of $E_{\text{net}} = 16\%$. **(B)** Spike rasters depicting the activity of each population (only half of the neurons are depicted). All pre-neurons act both excitatorily and inhibitorily.

of $J_i(x_1, x_2)$, and systematic biases in the weight solver.³¹ Leaving this question open, a pragmatic solution to this problem is to simply change the tuning curves of the target population. However, this affects the class of functions that can be decoded from the target population.

The bottom portion of Table 3.4 labelled “adapted parameters” lists the results of an experiment where we tune the target-population x -intercepts ξ_0 to be strictly negative. As suggested by preliminary experiments, we furthermore reduce λ to 10^{-6} and do not account for Dale’s principle—this effectively increases the number of pre-neurons.

With these changes, we are able to reach a mean dynamic network error E_{net} of about 20%, albeit with high inter-trial standard deviations of $\pm 5\%$. Some trials reach errors below 15% (cf. Table B.8). Figure 3.44 depicts an example trial. Errors are mostly visible at the extrema, that is, the target population does not quite reach one and minus one. Overall three-compartment neurons seems to be able to, qualitatively speaking, compute four-quadrant multiplication quite well.

³¹We are able to reach smaller errors with both L-BFGS-B and Adam for these particular tuning curves. However, we found in preliminary experiments (data not shown) that using L-BFGS-B with 100 epochs to solve for synaptic weights tends to *increase* the overall network error E_{net} .

3.6 Summary

In this chapter, we discussed dendritic computation from the perspective of function approximation. Specifically, we assumed that neurons with dendritic structures can be modelled as multivariate nonlinearities $\sigma(\xi_1, \dots, \xi_k)$, where each ξ_i constitutes a separate input channel. While such multivariate functions cannot be used as universal approximators, we argued that they can sometimes outperform resource-constrained two-layer networks.

To test whether this hypothesis holds in functional spiking neural networks, we extended the Neural Engineering Framework (NEF) to account for additional constraints required to build networks with biologically plausible multi-channel neurons. Specifically, we presented techniques for solving for synaptic weights in current-space while complying with Dale’s principle. We further introduced “subthreshold relaxation”, an extension to the weight optimisation problem that exploits the rectifier nature of biological neurons to improve the superthreshold current-approximation accuracy. Finally, we discussed a general framework for incorporating multi-channel neurons into the NEF using a dendritic nonlinearity H .

We continued by describing the n -LIF family of multi-compartment neurons, a model of spiking neurons with passive dendrites. While it is not possible to state a closed form dendritic nonlinearity model H that characterises these neurons, we derived a parametrised surrogate model by harnessing the equilibrium state of the n -LIF dynamical system. We systematically characterised the computational properties of the surrogate model in terms of product terms and divisive interaction between input channels.

A particular focus of our studies was on the two-compartment LIF neuron, the simplest, non-trivial n -LIF model. This model features two conductance-based input channels with divisive interaction. Notably, we can solve for synaptic weights in two-compartment LIF neurons by minimising a loss function in rational least-squares form, which in turn can be approximated as a convex linear least-squares problem. Spiking neural networks with two-compartment neurons outperform two-layer networks for a variety of benchmark functions, as long as these functions do not implicitly require solving the XOR problem.

Solving for surrogate model parameters and synaptic weights in arbitrary n -LIF neurons is a hard nonconvex optimisation problem. We demonstrated that it is possible to exploit the structure of the surrogate dendritic nonlinearity by constructing a sequential quadratic program (SQP) that reaches relatively small losses after a few iterations and that often outperforms gradient-based methods such as L-BFGS-B and Adam.

The methods presented here can approximate XOR-like functions such as four-quadrant multiplication using n -LIF neurons with three or more compartments. Furthermore, these neurons can be integrated into spiking neural networks. Despite numerous simplifying assumptions in the derivation of our surrogate model, they typically outperform two-layer networks. However, the improvement in function approximation accuracy compared to two-compartment neurons is limited by the surrogate dendritic nonlinearity becoming less accurate the more compartments are in the neuron.

Limitations of our work. While the work presented in this chapter is a step towards a general model of dendritic computation in functional neurobiological modelling frameworks, it has several limitations. Most importantly, we treat the dendritic nonlinearity H as time-independent. Correspondingly, we implicitly assume that synaptic dynamics typically dominate the overall neuronal dynamics. However, dendritic trees in biology—especially when considering active channels and dendritic spikes (Koch, 1999; Koch, Mo, et al., 2002)—possess filter properties and adaptation processes that are not accounted for in our model. We discuss techniques that could—in the future—exploit these dynamics in the next chapter.

Another shortcoming of our surrogate model H is the assumption that the average somatic membrane potential is constant. While we accounted for this by calibrating the model parameters, this calibration is specific to the working-regime of the neuron. Deviations from the modelled behaviour are particularly apparent in situations with small output rates (cf. Figures 3.25, 3.29, 3.30 and 3.38). Correspondingly, our surrogate dendritic nonlinearity may not be a suitable model for brain areas featuring extremely low maximum firing rates. Given that optimising for weights in arbitrary n -LIF neurons is a “hopeless” nonconvex optimisation problem anyway, it would be interesting to consider the use of a less constrained statistical model as a dendritic nonlinearity H , such as a small neural network.

In light of these limitations, we would like to re-emphasize that, as stated in the introduction of this chapter, our goal was not to provide a detailed mechanistic model of dendritic computation. Instead, we hope to provide a useful tool that captures essential aspects of dendritic computation—a nonlinear interaction between input channels—while being computationally cheap and mathematically tractable, but still grounded in biophysics.

In particular, we saw that both the two- and three-compartment neuron improve the multivariate function approximation performance of a single neuron population, while using more than three compartments resulted in diminishing returns. From a purely functional perspective, there thus seems to be no reason to use more complex models of *passive* dendritic computation beyond two and three-compartment neurons.

Future work. A potential application of our work outside neurobiological modelling is programming neuromorphic hardware (Section 2.3.2). Especially when considering mixed analogue-digital neuromorphic hardware, it should be possible to achieve a higher energy efficiency by implementing two- or three-compartment neuron models and performing local analog computation. Potential future work would be to validate our methods on a neuromorphic platform that implements dendritic trees, such as the *BrainScales 2* system (Schemmel, Kriener, et al., 2017), or analogue circuit emulators such as FPAAAs (e.g., George et al., 2016).

Other open threads that warrant further investigation include ruling out that the diminishing returns for three- and four-compartment neurons are not just due to our weight solver converging to suboptimal solutions compared to other solvers. It could also be interesting to find ways to improve the prediction of the x -intercept in our surrogate dendritic nonlinearity, and to further strengthen the mathematical foundation of the material in this chapter.

Chapter 4

Temporal Tuning

Complex effects, such as representing visual motion, are the outcome of the dynamics of neural networks. This means that while network properties are dependent on the properties of the neurons in the network, they are nevertheless not identical to cellular properties, nor to *simple* combinations of cellular properties. Interactions of neurons in networks is required for complex effects, but it is dynamical, not a simple wind-up doll affair.

— PARTICIA S. CHURCHLAND AND TERRENCE J. SEJNOWSKI
The Computational Brain (1992)

Material in this chapter from prior publications

Some of the material in this chapter has previously been published in the form of two technical reports, namely Stöckel (2021a), where we derive feedback matrices for polynomial temporal basis functions, and Stöckel (2021b), where we systematically compare different temporal function bases.

Contributions in this chapter

This chapter has in part been inspired by Volker and Eliasmith’s work on the Legendre Delay Network (LDN) and integrating higher-order synapses into the NEF (Voelker and Eliasmith, 2018; Voelker, 2019), as well as Chilkuri and Eliasmith’s effort on efficiently training deep neural networks that utilise the Legendre Delay Network (Chilkuri and Eliasmith, 2021). Novel work in this chapter includes generalising the third NEF principle to take temporal tuning into account, as well as demonstrating that it is possible to build networks that fulfil desired temporal tuning in networks with heterogeneous synaptic filters by solving a linear least-squares optimisation problem. Furthermore, we formalise the notion of LTI systems approximating sliding-window transformations and suggest the “information erasure” method for establishing a rectangle window. We use these ideas for an alternative derivation of the LDN LTI system from first principles and propose a similar LTI system that generates a “modified Fourier basis”. We show that this modified Fourier basis often outperforms the LDN LTI and has more favourable computational properties under Runge-Kutta integration.

In the previous chapter, we exploited the dynamics of multi-compartment n -LIF neurons to compute a larger class of functions than previously possible with a single layer of LIF neurons. However, we primarily focused on the equilibrium state of the n -LIF system. In other words, our dendritic nonlinearity H did not expose the underlying dynamics to the network-level. This can be problematic for two reasons. First, when approximating dynamical systems using NEF principle three (Section 2.3.5), not compensating for intrinsic neural dynamics may result in deviations from the desired outcome (cf. Section 2.3.6). Second, the intrinsic dynamics could in theory help establish some desired dynamics; they may be viewed as a temporal resource that can be *harnessed* for computation, and not just *compensated* for.

The goal of this chapter is to extend the NEF to more readily take a variety of temporal resources into account. While we still focus on synaptic filters as a primary source of neural dynamics, the same approach can in principle accommodate intrinsic neural dynamics as well. More precisely, the central hypothesis of this chapter—and, if you will, a theoretical neuroscience *theory*—is that temporal resources in biological neural networks give rise to the *temporal tuning* of individual neurons. These underlying temporal resources include synaptic filters, intrinsic neural dynamics, transmission delays, and the temporal tuning of neighbouring neurons. Such resources then establish diverse temporal tuning curves, and are recombined by synaptic weights to ultimately support high-level functions such as interpreting visual motion, forming working memory, or generating motor trajectories. Mathematically speaking, we can decode higher-order functions f from a temporally tuned neuron population that not only depends on the current represented value $\mathbf{x} \in \mathbb{R}^d$, but on a *signal* $\mathfrak{x}(t)$ (cf. Table 1.1).

Notably, our temporal tuning curve approach naturally generalises the NEF dynamics principle (Eliasmith and Anderson, 2003, Chapter 8). That is, using temporal tuning curves, we can realise any dynamical system that could be realised using the dynamics principle. At the same time, temporal tuning curves facilitate taking the temporal properties of pre-populations and intermediate filters into account, including heterogeneous and higher-order synapses.

However, the most important contribution of our temporal tuning curve approach is a shift in perspective. Just as we harness non-temporal tuning curves and neural nonlinearities to approximate spatial functions in neural networks, we are now able to harness the temporal tuning of pre-populations and intermediate filters to realise the temporal tuning of post-populations. Perhaps surprisingly, this includes recurrent neuron populations. We furthermore separate *temporal* from *spatial* tuning. A neuron can be tuned to a d -dimensional spatial quantity, and, at the same time, possess some temporal tuning, typically of finite order q .

Crucially, this allows modellers to directly specify some desired neural behaviour. For example, a modeller could choose to assign time-cell like tuning (Pastalkova et al., 2008; Tiganj et al., 2016) to one population, and visual spatiotemporal receptive fields (Carandini et al., 1999) to another. In theory, these tuning properties can be provided in the form of sampled data. So far, and as criticised by Nicola et al. (2017), this was more difficult to accomplish using the NEF dynamics principle—while the representation and transformation principle facilitate incorporating empirical data, the dynamics principle typically relies on modellers providing

closed-form equations (although partially data-driven dynamics were possible, e.g., the handwriting generation in SPAUN; cf. Eliasmith, Stewart, et al., 2012; Choo, 2018, Section 3.2.2).

Almost more strikingly, temporal tuning is not just useful for modelling neurobiological systems. Introducing the right kind of dynamics into *artificial* neural networks results in architectures that outperform successful approaches in machine learning by wide margins. Specifically, we show that a layer of Legendre Memory Units (LMUs; Voelker, Kajić, et al., 2019) corresponds to a spatiotemporally tuned NEF population.

Structure. We start Section 4.1 with a review of “temporal tuning” in biology, and extend the NEF by introducing the concept of a “temporal tuning curve”. Specifically, we propose a linear least-squares optimisation problem that realises the desired temporal tuning by taking synaptic filters and the pre-population tuning into account. Under ideal conditions, temporal tuning curves can be used to construct recurrent neural networks that realise LTI systems while compensating for heterogeneous and higher-order synaptic filters.

Given that we can tune neurons to arbitrary LTI systems, we ask in Section 4.2 what LTI system we should optimally implement to obtain a maximally diverse set of temporal tuning curves. This leads us to the concept of LTI systems approximating sliding-window spectra. We introduce a simple technique for constructing such LTI systems and re-derive the Legendre Delay Network (LDN) using this approach. Furthermore, we discover a “modified Fourier system” as a potential alternative to the LDN.

In Section 4.3, we demonstrate that our temporal tuning curve approach still works in less ideal spiking neural networks. We show that, just like the LDN, our modified Fourier system generates “time cells” when mapped onto a spiking neural network. Furthermore, we use spatiotemporal encoding matrices to generate multi-dimensional spatiotemporal tuning and demonstrate decoding nonlinear functions over both space and time.

Finally, in Section 4.4, we draw parallels between LMU layers and our aforementioned NEF populations with spatiotemporal tuning. We show that the choice of the sliding window-spectrum is secondary to the performance of the LMU and demonstrate that convolving input signals with the modified Fourier system can offer a significant improvement in terms of asymptotic time-complexity compared to the LDN. We close with a discussion in Section 4.5.

Notation

We sometimes use Fraktur letters (e.g., \mathfrak{a} , \mathfrak{b} , ...) to convey that the mathematical objects we are referring to are now *functions* over time. That is, we write a temporal encoder as \mathfrak{e} instead of e . Note that $\mathfrak{e}(t)$ refers to the result of evaluating the function \mathfrak{e} at a specific point in time t , whereas \mathfrak{e} refers to the function object as a whole. In other words, \mathfrak{e} is a map $\mathbb{R} \rightarrow \mathbb{R}^d$, whereas $\mathfrak{e}(t)$ is an element in \mathbb{R}^d . This distinction is important when expressing operations on functions, such as convolution. We furthermore assume in most of our definitions that $t = 0$ corresponds to the current point in time.

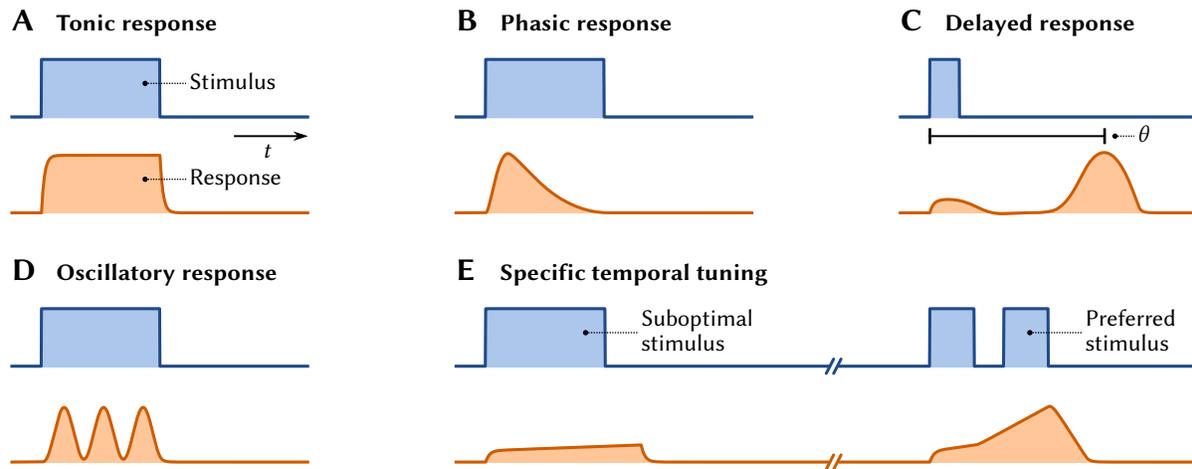


Figure 4.1: Illustration of some possible *in situ* temporal response patterns. **(A)** Neuron without pronounced temporal response. Pure tonic responses are relatively rare (e.g., Ruffini mechanoreceptors, cf. Kandel et al., 2012, Figures 22-5 & 23-9). **(B)** Neuron with a phasic response to a stimulus. Most sensory neurons are at least partially phasic (cf. Kandel et al., 2012, Part V). **(C)** Neuron which reaches its maximum firing rate θ seconds after the stimulus onset (e.g., time-cells, see below). **(D)** Neuron which oscillates rhythmically in response to a stimulus (e.g., Friedman-Hill, Maldonado, and Gray, 2000). **(E)** Neuron tuned to a specific temporal input pattern (see below).

4.1 Temporal Tuning Curves

Some neurons in the brain exhibit pronounced temporal activity patterns. For example, as is illustrated in Figure 4.1, their activities may extend beyond the stimulus period, cease before the stimulus has ended, repeat rhythmically, be tuned to specific spatiotemporal patterns, or any combination thereof. Theoretical work suggests that diverse temporal responses form a representation of the past stimulus history (Grossberg and Schmajuk, 1989; Howard et al., 2014; Voelker and Eliasmith, 2018) and could, for example, support delay learning in the cerebellum (Fujita, 1982; Medina and Mauk, 2000; see also Chapter 5). While spatiotemporal neural responses have originally been extensively studied in visual cortex (e.g., DeAngelis, Ohzawa, and Freeman, 1993; Carandini et al., 1999), more recent evidence indicates that various brain areas—including hippocampus, striatum, and cerebellum—possess neurons with pronounced temporal responses (e.g., Lusk et al., 2016).

In many cases, it is uncertain how these temporal response patterns come to be; that is, whether the temporal responses are a result of intrinsic neural properties, network-level effects, or a combination of both. In this section, we build upon a linear filter model of temporal tuning (Watson and Ahumada, 1983; Adelson and Bergen, 1985) that is agnostic to these questions and that can be easily integrated into the NEF as a generalisation of the dynamics principle. We first review some types of temporal response patterns observed in biology, describe our extension to the NEF, and then discuss examples of how this theory can be used in practice.

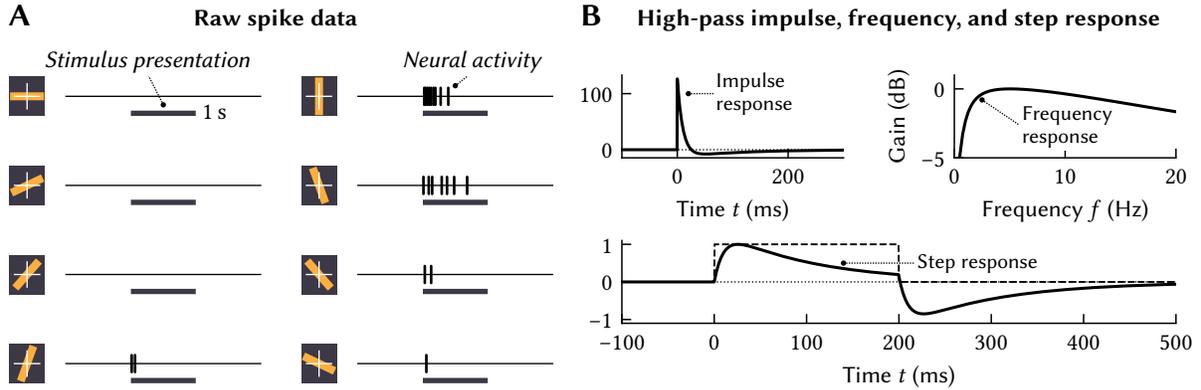


Figure 4.2: Temporal properties of cells in visual cortex. **(A)** Spike times from the Hubel and Wiesel experiment (same data as in Figure 2.19). The neuron is slightly phasic. Orange bar corresponds to the stimulus orientation. Adapted from Hubel and Wiesel (1959, Figure 3). **(B)** Illustration of a second-order high-pass filter. The step-response of such a filter (*bottom*) could model the current flowing into the neuron, and thus the decaying activity.

4.1.1 Time Cells and Temporal Tuning

Looking back at the data from the original experiment by Hubel and Wiesel (1959; Figure 4.2A), as well as the temporal properties of other cortical sensory neurons (cf. Kandel et al., 2012, Part V), we find that these neurons tend to be *phasic* (cf. Figure 4.1B); that is, the observed spike rate decays after the stimulus onset. Put differently, the neuron *adapts* to the stimulus.

Modelling phasic activity using high-pass filters. As is depicted in Figure 4.2B, this activity profile could in principle be modelled by applying a high-pass filter to the neuron’s somatic current. For example, the impulse response $h(t)$ of a second-order high-pass filter can be constructed by subtracting two exponential low-pass filters with time-constants τ_1 , τ_2 :

$$h(t) = \frac{1}{\tau_1} \exp\left(-\frac{t}{\tau_1}\right) - \frac{\alpha}{\tau_2} \exp\left(-\frac{t}{\tau_2}\right) \quad \text{for } t \geq 0, \text{ where } \alpha, \tau_1, \tau_2 > 0, \text{ and } \alpha\tau_1 < \tau_2. \quad (4.1)$$

The idea behind “temporal tuning curves” (formalised below) is to treat $h(t)$ as a part of the neuron’s tuning properties. That is, we replace the encoder, or “preferred stimulus”, \mathbf{e} with a temporal encoder $\mathbf{e}(t) = \mathbf{e} \cdot h(t)$, resulting in a “temporal preferred stimulus”.

Notably, in doing this, we do not subscribe to any specific mechanism that produces the high-pass characteristics. In fact, this response pattern could be the result of intrinsic neural or synaptic dynamics, or instead stem from network-level effects. As we discussed in Section 2.2.4, this is one of the central ideas of a tuning curve—we characterise the behaviour of a neuron responding to an external stimulus *in situ*, capturing both the computation performed by the network leading up to an individual neuron and the intrinsic properties of the neuron itself. In the context of the Neural Engineering Framework, we would then use available neural resources to systematically realise these desired tuning properties.

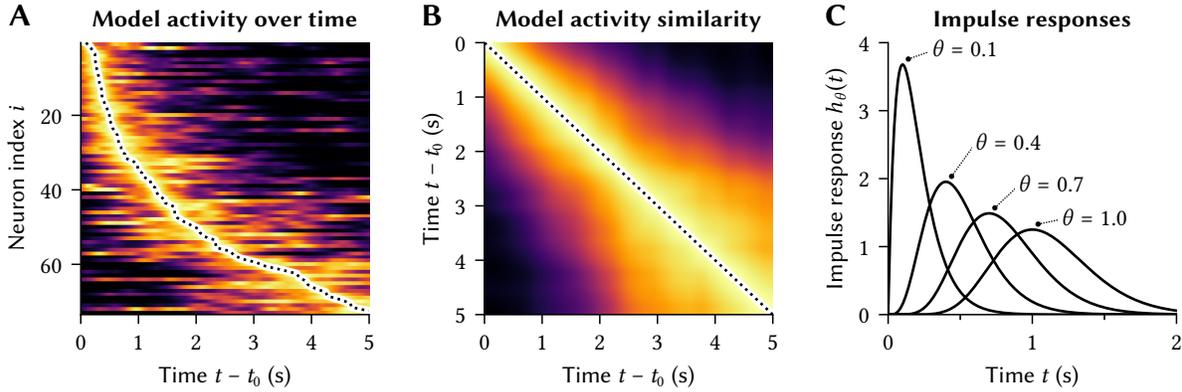


Figure 4.3: Illustration of time cells. The depicted simulation results (using the technique described by Howard et al., 2014) resemble the empirical data presented in Tiganj et al. (2016). Lighter colours correspond to larger values. **(A)** Normalised firing rate of 73 simulated time cells with $\theta \in [0 \text{ s}, 5 \text{ s}]$. Cells are sorted by the observed $\hat{\theta}$ (dotted line). Data obtained by feeding the impulse responses $h_\theta(t)$ depicted in **(C)** into an ensemble of LIF neurons with noisy somatic currents. **(B)** Cosine-similarity between the activity vectors for each time-pair. The similarities spread out for larger t , making it harder to decode t . **(C)** Impulse responses proposed by Howard et al. (2014) for $\tau = 0.1 \text{ s}$.

Time cells. A more interesting example of temporal response patterns are so-called “time cells”, first described by Pastalkova et al. (2008) in rat hippocampus. As is illustrated in Figure 4.1C, time cells reach their peak activity a certain time θ after the stimulus onset at t_0 .¹ This delay θ is specific to each cell and can be in the millisecond to seconds range. Populations of time cells encode the time since the stimulus onset $t - t_0$, or, assuming continuous input signals, a compressed version of the stimulus history. We discuss this in more detail below in Section 4.2.

Time cells have also been described in other brain regions, including medial prefrontal cortex (Tiganj et al., 2016), striatum, and the cerebellum (see Lusk et al., 2016 for a review). Qualitatively, the normalised neural activities resemble the data depicted in Figure 4.3A. As time progresses, the population activities tend to become more similar (Figure 4.3B).

Howard et al. (2014) propose the following impulse response $h_\theta(t)$ (cf. Figure 4.3C) that can be approximated as a weighted sum of first-order low-pass filters:

$$h_\theta(t) = \alpha_\theta t^{\frac{\theta}{\tau}} e^{-\frac{t}{\tau}} \approx \sum_{i=1}^q w_i e^{-\frac{t}{\tau_i}} \quad \text{for } t \geq 0 \text{ and where } \alpha_\theta \text{ is such that } \int_0^\infty h_\theta(t) dt = 1.$$

This model is biologically plausible, qualitatively reproduces empirical data, and has been proposed as a model for spatial representations in hippocampus as well. Still, as we discuss below, there are theoretical reasons why relying on linear combinations of exponential filters can be suboptimal. Voelker and Eliasmith (2018) demonstrate that the recurrent Legendre Delay Network (LDN) similarly reproduces time cell data (see Section 4.3.2).

¹In the original Pastalkova et al. (2008) experiment, the “stimulus onset” corresponds to the animal beginning to perform some behavioural task, namely running in a wheel.

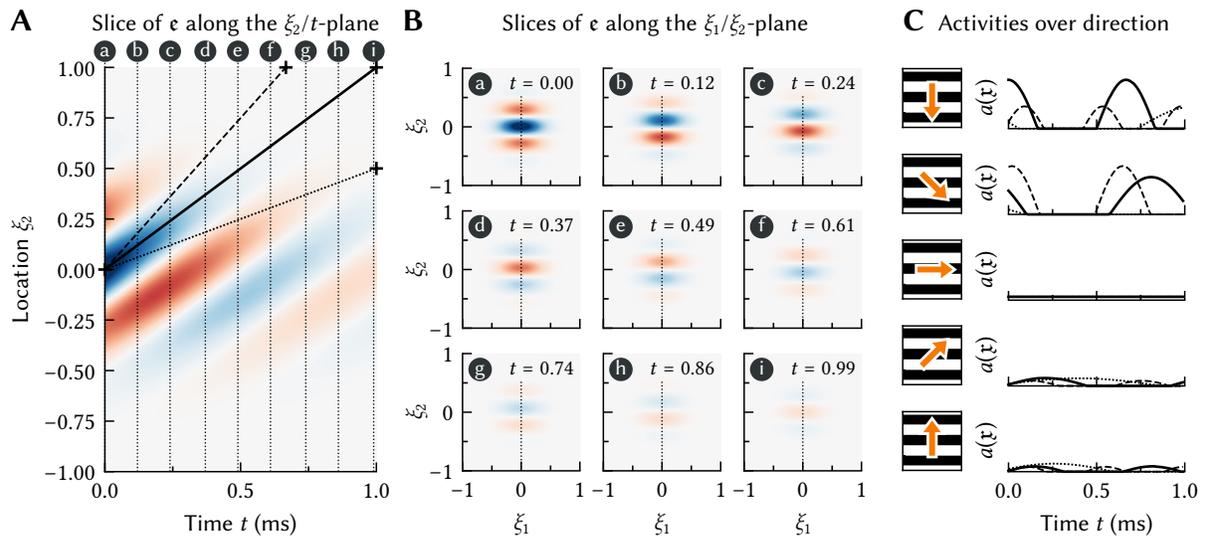


Figure 4.4: Example of a space-time receptive field $\epsilon(t; \xi_1, \xi_2)$ tuned to downwards motion. The field is a Gabor filter (cf. Section 2.2.4) with shifting phase and exponential decay. **(A, B)** Different slices through ϵ . Blue corresponds to positive, red to negative values. **(C)** Computing the neural activity for a grating pattern \mathfrak{x} moving in the arrow direction. Line styles encode the pattern velocity (cf. diagonal lines in (A) for vertical motion); the solid line is matched to the velocity of the receptive field. Downwards motion results in the strongest response. Oscillatory behaviour encodes the phase.

Space-time receptive fields in visual cortex. Linear models of simple cells in visual cortex were our main motivation for modelling tuning curves in the NEF as the dot-product between the stimulus \mathfrak{x} and a normalised encoder ϵ (cf. Section 2.2.4). Similarly, linear models of the temporal behaviour of simple cells—beyond the previously discussed phasic response—serve as our basis for extending the NEF toward temporal tuning.

The basic observation is that some cells in visual cortex reach their maximum firing rate for specific stimulus signals. For example, some cells only react to edges moving in a certain direction at a specific velocity. One possible way to construct a model of this spatiotemporal tuning is to use a *space-time receptive field* $\epsilon(t; \xi_1, \xi_2)$, where ξ_1 and ξ_2 are coordinates in the visual field and $t \geq 0$ is time (Watson et al., 1983; Adelson et al., 1985; see Carandini et al., 1999, for a review). Given stimulus history $\mathfrak{x}(t; \xi_1, \xi_2)$, the neural activity $a(\mathfrak{x})$ at $t = 0$ can be modelled by convolving over the instantaneous inner products between \mathfrak{x} and ϵ . Borrowing the gain and bias terms α, β , as well as the response curve G from the NEF, we have

$$a(\mathfrak{x}) = G \left[\alpha \iiint_0^\infty \epsilon(\tau; \xi_1, \xi_2) \mathfrak{x}(-\tau; \xi_1, \xi_2) d\tau d\xi_1 d\xi_2 + \beta \right] = G \left[\alpha \int_0^\infty \langle \epsilon(\tau), \mathfrak{x}(-\tau) \rangle d\tau + \beta \right]. \quad (4.2)$$

The space-time receptive fields of individual neurons (sometimes referred to as “weighting functions”) can be reconstructed by exposing animals to computer generated imagery (McLean and Palmer, 1989). An example of an artificial space-time receptive field that qualitatively resembles empirical data (cf. DeAngelis et al., 1993) is depicted in Figure 4.4.

4.1.2 Temporal Tuning Curves and the Neural Engineering Framework

The above summary of biological temporal response patterns provides some motivation as for why modelling temporal tuning can be desirable. We now define this concept more rigorously, and, in the next subsection, compare it to the original NEF dynamics principle.

The most general definition of a “temporal tuning curve” is “a function that maps the past history of a d -dimensional stimulus onto the average expected activation of a neuron”.

Definition 4.1. A temporal tuning curve $a(\mathfrak{x})$ is a second-order function mapping from the past stimulus history \mathfrak{x} onto the average neural activity at $t = 0$ (i.e., the present). In other words, $a : (\mathbb{R}^- \rightarrow \mathbb{R}^d) \rightarrow \mathbb{R}$, where $\mathbb{R}^- = \{x \mid x \in \mathbb{R} \text{ and } x \leq 0\}$.

Of course, this definition includes a wide range of imaginable functions and is thus not very practical. One particular class of temporal tuning curves that fits the NEF well are “linear temporal tuning curves”, inspired by the space-time receptive fields discussed above.

Definition 4.2. Linear temporal tuning curves are a special class of temporal tuning curves. Let $G[J]$ be a neural response curve, then the average activity of the i th post neuron is modelled as

$$a_i(\mathfrak{x}) = G \left[J_i \left(\int_0^\infty \langle \mathfrak{e}_i(\tau), \mathfrak{x}(-\tau) \rangle d\tau \right) \right] = G \left[\alpha_i \int_0^\infty \langle \mathfrak{e}_i(\tau), \mathfrak{x}(-\tau) \rangle d\tau + \beta_i \right], \quad (4.3)$$

where the temporal encoder \mathfrak{e}_i is a function $\mathfrak{e}_i : \mathbb{R}^+ \rightarrow \mathbb{R}^d$ over time.

Limiting the temporal encoder to nonnegative times ensures that \mathfrak{e}_i is a causal filter. As before, we assume that the current translation function $J_i(\xi)$ is affine with gain α_i and bias β_i .

As with standard tuning-curves, eq. (4.3) is a *normative constraint* (cf. Section 2.3.3). However, in addition to what we had before, the temporal encoder \mathfrak{e}_i allows modellers to define some desired *dynamics*. This effectively unifies the NEF dynamics and representation principles; as we discuss below, the weight solver can draw upon the dynamics of the pre-population as well as temporal resources such as synaptic filters to approximate the desired dynamics.

Normalisation and preferred temporal stimuli. In many cases, it may be useful to normalise the magnitude of the temporal encoder to an integral of one over time:

$$\int_0^\infty \|\mathfrak{e}(\tau)\| d\tau = 1. \quad (4.4)$$

In this way, and with some squinting, the temporal encoder can be interpreted as a *preferred temporal stimulus*. Note that normalisation is not possible for temporal encoders with an infinite energy, such as oscillators and integrator dynamics without decay.²

²Both the signal \mathfrak{x} and the encoder \mathfrak{e} must be normalised for the convolution between the encoder and the stimulus to correspond to a cosine similarity. However, given its unbounded nature, it is unclear how the signal history \mathfrak{x} should be normalised. Essentially, we would have to normalise with respect to some window function.

Synaptic filters as temporal encoders. We can easily confirm that our notion of temporal tuning is compatible with the standard NEF population dynamics. Remember from Section 2.3.5 that we typically assume a linear synaptic filter h . Correspondingly, given some time-invariant normalised encoding vector \mathbf{e}_i , and assuming that the synapses are modelled as a first-order low-pass filter, a population naturally possesses the following temporal encoder

$$\mathbf{e}_i(t) = h(t)\mathbf{e}_i, \quad \text{with } h(t) = \tau^{-1} \exp(-t\tau^{-1}) \quad \text{for } t > 0. \quad (4.5)$$

This temporal encoder fulfils the normalisation constraint in eq. (4.4). Furthermore, as one would expect, combining eq. (4.5) and eq. (4.3) yields the standard NEF population dynamics:

$$a_i(\mathbf{x}) = G \left[\alpha_i \int_0^\infty \langle h(\tau)\mathbf{e}_i, \mathbf{x}(-\tau) \rangle d\tau + \beta_i \right] = G \left[\alpha_i \langle \mathbf{e}_i, (\mathbf{x} * h)(0) \rangle + \beta_i \right]. \quad (4.6)$$

Importantly, this does not imply that the temporal encoder is in any way defined by the synaptic filter h ; modellers can still choose any \mathbf{e}_i they desire. However, unless there are other temporal resources in the network—such as diverse temporal tuning of the pre-population or recurrent connections, these are the only dynamics that can be well approximated.

General weight optimisation problem. Naturally, this raises the question of how to ensure that a pre-population possesses diverse temporal tuning. While we could try to manually diversify synaptic filters, remember that one of the fundamental ideas of the NEF is to specify tuning as a normative constraint, and to solve for connection weights that realise this tuning.

To this end, we propose a temporal variant of the current-space least-squares loss from Section 3.2. If so desired, we can combine this loss function with subthreshold relaxation (Section 3.2.3), nonnegativity constraints (Section 3.2.2), and dendritic nonlinearities (Section 3.2.4).

As a first step, consider the loss function for connecting two neuron populations representing d -dimensional quantities without any transformation f . Both neuron populations will represent the same quantity, but with potentially different temporal tuning.

Let $a_j(\mathbf{x})$ be the temporal tuning curve of the j th pre-neuron, J_i the current-translation function, and h_{ij} the synaptic filter between the j th pre- and i th post neuron; this may include axonal and synaptic transmission delays. Furthermore, let N be the number of input signals (i.e., samples), and m the number of pre-population neurons. We have

$$E = \sum_{k=1}^N \left[J_i \left(\int_0^\infty \langle \mathbf{e}_i(\tau), \mathbf{x}_k(-\tau) \rangle d\tau \right) - \sum_{j=1}^m w_{ij} \int_0^\infty h_{ij}(\tau) a_j(\mathbf{x}_k * \delta_\tau) d\tau \right]^2, \quad (4.7)$$

where $\delta_\theta(t) = \delta(t - \theta)$ is a shifted Dirac delta. Correspondingly $(\mathbf{x}_k * \delta_\theta)(t) = \mathbf{x}_k(t - \theta)$, and so $a_j(\mathbf{x}_k * \delta_\tau)$ is the activity of the j th pre-neuron at time $-\tau$.

The term before the minus sign in eq. (4.7) is the current required to obtain our desired temporal tuning from eq. (4.3); the term after the minus sign uses the pre-population temporal

tuning and the synaptic filters as a *spatiotemporal function basis* to decode the desired currents. While we know that randomly choosing spatial encoders \mathbf{e}_j results in diverse *spatial* pre-population tuning (Section 3.1), we discuss in Section 4.2 how this basis can be made as *temporally* diverse as possible. Note that our optimisation problem is similar in principle to optimal Wiener filters (Wiener, 1949; Haykin, 2014, Chapter 2). We discuss this in more detail in the context of *adaptive* filters in Section 4.3.4 and Chapter 5.

Lastly, note that our loss function is *not* an integral over time—as is for example common in optimal control theory (e.g., Brogan, 1991, Chapter 14). Instead, we take advantage of the fact that our temporal tuning curves are defined as the average activity of the neuron in the *present*, i.e., at time $t = 0$. Correspondingly, we only minimise the error at a single point in time, namely $t = 0$, but for a large set of input samples \mathbf{r}_k . We can thus tabulate eq. (4.7) as

$$\mathbf{w}_i = \arg \min_{\mathbf{w}_i} \|\mathbf{A}_i \mathbf{w}_i - \mathbf{J}_i\|_2^2 \quad \Rightarrow \quad \mathbf{w}_i = \mathbf{A}_i^+ \mathbf{J}_i, \quad \text{where } \mathbf{A}_i \in \mathbb{R}^{N \times m}, \quad \mathbf{J}_i \in \mathbb{R}^n,$$

and where \mathbf{A}_i^+ is the Moore-Penrose pseudo-inverse. Put differently, and assuming that \mathbf{r}_k is finite and that all quantities have been appropriately sampled, minimising eq. (4.7) is the same kind of convex optimisation problem previously encountered in Section 3.2.

The primary challenge in solving this optimisation problem lies in selecting suitable input signals \mathbf{r}_k (cf. Verhaegen and Verdult, 2007, Section 10.2.5)—we discuss this in more detail in Section 4.3.1. Choosing \mathbf{r}_k is particularly important if the desired tuning cannot be realised precisely; the frequency content of the training signals then determines the regime over which the approximation works well. In our examples we use low-pass filtered white noise as \mathbf{r}_k .

Finally, consider the case where we additionally solve for a spatiotemporal transformation, that is, a function f that maps from a d -dimensional signal \mathbf{r} onto a d' -dimensional quantity, i.e., $f : (\mathbb{R}^d \rightarrow \mathbb{R}^{d'}) \rightarrow \mathbb{R}^{d'}$ (cf. Table 1.1; see Section 4.3.3 for examples). We have:

$$E = \sum_{k=1}^N \left[J_i \left(\int_0^\infty \langle \mathbf{e}_i(\tau), f(\mathbf{r}_k * \delta_\tau) \rangle d\tau \right) - \sum_{j=1}^m w_{ij} \int_0^\infty h_{ij}(\tau) a_j(\mathbf{r}_k * \delta_\tau) d\tau \right]^2. \quad (4.8)$$

As we explain in far more detail below, this equation unifies the NEF representation, transformation, and dynamics principles. To summarise:

1. *Representation.* To find identity decoders $\mathbf{D}_{\theta'}$, use eq. (4.7). Set $J_i(\xi) = \xi$ and $\mathbf{e}_i = (\mathbf{I})_i \delta(-\theta')$. Choosing $\theta' < 0$ queries information present in the neuron population about the past.
2. *Transformation.* Directly use eq. (4.8); we can compute function decoders \mathbf{D}^f as above.
3. *Dynamics.* To realise linear dynamics, choose $\mathbf{e}_i(t) = \langle \mathbf{e}_i^t, \exp(\mathbf{A}t)\mathbf{B} \rangle$ where $\mathbf{e}_i^t \in \mathbb{R}^q$.³

³As mentioned in the introduction, realising nonlinear dynamics is technically possible, but notationally inelegant. We would have to rely on the general definition of temporal tuning (Definition 4.1) and solve for $G_i^{-1}[a_i(\mathbf{r}_k)]$. Also, note that realising nonlinear dynamics is less important than it may seem; we can decode arbitrary nonlinear spatiotemporal f from neurons with linear tuning due to the nonlinear response curve G_i .

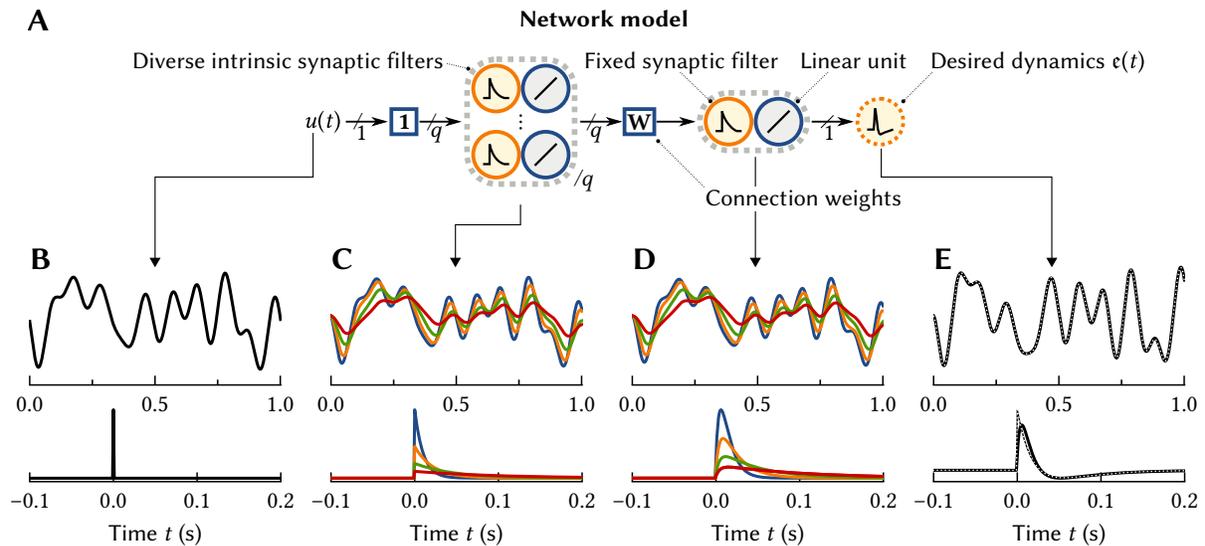


Figure 4.5: Solving for desired dynamics in a feed-forward linear network. **(A)** Overview of the network architecture. A collection of q synaptic filters diversifies the temporal response in a pre-population. **(B, C, D, E)** Example with $q = 4$ first-order low-pass filters (time-constants 10 ms, 22 ms, 46 ms, and 100 ms). The desired dynamics are a high-pass filter (eq. 4.1, $\tau_1 = 20$ ms, $\tau_2 = 40$ ms, $\alpha = 1$). *Top:* System response to a band-limited white noise signal $u(t)$. *Bottom:* Impulse response. Dotted line in (E) is the desired response, solid line is the response when solving for W using least-squares.

4.1.3 Implementing Linear Time-Invariant Systems

For the sake of simplicity, we first focus on linear networks and dynamical systems. In particular, we discuss realising linear time-invariant (LTI) systems in networks with linear response curves and current-translation (i.e., $G_i[J] = J$ and $J_i(\xi) = \xi$). Essentially, each “neuron” corresponds to a separate represented dimension.

This restriction is less severe than it may seem. LTI systems are quite useful, and we can use the NEF representation and transformation principles (Sections 2.3.3 and 2.3.4) to emulate linear networks with nonlinear neurons. Still, we discuss examples with more complex nonlinear networks and dynamics in Section 4.3.

Feed-forward dynamics. Assume that a pre-population possesses diverse temporal tuning. We can now use eq. (4.8) to solve for desired post-population dynamics in feed-forward networks. This is possible because the diverse temporal tuning of the pre-neurons forms a *temporal basis*—just like the time-invariant tuning of an NEF population forms a basis from which arbitrary functions f can be decoded (cf. Figure 2.26).

Figure 4.5 provides an example of this concept. Each pre-population neuron possesses a slightly different synaptic filter time-constant, and we solve for weights that result in high-pass filter tuning of the post-population. This is the same idea as in the time-cell model by Howard et al. (2014), although we take the synaptic filter of the post-population into account as well.

Limitations of feed-forward dynamics. As we alluded to when discussing the Howard et al. (2014) time-cell model, there are major limitations to approximating dynamics in feed-forward networks with low-pass filter synapses. Specifically, network dynamics slow down as the network depth increases, the memory of a fixed-depth network is finite, and, generally speaking, the set of linear dynamics that can be realised is rather limited.

Phrasing the first limitation more precisely, maintaining uniform dynamics across a series of chained populations becomes more difficult as the length of the chain increases. This is illustrated in Figure 4.6. To reduce approximation errors, the desired temporal tuning for later populations must possess longer time-constants. This is despite it being, perhaps counter-intuitively, possible to exploit diverse synaptic filters and temporal tuning to realise dynamics with *faster* time-constants than the fastest synaptic filter in the post-population.

Conversely, pertaining to our second limitation, it is not possible to solve for dynamics with *slower* time-constants than what is dictated by the longest chain of synaptic filters. Correspondingly, feed-forward dynamics cannot realise infinite impulse response (i.e., not exponentially decaying) dynamics such as integrators and oscillators, and they are not a good model of phenomena such as working memory that rely on sustained neural activity.⁴ This is what Churchland et al. (1992, Chapter 1, p. 4) refer to when they claim that “complex effects are the outcome of the dynamics of neural networks” (see the title page of this chapter).

Regarding the third issue, the set of possible dynamics that can be realised in feed-forward networks is severely limited. We can analyse this by performing a principal component analysis (PCA) of low-pass filter impulse responses. This “uncovers” the set of orthogonal temporal basis functions implicitly spanned by the low-pass filters; the associated singular values determine the contribution of each orthogonal basis function. In the case of low-pass filters, and as is depicted in Figure 4.7, the number of orthogonal basis functions that substantially contributes to spanning the space is limited to about four; and those basis functions are most diverse over the first hundred milliseconds, indicating that this is the region where the basis is most “expressive”. We analyse temporal bases in more detail in the next section.

Recurrent connections. Recurrent dynamics do not possess any of these limitations. Specifically, recurrence can diversify the temporal tuning, even if, as in the NEF dynamics principle, the only temporal resources in a network are homogeneous synaptic filters.

Surprisingly, we do not have to take special precautions when solving for recurrent connection weights using eq. (4.8)—we simply treat the recurrent population as its own pre-population. Correspondingly, as with any other pre-population, we assume that the population possesses the temporal tuning defined by the tuning-curve constraint (eq. 4.3). We thus foster a self-fulfilling prophecy: we solve for dynamics assuming that the dynamics are already realised.

⁴Indeed, as elaborated by Eliasmith and Anderson (2003, Section 8.4.1), local recurrent (and *not* feed-forward) connections are known to generate sustained activity in biology (e.g., in eye-position control in goldfish, cf. Aksay et al., 2001). However, recent studies also suggest that momentary synaptic adaptation outweighs persistent activities as a primary mechanism supporting working memory (Lundqvist, Herman, and Miller, 2018).

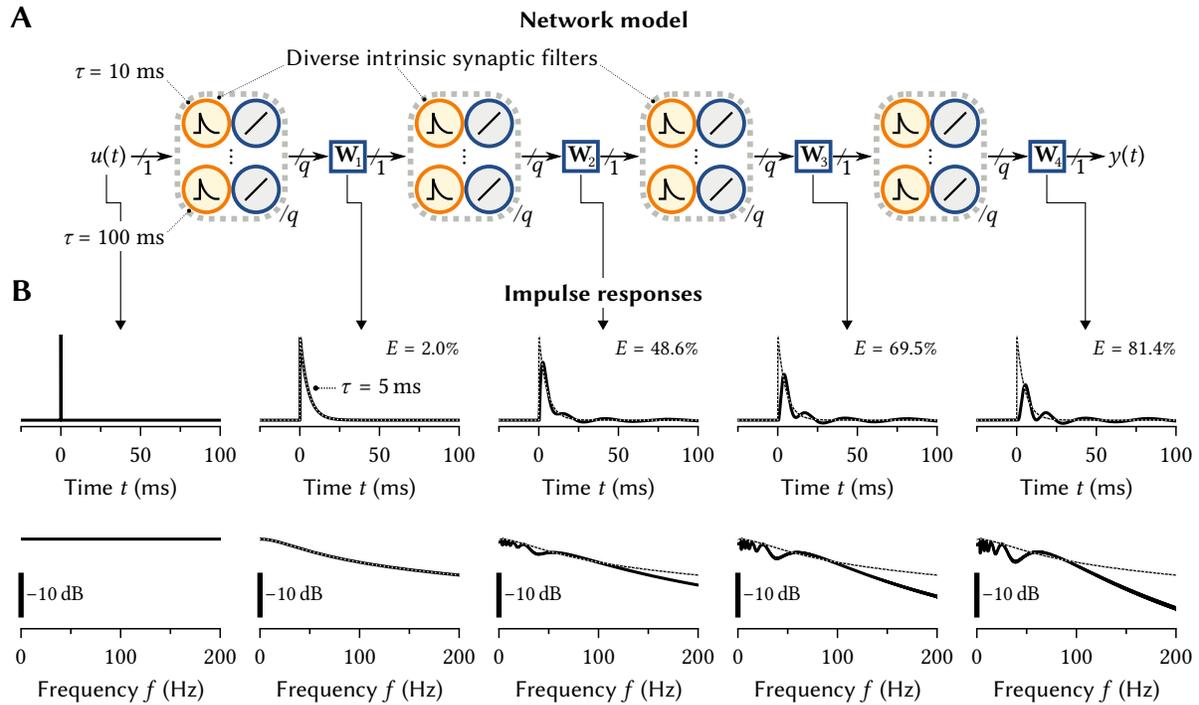


Figure 4.6: Chaining populations with diverse temporal tuning. **(A)** Overview of the network architecture. Each population possesses $q = 100$ synaptic filters with time-constants between 10 ms to 100 ms. The desired temporal tuning ϵ_i of each neuron is a 5 ms first-order low-pass filter. **(B)** Impulse responses at different points in the network. Dotted lines correspond to the desired ϵ_i . Connection weights were computed with perfect knowledge of the actual dynamics. Although the 5 ms filter can be decoded from the first population, the decoding error E (NRMSE) increases in subsequent populations (*top*) and, as is apparent from the depicted spectra (*bottom*), becomes a stronger low-pass.

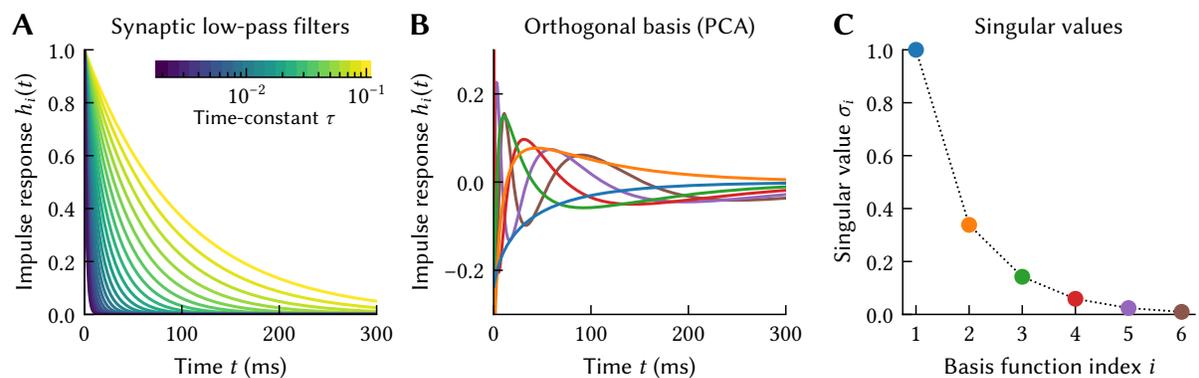


Figure 4.7: Principal component analysis of a set of low-pass filters. **(A)** Impulse responses of $q = 20$ synaptic low-pass filters. **(B, C)** First-order low-pass filter principal components and the corresponding normalised singular values. Most of the information is contained within the first 100 milliseconds and only the first two to four orthogonal basis functions can be decoded stably. Increasing q or increasing the range of possible τ does not substantially improve these numbers.

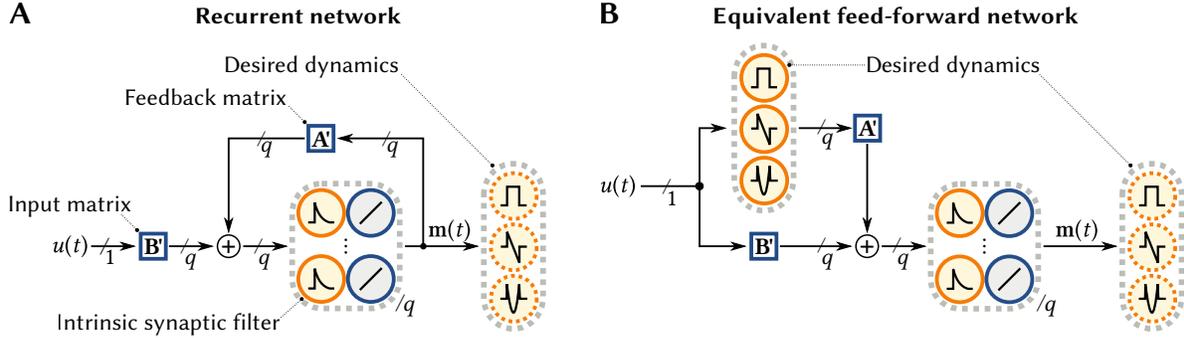


Figure 4.8: Solving for some desired temporal tuning in a recurrent neural network. **(A)** Overview of the recurrent neural network. We need to solve for an input matrix \mathbf{A} and a feedback matrix \mathbf{B} that realise the desired dynamics for the given intrinsic synaptic filter. **(B)** Assuming that the desired dynamics have already been realised, we obtain a feed-forward network. Solving for weight matrices is now a simple least-squares problem. Note that the placement of the weight matrices \mathbf{A} , \mathbf{B} and the synaptic filters can be swapped because of linearity.

Notably, this assumption effectively turns the system into a feed-forward network, which greatly simplifies solving for recurrent weights (cf. Figure 4.8). In the special case of linear networks, solving for weights directly results in input and feedback matrices \mathbf{B}' , \mathbf{A}' . For first-order synapses these matrices are equal to those returned by the NEF dynamics principle.

Theorem 4.1. Let $\mathbf{A} \in \mathbb{R}^{n \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times m}$ (w.l.o.g. $m = 1$) represent an LTI system, ϵ_i be the impulse response of its i th state dimension, and $h(t)$ be the impulse response of a first-order low-pass:

$$\dot{\mathbf{m}}(t) = \mathbf{A}\mathbf{m}(t) + \mathbf{B}u(t), \quad \epsilon_i(t) = (\exp(\mathbf{A}t)\mathbf{B})_i, \quad h(t) = \tau^{-1}e^{-t/\tau}.$$

Let $\mathfrak{x}_k : \mathbb{R}^- \rightarrow \mathbb{R}$ be one of $N \geq n + 1$ arbitrary (with some mild constraints; see proof) input signals. If all ϵ_i and \mathfrak{x}_k are pairwise linearly independent, the following least-squares loss has a unique minimum at $(b'_i, a'_{i,1}, \dots, a'_{i,n}) = (\mathbf{B}', \mathbf{A}')_i$ for all $i \in \{1, \dots, n\}$, where $\mathbf{A}' = \tau\mathbf{A} + \mathbf{I}$, $\mathbf{B}' = \tau\mathbf{B}$:

$$E = \sum_{k=1}^N \left(\int_0^\infty \epsilon_i(\tau)\mathfrak{x}_k(-\tau) - b'_i h(\tau)\mathfrak{x}_k(-\tau) - \sum_{j=1}^n a'_{ij} h(\tau) \int_0^\infty \mathfrak{x}_k(-\tau - \tau')\epsilon_j(\tau') d\tau' d\tau \right)^2.$$

This loss function is equal to eq. (4.7) in the case of a linear network with n neurons that recurrently connect to themselves, and a “zeroth” pre-neuron with no temporal tuning providing the input, i.e., $a_0(t) = \mathfrak{x}_k(t)$. We provide a proof for this in Appendix A.3.1. Similarly, as we discuss in Appendix A.3.2, we can use temporal tuning curves to obtain the same results as the NEF in the case of nonlinear dynamics and neurons. Temporal tuning curves are a *generalisation* of the NEF dynamics principle.

Examples of LTI systems realised by solving the least-squares loss are depicted in Figure 4.9. Note that there are some minor discrepancies between the numerical and the ideal solution (error is on the order of 10^{-3}). These errors are mostly due to time-discretisation at $\Delta t = 1$ ms, and would be less noticeable for exponentially decaying impulse responses ϵ_i .

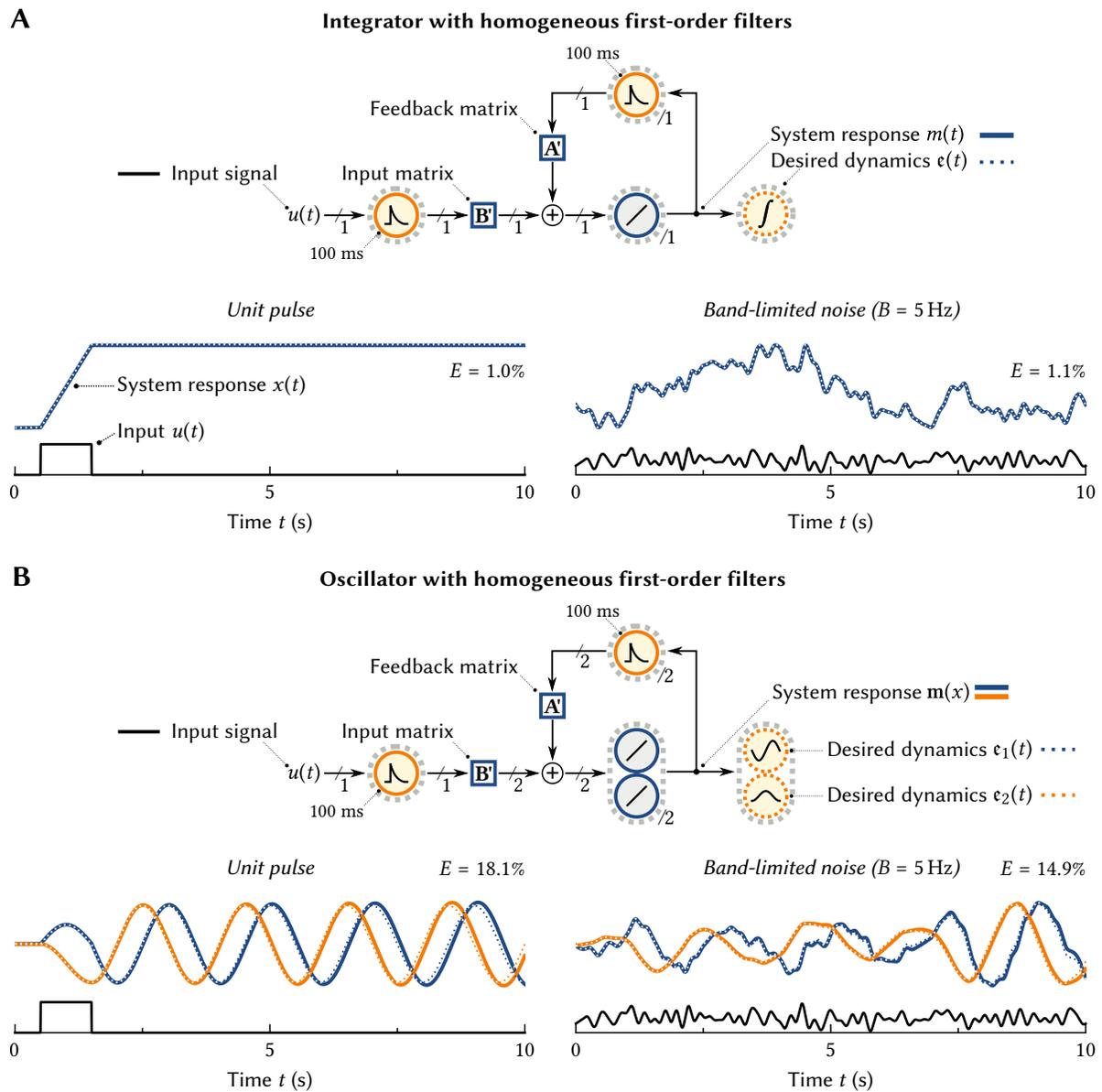


Figure 4.9: Realising LTI systems using temporal tuning curves. The depicted networks possess homogeneous first-order low-pass filters as synaptic filters. The computed matrices \mathbf{A} , \mathbf{B} are equal to the matrices obtained using the NEF dynamics principle down to two decimal places. We use $N = 1000$ input signals x_k of length $T = 10$ s sampled at $\Delta t = 1$ ms consisting of white noise filtered with a second-order low-pass filter (time-constant $\vartheta = 100$ ms). Error values E are the NRMSE. **(A)** Realising an integrator (step impulse-response). Note that the system settles at a value slightly smaller than one for a unit pulse input (*left*) due to the aforementioned imprecisions. For a white-noise input (*right*) the result is effectively indistinguishable from the ground-truth solution. **(B)** Realising a harmonic oscillator by solving for the impulse response of the two-state variables. Again, the solution is slightly imprecise causing a small frequency shift. In addition, in the case of the unit pulse input (*left*), the system slowly diverges.

4.1.4 Realising LTI Systems in Networks with Complex Recurrence

Admittedly, the results presented in the previous subsection are slightly underwhelming. Solving for the dynamics numerically is computationally expensive,⁵ and the resulting matrices \mathbf{A} , \mathbf{B} are only precise to about two decimal places (i.e., an error of about 10^{-3}).

Still, there are two reasons why our approach is useful. First, the imprecisions are negligible compared to those resulting from implementing a linear transformation in a spiking neural network.⁶ Second, and most importantly, the strength of our approach lies in being agnostic to the network architecture; we can approximate connection weights for networks with higher-order and heterogeneous synaptic filters in the feedback and input paths.

Realising integrators and low-pass filters in heterogeneous networks. To demonstrate this, we next analyse networks with second-order synaptic filters (cf. Section 2.2.1; eq. 2.9) and long, heterogeneous time-constants τ . Results are depicted in Figures 4.10 and 4.11. Our goal is to implement an integrator and a short first-order low-pass filter with $\tau = 10$ ms. Importantly, being able to realise these systems implies that we can implement any LTI system.⁷

As is depicted in Figure 4.11, the system performance critically depends on placing *multiple* filters with different time-constants in the input and feedback path.⁸ Curiously, this is *more* biologically plausible, since there typically is some variance in synaptic time-constants in biology (Jones et al., 2014). For the integrator, increasing the number of filters reduces the error by up to two orders of magnitudes, well below what could be realised in a spiking neural network for input bandwidths up to 100 Hz. The “short low-pass” system only performs well for input frequencies up to 10 Hz, but similarly requires at least two filters per path.

The necessity for multiple filters follows from Voelker and Eliasmith (2018, Section 4.4, p. 591). To realise a dynamical system with an n th order synaptic filter, we must generally provide the first $n - 1$ time-derivatives of the input to the network, i.e., $d/dt x$, ..., $d/dt^{n-1} x$. Accounting for heterogeneous filters similarly requires access to the derivative (cf. Appendix A.3.3).

The different filters in the feed-forward path can be thought of as being linearly combined to *approximate* these differentials, akin to the “dual-time-constant networks” explored by Tripp and Eliasmith (2010). Numerically solving the weight optimisation problem in eq. (4.8) automatically determines the weights that implicitly decode the differentials and that, at the same time, *approximate* the desired dynamics.

⁵On the order of one second per processor core, neuron and synaptic filter for $N = 1000$, $T = 1$ s, $\Delta t = 1$ ms.

⁶For $n \approx 100$ neurons, errors due to noise are on the order of 10^{-2} , whereas errors due to static distortion are on the order of 10^{-3} (cf. Eliasmith and Anderson, 2003, Section 2.2.2 and Figure 2.6, note the squared errors).

⁷This follows from the controllable canonical form (cf. Verhaegen et al., 2007, Section 3.4.4), which can be interpreted as chain of leaky integrators (i.e., low-pass filters).

⁸Multiple synaptic filters per connection are technically not compatible with the way our optimisation problem is phrased in eq. (4.8). However, we can assume that the different feed-forward filters correspond to diverse temporal tuning of different pre-population neurons, and the different recurrent filters are obtained by simply splitting our single neuron into three with the same desired temporal tuning.

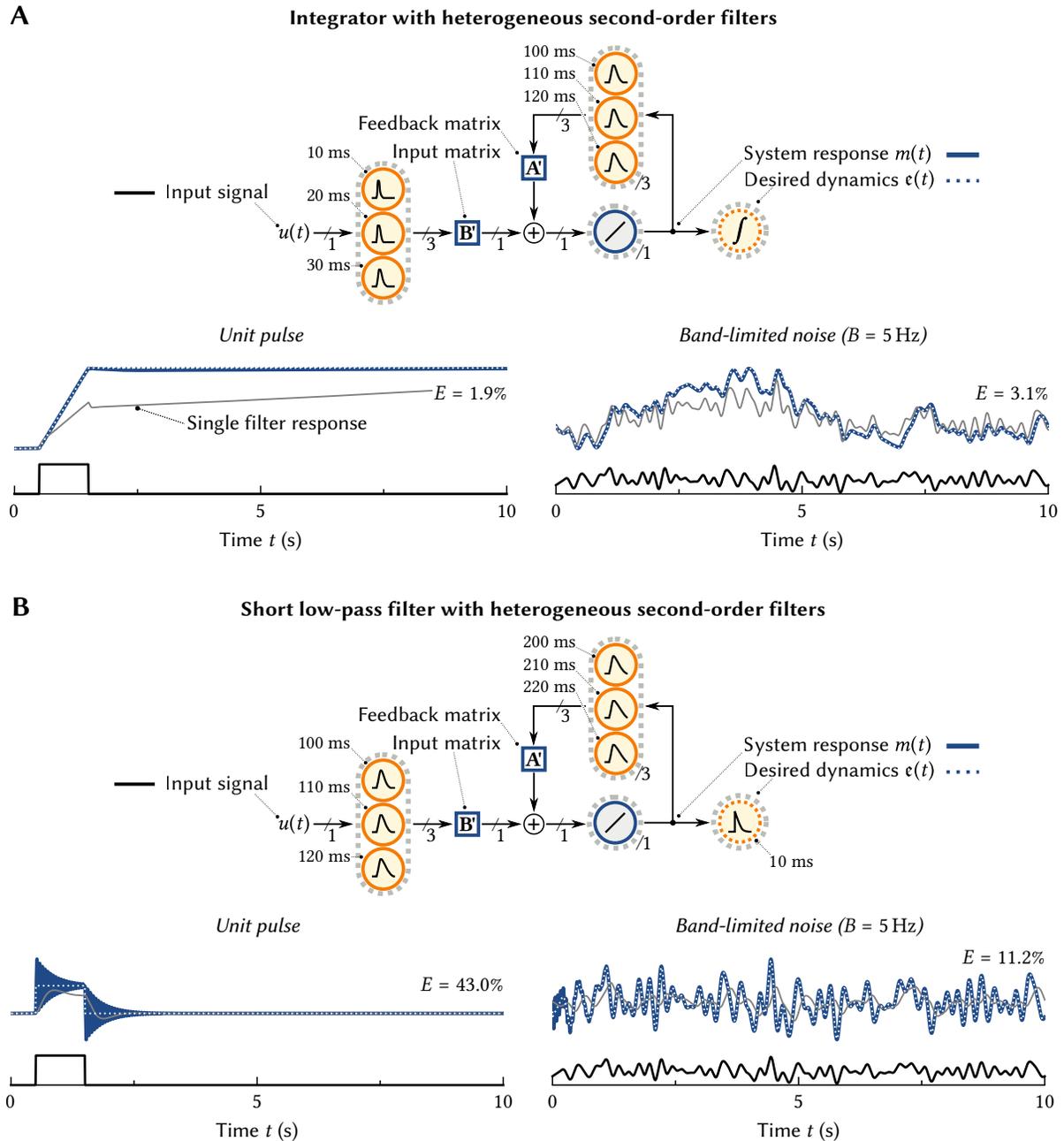


Figure 4.10: Realising LTI systems in networks with heterogeneous second-order filters. The depicted networks possess a single linear neuron with multiple heterogeneous synaptic filters. The grey line corresponds to a reference solution obtained with a single second-order filter in each path. Same setup and legend as described in Figure 4.9. **(A)** Implementing an integrator; the pre-synaptic time-constants are much shorter than the post-synaptic time-constants. Using eq. (4.8), it is possible to solve for quite precise integrator dynamics. **(B)** Implementing a first-order low-pass with $\tau = 10$ ms in a network with much longer synaptic filters. While the response to band-limited white noise (*right*) is reasonably good, there are severe ringing artefacts at input transients, e.g., in the unit pulse (*left*).

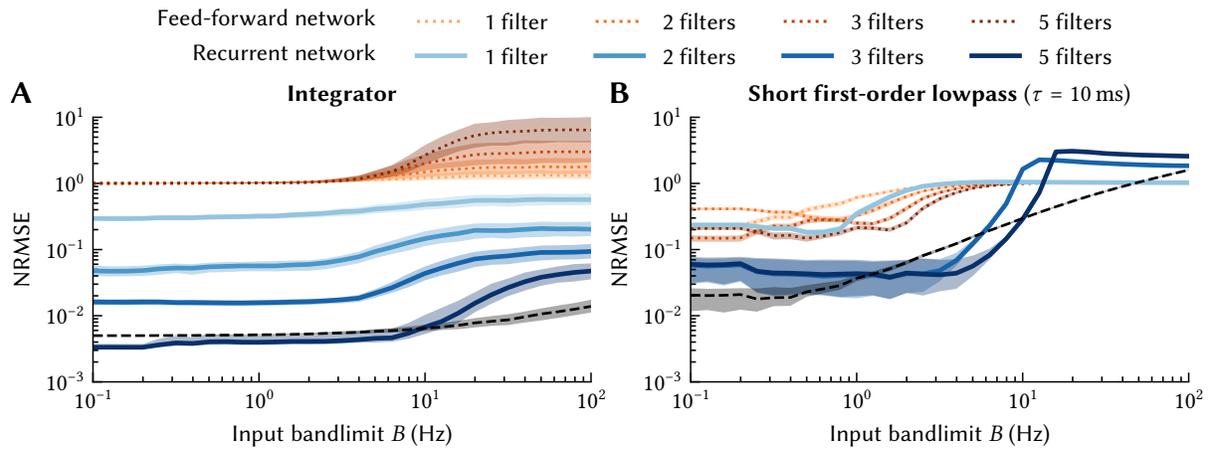


Figure 4.11: Approximation errors in heterogeneous networks with second-order synaptic filters. Errors are the median NRMSE between the actual and desired response for bandlimited white-noise input over 1000 trials. Dotted orange lines correspond to feed-forward networks; blue lines to recurrent networks. See Figure 4.10 for the architecture and the minimum/maximum filter time-constants in each path. Dashed line is a reference network with homogeneous first-order filters ($\tau = 100$ ms). Shaded areas are the 25th and 75percentiles. Same training procedure as in Figure 4.9; training signals \mathbf{r}_k are white noise filtered with a second-order low-pass filter with time-constant 100 ms. **(A)** Implementing an integrator. Recurrence is required to implement the integrator. **(B)** Emulating a 10 ms first-order low-pass requires recurrence and two or more filters per path (data for two and three filters are equal).

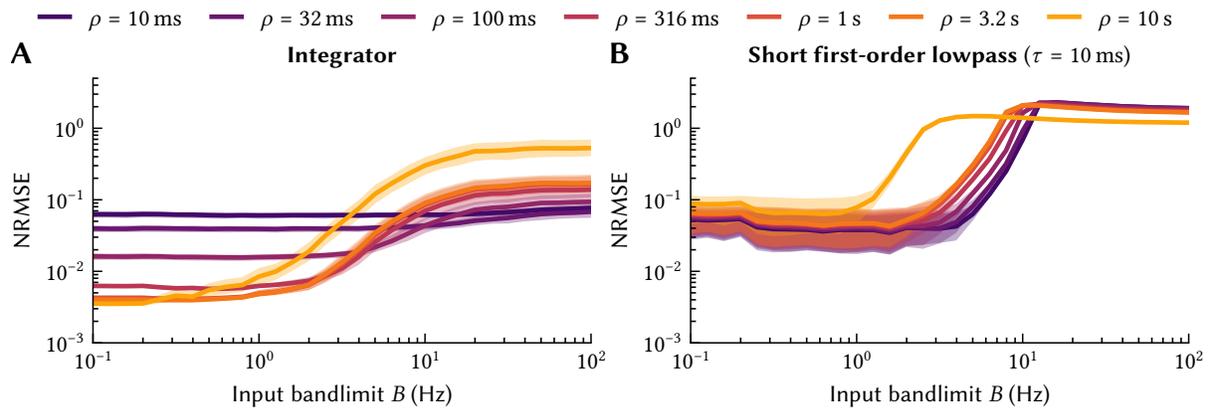


Figure 4.12: Effect of the training signal distribution on the system error. Same setup as in Figure 4.11, but for a recurrent network with three filters and with training signals \mathbf{r}_k of varying bandwidth. Training data are generated by low-pass filtering white noise with a second-order low-pass; the time-constant ρ of that filter corresponds to the different line colours (dark blue corresponds high-bandwidth signals, orange to low-frequency signals). Data is the median over 1000 trials, shaded areas are the 25th and 75th percentile. **(A)** For the integrator, there is a trade-off between lower errors in a small frequency range, and larger, but consistent errors over a larger frequency range. **(B)** The training distribution has a smaller effect on this network; the optimal solution is mostly independent of the training data.

Impact of the training samples \mathbf{r}_k . The emphasis on “approximate” is important here. In contrast to the heterogeneous networks with first-order synaptic filters discussed in the context of Theorem 4.1, there are no guarantees that the final network dynamics are *exactly* the desired dynamics. Moreover, the ability of the network to generalise depends on the input signals \mathbf{r}_k used for training. While this is the norm in most machine learning scenarios, the network in Theorem 4.1 did not have this constraint (cf. Appendix A.3).

We explore the effect of the training signals on the final network in Figure 4.12. Specifically, we generate \mathbf{r}_k with different frequency content by varying the time-constant ρ of the second-order low-pass filter that is applied to the generated white noise. Unsurprisingly, only supplying low-frequency inputs (i.e., using a large ρ) results in networks that achieve low errors for low-frequency inputs. A more wide-band input (i.e., using a small ρ) results in a compromise-solution with smaller errors for wide-band inputs, but (at least in the case of the integrator) slightly higher errors for low-frequency inputs.

Relationship to system identification. When solving for weights in a recurrent system using eq. (4.8), we do not have any formal guarantee that the resulting system is stable—that is, unless the desired dynamics are stable and can be approximated *precisely*. While, judging from our experience, exponentially decaying temporal tuning curves $\epsilon_i(t)$ result in stable networks, it would be good to be able to characterise this more thoroughly.

While we are still working on a more thorough analysis, we can at least draw some interesting parallels to the problem of least-squares system identification (cf. Verhaegen et al., 2007, specifically Chapters 7-10). That is, we can treat the desired dynamics ϵ_i as if they were the output of a black-box system in response to our input signals \mathbf{r}_k . Our goal is to find model parameters w_{ij} that predict the observed output.⁹

In fact, our optimisation problem in eq. (4.8) is a variant of the “integral-equation approach to system identification” presented by Whitfield and Messali (1987), and, in less developed form, by Squire (1971). The primary difference to our approach is that our networks do not contain perfect integrators—correspondingly, we convolve with the synaptic filter instead of integrating. This is possible because the integral-equation approach is—at least if the system can be perfectly realised—invariant to convolution with a non-zero filter.

A central challenge of the integral-equation approach is robustness to noise in the measurements, which—even in the case of zero-mean white noise—causes asymptotic biases in the resulting system (Sagara and Zhao, 1989). That is, even as the number of samples N goes to infinity, the estimated system parameters differ from the true system parameters. Given that we mostly work with artificially generated data, the primary source of noise is time-discretisation. It may be possible to incorporate more modern weight solving approaches with imposed stability guaranties, as is discussed in Verhaegen et al. (2007).

⁹Our specific system identification problem is less complex than a general problem. We directly measure the state, and we do not have instantaneous impulse responses. Put differently, the output matrix \mathbf{C} of our LTI system is the identity, and the feedthrough matrix \mathbf{D} is zero.

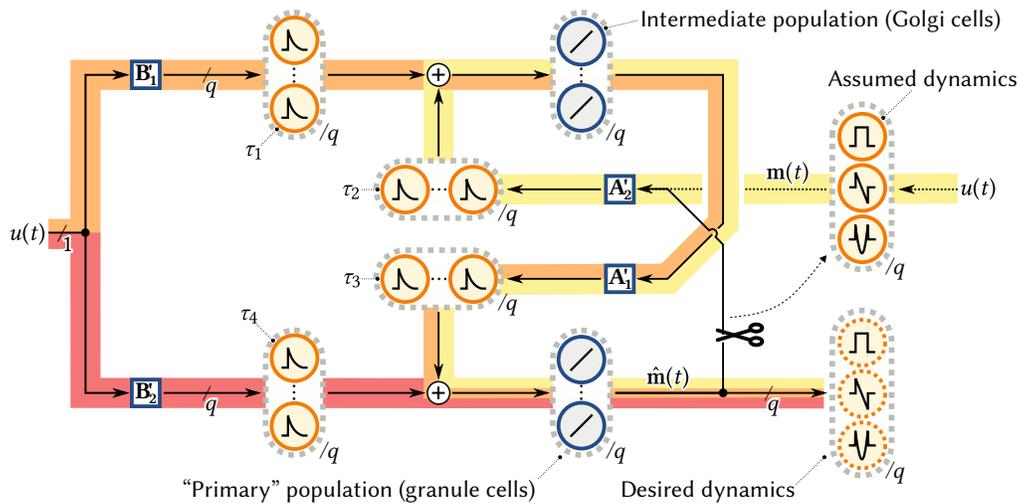


Figure 4.13: Approximating dynamical systems across intermediate populations. Both neuron populations receive input $u(t)$ through a set of synaptic filters with time-constants τ_1 , τ_4 ; the upper, or “intermediate” population then projects onto the lower population, the lower population provides the output and projects onto the upper population. Each feedback path passes through another set of filters with time-constant τ_2 , τ_3 . As before, we can solve for connection weights A'_1 , A'_2 , B'_1 , B'_2 by assuming that the desired dynamics have already been realised (scissor symbol and dotted lines). We must then account for all possible paths (coloured backgrounds) from the input $u(t)$ to the output $\mathbf{m}(t)$.

Realising dynamics in recurrences with intermediate populations. So far, we assumed that recurrent connections are tight loops that originate in a certain population and laterally target neurons in the same population. This is not necessarily the case in biology. As we mentioned in the context of Dale’s principle in Sections 2.3.6 and 3.2.2, inhibitory input passes through interneuron populations. Fortunately, the interneurons typically do not dramatically affect the overall network dynamics (Parisien et al., 2008).

There are, of course, exceptions to this. As we discuss in more detail in Chapter 5, the Granule-Golgi microcircuit in the cerebellum possesses relatively slow synapses in the connection from the excitatory granule cells to the inhibitory Golgi cells.¹⁰ Additionally, the interneurons receive the same input $u(t)$ as the excitatory population (D’Angelo et al., 2013), which does not fit our canonical picture of interneuron populations.

Figure 4.13 illustrates this “Granule-Golgi circuit” schematically. To realise an LTI system \mathbf{A} , \mathbf{B} in such a circuit, we must find weight matrices A'_1 , A'_2 , B'_1 , B'_2 that compensate for the network dynamics. We next discuss two ways to approach this problem.

Ascribing temporal tuning to the interneuron population. First, we could simply decide on some temporal tuning for the interneuron population and use eq. (4.8) to solve for connection weights.

¹⁰As reported by Dieudonné (1998), the synaptic dynamics of the granule to Golgi projection can be modelled using second order filter dynamics and time-constants of $\tau_1 \approx 31$ ms and $\tau_2 \approx 170$ ms (cf. eq. 2.9); this corresponds to a first-order low-pass filter with a time-constant of about 60 ms to 70 ms.

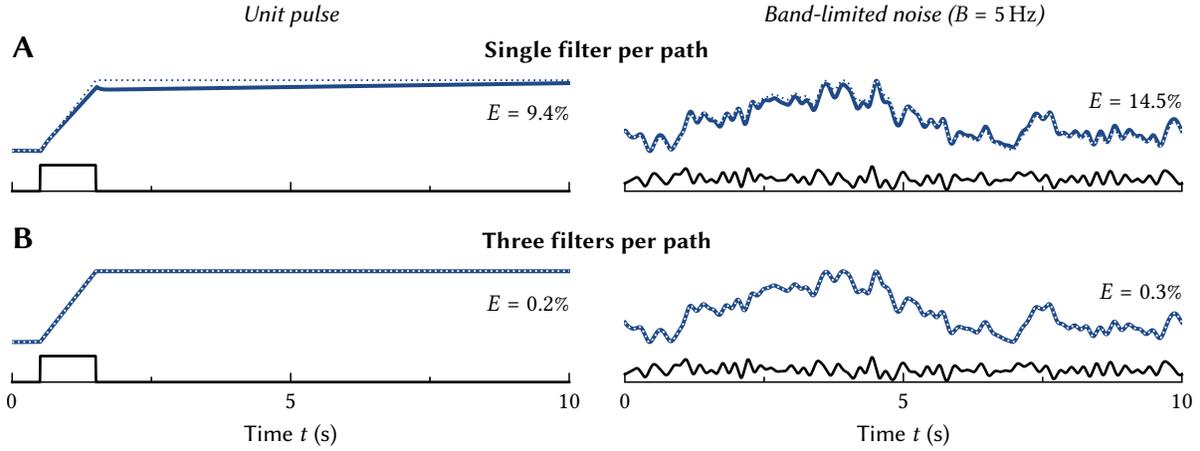


Figure 4.14: Realising an integrator in a recurrent network with an intermediate population and heterogeneous synaptic filters. Same experimental setup as in Figure 4.9; errors E are the NRMSE. **(A)** A single filter per path ($\tau_1 = \tau_4 = 10$ ms, $\tau_2 = \tau_3 = 100$ ms). Errors are relatively large, since we do not compensate for the second-order filter formed by the filter-chain. **(B)** Placing three filters in each path (for a total of 21 weights) results in smaller errors (time-constants are between 10 ms to 30 ms in the input path, and 100 ms to 120 ms in the recurrent path). We sidestep splitting the computed weight matrices by simulating a partially collapsed network; this has no influence on the network function.

If both neuron populations possess the same temporal tuning, then the interneurons appear as a pass-through to the primary population. Assuming homogeneous first-order synaptic filters with time-constant τ , the closed-form solution is

$$\mathbf{A}'_1 = \mathbf{A}'_2 = \tau \mathbf{A} + \mathbf{I}, \quad \mathbf{B}'_1 = \mathbf{B}'_2 = \tau \mathbf{B}. \quad (4.9)$$

Put differently, we use the closed-form solution from the NEF dynamics principle in both paths. This makes intuitive sense due to symmetry, but see Appendix A.3.4 for a detailed derivation.

Unrolling the network. An alternative approach is to solve for all weights simultaneously. Assuming that we have already realised the desired tuning at the output of the network, and because all our neurons and filters are linear (cf. the beginning of Section 4.1.3), we can “unroll” the network into several feed-forward paths (coloured in Figure 4.13).

Let h_1, \dots, h_4 denote the individual synaptic filters. Furthermore, to simplify our notation, let ϵ be a vectorial function $\epsilon : \mathbb{R}^+ \rightarrow \mathbb{R}^q$, and let “ $*$ ” denote elementwise convolution. We obtain the following loss function:

$$E = \sum_{k=1}^N \left\| (\mathbf{x}_k * \epsilon)(0) - \mathbf{B}'_2(h_4 * u)(0) - \mathbf{A}'_1 \mathbf{B}'_1(h_3 * (h_1 * u))(0) - \mathbf{A}'_1 \mathbf{A}'_2(h_3 * (h_2 * \epsilon))(0) \right\|_2^2. \quad (4.10)$$

Minimising this loss function results in the matrix products $\mathbf{A}'_1 \mathbf{A}'_2$ and $\mathbf{A}'_1 \mathbf{B}'_1$ instead of the individual matrices; however, the weights can be split arbitrarily in a post-processing step. Figure 4.14 depicts the results of realising an integrator using this method.

4.2 Realising Temporal Bases

In the previous section, we demonstrated that feed-forward networks alone are insufficient to explain the complex dynamics observed in biological systems. However, we also demonstrated that we can realise arbitrary LTI systems of the form $\dot{\mathbf{m}}(t) = \mathbf{A}\mathbf{m}(t) + \mathbf{B}u(t)$ in recurrent *linear* networks. While we show in the next section that we can successfully solve for dynamics in networks with nonlinear neurons, our goal for now is to consider what specific temporal tuning we should *optimally* realise, and how to translate this tuning into an LTI system.

As we already mentioned above, implementing an LTI system in a recurrent connection can diversify the available temporal tuning. This in turn allows us to decode a wider range of dynamics in ensuing feed-forward sections of the network. To maximise the set of dynamics that can be decoded, and without further knowledge about the input signals, we should optimally realise q orthonormal temporal encoders $\epsilon_1(t), \dots, \epsilon_q(t)$ over an interval $t \in [0, \theta]$. Orthonormality ensures that the ϵ_i are not redundant, maximising information under noise.

We first describe our overall goal more precisely, namely constructing LTI systems that approximate sliding-window spectra. We then review common continuous and discrete bases, and discuss two methods for constructing LTI systems that generate these bases: numerically optimising an autoregressive model, and analytically solving for LTI systems generating the Legendre basis. This latter approach yields a novel derivation of the Legendre Delay Network (LDN; Voelker and Eliasmith, 2018) and is conceptually similar to work by Gu et al. (2020). Finally, we compare the bases generated by these systems in terms of their expressiveness.

4.2.1 Computing Sliding-Window Spectra Using Linear Time-Invariant Systems

Given a function basis spanning a space such as $L^2(0, \theta)$ (e.g., Young, 1988), any function $f \in L^2(0, \theta)$ can be represented as an infinite weighted sum of basis functions \mathbf{b}_n . In the case of an orthonormal function basis (i.e., $\langle \mathbf{b}_i, \mathbf{b}_j \rangle = \delta_{ij}$), the corresponding weights χ_i *uniquely* define each f and are referred to as *generalised Fourier coefficients* or the *spectrum* of f :

Definition 4.3 (Generalised Fourier series and coefficients; cf. Young, 1988, Definition 4.3). *Consider an orthonormal function basis $(\mathbf{b}_n)_{n \in \mathbb{N}}$, where each $\mathbf{b}_n \in L^2(0, \theta)$, as well as a function $f \in L^2(0, \theta)$. Then the series*

$$f = \sum_{i=0}^{\infty} \langle f, \mathbf{b}_i \rangle \mathbf{b}_i = \sum_{i=0}^{\infty} \chi_i \mathbf{b}_i \quad (4.11)$$

is the generalised Fourier series of the given f and χ_i are the generalised Fourier coefficients.

Note on using a finite number of basis functions. In practice, it is often infeasible to realise an infinite number of basis functions. While the \mathbf{b}_n are not necessarily ordered—any permutation $(\mathbf{b}_{\pi(n)})_{n \in \mathbb{N}}$ still forms a valid basis—the basis functions we discuss in this section are arranged such that larger n correspond to higher frequency content. Correspondingly, only using the first q basis functions results in a negligible reconstruction error if our signals are bandlimited, or, equivalently, our signals and bases are discretised appropriately (see below).

Sliding-window spectrum. Coming back to the generalised Fourier coefficients χ_i , note that the inner product $\langle \mathbf{f}, \mathbf{b}_i \rangle$ is—time-reversal aside—merely a convolution between \mathbf{f} and \mathbf{b}_i :

$$\chi_i = \langle \mathbf{f}, \mathbf{b}_i \rangle = \int_0^\theta \mathbf{f}(\tau) \mathbf{b}_i(\tau) \, d\tau = (\bar{\mathbf{f}} * \mathbf{b}_i)(0) \quad \text{where } \bar{\mathbf{f}}(t) \mapsto \mathbf{f}(-t).$$

Correspondingly, if we were able to realise an LTI system that has q orthonormal basis functions \mathbf{b}_i as an impulse response over a window $[0, \theta]$, then the state $\mathbf{m}(t)$ of that system would correspond to the generalised Fourier coefficients χ_i representing the past θ seconds of $u(t)$, which we denote as $u_{[t-\theta, t]} : [-\theta, 0] \rightarrow \mathbb{R}$ with $u_{[t-\theta, t]}(\tau) \mapsto u(t + \tau)$:

$$m_i(t) = \int_0^\theta u(t - \tau) \mathbf{b}_i(\tau) \, d\tau = \int_0^\theta u_{[t-\theta, t]}(-\tau) \mathbf{b}_i(\tau) \, d\tau = \langle \bar{u}_{[t-\theta, t]}, \mathbf{b}_i \rangle.$$

This is sometimes referred to as a *sliding-window spectrum* (Bastiaans, 1985; den Brinler, 1996).¹¹ We illustrate this concept in Figures 4.15A to 4.15C.

Transforming the sliding-window spectrum. Another way to interpret $\mathbf{m}(t)$ is as a compressed representation of $u_{[t-\theta, t]}$. That is, realising q temporal basis functions as an LTI system continuously compresses $u_{[t-\theta, t]}$ into a q -dimensional vector $\mathbf{m}(t)$. Specifically, for $q \rightarrow \infty$, and as is suggested by eq. (4.11), we can perfectly reconstruct $u_{[t-\theta, t]}$ from \mathbf{m} :

$$u_{[t-\theta, t]}(-\tau) = \sum_{i=0}^{\infty} m_i \mathbf{b}_i(\tau) = \sum_{i=0}^{\infty} \langle \bar{u}_{[t-\theta, t]}, \mathbf{b}_i \rangle \mathbf{b}_i(\tau) \quad \text{where } 0 \leq \tau \leq \theta.$$

Similarly, we can compute $u(t)$ convolved with an arbitrary kernel $\mathbf{c} : [0, \theta] \rightarrow \mathbb{R}$ by linearly combining the entries of $\mathbf{m}(t)$. Let $\mathbf{c} = (c_0, c_1, \dots) = \langle \mathbf{c}, \mathbf{b}_i \rangle$ be the generalised Fourier coefficients of \mathbf{c} . Then, $\langle \mathbf{m}(t), \mathbf{c} \rangle = (u * \mathbf{c})(t)$:

$$\begin{aligned} \langle \mathbf{m}(t), \mathbf{c} \rangle &= \sum_{i=0}^{\infty} m_i(t) c_i = \sum_{i=0}^{\infty} \langle \bar{u}_{[t-\theta, t]}, \mathbf{b}_i \rangle \langle \mathbf{c}, \mathbf{b}_i \rangle = \sum_{i=0}^{\infty} \int_0^\theta \int_0^\theta (\bar{u}_{[t-\theta, t]}(\tau) \mathbf{b}_i(\tau)) (\mathbf{c}(\tau') \mathbf{b}_i(\tau')) \, d\tau \, d\tau' \\ &= \int_0^\theta \left(\sum_{i=0}^{\infty} m_i(t) \mathbf{b}_i(\tau) \right) \left(\sum_{i=0}^{\infty} c_i \mathbf{b}_i(\tau) \right) \, d\tau = \int_0^\theta u(t - \tau) \mathbf{c}(\tau) \, d\tau = (u * \mathbf{c})(t). \end{aligned}$$

Where the step from the first to the second line follows from $\langle \mathbf{b}_i, \mathbf{b}_j \rangle = \delta_{ij}$.

An example of this is illustrated in Figure 4.15D, where we decode a delay $\mathbf{c}(t) = \delta(t - \theta/2)$ by linearly combining the three state dimensions $m_i(t)$. Since $q = 3$ is finite, we cannot use \mathbf{c} as defined above, but must instead solve the least-squares problem discussed in the context of approximating dynamics in feed-forward networks (i.e., eq. 4.8).

For $q \rightarrow \infty$, $\mathbf{m}(t)$ represents all information in the windowed input signal $u_{[t-\theta, t]}$. Correspondingly, we can compute any nonlinear function over $u_{[t-\theta, t]}$ by transforming $\mathbf{m}(t)$ nonlinearly. We could, for example, represent \mathbf{m} in a neuron population and decode a function $f(\mathbf{m})$; if we learn this decoder online, we have an *adaptive filter* (cf. Section 4.3.4).

¹¹The cited publications use this term in the context of discrete signals, but the concept is the same.

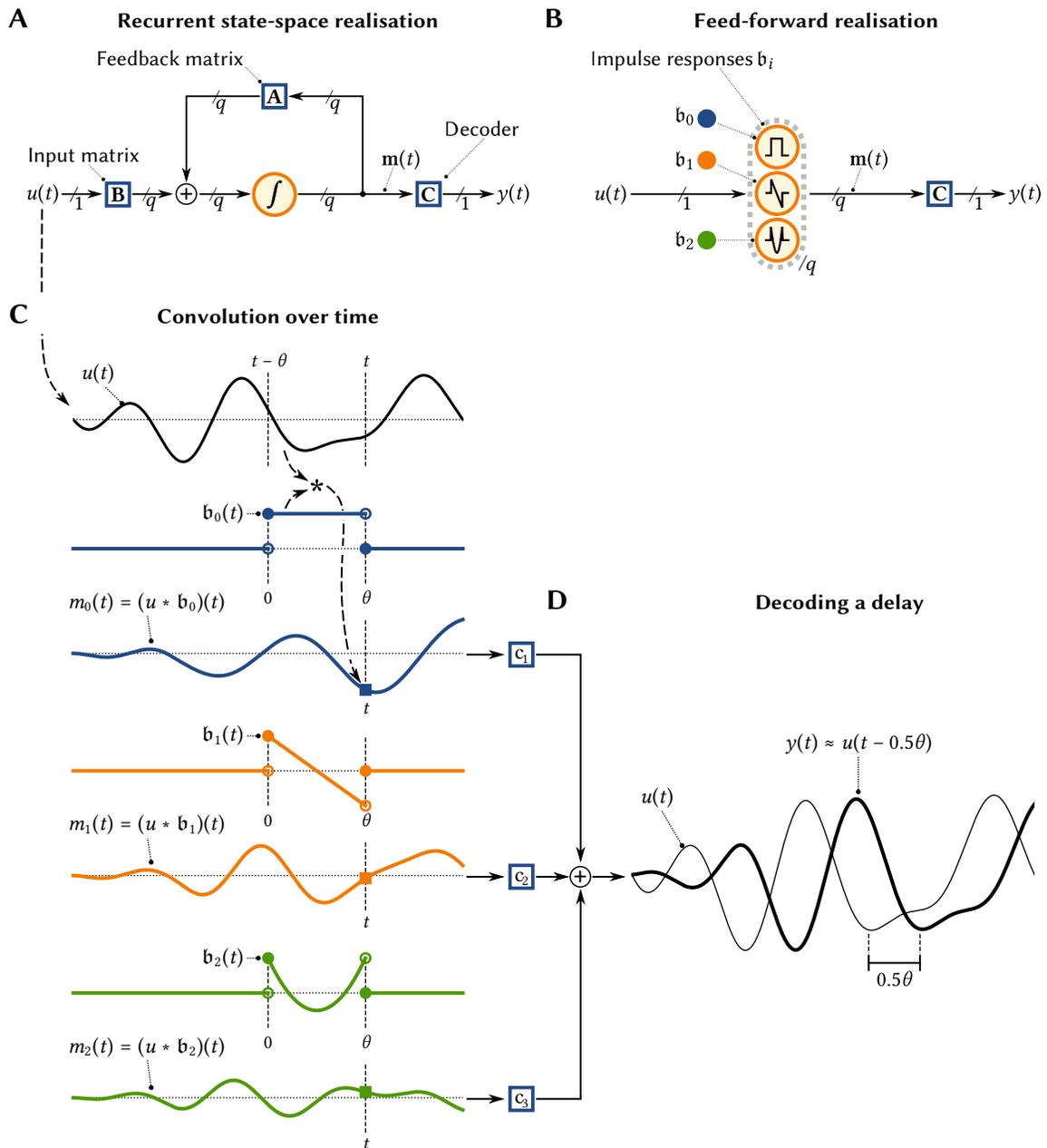


Figure 4.15: Using an LTI system $\dot{\mathbf{m}}(t) = \mathbf{A}\mathbf{m}(t) + \mathbf{B}u(t)$ to compute a sliding-window spectrum. **(A)** Any LTI system can either be implemented as using an integrator with feedback, or, as is depicted in **(B)**, by convolving $u(t)$ with the impulse response b_i of each state dimension. **(C)** Assume that b_i was a basis function over a window $[0, \theta]$. Then, each state dimension $m_i(t)$ is the convolution between b_i and a slice of the input signal $u_{[t-\theta, t]}$. Correspondingly, $m_i(t)$ is the generalised Fourier coefficient χ_i describing $u_{[t-\theta, t]}$ in terms of the function basis at each point in time. In other words, $\mathbf{m}(t)$ contains a compressed representation of $u_{[t-\theta, t]}$. **(D)** By linearly combining $\mathbf{m}(t)$ using a “decoding” matrix C , we can approximate another LTI system, for example a delay $y(t) \approx u(t - \theta/2)$.

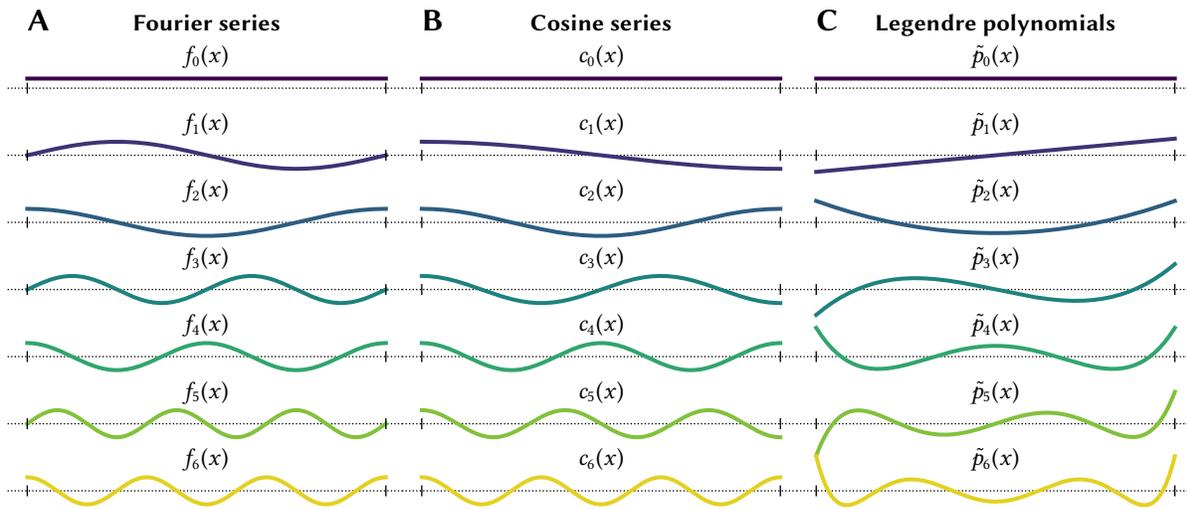


Figure 4.16: Orthonormal basis functions. All plots are to scale, the dotted line is zero, ticks are 0, θ .

4.2.2 Continuous and Discrete Orthonormal Function Bases

We next review three orthonormal bases spanning $L_2(0, 1)$: the Fourier and cosine series, as well as the shifted orthonormal Legendre polynomials.

Fourier and cosine series. The most prominent function basis for temporal representations is the Fourier series. This basis is closely related to the Fourier transform.¹²

Definition 4.4 (Fourier series). *The Fourier series $(f_n)_{n \in \mathbb{N}}$ over $[0, 1]$ is given as*

$$f_0(x) = 1, \quad f_{2n+1}(x) = \sqrt{2} \sin(2\pi nx), \quad f_{2n} = \sqrt{2} \cos(2\pi nx). \quad (4.12)$$

The first f_i are depicted in Figure 4.16A. Note the sine and cosine pairs, and that each function in the Fourier series is periodic over $[0, 1]$, that is $f_n(0) = f_n(1)$. Correspondingly, if we truncate the Fourier series to q terms, we can only represent periodic functions.

An arguably simpler alternative to the Fourier series is the cosine series. This series skips the “sine” terms of the Fourier series and increments the frequency in steps of π instead of 2π .

Definition 4.5 (Cosine series). *The cosine series $(c_n)_{n \in \mathbb{N}}$ over $[0, 1]$ is given as*

$$c_0(x) = 1, \quad c_n(x) = \sqrt{2} \cos(\pi nx). \quad (4.13)$$

Again, the first c_i are depicted in Figure 4.16B. Since the cosine basis is aperiodic ($f_i(0) \neq f_i(1)$ for odd i), it is better suited to representing aperiodic functions if we only have a truncated basis with q terms. Natural signals are typically aperiodic with respect to a fixed sliding window.

¹²While related, the Fourier series and Fourier transform are different concepts. The Fourier series spans a function space over $[0, 1]$, whereas the Fourier transform maps functions $f : \mathbb{R} \rightarrow \mathbb{C}$ onto functions $\mathcal{F}(f) : \mathbb{C} \rightarrow \mathbb{C}$. Still, the generalised Fourier coefficients χ_i of a function f with respect to the Fourier series can be interpreted as the real and imaginary coefficients of the Fourier transform for specific frequencies.

Polynomial bases. An alternative to trigonometric bases are polynomial bases. Examples include the Legendre, Chebyshev, Laguerre, Hermite, and Jacobi polynomials (e.g., Press et al., 2007, Section 4.6.1; Arfken and Weber, 2005, Chapter 12). For our purposes, the Legendre basis is most interesting. It is aperiodic, and in contrast to the other listed polynomial bases, orthogonal with respect to a constant weighting $W(x) = 1$ (as we would intuitively expect).

Definition 4.6 (Legendre polynomials). *Legendre polynomials are uniquely defined as a sequence of functions $(p_n)_{n \in \mathbb{N}}$ over $[-1, 1]$ with the following properties: (1) p_n is a polynomial of order n , (2) the p_n are orthogonal ($\langle p_i, p_j \rangle = 0$ exactly if $i \neq j$), and (3) $p_n(1) = 1$.*

Recurrence relation. Starting with the base cases $p_0(x) = 1$ and $p_1(x) = x$, the Legendre polynomials $p_n(x)$ are typically defined as a recurrence relation (Press et al., 2007, Section 4.6.1):

$$(n + 1)p_{n+1}(x) = (2n + 1)xp_n(x) - np_{n-1}(x). \quad (4.14)$$

Closed form equation. Alternatively, the *orthonormal* shifted Legendre polynomial \tilde{p}_n over $[0, 1]$ are given in closed form as

$$\tilde{p}_n(x) = \sqrt{2n + 1} (-1)^n \sum_{i=0}^n (-1)^i \binom{n}{i} \binom{n + i}{i} x^i. \quad (4.15)$$

This basis is depicted in Figure 4.16C. It spans $L^2(0, 1)$, just like the Fourier and cosine basis.

Discrete Orthonormal Function Bases. Continuous basis functions are useful for mathematical analysis. However, in practice, we often divide the interval $[0, 1]$ into N individual samples. Specifically, we use such discrete bases in our numerical method for deriving **A**, **B**, and in Section 4.4, when we discuss applications to machine learning.

Of course, from a mathematical perspective, the notion of “discrete function bases” is at most moderately exciting—once we discretise functions over an interval, we obtain finite-dimensional vector spaces over \mathbb{R}^N . Still, there is some room for defining the concept of “discrete function bases” in relation to their continuous counterparts more rigorously. Although these definitions are non-canonical, our notation roughly follows Neuman et al. (1974).

Definition 4.7 (Discrete Function Basis). *A discrete function basis with an associated continuous function basis $(\mathfrak{b}_n)_{n \in \mathbb{N}}$ over $[0, 1]$ is a finite sequence of discrete basis functions $(E_n(k; N))_{n < N}$. $n \in \{0, \dots, N - 1\}$ is the basis function index, $k \in \{0, \dots, N - 1\}$ is the sample index, and $N \geq 2$ is the number of samples. E_n must uniformly converge to \mathfrak{b}_n as $N \rightarrow \infty$:*

$$\sqrt{N}E_n\left(\left\lfloor t\left(N - \frac{1}{2}\right)\right\rfloor; N\right) \rightrightarrows \mathfrak{b}_n(t), \text{ and } \sum_{k=0}^{N-1} E_i(k; N)E_j(k; N) = \delta_{ij} \text{ if } E \text{ is orthonormal.} \quad (4.16)$$

We call a matrix $\mathbf{E} \in \mathbb{R}^{q \times N}$ with $(\mathbf{E})_{ij} = E_i(j; N)$ a basis transformation matrix. We refer to \mathbf{E} as orthonormal if $\mathbf{E}^T \mathbf{E} = \mathbf{I}$, even if \mathbf{E} is not square ($q < N$).

Note that for $q = N$ an orthonormal basis transformation matrix \mathbf{E} is an orthonormal basis of \mathbb{R}^N . For $q < N$, the operation $\mathbf{E}\mathbf{x}$ lossily compresses $\mathbf{x} \in \mathbb{R}^N$ into a q -dimensional space.

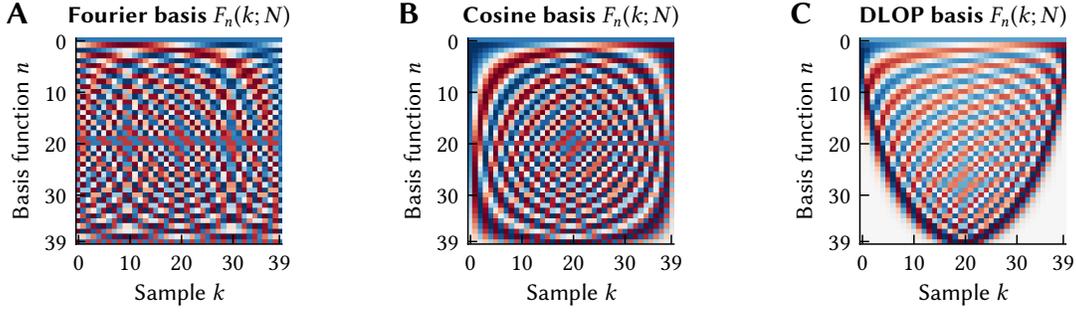


Figure 4.17: Visualisation of the discrete Fourier, cosine, and DLOP (Legendre) basis for $q = N = 40$. Each pixel is a separate sample. Red corresponds to negative values, blue to positive, grey to zero.

Discrete Fourier and cosine basis. Choosing the sample points carefully, we obtain the discrete Fourier F_n and cosine bases C_n used in the discrete Fourier and cosine transforms (DFT and DCT-II) and visualised in Figures 4.17A and 4.17B:¹³

$$\begin{aligned}
 F_0(k; N) &= \frac{1}{\sqrt{N}}, & F_{2n+1}(k; N) &= \frac{\sqrt{2}}{\sqrt{N}} \sin\left(2\pi n \frac{k + \frac{1}{2}}{N}\right), & F_{2n}(k; N) &= \frac{\sqrt{2}}{\sqrt{N}} \cos\left(2\pi n \frac{k + \frac{1}{2}}{N}\right), \\
 C_0(k; N) &= \frac{1}{\sqrt{N}}, & C_n(k; N) &= \frac{\sqrt{2}}{\sqrt{N}} \cos\left(\pi n \frac{k + \frac{1}{2}}{N}\right).
 \end{aligned} \tag{4.17}$$

In both cases, fast algorithms exist for computing $\mathbf{F}\mathbf{x}$ and $\mathbf{C}\mathbf{x}$ in $\mathcal{O}(N \log N)$ instead of $\mathcal{O}(N^2)$, namely the FFT and FCT (Cooley and Tukey, 1965; Makhoul, 1980; Press et al., 2007, Chapter 12).

Discrete Legendre Orthogonal Polynomials (DLOPs). A discrete version of the Legendre polynomials can be generated using the following recurrence relation (depicted in Figure 4.17C):

$$\begin{aligned}
 P_0(k; N) &= \frac{1}{\sqrt{N}}, & P_1(k; N) &= \frac{(2k - N + 1)}{N - 1} \sqrt{\frac{3(N - 1)}{N(N + 1)}}, \\
 P_n(k; N) &= P_{n-1}(k; N) \frac{(2n - 1)(N - 2k - 1)}{n(N - n)} \sqrt{\frac{\alpha_n(N)}{\alpha_{n-1}(N)}} - P_{n-2}(k; N) \frac{(n - 1)(N + n - 1)}{n(N - n)} \sqrt{\frac{\alpha_n(N)}{\alpha_{n-2}(N)}}.
 \end{aligned}$$

Where the normalisation factor $\alpha_n(N)$ is given as

$$\alpha_n(N) = \frac{(2n + 1)(N - 1)^{(n)}}{(N + n)^{(n+1)}} \quad \text{where } k^{(i)} = \prod_{j=0}^{i-1} (k - j) = \frac{k!}{(k - i)!} \text{ is the } i\text{th falling factorial of } k.$$

These ‘‘Discrete Legendre Orthogonal Polynomials’’ (DLOPs) have been proposed by Neuman et al. (1974). Our modified recurrence relation above is numerically stable and directly produces an orthonormal basis. We provide a detailed discussion in Stöckel (2021b).

¹³For even N , F_{N-1} must be rescaled to maintain orthonormality; we ignore this in our definition for terseness.

4.2.3 Numerically Solving for Linear Time Invariant Systems

Our next goal is to construct LTI systems $\dot{\mathbf{m}}(t) = \mathbf{A}\mathbf{m}(t) + \mathbf{B}u(t)$ that generate the above bases using only q state dimensions. To this end, we first describe a numerical method, and then, in the next subsection, analytically solve for \mathbf{A} , \mathbf{B} in the specific case of the Legendre basis.

Autoregressive system identification. Consider an orthonormal basis transformation matrix $\mathbf{M} \in \mathbb{R}^{q \times N}$. The i th column (with $i \in \{1, \dots, N\}$) of \mathbf{M} , here denoted as \mathbf{m}_i , is a q -dimensional vector describing the impulse response of the desired system at $t = \Delta t i$, where $\Delta t = \theta/N$. If the state evolution is indeed the result of a time-invariant linear process, then we have

$$\mathbf{m}_0 = \tilde{\mathbf{B}}, \quad \mathbf{m}_{i+1} = \mathbf{m}_i + \Delta t \tilde{\mathbf{A}} \mathbf{m}_i \quad \Leftrightarrow \quad \tilde{\mathbf{A}} \mathbf{m}_i = \frac{\mathbf{m}_{i+1} - \mathbf{m}_i}{\Delta t} \quad \text{for all } 1 \leq i < N,$$

where $\tilde{\mathbf{B}}$ describes the influence of the initial impulse on the state, and $\tilde{\mathbf{A}}$ is the state-transition matrix. Finding a matrix $\tilde{\mathbf{A}}$ with this property can be written as a linear least-squares problem:

$$\tilde{\mathbf{A}} = \arg \min_{\tilde{\mathbf{A}}} \sum_{i=1}^{N-1} \left\| \tilde{\mathbf{A}} \mathbf{m}_i - \frac{\mathbf{m}_{i+1} - \mathbf{m}_i}{\Delta t} \right\|_2^2. \quad (4.18)$$

This is a standard autoregressive linear model (cf. Verhaegen et al., 2007, Chapter 8); of course, we could use the derivative of a continuous basis instead of the difference quotient. Assuming that $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$ are the result of discretising the LTI system with a zero-order hold assumption (e.g., Brogan, 1991, Section 9.8), we obtain a continuous system \mathbf{A} , \mathbf{B} as follows:

$$\mathbf{A} = \frac{1}{\Delta t} \log(\mathbf{I} + \Delta t \tilde{\mathbf{A}}), \quad \mathbf{B} = \sqrt{N} \tilde{\mathbf{B}}, \quad (4.19)$$

where “log” is the matrix logarithm, the inverse operation of taking the matrix exponential.

Figure 4.18 depicts the impulse responses of LTI systems constructed in this manner. The method works well for the Fourier and the Legendre basis, but results in large errors in the case of the cosine basis; particularly for later basis functions. Since the Fourier system consists of $(q-1)/2$ harmonic oscillators that require two state-dimensions each, q must be odd.

Rather unsurprisingly, the impulse response of each generated LTI system continues past the time-window θ . In the case of the Fourier basis the oscillation just continues, whereas the Legendre and cosine systems diverge. To prevent this, we need to modify the system to rapidly decay to zero, preferably *exactly* at time θ .

Window functions. One possible solution is to multiply the desired basis with a window function $w(t) \in [0, 1]$, that is, $\mathbf{b}'_i(t) = w(t)\mathbf{b}_i(t)$ for $t \in [0, \theta]$. Unfortunately, the resulting basis is no longer orthonormal, and decoding information in regions with small $w(t)$ is more difficult.

Figure 4.19 depicts results for a one-sided Bartlett window $w(t) = 1 - x\theta^{-1}$ (cf. Oppenheim et al., 2009, Section 7.5). Again, we are able to realise the desired dynamics well for the Fourier and Legendre basis, albeit with a large increase in error due to having to decode the window dynamics. Furthermore, this method does not work particularly well for the cosine basis. Although the final system is stable, we can still observe substantial oscillations for $t > \theta$.

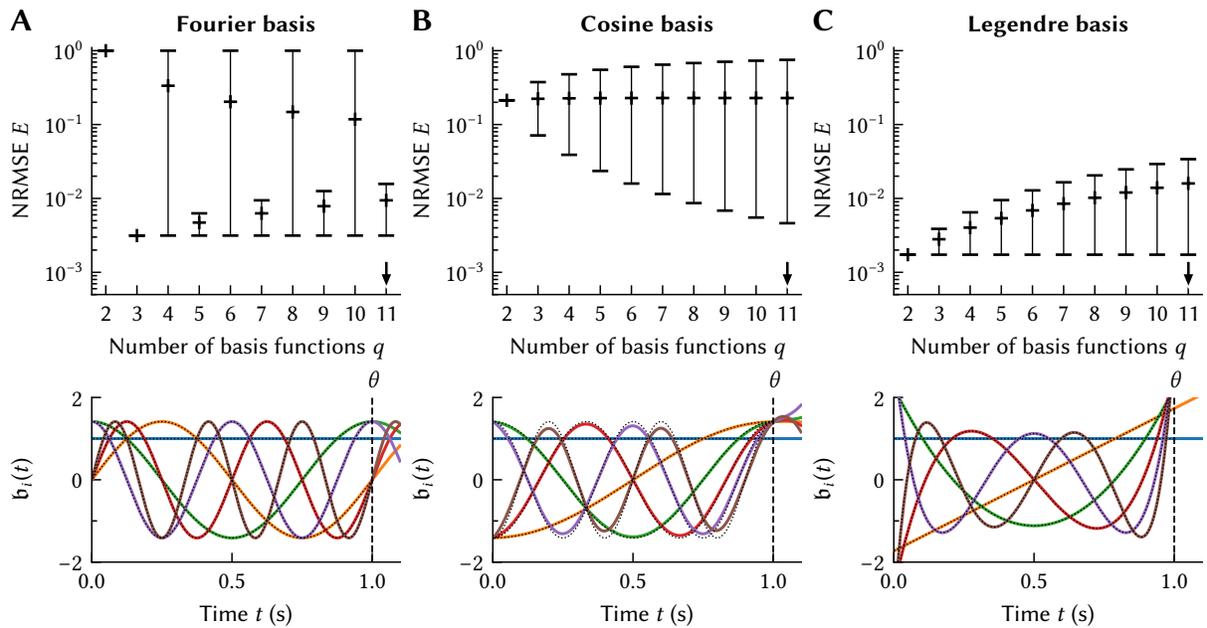


Figure 4.18: LTI systems generating the Fourier, cosine and Legendre basis. Constructed using eqs. (4.18) and (4.19) and $N = 1000$ samples. *Top:* NRMSE E over $[0, \theta]$ between the LTI impulse response and the reference basis functions (excluding the constant b_0). Black cross is the mean, error bars the minimum and maximum error over $q - 1$ basis functions. *Bottom:* First six dimensions of the reconstructed LTI system impulse response for $q = 11$ basis functions (cf. above arrow); dotted line is the reference. **(A)** The Fourier basis can be constructed well by an LTI system for odd q (mean $E < 10^{-2}$). **(B)** The cosine basis can only be realised as an LTI system with relatively large errors (mean $E > 10^{-1}$). Adding more basis functions reduces the error for earlier basis functions. **(C)** The Legendre basis can be realised well with small errors for all q (mean $E < 10^{-2}$).

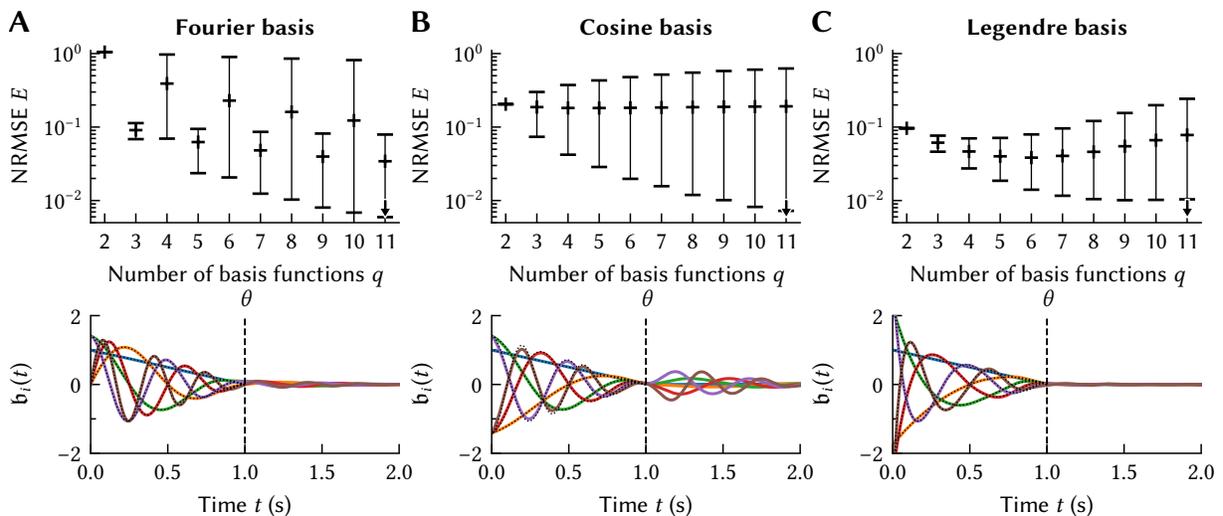


Figure 4.19: Windowed LTI systems generating the Fourier, cosine and Legendre basis. Same experiment as in Figure 4.18, but the target functions are multiplied with a linear window over $[0, \theta]$.

Establishing a rectangle window through “information erasure”. Optimally, we would like to realise a rectangle window. That is, the impulse response of our system should exactly trace out $\mathbf{b}_i(t)$ for $t \in [0, \theta]$, and then, at $t = \theta$, drop to zero. Phrasing this more colloquially, we must ensure that input older than θ seconds (or, in the discrete case, N samples) is “forgotten”.

Accomplishing this is indeed possible, and the fact that we are realising a function basis is key. Remember from Section 4.2.1 that we can decode a delayed version of the input signal $u(t - \theta)$ from the system state $\mathbf{m}(t)$. We can furthermore compute the specific contribution $\tilde{\mathbf{m}}(t)$ of $u(t - \theta)$ to $\mathbf{m}(t)$. Subtracting $\tilde{\mathbf{m}}(t)$ from $\mathbf{m}(t)$ effectively “erases” any information about $u(t - \theta)$ from the state vector, resulting in a rapidly decaying and almost finite impulse response.

We discuss this more thoroughly for continuous bases in the next subsection. For now, in the discrete case, assume that the impulse response of the discrete LTI system $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}$ perfectly reconstructs the basis transformation matrix $\mathbf{E} \in \mathbb{R}^{q \times N}$. We can then reconstruct the input sample u_{t-N+1} from \mathbf{m}_t using the Moore-Penrose pseudo-inverse \mathbf{E}^+ :

$$u_{t-N+1} = (\mathbf{E}^+)_N \mathbf{m}_t, \quad \text{where } \mathbf{E}^+ = (\mathbf{E}^T \mathbf{E})^{-1} \mathbf{E}^T.$$

Let “ \otimes ” denote the outer product. The contribution of u_{t-N+1} to the state \mathbf{m}_t is then given as

$$\tilde{\mathbf{m}}_t = (\mathbf{E}^T)_N u_{t-N+1} = (\mathbf{E}^T)_N \otimes (\mathbf{E}^+)_N \mathbf{m}_t.$$

Sequentially interleaving an “update” and an “erasure” step we obtain

$$\mathbf{m}'_t \leftarrow \tilde{\mathbf{A}} \mathbf{m}_{t-1} + \tilde{\mathbf{B}} u_t, \quad (\text{Update})$$

$$\mathbf{m}_t \leftarrow \mathbf{m}'_t - \tilde{\mathbf{m}}_t = (\mathbf{I} - (\mathbf{E}^T)_N \otimes (\mathbf{E}^+)_N) \mathbf{m}'_t. \quad (\text{Erasure})$$

$$\mathbf{m}_t \leftarrow (\mathbf{I} - (\mathbf{E}^T)_N \otimes (\mathbf{E}^+)_N) \tilde{\mathbf{A}} \mathbf{m}'_{t-1} + (\mathbf{I} - (\mathbf{E}^T)_N \otimes (\mathbf{E}^+)_N) \tilde{\mathbf{B}} u_t = \tilde{\mathbf{A}}' \mathbf{m}_{t-1} + \tilde{\mathbf{B}}' u_t. \quad (4.20)$$

Applying the inverse discretisation from eq. (4.19) to $\tilde{\mathbf{A}}', \tilde{\mathbf{B}}'$ results in a dampened, continuous LTI system. Importantly, this is only an optimal solution if $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$ perfectly reconstruct the discrete function basis. Furthermore, this technique cannot work with periodic bases (i.e., the Fourier basis), since there is technically no difference between decoding $u(t)$ and $u(t - \theta)$.

Figure 4.20 depicts the effects of this “information erasure” procedure on the LTI systems generating a modified Fourier, cosine and Legendre bases. Specifically, we slow the oscillations of the Fourier basis by 10%, making the basis aperiodic and compatible with information erasure. This new *modified Fourier basis* is no longer orthonormal, and thus information-theoretically suboptimal. For both the modified Fourier and Legendre basis, our method indeed approximates a rectangle window, yet introduces ringing artefacts. The method stabilises the cosine basis, but there are still residual oscillations for $t > \theta$ —this is likely because the cosine basis is not realised well by our LTI system (see previous experiments).

The feedback matrices \mathbf{A} are depicted in Figure 4.21; their regularity suggests that we might be able to construct these matrices analytically. For example, the Fourier system feedback matrices describe harmonic oscillators, with information erasure adding a dampening term. As we show next, the Legendre system can similarly be derived from first principles.

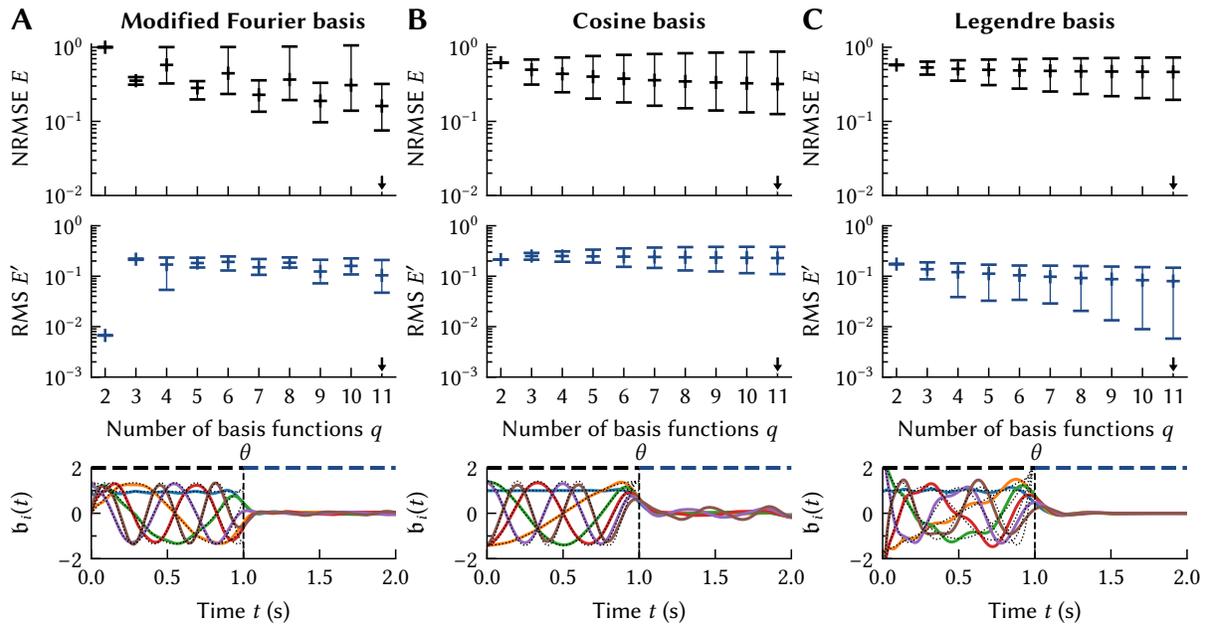


Figure 4.20: Effect of the “information erasure” technique (eq. 4.20) on LTI systems generating function bases. *Top, bottom:* Same analysis as in Figure 4.19. *Middle:* RMS E' of the response over $[\theta, 2\theta]$. Blue cross is the mean, bars are the minimum and maximum over the last $q - 1$ basis functions. **(A)** We use a modified Fourier basis with halved frequency (see text). Reconstruction errors E are small compared to the other bases; the basis quickly decays to zero. **(B)** The dampened LTI system generating the cosine basis is stable, but possesses substantial oscillations for $t > \theta$. **(C)** Dampening introduces substantial ringing artefacts in the Legendre basis, but the impulse response quickly decays to zero.

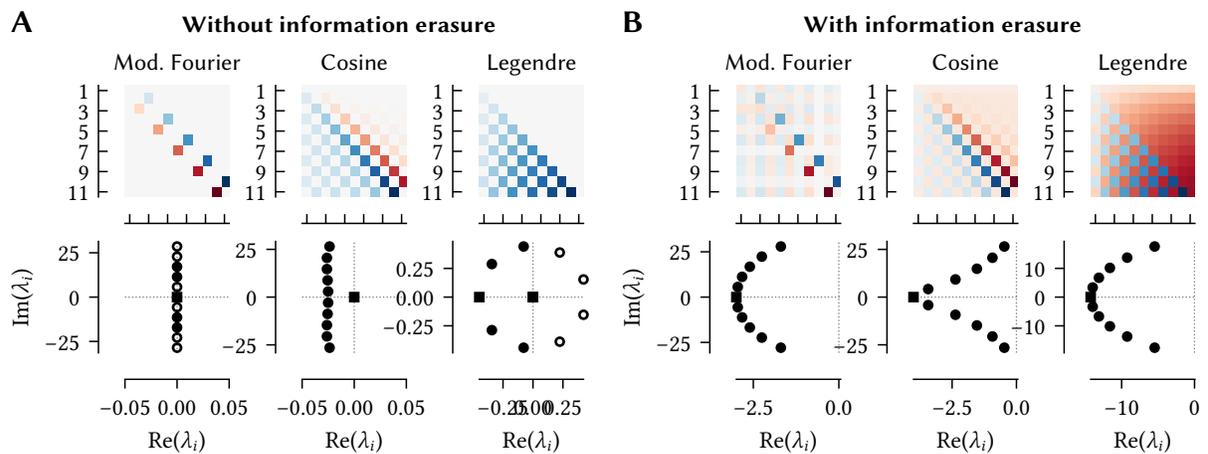


Figure 4.21: Feedback matrices generating LTI systems and their eigenvalues. *Top:* Visualisation of the feedback matrices A for $q = 11$, red corresponds to negative, blue to positive values, grey is zero. *Bottom:* Real and imaginary components of the corresponding eigenvalues. **(A, B)** With and without information erasure. Note the regular structure of **A**. Information erasure ensures stability, i.e., negative $\text{Re}(\lambda_i)$. The eigenvalues of the dampened Fourier and Legendre systems are qualitatively similar.

4.2.4 Closed-Form Solution for the Legendre Basis

While our numerical method for approximating function basis generating LTI systems works reasonably well, we can, in some cases, analytically determine LTI systems that generate bases over an approximated rectangle window. For example, as we show next, we can analytically construct an LTI system \mathbf{A}' , \mathbf{B}' that *exactly* traces out the Legendre polynomials. To this end, we use a continuous variant of our above information erasure technique to construct a dampening term Γ that limits the impulse response of the system to $[0, \theta]$. Subtracting Γ from \mathbf{A}' results in a rescaled version of the LTI system underlying the Legendre Delay Network (LDN; Voelker and Eliasmith, 2018). This idea is similar to Gu et al. (2020) and can, as we discuss in more detail in Stöckel (2021a), be applied to arbitrary polynomial bases.

Lemma 4.2. *The impulse response of the linear time-invariant system $\theta \dot{\mathbf{m}}(t) = \mathbf{A}'\mathbf{m}(t) + \mathbf{B}'u(t)$ with $\mathbf{m} \in \mathbb{R}^q$, $\mathbf{A}' \in \mathbb{R}^{q \times q}$ and $\mathbf{B}' \in \mathbb{R}^{q \times 1}$*

$$(\mathbf{A}')_{ij} = (4j - 2) \begin{cases} 0 & \text{if } i \leq j \text{ or } i + j \text{ is even,} \\ 4j - 2 & \text{if } i > j \text{ and } i + j \text{ is odd,} \end{cases} \quad (\mathbf{B}')_i = (-1)^{i+1},$$

are the first q shifted Legendre polynomials $\tilde{P}_n(t\theta^{-1})$ scaled to $\tilde{P}_n(1) = 1$ over $t \in [0, \theta]$.

We provide a proof of this lemma in Appendix A.3.5; see Figure 4.22A for an example. As before, we now need to limit the impulse response of this system to a rectangle window $[0, \theta]$. This would be trivial if we had access to a delayed version $u(t - \theta)$ of the signal:

Lemma 4.3. *Let \mathbf{A} , \mathbf{B} describe an LTI system and let $u(t)$ be some input signal. The impulse response of the following modified system is unchanged compared to the original LTI system for $0 \leq t < \theta$ but zero for all $t \geq \theta$*

$$\dot{\mathbf{m}}(t) = \mathbf{A}\mathbf{m}(t) + \mathbf{B}u(t) - \mathbf{e}(\theta)u(t - \theta), \quad \text{where } \mathbf{e}(\theta) = \exp(\mathbf{A}\theta)\mathbf{B}.$$

Of course, we usually do not have access to the delayed input signal $u(t - \theta)$. However, as we discussed in Sections 4.2.1 and 4.2.3, we can decode an approximate $u(t - \theta')$ from the state $\mathbf{m}(t)$ using a delay decoder $\mathbf{d}(\theta')$. Below, we define the term “delay decoder” more precisely for input signals $\hat{u}(t - \theta)$ that are expressible using q basis functions.

Definition 4.8 (Delay decoder). *Let $\hat{u} : [0, \theta] \rightarrow \mathbb{R}$ be expressible as a linear combination of q basis functions $\mathbf{b}_n : [0, \theta] \rightarrow \mathbb{R}$ with generalised Fourier coefficients χ_i . Let $\mathbf{m}(t)$ be the state of a q -dimensional system with impulse responses \mathbf{b}_n and input \hat{u} , i.e.,*

$$m_i(t) = \int_0^\theta \mathbf{b}_i(\tau) \hat{u}(t - \tau) d\tau = \int_0^\theta \mathbf{b}_i(\tau) \sum_{j=0}^{q-1} \chi_j \mathbf{b}_j(t - \tau) d\tau.$$

Then $\mathbf{d}(\theta') = (d_0(\theta'), \dots, d_{q-1}(\theta'))$ is called a delay decoder if

$$\hat{u}(\theta - \theta') = \sum_{j=0}^{q-1} \chi_j \mathbf{b}_j(\theta - \theta') = \langle \mathbf{d}(\theta'), \mathbf{m}(\theta) \rangle = \sum_{i=0}^{q-1} d_n(\theta') \int_0^\theta \mathbf{b}_i(\tau) \sum_{j=0}^{q-1} \chi_j \mathbf{b}_j(\theta - \tau) d\tau. \quad (4.21)$$

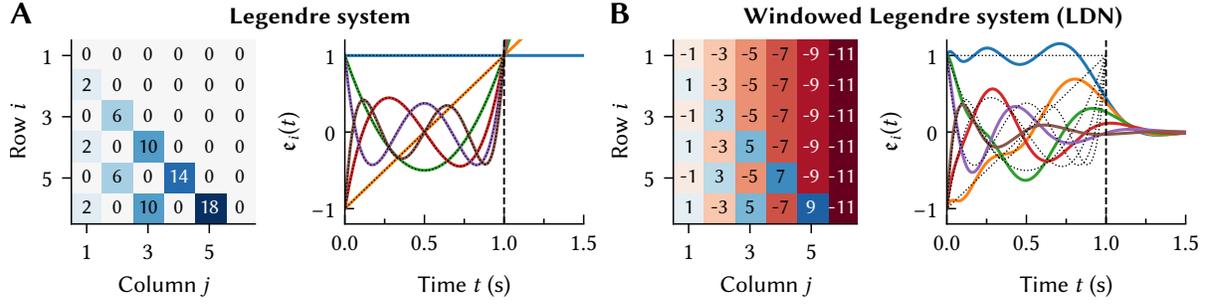


Figure 4.22: Feedback matrices and impulse responses of the Legendre system. *Left:* The feedback matrix of the corresponding system for $q = 6$ and $\theta = 1$ s. *Right:* The corresponding impulse response. **(A)** The undamped Legendre system \mathbf{A}' , \mathbf{B}' as described in Lemma 4.2. **(B)** The Legendre system \mathbf{A} , \mathbf{B} with approximate rectangle window (cf. eq. 4.24).

Lemma 4.4. For the shifted Legendre polynomials $\tilde{P}_i(t\theta^{-1})$ with $\tilde{P}_i(1) = 1$, the delay decoder is

$$d_i(\theta') = \frac{2i+1}{\theta} \tilde{P}_i(\theta'\theta^{-1}).$$

This follows from the orthogonality of $\tilde{P}_i(t)$. We provide a proof in Appendix A.3.7.

Given the concept of a “delay decoder” we can construct an approximate version of the windowed LTI system in Lemma 4.3 that reconstructs $u(t - \theta)$ from the system state $\mathbf{m}(t)$:

$$\dot{\mathbf{m}}(t) = \mathbf{A}\mathbf{m}(t) + \mathbf{B}u(t) - \mathbf{e}(\theta)\langle \mathbf{d}(\theta), \mathbf{m}(t) \rangle = (\mathbf{A} - \mathbf{\Gamma})\mathbf{m}(t) + \mathbf{B}u(t), \quad (4.22)$$

where the dampening term $\mathbf{\Gamma} = \mathbf{e}(\theta) \otimes \mathbf{d}(\theta)$ is the *delay re-encoder*. For the shifted Legendre polynomials $\tilde{P}_i(t\theta^{-1})$ with $\tilde{P}_i(1) = 1$ the resulting delay re-encoder is given as

$$(\mathbf{\Gamma})_{ij} = e_i(\theta)d_j(\theta) = \frac{2j+1}{\theta} \tilde{P}_{i-1}(\theta\theta^{-1})\tilde{P}_{j-1}(\theta\theta^{-1}) = \frac{2j+1}{\theta}. \quad (4.23)$$

Correspondingly, the feedback matrix $\mathbf{A} = \mathbf{A}' - \mathbf{\Gamma}$ and the input vector \mathbf{B} of the LTI system generating the Legendre polynomials with approximated rectangle window are given as

$$(\mathbf{A})_{ij} = \frac{(2j-1)}{\theta} \begin{cases} -1 & \text{if } i \leq j, \\ (-1)^{i-j+1} & \text{if } i > j, \end{cases} \quad (\mathbf{B})_i = \frac{(-1)^{i+1}}{\theta}. \quad (4.24)$$

The impulse response for $q = 6$ of this system is depicted in Figure 4.22B.

If we divide each state dimension by $(2i - 1)$, we obtain exactly the same system as the one described by Voelker (2019, Section 6.3.1, pp. 133-135) in the context of the Legendre delay network. Notably, this division operation simply swaps the $(2j - 1)$ scaling factor in \mathbf{A} with $(2i - 1)$. The LDN system has been derived from the Padé approximants of a Laplace domain delay $e^{-\theta s}$ and a subsequent coordinate transformation (Voelker, 2019). From this perspective, the similarity of the impulse response to the Legendre polynomials is rather coincidental; our derivation strengthens the connection between the Legendre polynomials and the LDN.

4.2.5 Decoding Delays as a Benchmark for Temporal Bases

In theory, the different continuous and discrete function bases discussed in Section 4.2.2 possess the same representational power. The continuous bases span the function space $L^2(0, \theta)$, and the discrete bases span the N -dimensional vector space \mathbb{R}^N . However, we expect to see differences in how well we can decode functions as soon as we truncate the bases to q terms, and even more so when generating the bases as the impulse response of an LTI system of order q .

One way to characterise such non-ideal temporal bases is to measure in how far we can decode delayed versions of the input signal $u(t)$ from the generalised Fourier coefficients $\mathbf{m}(t)$ (cf. Voelker and Eliasmith, 2018). This is a reasonable benchmark, since performing well in this task has implications beyond just computing delays. Remember that a convolution of $u(t)$ with some kernel $\mathbf{c}(t) : [0, \theta] \rightarrow \mathbb{R}$ is a weighted integral over delayed $u(t)$:

$$(u * \mathbf{c})(t) = \int_0^\theta u(t - \theta') \mathbf{c}(\theta') d\theta' = \int_0^\theta \langle \mathbf{m}(t), \mathbf{d}_{\theta'} \rangle \mathbf{c}(\theta') d\theta' = \langle \mathbf{m}(t), \mathbf{c} \rangle,$$

where the vector \mathbf{c} is the “filter decoder” mentioned in Section 4.2.1. Hence, being able to decode delays well implies that we can approximate any linear filter \mathbf{c} with a small error.

Comparing windowed LTI systems to a low-pass filter basis. Figure 4.23 depicts delayed versions of a low-pass filtered white-noise signal $u(t)$ (cutoff frequency at 5 Hz)¹⁴ decoded from a $q = 11$ dimensional state vector $\mathbf{m}(t)$ that has been obtained by simulating state-space LTI systems. The first-order low-pass filters (Figure 4.23A) are implemented in terms of their differential equations. For the cosine and modified Fourier basis (i.e., a Fourier basis with a 10% lower frequency) we use autoregression with information erasure to approximate the LTI state-space matrices. We use the LDN system eq. (4.24) to generate the Legendre basis.¹⁵

We compute the delay decoders $\mathbf{d}_{\theta'}$ using linear least-squares for independent training signals. The reported error E is the mean NRMSE between the ground-truth signal $u(t - \theta')$ (with $u(t) = 0$ for $t < 0$) and the decoded delay $\hat{u}(t - \theta')$ for $\theta' \in [0, \theta]$ and $\theta = 1$ s.

In this particular example, the modified Fourier system ($E \approx 29\%$) outperforms both the cosine and the Legendre system, with the latter two reaching similar errors ($E \approx 41\%$). As was already suggested by our earlier principal component analysis (cf. Section 4.1.3), and as is clearly visible in these results, low-pass filters ($E \approx 80\%$) do not form a suitable basis for decoding delays—we hence ignore the low-pass filter basis in the following experiments.

¹⁴Note that the “cutoff frequency” is a technical term from signal processing and, by convention, denotes the frequency at which the output is dampened by -3 dB; this is not to be confused with the *band-limited* signals we used before. In particular, we use a fourth-order Butterworth filter (e.g. Oppenheim et al., 2009, Section 7.3).

¹⁵The LDN system as derived in Section 4.2.4 exactly traces out the standard shifted Legendre polynomials (i.e., with the normalisation $\tilde{P}_i(1) = 1$), while the original LDN (Voelker, 2019) scales the i th state dimension by $(2i + 1)$. Hence, our normalisation requires larger decoding weights, increasing the regularisation error. However, the incurred additional decoding error is negligible in our examples (about 10^{-4} for $q = 63$).

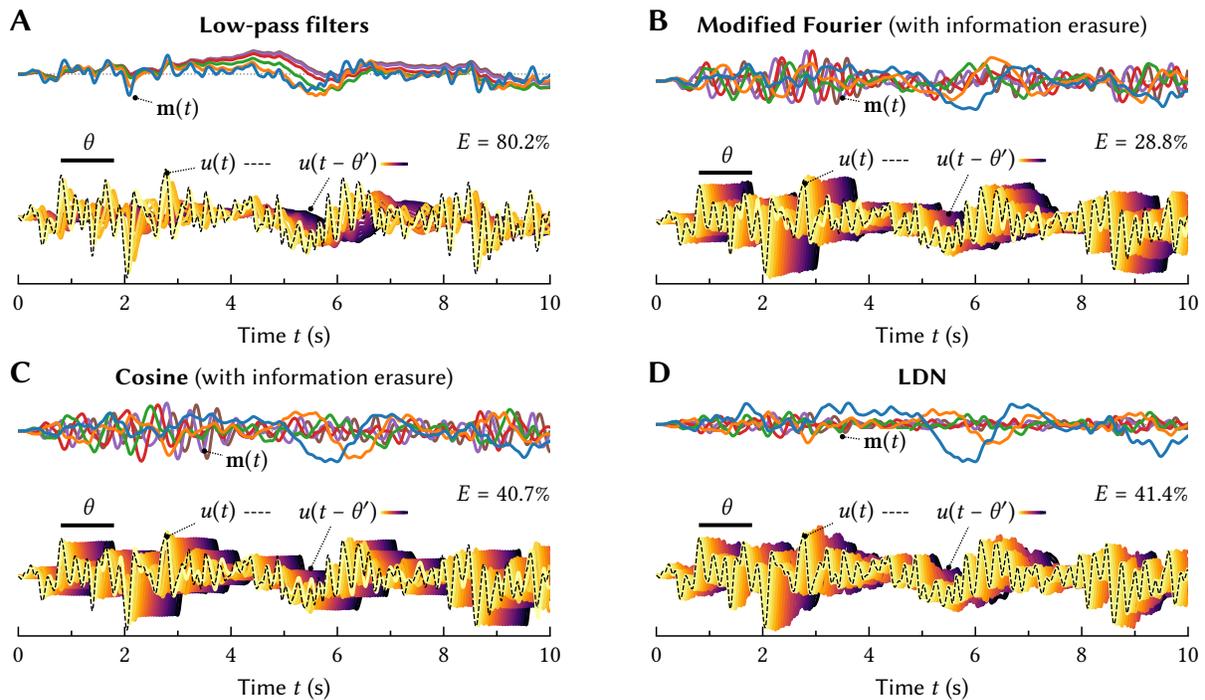


Figure 4.23: Computing delays as a function basis benchmark. *Top:* Six state dimensions $\mathbf{m}(t)$ of the underlying LTI system (order $q = 11$) for an input $u(t)$. *Bottom:* Input $u(t)$ (dashed black line), and delayed versions decoded from $\mathbf{m}(t)$ (coloured lines). Depicted error E is the mean NRMSE between the ground-truth $u(t - \theta')$ (assuming $u(t) = 0$ for $t < 0$) and the decoded delays. **(A)** Low-pass filters with time-constants τ between 10 ms and 10 s. Only small delays can be decoded well. **(B, C, D)** Systems generating the modified Fourier, cosine, and Legendre bases with approximated rectangle windows through information erasure. The modified Fourier system achieves the smallest error.

Systematic analysis. Figures 4.24 and 4.25 depict a more thorough analysis of the different basis functions and windowing methods discussed in this section, including the original Fourier basis. We perform the same experiment as above, but also sweep over $q \in \{1, 3, 5, \dots, 63\}$ and repeat the experiment for $N = 100$ randomly sampled inputs $u(t)$ (same parameters as above).

We first compare the different bases under the assumption that we can realise a perfect rectangle window (Figure 4.24A). That is, we keep the entire signal history over $[0, \theta]$ in memory and convolve with the ideal basis functions. There is no significant difference between the Fourier and cosine bases (Figure 4.25A) while the Legendre basis performs slightly worse. In particular, note the “n”-shape of the error for the Legendre basis—near $\theta'/\theta = 0.5$ this basis can only represent lower-frequency content; it is not as “expressive”, leading to higher errors.

Of course, realising the bases as LTI systems increases the measured error (Figures 4.24B and 4.24C). The decoding error over θ' is (noise aside) strictly monotone. Since we do not have $u(t)$ in memory, information lost from $\mathbf{m}(t)$ cannot be recovered; the error for the Legendre system only increases past $\theta'/\theta = 0.5$. As predicted, the cosine basis cannot be realised well, and forcing a rectangle-window onto the unmodified Fourier basis results in large errors.

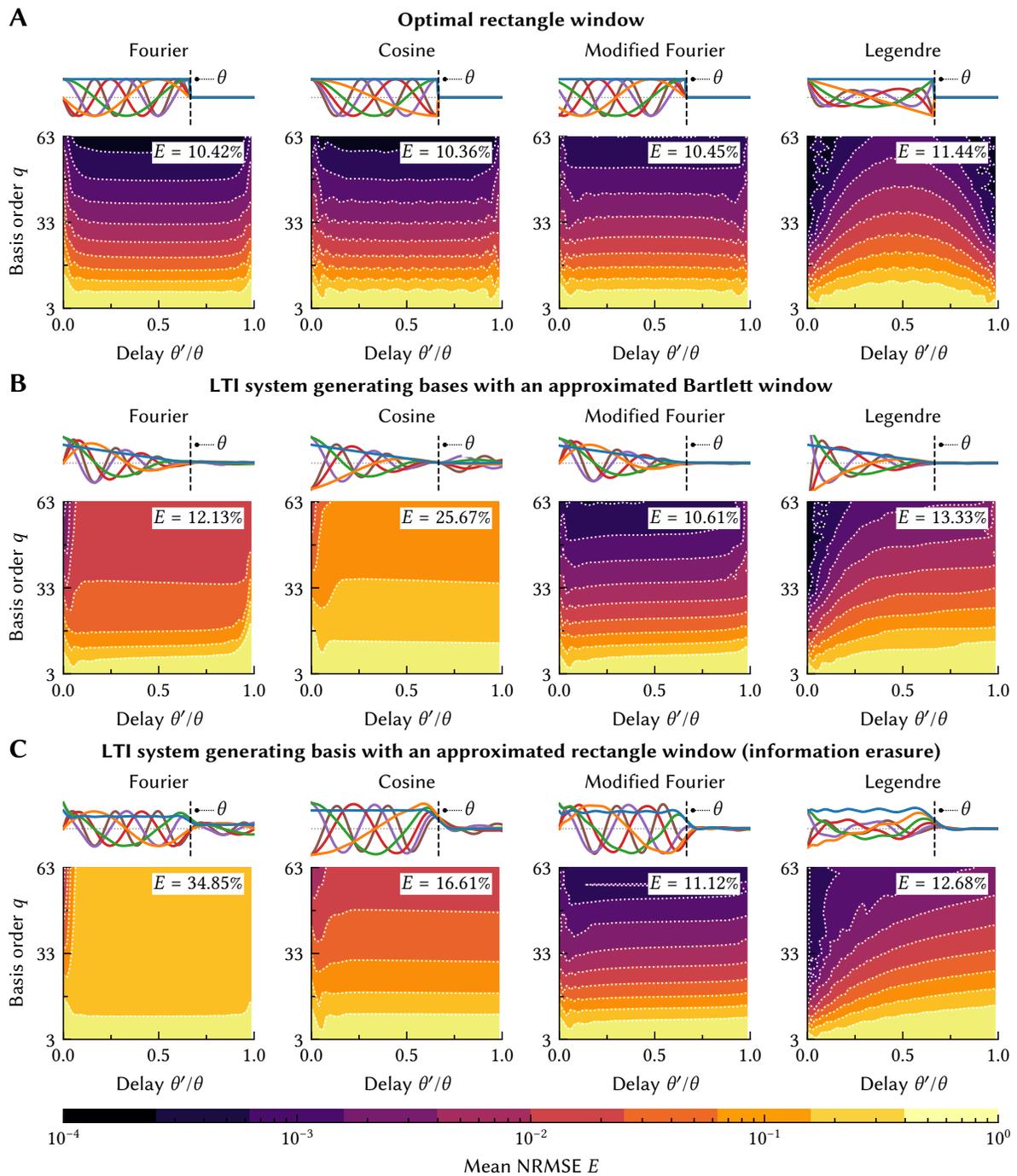


Figure 4.24: Systematic analysis of different bases and windowing methods. *Top:* First six basis functions (A) or the impulse response of the first six state dimensions (B, C). *Bottom:* Mean decoding error for different delays θ' and basis function counts q over $N = 100$ trials. E is the mean error over the entire area. (A) Using basis functions with an optimal rectangle window. (B, C) Using the impulse response of an LTI system with a one-sided Bartlett window (B), or the approximated rectangle window (C). We use the analytical LDN system (eq. 4.24) in (C). A statistical analysis of is provided in Figure 4.25.

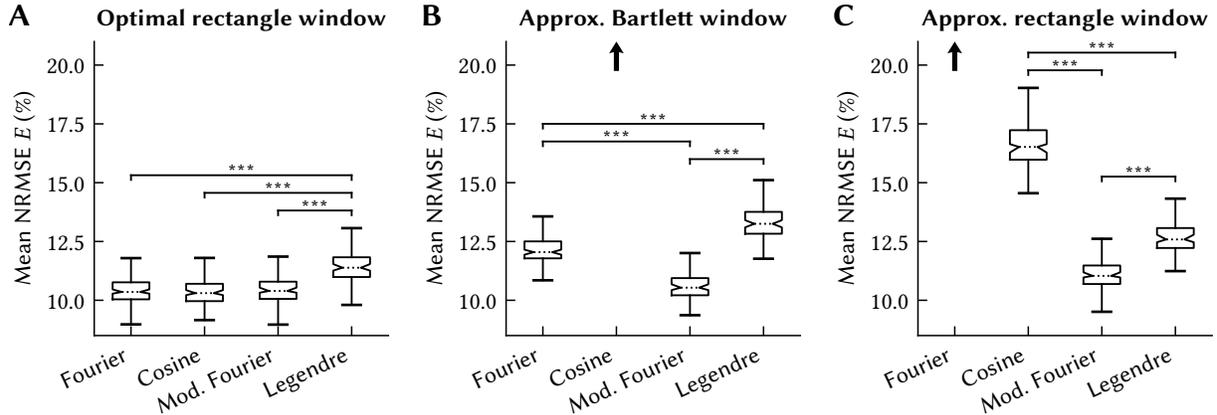


Figure 4.25: Mean delay decoding error statistics. Statistics of the mean NRMSE E depicted in Figure 4.24. Each box plot is over the $N = 100$ different test signals $u(t)$; the mean error values are over all delays θ' and q for each test signal. Boxes are the quartiles, notches the 99% confidence interval. Three stars indicate statistical significance according to a Kolmogorov-Smirnov test ($p < 0.1\%$).

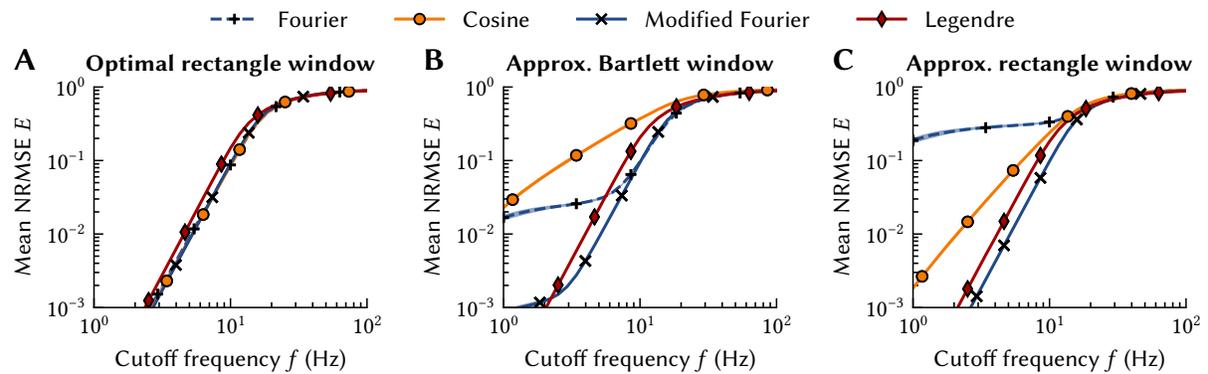


Figure 4.26: Mean delay decoding error over the input signal frequency. Depicted is the median over $N = 100$ input signals $u(t)$ of the mean delay decoding error with respect to the tested $\theta'/\theta \in [0, 1]$; q is fixed at $q = 31$. Shaded areas (barely visible) are the 25th and 75th percentile.

Surprisingly, at least in this noise-free environment, implementing the rather simplistic Bartlett window only incurs a small penalty for larger θ' ; we can still decode information near $\theta'/\theta = 1$. Furthermore, the modified Fourier system consistently reaches significantly smaller errors than the Legendre system (Figures 4.25B and 4.25C). This is still true when keeping q constant and sweeping over frequency content of $u(t)$ (Figure 4.26).

Discussion. The modified Fourier system outperforming the LDN is at odds with the LDN system being an *optimal* approximation of a state-space delay (Voelker, 2019, Section 6.1.1). One potential caveat is that the LDN is only optimal for input $u(t)$ generated by a system of order $2q - 1$, while our low-pass filtered $u(t)$ is technically of infinite order. Using band-limited $u(t)$ indeed reduces the delay decoding error (Appendix B.3.1), yet the modified Fourier basis still significantly outperforms the LDN. The exact reason for this is unclear; future research should characterise the exact conditions under which these LTI systems are optimal.

4.3 Accounting for Neural Nonlinearities

Minimising the linear quadratic loss in eq. (4.8) results in weights that—in some *linear* networks (cf. Section 4.1.3)—optimally realise LTI systems while at the same time compensating for synaptic filters. As we mentioned before, this focus on linear networks is not a major obstacle. Linear activations can be emulated in networks of nonlinear neurons using NEF identity decoders (cf. Section 2.3.3). Being able to do this is not just of theoretical interest. To the contrary, implementing linear dynamics on top of nonlinear networks is useful in practice. As we discuss in Chapter 5, we can for example build an adaptive filter by transforming the state $\mathbf{m}(t)$ represented in the network.

Still, it can also be beneficial to solve for dense weight matrices that realise temporal tuning in networks of nonlinear neurons. First and foremost, this allows modellers to finely control the temporal properties of individual neurons. Furthermore, the temporal tuning curve paradigm suggests a natural way to represent multi-dimensional quantities over time. Lastly, we can take heterogeneous synaptic filters into account, now at the level of individual synapses (cf. Figure 2.30B).

In this section, we discuss solving for synaptic weights in nonlinear networks. To this end, we propose a method for generating input signals \mathbf{x}_k that uniformly cover the neural activity space. We then confirm in several experiments that our methods can be used to successfully build networks with the intended properties.

4.3.1 Sampling Input Signals and Temporal Encoders

On the surface, switching from linear units to neurons with nonlinear response curves, G , does not change our weight optimisation problem. Recall that our least-squares loss was given as

$$E = \sum_{k=1}^N \left[J_i \left(\int_0^\infty \langle \mathbf{e}_i(\tau), f(\mathbf{x}_k * \delta_\tau) \rangle d\tau \right) - \sum_{j=1}^m w_{ij} \int_0^\infty h_{ij}(\tau) a_j(\mathbf{x}_k * \delta_\tau) d\tau \right]^2. \quad (4.8)$$

In linear networks, the current-translation function J_i and the pre-population tuning curve \mathbf{a}_j are linear. While we continue to assume that J_i is linear,¹⁶ the tuning-curve \mathbf{a}_j is now nonlinear in \mathbf{x}_k due to the neural response curve G . Additionally, each population now consists of hundreds of neurons, compared to the few dozen linear units in our previous examples.

This poses two new challenges. First, it is unclear how to assign temporal tuning curves to each neuron. Before, we had a one-to-one mapping between linear units and the impulse responses of the LTI system we wanted to realise; we now need to assign q impulse responses to n neurons, where typically $n \gg q$. Second, the selection of input signals \mathbf{x}_k is more difficult. Whereas the magnitude of \mathbf{x}_k played no role in linear networks, scaling \mathbf{x}_k in nonlinear networks profoundly impacts neural activities. We must sample $\mathbf{a}_j(\mathbf{x}_k)$ densely enough to capture the curvature of the nonlinearity above its threshold.

¹⁶Remember that we saw in Section 3.2.1 that we can usually decode biases directly from the pre-population.

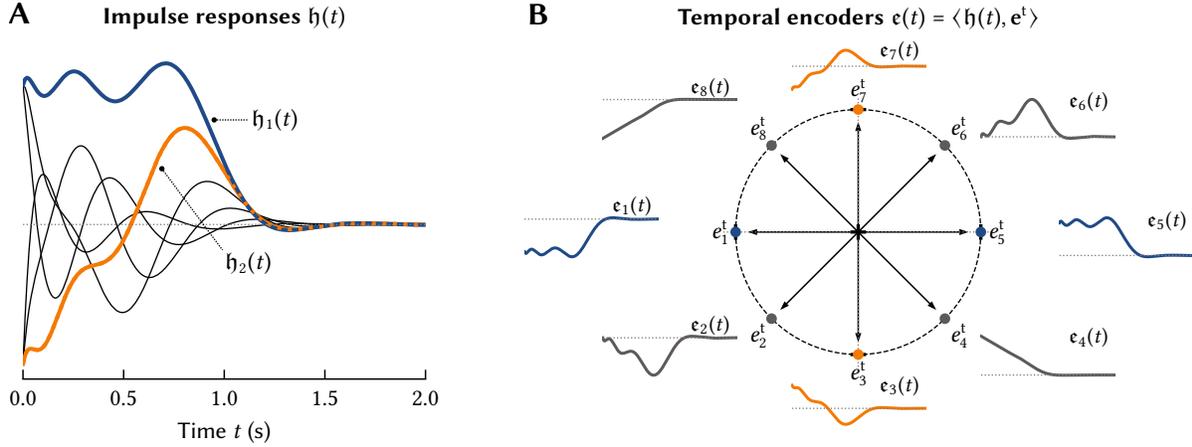


Figure 4.27: Linearly combining impulse responses to obtain temporal encoders. **(A)** Starting with a set of impulse responses $h_i(t)$, we can generate temporal tuning for each neuron by linearly combining these basis functions using an encoding vector e^t . **(B)** When choosing normalised e^t (e.g., $\|e^t\| = 1$), this vector is directly equivalent to the spatial encoder e when realising an LTI system as multiple spatial dimensions in the NEF; each “direction” corresponds to a different temporal encoding.

Temporal encoding vectors. One method for realising a q -dimensional LTI system with state-space matrices A , B in a nonlinear network with n neurons is to simply set each c_i to a linear combination of the system impulse responses h_j (cf. Figure 4.27):

$$c_i(t) = \sum_{j=1}^q e_j^t \exp(At)B = \sum_{j=1}^q e_j^t h_j(t) = \langle e_i^t, h(t) \rangle, \quad (4.25)$$

where $e_i^t \in \mathbb{S}^q$ is a *temporal encoding vector*, and \mathbb{S}^q is the q -dimensional unit sphere. Since e_i^t is normalised to unit length, e_i^t is *exactly equivalent* to the spatial encoding vector e_i when constructing an LTI system of order q using the NEF dynamics principle.

The difference between the dynamics principle and temporal tuning curves is largely conceptual. Using the NEF dynamics principle, our neuron population represents a q -dimensional quantity, while, using temporal tuning, the population represents a scalar—the individual neurons possess different temporal tuning, which we can exploit when solving for weights.

Spatiotemporal tuning. Interestingly, the temporal tuning paradigm suggests a way to combine temporal and spatial encoding (i.e., $e_i^t \in \mathbb{S}^q$ and $e_i \in \mathbb{S}^d$). Remember from Definition 4.2 that c_i is a *vectorial* quantity over time. A valid choice for c_i is thus

$$c_i(t) = e_i \langle e_i^t, h(t) \rangle = (e_i \otimes e_i^t) h(t) = E_i^t h(t), \quad (4.26)$$

where “ \otimes ” is the outer product, and $E_i^t \in \mathbb{R}^{d \times q}$ is a *spatiotemporal encoding matrix* with $\|E_i^t\|_F = 1$. Populations with this tuning represent d -dimensional quantities with temporal tuning of order q and can be harnessed to approximate *nonlinear* functions over space and time. We discuss this in more detail in Section 4.3.3. Curiously, the matrix E_i^t is also implicitly used in the Legendre Memory Unit (LMU; Voelker, Kajić, et al., 2019); we discuss this in Section 4.4.1.

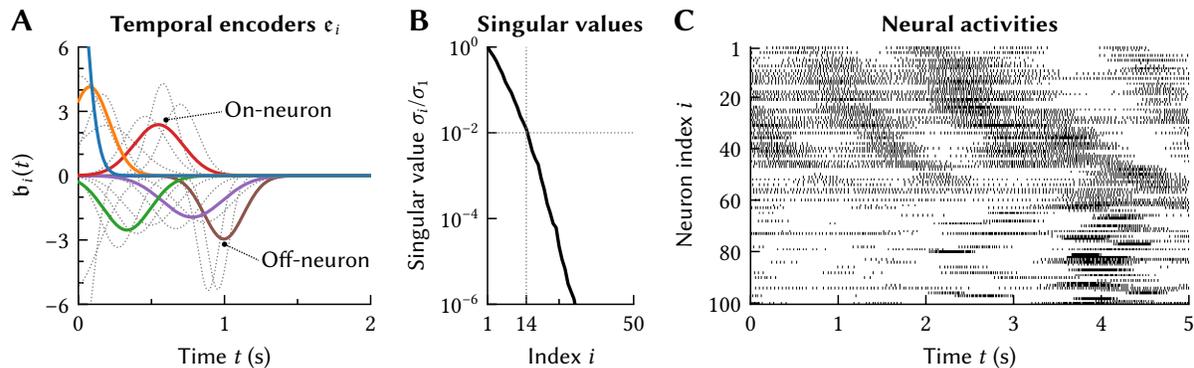


Figure 4.28: Example of non-orthogonal basis functions emulating time cells. **(A)** Radial basis functions with randomly distributed peak position, width, and sign (i.e., “on” and “off” neurons). Each temporal encoder is normalised to an area of one. **(B)** As indicated by the singular values, this set of functions is not of full rank; the first 14 singular values account for 99% of the variance. **(C)** Spike raster plot of a spiking neural network realising these temporal encoders. Each black line is a spike event. Note the diagonal “stripes” corresponding to different delayed versions of the band-limited noise input.

Implicitly solving for temporal bases. One of our goals is to implement networks that generate q windowed temporal basis functions b_1, \dots, b_q . To this end, we can proceed as discussed in Section 4.2: that is, we solve for an LTI system that approximately generates this basis, and use temporal encoding vectors \mathbf{e}_i^t to map the impulse response of this system onto neurons.

An alternative method is to directly set the temporal encoders to a linear combination of our desired windowed basis functions, in other words $\epsilon_i(t) = \langle \mathbf{e}_i^t, \mathbf{b}(t) \rangle w(t)$. This way, we implicitly solve for the dynamics of an LTI system that generates this system.¹⁷ Either way, the neurons in the population are tuned to a linear combination of q state dimensions.

As we explained in Section 4.1.1, we can also choose the ϵ_i according to empirical constraints. If we would like to emulate temporal tuning in visual cortex, then we select temporal encoders as in Figure 4.4. Or, to model time cells, we could explicitly set the temporal encoder of each neuron to a bell-shaped function with appropriately distributed peak times θ_i , akin to a radial basis (Broomhead et al., 1988; Stöckel, 2020). This works (cf. Figure 4.28), but, curiously, implementing temporal bases similarly induces time cells (see below).

From our experience, networks constructed by explicitly choosing ϵ_i can perform similarly (in terms of delay decoding errors), but do not outperform networks linearly combining orthogonal basis functions. This is likely because this approach often implies using highly redundant and non-orthogonal basis functions. Typically, the first few singular values of these (discretised) bases decays rapidly (cf. Figure 4.28B). Hence, although we can theoretically decode more functions (fewer singular values are zero), this requires larger decoding weights compared to an orthonormal basis, thus amplifying the noise present in spiking neural networks.

¹⁷In practice, this works particularly well for gradually decaying dynamics, for example exponential or Bartlett windows $w(t)$. Rectangle windows are better realised using our information erasure technique.

Uniform activation sampling. Given the above techniques for selecting temporal encoders $\mathbf{e}_i(t)$, the next challenge is to sample input signals $\mathbf{x}_k : [-T, 0] \rightarrow \mathbb{R}$ such that all neurons are activated uniformly. So far, we mostly relied on low-pass filtered white noise. We can generate such “coloured” noise \mathbf{x}_k with bandwidth ρ by sampling spectral coefficients X_ℓ :¹⁸

$$\mathbf{x}_k(t) = \sum_{\ell=0}^{\infty} X_{k\ell} f_\ell(-tT^{-1}) \quad \text{where } X_\ell \sim \mathcal{N}(0, \exp(-F_\ell^2 \rho^{-2})) \text{ and } F_\ell = \lfloor (\ell + 1)/2 \rfloor. \quad (4.27)$$

Here, f_ℓ is a basis function in the Fourier series (eq. 4.12), and F_ℓ is the corresponding frequency.

Now, assuming that we realise an LTI system of order q in a network with temporal encoding vectors \mathbf{e}_i^t (eq. 4.26), and that $J_i(\xi)$ is part of the response curve $G_i(\xi)$ (where ξ is the “activation”), we can write the temporal tuning curve $a_i(\mathbf{x}_k)$ (cf. Definition 4.2) for scalar \mathbf{x}_k as

$$a_i(\mathbf{x}_k) = G_i \left[\sum_{j=0}^q \mathbf{e}_j^t \int_0^T \mathbf{x}_k(-\tau) \mathfrak{h}_j(\tau) d\tau \right] = G_i \left[\sum_{j=0}^q \mathbf{e}_j^t \sum_{\ell=0}^{\infty} X_{k\ell} H_{j\ell} \right], \quad \text{where } \mathfrak{h}_j(t) = \sum_{\ell=0}^{\infty} H_{j\ell} f_\ell(tT^{-1}).$$

Here, we exploit that the product of the generalised Fourier coefficients of two signals (with one being time reversed) corresponds to evaluating their convolution at $t = 0$. Making the equivalence of linear temporal tuning to the NEF dynamics principle more obvious, we can write this as $a_i(\mathbf{x}_k) = G_i[\langle \mathbf{e}^t, \mathbf{x}^t(\mathbf{x}_k) \rangle]$, where $\mathbf{x}^t(\mathbf{x}_k) = \mathbf{x}_k^t$ is the result of convolving \mathbf{x}_k with all \mathfrak{h}_j .

Notably, for most LTI systems, projecting the coloured noise signals from eq. (4.27) onto the impulse responses does not result in uniformly distributed \mathbf{x}_k^t (cf. Figure 4.29A). In contrast, when solving for non-temporal decoders within the NEF, we typically uniformly sample \mathbf{x} from a unit-ball \mathbb{B}^d (cf. section 2.3.3). This ensures that $\xi = \langle \mathbf{e}, \mathbf{x} \rangle$, is approximately uniform; correspondingly, the curvature of the response curve $G_i[\xi]$ is well captured in the training samples.

One method for generating uniformly distributed \mathbf{x}_k^t is to set \mathbf{x}_k to a linear combination of the impulse responses \mathfrak{h}_j . However, in low-order systems, this biases the weight solver to a relatively small repertoire of potential inputs.

We instead propose to sample both \mathbf{x}_k^t and \mathbf{x}_k , and to then solve for an \mathbf{x}'_k such that $\mathbf{x}^t(\mathbf{x}'_k) = \mathbf{x}_k^t$. In practice, we sample \mathbf{x}_k^t from an optimal Halton sequence uniformly mapped onto the unit-ball (Chi, Mascagni, and Warnock, 2005; Fang and Wang, 1994, Section 1.5).¹⁹ Finding the spectral coefficients $X'_{k\ell}$ can be phrased as a linearly-constrained least-squares problem:

$$\min \sum_{\ell=0}^{\infty} \exp(F_\ell^2 \rho^{-2}) (X'_{k\ell} - X_{k\ell})^2 \quad \text{subject to} \quad \sum_{j=1}^q \sum_{\ell=0}^{\infty} X'_{k\ell} H_{j\ell} = (\mathbf{x}_k^t)_\ell,$$

where ρ is the bandwidth and F_ℓ the frequencies from eq. (4.27). Weighting the quadratic term by these factors ensures that high-frequency coefficients are not altered over-proportionally. This optimisation problem can be easily solved using Lagrange multipliers (cf. Boyd et al., 2004, Chapter 5). Results of using this method are depicted in Figure 4.29B, and the effect on the computed weights in the context of an integrator is illustrated in Figure 4.30.

¹⁸In practice, the infinite sums over the spectral coefficients are limited to a few dimensions; $X_{k\ell}$ tends to be negligibly small for larger ℓ due to the exponentially decaying power spectrum.

¹⁹This is inspired by a similar approach in “NengoLib” (<https://github.com/arvoelke/nengolib>).

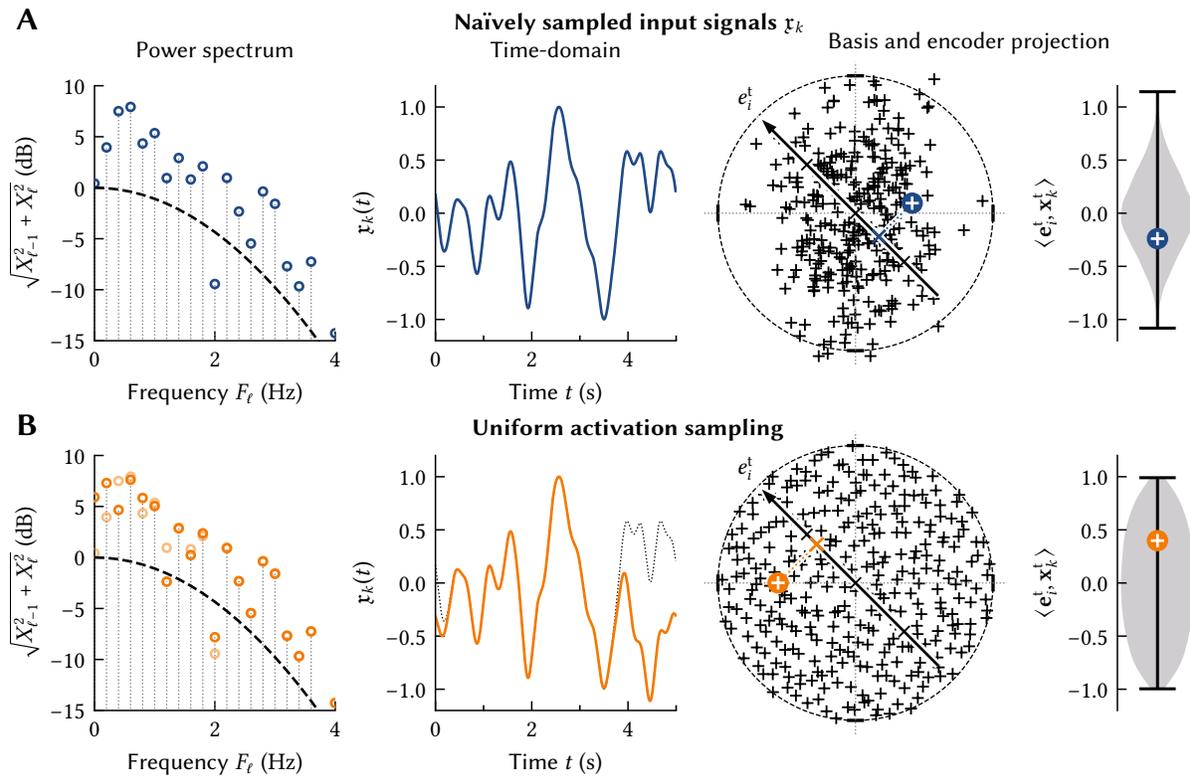


Figure 4.29: Illustration of uniform activation sampling. **(A)** Sampling low-pass filtered noise input signals \mathfrak{x}_k (example power spectrum and time-domain signal in the left half) and computing the projection onto the LTI system impulse responses \mathbf{x}^t (black crosses; same underlying impulse responses as in Figure 4.27) reveals a bias toward small values; same for the neural activation $\xi_k = \langle \mathbf{x}_k^t, \mathbf{e}^t \rangle$ (cf. violin plot on the right; $N \approx 2000$). Circled white cross corresponds to the exemplary signal to the left. **(B)** Uniform activation sampling modifies the power spectrum to obtain the desired projection \mathbf{x}_k^t . As a result, the neural activation ξ_k is spread more uniformly within $[-1, 1]$.

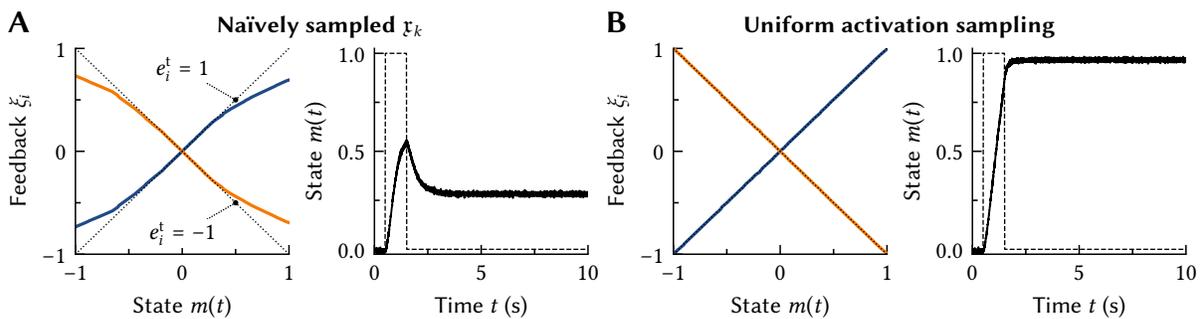


Figure 4.30: Impact of uniform activation sampling on realising an integrator. Recurrent network of $n = 100$ spiking LIF neurons with a first-order synaptic filter $\tau = 100$ ms. *Left:* Learned feedback signal. *Right:* System response to a rectangle input. **(A)** Passing randomly generated signals \mathfrak{x}_k (here with a small RMS to exaggerate the effect) results in a nonlinear feedback function. **(B)** Using uniform activation sampling results in the expected linear functions (dotted lines).

Challenges with uniform activation sampling. There are some potential downsides to uniform activation sampling. First, if the impulse responses h_j are linearly dependent, then the optimisation problem may not have a solution. Even if the h_j are merely non-orthogonal, the problem can become ill-conditioned as q increases. In practice, we avoid this by randomly selecting three linearly independent h_j and only constraining r_k along these axes.

Second, uniform activation may simply not be desirable. If we know that the input signals naturally possess a certain distribution, then we should sample from that distribution. The weight solver can then exploit neurons only being active within a specific regime.

4.3.2 Realising and Comparing Temporal Bases in Spiking Neural Networks

Given the techniques for sampling input signals r_k and temporal encoders above, we now investigate in how far we are able to realise temporal bases in spiking neural networks. Similar to Voelker and Eliasmith (2018), we qualitatively compare the resulting neural activities to those of biological “time cells” (cf. Section 4.1.1).

Modified Fourier networks. In the previous section, we saw that the modified Fourier system—both with Bartlett and rectangle windows—can outperform the LDN system. While the LDN has been constructed with spiking neural networks in mind (Voelker and Eliasmith, 2018), we have so far not tested the modified Fourier system in a neural network context.

Correspondingly, we first ensure that the two Fourier systems can be mapped onto a spiking neural network with the architecture depicted in Figure 2.30A using the NEF dynamics principle. Specifically, we assume first-order synaptic filters with $\tau = 100$ ms, and $n = 1000$ LIF neurons with maximum rates between 50 s^{-1} to 100 s^{-1} . We obtain delay decoders $d_{\theta'}$ by feeding a training signal $r(t)$ into the network, and solving for weights that project the spiking activity onto $r(t - \theta')$. We then compute the mean delay decoding error for a test signal.²⁰

Results for a bandlimited noise input signal ($\rho = 3$ Hz; eq. 4.27) and $q = 7$ state dimensions are depicted in Figure 4.31. In this particular example, all three networks achieve similar mean delay decoding errors (NRMSEs between 20% and 26%), and the modified Fourier system outperforms the LDN system by a few percent, consistent with previous findings (cf. Figure 4.25).

Systematic evaluation. Results of a more systematic experiment are depicted in Figure 4.32. Specifically, we vary the number of neurons in the network n , the number of state dimensions q , and compare the NEF dynamics principle to our temporal tuning curve optimisation scheme (eq. 4.8) with and without uniform activation sampling (see above). We use the default weight solver implemented in Nengo for the standard NEF (i.e., $N = \max\{500, \min\{2500, 500q\}\}$ training samples), and $N = 1000$ samples for the temporal-tuning curve based methods. All other parameters and methods are as discussed above.

²⁰The spike trains are low-pass filtered ($\tau = 25$ ms) to approximate momentary activities; the same low-pass filter is applied to r before computing errors and decoders.

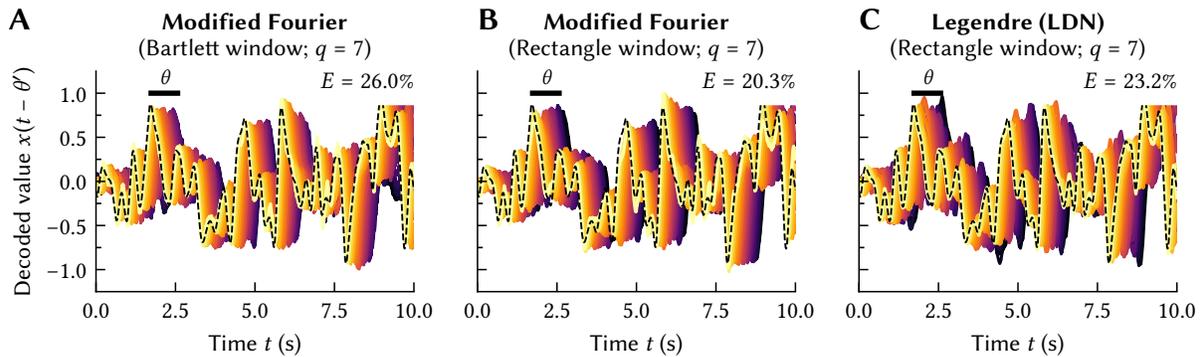


Figure 4.31: Realising temporal bases and decoding delays in a spiking neural network using the “standard NEF” dynamics principle for weight computation. Same colour scheme as in Figure 4.23. Mean delay decoding errors are the mean NRMSE for 20 different θ' . See text for a detailed description.

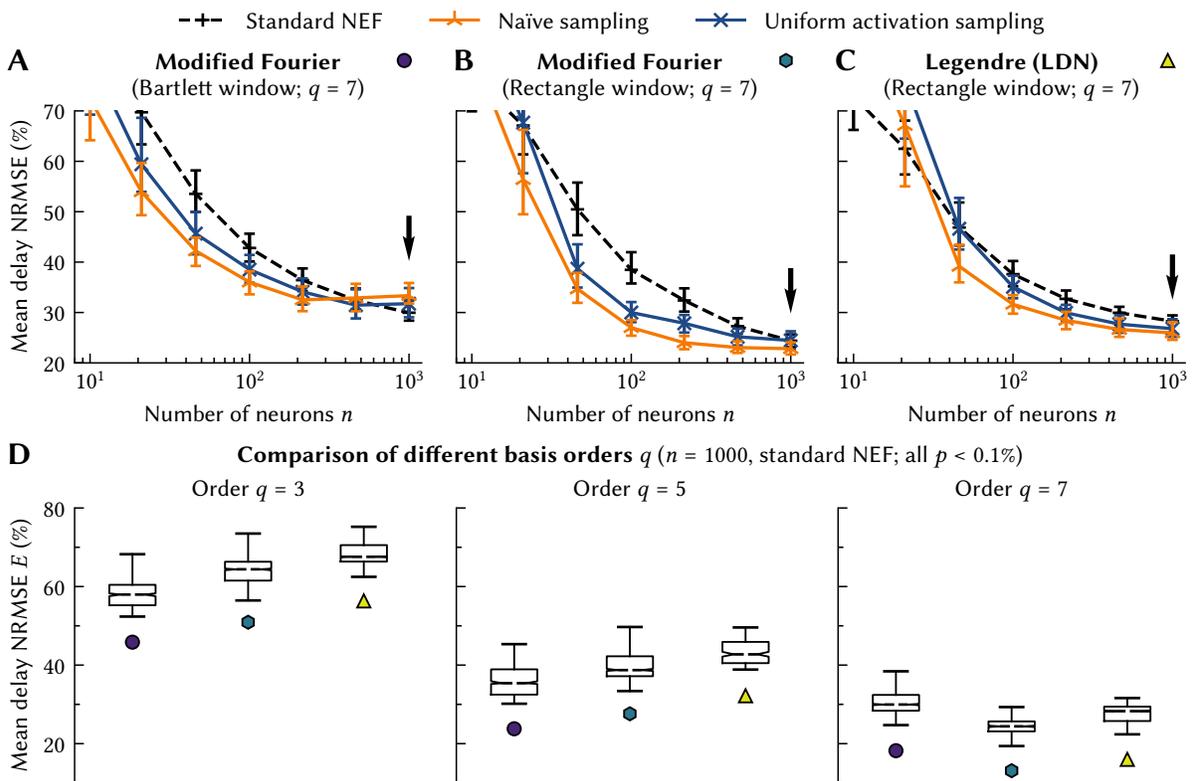


Figure 4.32: Systematic comparison of different temporal bases and synaptic weight optimisation methods. **(A-C)** Varying the number of neurons n in the network. Lines are the median over $N = 100$ trials (at 10 test signals per trial), error bars indicate the 25th and 75th percentile. Arrow points at the configuration used in the previous experiment and in **(D)**. The two sampling methods typically outperform the standard NEF method for the chosen distribution of input signals. **(D)** Comparison of the different function bases (coloured symbols) for different q using the standard NEF. Boxes indicate the quartiles, notches the 99% confidence interval, whiskers the minimum and maximum. The Fourier bases outperform the LDN; for small q the modified Fourier basis with Bartlett window can be better.

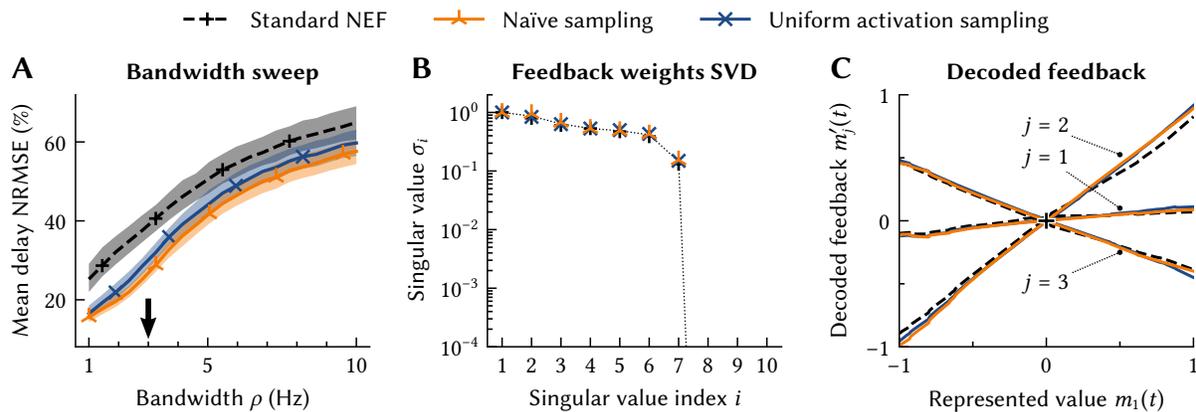


Figure 4.33: Analysing the feedback weight matrices obtained using different optimisation methods. All data is for the modified Fourier basis with rectangle window for $q = 7$ at $n = 100$ neurons. **(A)** Sweep over the test signal bandwidth ρ (training is at $\rho = 3$ Hz; cf. arrow); lines are the median over $N = 100$ networks and ten test signals each, shaded areas indicate quartiles. The temporal tuning-curve methods outperform the standard NEF for all ρ , however, the discrepancies are higher for small ρ . **(B)** Normalised singular values of an exemplary feedback matrix \mathbf{W} . All methods generate a matrix of effective rank $q = 7$ (i.e., $\sigma_i < 10^{-10}$ for $i > 7$). **(C)** Effect of the feedback matrix \mathbf{W} on the first three state dimensions j when varying $m_1(t)$. The functions obtained using the temporal tuning methods are smoother.

Results. Notably, our temporal tuning-curves based weight optimisation methods outperform the standard NEF method for a wide range of neural population sizes n . For all tested combinations of q and n the modified Fourier network with rectangle window significantly ($p < 0.1\%$) outperforms the LDN. Interestingly, using uniform activity sampling typically results in a slightly higher error than using naïve sampling.

Discussion. It is unclear why our temporal tuning based approach performs better. Analysing the differences more thoroughly, the weights obtained using the temporal tuning curve methods are, as predicted, slightly overfit to the training signal bandwidth (cf. Figure 4.33A). Both the standard NEF solver and uniform activation sampling likely generalise better to other input distributions, hence the higher errors compared to naïve sampling. However, we were not able to observe this in practice for synthetic input signals.

Apart from this, all methods result in a rank- q feedback weight matrix (cf. Figure 4.33B) with similar regularisation errors.²¹ One culprit regarding the worse performance of the NEF dynamics principle is the quality of the decoded linear functions, which are slightly smoother for the weight matrices obtained using our approaches (cf. Figure 4.33C). This may be due to sampling \mathfrak{r} from different distributions (i.e., using Halton sequences for uniform activation sampling); however, the results in Figure 4.33C specifically are for $N = 100$ samples, whereas Nengo uses $N = 2500$, achieving a higher sampling density.

²¹Regularisation factors were matched for the L_2 regularisation error $\|\mathbf{W}\|_{\mathbb{F}}^2$ to be approximately equal (about 4×10^{-4} for the standard NEF and 5×10^{-4} for the tuning-curve approach in the above example).

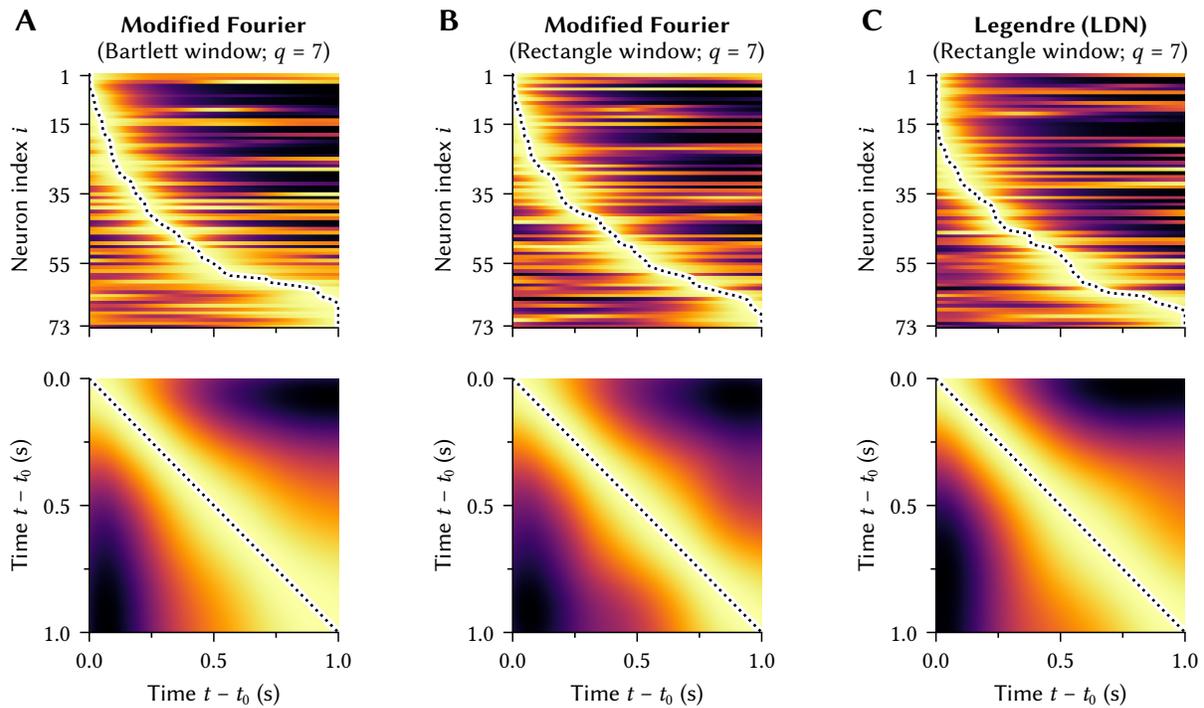


Figure 4.34: Temporal fields and activity vector similarity of neurons in networks generating temporal bases. Same analysis as in Figure 4.3. Lighter colours correspond to larger values. *Top:* Activity of 73 non-silent neurons randomly (selected from $n = 120$ neurons) in response to a short rectangle pulse (100 ms, magnitude 10). Neurons are sorted by their specific delay θ_i (dotted line). *Bottom:* Cosine similarity between the population activity vectors at different times; line is the diagonal.

Time cells. Voelker and Eliasmith (2018) find that the activities of neurons in the Legendre delay network resemble those of empirical time cells. In particular, the LDN qualitatively matches recordings from prefrontal cortex (cf. Tiganj et al., 2016 and Figure 4.3).

As we discussed in Section 4.1.1, time cells are neurons that reach their peak activity a fixed delay θ_i after an event at time t_0 . Empirical data suggests (e.g. MacDonald et al., 2011; Tiganj et al., 2016) that the specificity of this temporal representation decays over time; it becomes progressively harder to determine how much time has passed since t_0 .

Figure 4.34 depicts our results for repeating the analysis performed by Voelker and Eliasmith (2018) for the two modified Fourier and Legendre bases. Methods and network setup are the same as before in this section; we use $n = 120$ neurons and the standard NEF solver. Minor differences can be observed in the spread of the activity similarity over time. We observe a larger spread (i.e., lower temporal specificity) for both the modified Fourier basis with Bartlett window and the Legendre basis, while the Fourier system with rectangle window maintains a higher specificity. Still, overall, the three systems produce time cells with similar distributions of specific delays θ_i .

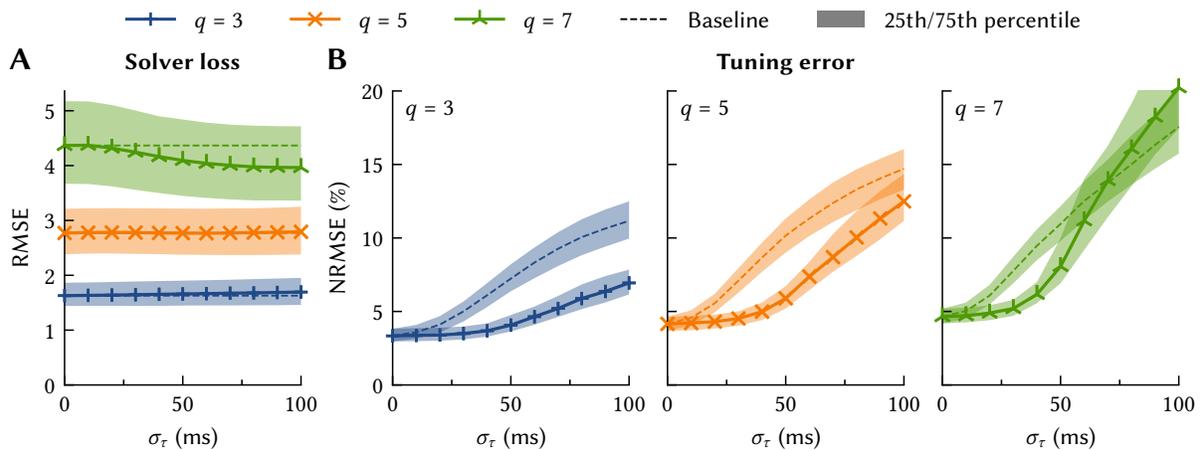


Figure 4.35: Compensating for heterogeneous synapses using temporal tuning. Solid lines are the median over $N = 100$ trials with ten test signals each. **(A)** Solver loss \sqrt{E} (cf. eq. 4.8) for different q over varying σ_τ . Dashed line is the baseline for $\sigma_\tau = 0$ ms. On average, the solver loss decreases slightly as σ_τ increases. **(B)** Tuning error (difference between the expected and actual firing rate) for a test input. The dashed line is the result when not accounting for recurrent heterogeneous synapses while solving for weights. We typically achieve lower errors if we account for the heterogeneous recurrent synapses.

Heterogeneous synaptic filters. As we saw in Section 4.1.4, optimising eq. (4.8) can implicitly compensate for heterogeneous synaptic filters. However, our technique of mapping linear onto nonlinear networks only allows for limited heterogeneity. In contrast, if we solve for weights directly within the nonlinear network, every synaptic filter h_{ij} can be different.

To explore in how far we can compensate for variations in synaptic filters, we map the modified Fourier network with rectangle window onto a spiking neural network with $n = 100$ neurons. While the overall setup is the same as before, we now sample the time-constant τ_{ij} of each synapse from a truncated normal distribution with mean 100 ms and a minimum of 1 ms.²² To account for the fact that compensating for heterogeneous synapses implicitly requires access to the input differential (cf. Appendix A.3.3), we emulate a pre-population with diverse temporal tuning by passing the input signal through eleven different low-pass filters with τ between 1 ms to 400 ms (spaced logarithmically).

Results for an experiment where we vary σ_τ are depicted in Figure 4.35. Increasing σ_τ barely affects the solver loss. However, increasing σ_τ generally leads to an increase in *tuning error*, the difference between the actual and expected spike rate. Importantly, when taking heterogeneous recurrent synapses into account, this increase is quite slow over a wide range of σ_τ . For larger σ_τ , the error approaches or even surpasses (for $q = 7$) the error for not taking heterogeneity into account. The error being surpassed is likely due to compensating for heterogeneous filters requiring larger weight magnitudes, amplifying noise in the system.

²²We discretise the sampled τ_{ij} to reduce the computational costs of our simulations—the fewer different τ_{ij} , the more operations are shared between neurons. Specifically, we compute $\tau'_{ij} = \exp(\lfloor 10 \log(\tau_{ij}) \rfloor / 10)$.

4.3.3 Spatiotemporal Neuron Populations

As we suggested in Section 4.3.1, neurons can be simultaneously tuned spatially and temporally. Specifically, we proposed in eq. (4.25) to construct temporal encoders ϵ_i by sampling separate temporal and spatial encoding vectors $\mathbf{e}_i^t \in \mathbb{S}^q$, $\mathbf{e}_i \in \mathbb{S}^d$. This way, we obtain a rank one spatiotemporal encoding matrix $\mathbf{E}_i^t \in \mathbb{R}^{d \times q}$, where $\mathbf{E}_i^t = \mathbf{e}_i \otimes \mathbf{e}_i^t$ and $\epsilon_i = \mathbf{E}_i^t \mathbf{h}$ (cf. eq. 4.26; i is the neuron index, d the number of spatial dimensions, and \mathbf{h} is a temporal basis of order q).

In theory, for an infinite number of neurons and temporal basis dimensions, and \mathbf{E}_i^t being sampled uniformly, we can decode arbitrary continuous functions f over space and time. More precisely, f is a higher-order function $f : ([-\theta, 0] \rightarrow \mathbb{X}) \rightarrow \mathbb{R}^{d'}$, where $[-\theta, 0]$ is some temporal interval, $\mathbb{X} \subset \mathbb{R}^d$ is a compact spatial domain, and d' is the number of target dimensions.²³ Below, we explore two examples of such functions: delayed multiplication and the “recently travelled distance”. We furthermore highlight the difference between rank one and full-rank spatiotemporal encoding matrices \mathbf{E}_i^t .

Delayed multiplication with rank one \mathbf{E}_i^t . Consider the product of two delayed signals $\mathbf{x}^1, \mathbf{x}^2$:

$$f_{\theta'_1, \theta'_2}(\mathbf{x}^1, \mathbf{x}^2) = \mathbf{x}^1(-\theta'_1) \mathbf{x}^2(-\theta'_2).$$

This function is useful as a benchmark. Any approximation $\hat{f}_{\theta'_1, \theta'_2}$ with an NRMSE below 100% indicates that we can decode *nonlinear* functions over the delayed inputs. This is because, for zero mean \mathbf{x} , the optimal *linear* approximation is $\hat{f}_{\theta'_1, \theta'_2}(\mathbf{x}) = 0$ with an NRMSE of 100%.

Methods. To decode this “delayed multiplication” function in a spatiotemporal network, we tune $n = 1000$ LIF neurons with maximum firing rates between 50 Hz and 100 Hz to $d = 2$ spatial dimensions and $q = 5$ temporal LDN dimensions using separate temporal and spatial encoders $\mathbf{e}_i^t, \mathbf{e}_i$.²⁴ After solving for weights using $N = 1000$ naïvely sampled input signals \mathbf{x}_k (at $\rho = 3$ Hz), we feed a training and test signal of length $T = 100$ s and a band-limit of 1 Hz into the network. We use the training signal to compute decoders, and the test signal to compute the presented decoding errors.

Results. Results are depicted in Figure 4.36. It is possible to decode delayed versions of the individual input signals with similar errors as in previous experiments. Decoding delayed multiplication generally results in relatively large errors, with a minimum error of 36%. Most notably, errors are minimal along the diagonal, i.e., $\theta'_1 = \theta'_2$.

Discussion. Decoding errors for off-diagonal θ'_1, θ'_2 pairs being large is due to \mathbf{E}^t being of rank one. Each neuron is temporally tuned to a linear combination of the spatial input signal dimensions. For example, for some $\mathbf{e}_i, \mathbf{e}_i^t$, the neuron is most active if the time-course of the linear combination $\langle \mathbf{e}_i, \mathbf{x}(t) \rangle$ is similar to the time-course described by the function $\langle \mathbf{e}_i^t, \mathbf{h}(t) \rangle$.

²³This depends on the spatial encoding matrices uniformly covering the space of possible input signals and once again follows from Hornik et al. (1989). Intuitively, we can linearly combine the neural activities to assign a value to every non-pathological input signal $\mathbf{x} : [-\theta, 0] \rightarrow \mathbb{X}$ (cf. Appendix A.1.4).

²⁴In this context, using the LDN system resulted in slightly smaller errors than the modified Fourier system.

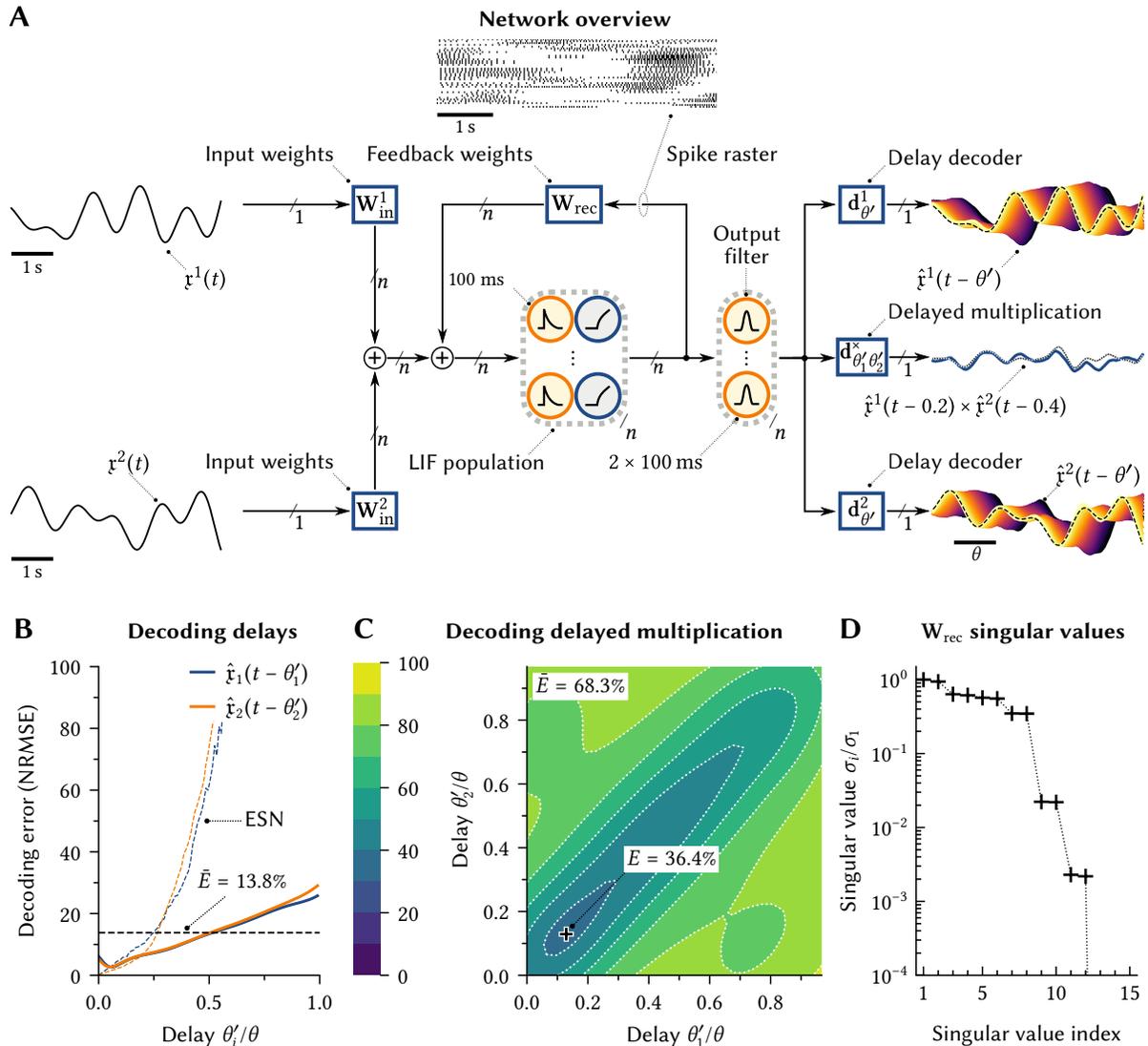


Figure 4.36: Two-dimensional quantities over time in a rank one spatiotemporal network. **(A)** Overview of the network and results for some exemplary inputs $x_1(t)$ and $x_2(t)$. The input signals are fed into a recurrent network where each neuron is tuned to a random linear combination of the inputs, and a linear combination of the LDN impulse responses for $q = 5$ and $\theta = 1$ s. We can approximately decode delayed versions of the inputs (a linear transformation) and the product of the two input signals for $\theta_1 = \theta_2$. **(B)** For band-limited noise inputs, delayed versions of the both input signal can be decoded with a small mean delay decoding error \bar{E} . Dashed lines are the delay decoding errors for an equivalent Echo State Network (ESN; Jaeger and Haas, 2004). **(C)** Decoding error (NRMSE) when systematically decoding delayed multiplication for delays θ_1, θ_2 . While the resulting error is quite large, it is substantially smaller than what is achievable with linear approximations of four-quadrant multiplication (i.e., an NRMSE of 100%). Decoding delayed multiplication works best for $\theta_1 = \theta_2$. For an ESN, all errors are above 100%. **(D)** Normalised singular values of the recurrent weight matrix. The matrix is of rank twelve, with the last four singular values being relatively small.

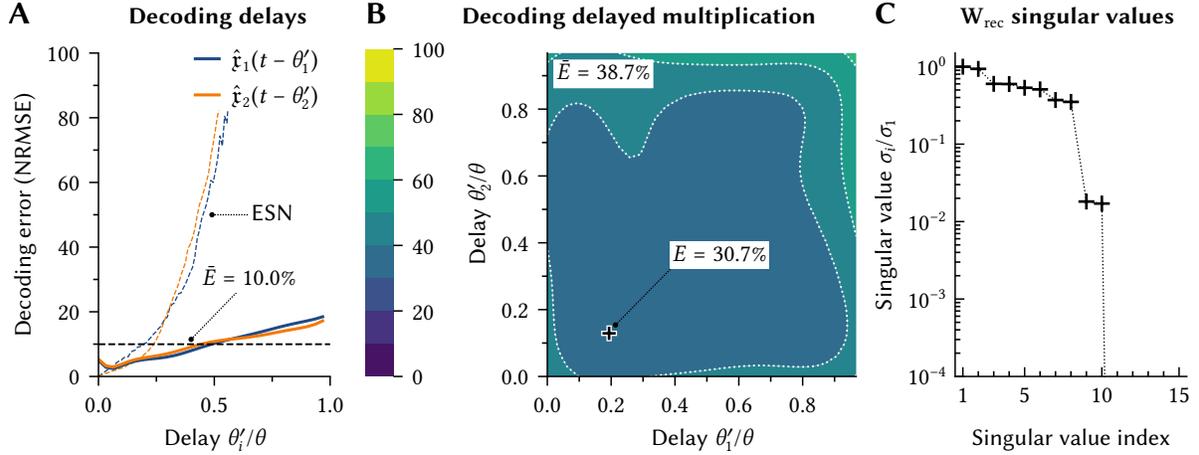


Figure 4.37: Decoding delayed multiplication in a network with full rank spatiotemporal encoders \mathbf{E}_i^t . Same architecture as in Figure 4.36A. **(A)** Decoding delayed versions of the two input dimensions. Note the slightly smaller mean delay decoding error compared to using a rank one spatiotemporal encoding matrix. **(B)** Delayed multiplication decoding errors are approximately uniform for all combinations of θ_1 and θ_2 . **(C)** The recurrent weight matrix now possesses $dq = 10$ non-zero singular values.

However, using rank one \mathbf{E}_i^t , it is impossible to generate neural tuning such that a neuron is tuned to different input dimensions having different temporal behaviour. As a result, we cannot decode functions that are nonlinear over several time-points. This is similar to not being able to decode nonlinear functions from neuron populations with axis-aligned spatial encoders (cf. Section 3.1.2). We reach NRMSEs below 100% for $\theta'_1 \neq \theta'_2$ when computing delayed multiplication because the product for the closest uniform delay is a decent approximation.

Delayed multiplication with full rank \mathbf{E}^t . We repeat the above experiment with full rank spatiotemporal encoding matrices \mathbf{E}^t . That is, instead of sampling spatial and temporal encoding vectors \mathbf{e}_i , \mathbf{e}_i^t , we generate random \mathbf{E}^t with Frobenius norm $\|\mathbf{E}^t\|_F = 1$. Specifically, we uniformly sample vectors from the hypersphere \mathbb{S}^{dq} , and reshape these vectors into matrix form.

Results. Errors for delay decoding and computing delayed multiplication are given in Figure 4.37. Note that the errors for decoding delayed versions of the individual input signals are slightly smaller than before (Figure 4.37A). The results for computing delayed computation show a drastic improvement (Figure 4.37B). Errors are between 30% to 40% for all θ'_1 , θ'_2 pairs.

Discussion. Improvements in the linear delay decoding task are due to a slight reduction in our solver loss from eq. (4.8); the more diverse tuning facilitates realising the desired tuning in the recurrent connection. The reduction in error for delayed multiplication is as expected.

Finally, note that the recurrent weight matrices are of low rank (Figures 4.36D and 4.37C). This suggests that it is possible to realise the same network in the NEF without temporal tuning. Indeed, as we show in Appendix B.3.2, this is possible. However, we must realise a qd -dimensional representation in the population; implementing dynamical systems with

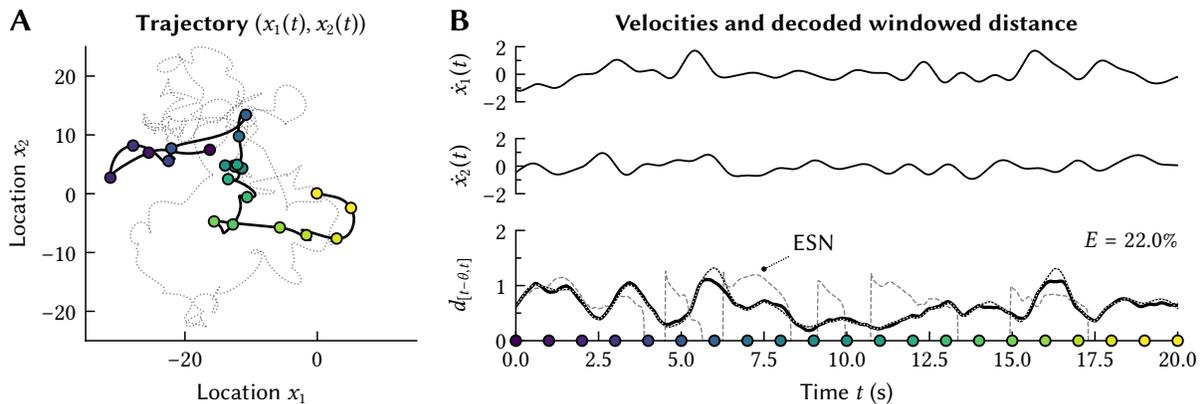


Figure 4.38: Decoding the recently travelled distance from a spatiotemporal network. Same experimental setup as in Figure 4.36, but decoding the distance travelled over the past second instead. **(A)** Random trajectory generated by integrating the input velocities in x_1 - and x_2 -direction. Highlighted section (black line and coloured circles) corresponds to the time range displayed in **(B)**. **(B)** Input velocities $\dot{x}_1(t)$ and $\dot{x}_2(t)$ and the decoded windowed distance $d_{[t-\theta, t]}$. Dotted line is the ground-truth. The NRMSE E is computed after subtracting the mean (without subtracting the mean $E = 8.1\%$). Again, NRMSEs are far above 100% when using an ESN (grey dashed line).

spiking neurons in such high dimensional spaces can be challenging. Our approach offers the advantage of biasing the weight solver towards portions of the activity space relevant for certain input signals and thus results in smaller errors in the delayed multiplication task.

Example: Recently travelled distance. As a slightly more practical example of a function that could be decoded from a spatiotemporal network, consider the distance an agent has travelled over the past θ seconds. For the sake of simplicity, let $\dot{x}_1(t)$, $\dot{x}_2(t)$ denote the velocity of the agent in a global coordinate space. We define the *recently travelled distance* $d_{[t-\theta, t]}$ as

$$d_{[t-\theta, t]}(\dot{x}_1, \dot{x}_2) = \int_{t-\theta}^t \sqrt{\dot{x}_1(\tau)^2 + \dot{x}_2(\tau)^2} d\tau.$$

We test decoding this function using the spike data recorded in the experiment with rank one E_i^t . Results are depicted in Figure 4.38. We obtain an NRMSE of $E \approx 22\%$ over $T = 100$ s. Interestingly, using the full rank E_i^t we obtain $E \approx 26\%$ (data not shown).

Discussion. This experiment indicates that it can be beneficial to restrict the encoders to a lower rank. We apply a linear operator (the integral) to a function that is only nonlinear over the same point in time τ —this is exactly the type of function supported by our rank one spatiotemporal encoding matrices.

Overall, our experiments demonstrate that it is possible to realise populations with spatiotemporal tuning in the NEF; our *spiking* networks even outperform the popular non-spiking Echo State Network (Jaeger and Haas, 2004) by a wide margin. A next step would be to use this technique to construct a neurophysiologically motivated model of spatiotemporal tuning.

4.3.4 Adaptive Filters

Another interesting application of neural networks with spatiotemporal tuning is using them to construct nonlinear *adaptive filters*. Here, we discuss an adaptive variant of the Wiener filter (Wiener, 1949; Haykin, 2014, Chapter 2). *Adaptive* refers to updating the usually static filter parameters on-the-fly, in response to an error signal $\varepsilon(t)$ while the system is operating. This is in principle comparable to learning and adaptation in animals.

The core idea of Wiener filters is to linearly combine filtered versions of an input signal with the goal of minimising a linear least-squares loss.²⁵ Let $h_i(t)$ be one of n filter kernels, $\mathbf{d} = (d_1, \dots, d_n)$ a set of weights, $u(t)$ an input signal, and $r(t)$ a desired output signal. Then, in the simplest case, this optimisation problem can be stated as follows:

$$E = \left(\int_0^T r(t) - \sum_{i=1}^n d_i (u * h_i)(t) dt \right)^2 = \left(\int_0^T r(t) - \sum_{i=1}^n d_i a_i(u * \delta_{-t}) dt \right)^2. \quad (4.28)$$

where, as in eq. (4.8), δ_{-t} is a shifted Dirac delta, and $a_i(u) = (u * h_i)(0)$.

As pointed out by Wiener (1949), this kind of optimisation problem can be interpreted in multiple ways, two of which are compatible with adaptation. First, eq. (4.28) can be used to find \mathbf{d} that extract “hidden” information $r(t)$ from $u(t)$ by removing unwanted noise—hence the term *filter*. Wiener refers to this as *interpolation*. Second, if $r(t)$ corresponds to a future point in time relative to $u(t)$, then these equations can be used for *prediction* or *extrapolation*.

Now, to adapt \mathbf{d} over time, we take the gradient of E in eq. (4.28) with respect to \mathbf{d} for the current signal history $u : \mathbb{R}^- \rightarrow \mathbb{R}$ and perform gradient descent over time:

$$\frac{\partial}{\partial d_i(t)} E \propto \left(y(t) - \sum_{i=1}^n d_i(t) a_i(u) \right) a_i(u) = \varepsilon(t) a_i(u) \quad \rightsquigarrow \quad \dot{d}_i(t) = -\eta \varepsilon(t) a_i(u). \quad (4.29)$$

Here, $\varepsilon(t)$ is the momentary error, and η is a learning rate. Crucially, instead of setting a_i to a linearly filtered version of u , we can use the activities a_i of a nonlinear neural network with temporal tuning. In this case, eq. (4.29) is the *delta learning rule*, a precursor to backpropagation that has been developed with nonlinear adaptive filters in mind (Widrow and Hoff, 1960).²⁶

MacNeil et al. (2011) propose a biologically plausible version of eq. (4.29) that can be integrated into spiking neural networks. We discuss this Prescribed Error Sensitivity (PES) rule in more detail in Section 5.3. For now, we would like to demonstrate that the combination of our spatiotemporal spiking NEF populations and the delta rule results in a powerful adaptive filter that can, for example, learn to predict the nonlinear dynamics of a forced pendulum over time. Forced pendulums are found in most robotic actuators and can, under some conditions, exhibit chaotic behaviour (Hubbard, 1999).

²⁵Notably, Wiener provides conditions under which an optimal \mathbf{d} can be obtained. Kalman (1960) broadens the optimality conditions by accounting for uncertainty and proposing an optimal update rule.

²⁶For nonlinear neurons, we have to multiply eq. (4.29) by a gradient term a'_i . We typically ignore this for LIF neurons, which, if not saturated, resemble ReLUs. Correspondingly, a'_i is a constant factor if the neuron is active.

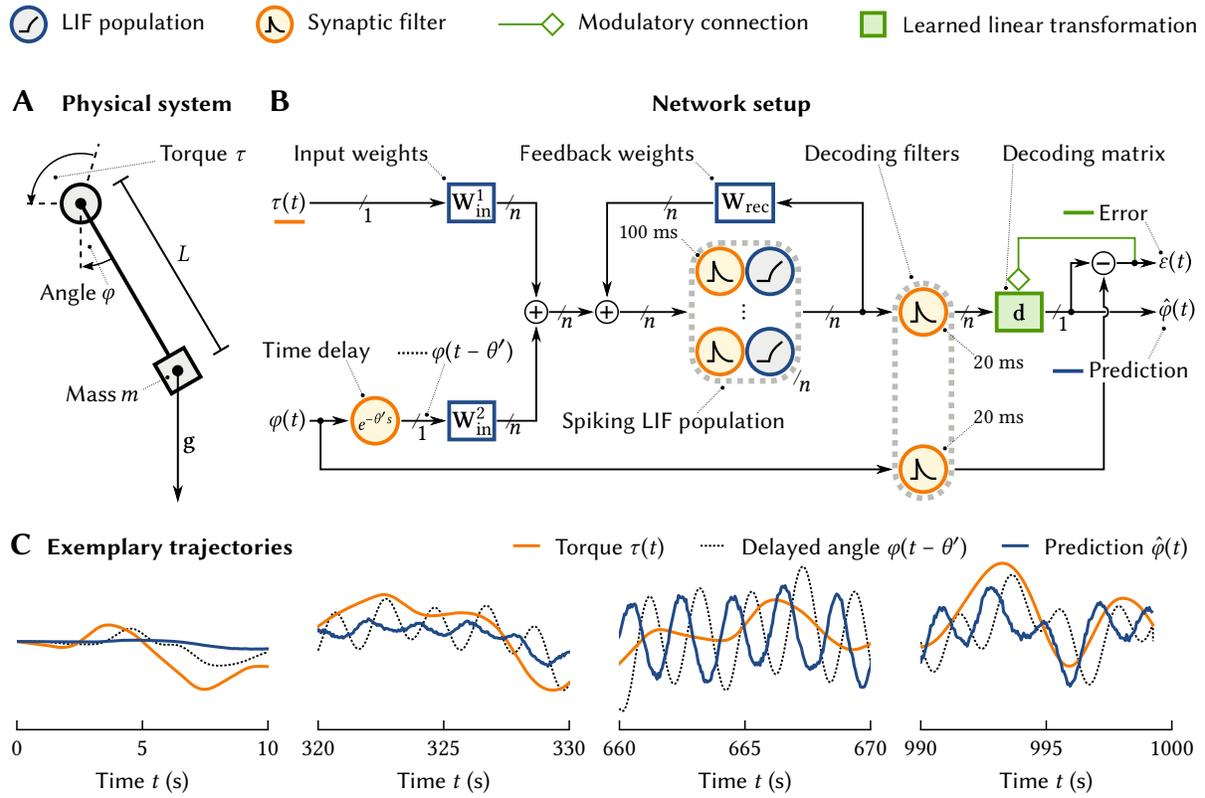


Figure 4.39: An adaptive filter learning pendulum dynamics. **(A)** Illustration of the nonlinear pendulum system. A point mass $m = 1$ kg under the influence of gravity $\|g\| = 9.81 \text{ m s}^{-2}$ is connected to a rotating joint through a rigid rod of length $L = 1$ m. A random torque $\tau(t)$ is applied to the joint; we measure the angle of the pendulum $\varphi(t)$. **(B)** The torque $\tau(t)$ and a delayed version of the angle $\varphi(t - \theta')$ are fed into the full-rank spatiotemporal NEF network from the previous subsection. After passing the spikes through a short low-pass filter, we decode the angle at time t using a decoding vector d . This decoding matrix is updated according to the delta rule; the network *learns* the nonlinear pendulum dynamics. **(C)** Examples of the pendulum dynamics and prediction after different amounts of training.

Methods. As is depicted in Figure 4.39, we apply a low-pass filtered white noise signal $\tau(t)$ as a torque to the rotational joint of a simulated pendulum.²⁷ Care has been taken to choose the frequencies and amplitude of $\tau(t)$ such that the pendulum regularly switches between a resonance mode—where the pendulum merely oscillates with its approximate resonance period $2\pi\sqrt{Lg^{-1}}$ —and a mode where the pendulum is more directly influenced by $\tau(t)$. These modes are to some degree visible in the two rightmost plots of Figure 4.39C.

Our spatiotemporal neuron population from the last subsection ($d = 2$, $q = 5$, $n = 1000$) receives both the torque $\tau(t)$ and a time-delayed version of the pendulum joint angle $\varphi(t - \theta')$, where $\theta' = 0.75$ s. Inputs to the network are rescaled to the interval $[-1, 1]$.

²⁷We use our library *PyKinSim* (<https://github.com/astoeckel/pykinsim>) as a kinematic chain simulator.

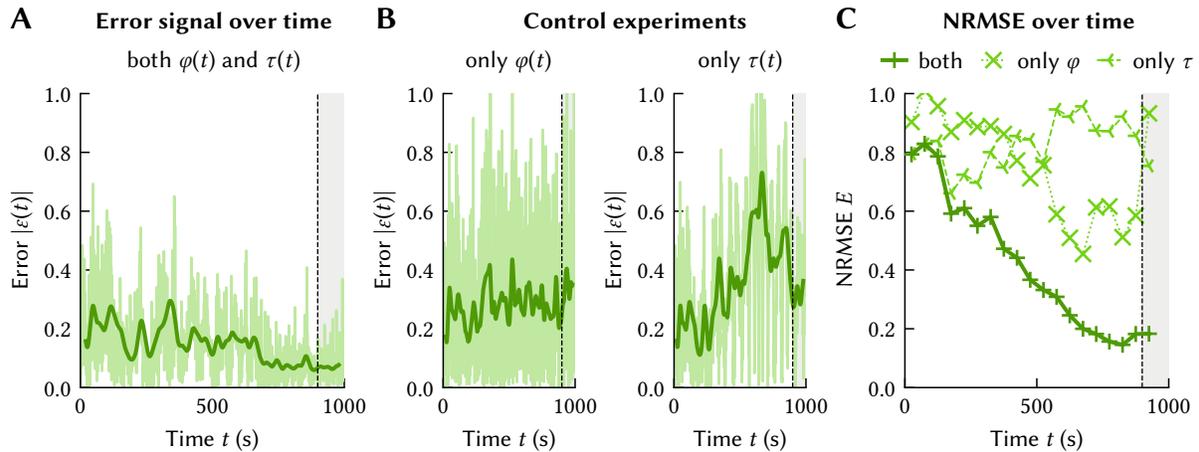


Figure 4.40: Adaptive filter error over time. Area right to the black dashed line is without adaptation. **(A)** Error signal $\varepsilon(t)$ over time; dark line is a low-pass filtered version. **(B, C)** Control experiment where only one of the two input dimensions are available to the network. Evidently, both are necessary to compute the function well. **(D)** Same data as before, but as NRMSE over bins of length 50 s.

We autoregressively learn a decoding vector \mathbf{d} using eq. (4.29) and the error signal $\varepsilon(t) = \varphi(t) - \hat{\varphi}(t)$, where $\hat{\varphi}(t)$ is the decoded and low-pass filtered spiking activity. Since the input angle $\varphi(t - \theta')$ is delayed, the system essentially learns to *predict* the motion of the pendulum θ' seconds into the future given the future torque trajectory. We set the learning rate η to 2×10^{-4} and train over 1000 s.²⁸

As a control experiment, we test only feeding one of the input dimensions—that is, either the angle $\varphi(t - \theta')$ or the torque $\tau(t)$ —into the system. This way, we can ensure that the predicted dynamics indeed depend on both the past pendulum state and the torque; for example, the torque $\tau(t)$ is mostly irrelevant if the pendulum is in its resonance mode.

Results. Examples of the inputs and the predicted output are depicted in Figure 4.39C. A more thorough analysis of the error $\varepsilon(t)$ over time is provided in Figure 4.40. Generally, the phase of the prediction is correct, but the amplitude of the peaks is often slightly over- or underestimated. The final NRMSE reached by the network is about 20%. Our control experiments show that both $\tau(t)$ and $\varphi(t - \theta')$ critically contribute to the prediction.

Discussion. Our nonlinear adaptive filter learns to predict the nonlinear pendulum dynamics reasonably well. Possible sources for error are the relatively small q , the limited neural resources, and the remaining spike noise. Interesting future work includes exploring the relationship between this kind of adaptive filter and hierarchical predictive coding networks based on a more general formalisation of Kalman filtering (Bastos et al., 2012). We discuss adaptive filters in the context of a biologically constrained model of the cerebellum in Chapter 5.

²⁸Larger learning rates—and shorter simulation times—are possible, but our setup generates enough data to construct a relatively noise-free learning curve, i.e., Figure 4.39C.

4.3.5 Accounting for Intrinsic Neural Dynamics

So far, we have focused on synaptic filters as the primary temporal resource in spiking neural networks. However, as we mentioned several times throughout this thesis, neurons themselves can be a significant source of dynamics. Our temporal tuning approach suggests a way to account for neural dynamics while solving for weights; however, this is by no means a one-size-fits-all solution for every neuron model. Below, we discuss the abstract optimisation problem, point out solution strategies, and demonstrate that we can account for ALIF dynamics.

Optimisation problem. Remember that our temporal tuning paradigm assigns tuning properties to each neuron i —given some input signal \mathbf{x}_k , we define a desired output rate $a_i(\mathbf{x}_k)$. Under the assumption that the pre-neurons j adhere to their normative tuning, we then solve for weights w_{ij} that decode a current $J_i(\mathbf{x}_k)$ evoking the desired rate. The mapping between currents and rates is according to a *static* response curve $G[J]$; it holds $a_i(\mathbf{x}_k) = G[J_i(\mathbf{x}_k)]$.

If we would like to account for intrinsic neural dynamics, then we can no longer assume static response curves. Instead, we must model the average firing rate of a neuron depending on its input current history; we have a *temporal* response curve $\Omega[\mathfrak{J}] : (\mathbb{R}^- \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^+$, where \mathfrak{J} is a signal describing the past post-synaptic currents, and $\Omega[\mathfrak{J}]$ is the rate at $t = 0$.

Since our weight optimisation problem is in current space, we need to map from the rate determined by the tuning curve $a_i(\mathbf{x}_k)$ onto a current. More precisely, we search for a function of the form $\Omega^{-1}[\mathbf{a}] : (\mathbb{R}^- \rightarrow \mathbb{R}^+) \rightarrow \mathbb{R}$:²⁹ given a desired output rate signal \mathbf{a} , this function returns the post-synaptic current J that should be injected into the neuron in the present, at $t = 0$. Consequently, our optimisation problem in eq. (4.8) becomes

$$E = \sum_{k=1}^N \left[\Omega^{-1}[\mathbf{a}_i(\mathbf{x}_k)] - \sum_{j=1}^m w_{ij} (h_{ij} * \mathbf{a}_j(\mathbf{x}_k))(0) \right]^2, \quad \text{where } (\mathbf{a}_i(\mathbf{x}_k))(t) = a_i(\mathbf{x}_k * \delta_{-t}). \quad (4.30)$$

While this optimisation problem is still in linear least-squares form, Ω^{-1} is typically a rather complex spatiotemporal function. Correspondingly, we can only expect to solve eq. (4.30) with a small error if the pre-population tuning forms an expressive spatiotemporal function basis.

Determining Ω^{-1} is generally difficult, and, for most neuron models, a research project of its own. One approach explored by Hunsberger (2016) is to employ linear-nonlinear models of the form $\Omega[\mathfrak{J}] = G[(\mathfrak{J} * \mathfrak{h})(0)]$. In this case, we do not necessarily have to use eq. (4.30), but can simply treat \mathfrak{h} as an additional filter next to the h_{ij} in eq. (4.8); this will implicitly solve for Ω^{-1} .

The following is another method for implicitly determining Ω^{-1} —however, as of writing, this still requires further investigation. The basic idea is to iteratively compensate for errors introduced by the intrinsic neural dynamics. That is, after solving for weights as in eq. (4.8), we can determine the actual population response $\hat{a}_i(\mathbf{x}_k)$ for synthetic pre-population spike trains

²⁹Note that we write Ω^{-1} only for notational purposes; this is not strictly speaking the mathematical inverse of Ω . Since Ω is unlikely to be bijective, Ω^{-1} is ill-defined. Furthermore, having Ω^{-1} only depend on the output history may not be sufficient for complex neuron models; the entire input history may play a role in determining the “correct” input current. It is unclear how this could be incorporated into our optimisation problem.

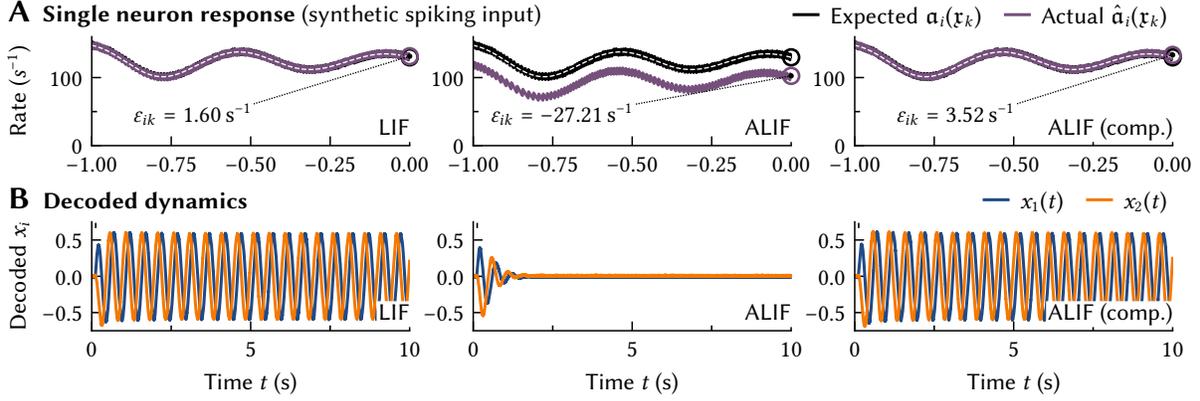


Figure 4.41: Accounting for ALIF dynamics. *Left:* Recurrent network implementing a 2 Hz oscillator with 100 LIF neurons (maximum rates at 200 s^{-1}). *Middle:* Same network with ALIF neurons; not taking dynamics into account. *Right:* Compensating for the ALIF dynamics. **(A)** Comparison between the actual response $\hat{a}_i(\mathbf{r}_k)$ of a neuron i for the k th sample \mathbf{r}_k to the expected response $a_i(\mathbf{r}_k)$. The neuron receives synthetic spike trains matching the expected pre-population rates. **(B)** Decoded system state of the freely running network. Without compensation, the low-pass dynamics of the ALIF neuron cause a rapid decay; with compensation the network behaves as expected.

that perfectly follow the desired tuning $a_j(\mathbf{r}_k)$ (cf. Figure 4.41A). Using the deviation from the desired output rate $\epsilon_{ik} = \hat{a}_i(\mathbf{r}_k) - a_i(\mathbf{r}_k)$, we estimate an offset current J_{ik} that compensates for ϵ_{ik} . We add this J_{ik} to eq. (4.8), solve for new weights, and repeat. This is similar to an approach described by Duggins (2017) that has been successfully applied to biologically detailed neurons.

Accounting for ALIF dynamics. $\Omega^{-1}(\mathbf{a})$ is readily available for the adaptive leaky integrate-and-fire model (ALIF). The ALIF model accounts for firing-rate adaptation observed in biological neurons (cf. Figure 2.17F) by increasing an exponentially decaying adaptation variable $n(t)$ by a constant amount n_0 every time the neuron generates a spike (Treves, 1993). The specific ALIF model discussed here applies this spike-triggered adaptation mechanism to a current-based LIF neuron by subtracting the adaptation variable $n(t)$ from the input current $J(t)$. Despite its simplicity, this model can be made to fit empirical spike data from cortical pyramidal cells well (Camera et al., 2004).

To account for ALIF dynamics within eq. (4.30), we merely need to decode the adaptation current from the pre-population. Given the adaptation decay time-constant τ_n , we have

$$\Omega^{-1}(\mathbf{a}) = G^{-1}[\mathbf{a}(0)] + n_0(\mathbf{a} * h_n)(0), \quad \text{where } h_n(t) = \tau_n^{-1} \exp(-t\tau_n),$$

and where $G^{-1}[a]$ is the current required to evoke a rate a according to the static LIF model. We demonstrate successfully accounting for ALIF dynamics ($\tau_n = 100 \text{ ms}$, $n_0 = 0.01 \text{ nA}$) in this way in Figure 4.41. The only requirement for this to work is that the pre-population forms a temporal basis from which h_n can be decoded. In a sense, if we tune our neurons to possess low-pass dynamics, then this approach allows us to *harness* the ALIF dynamics—after all, the ALIF neuron intrinsically implements a low-pass filter (cf. Figure 2.34).

4.4 Applications to Artificial Neural Networks

The Legendre Memory Unit (LMU)—originally proposed by Voelker, Kajić, et al. (2019)—is a building block for spatiotemporal computing in deep neural networks. The LMU, just like our models of spatiotemporal biological neuron populations, is a dynamical system that intrinsically performs stream-to-stream processing. At its core, each LMU consists of a recurrent linear layer implementing the LDN, and a nonlinear layer transforming the LDN state. By harnessing the ability of the LDN to approximate a sliding window spectrum of length θ , the LMU can, even with little training, integrate information over longer time-spans than competing architectures, while requiring a minimal amount of resources, both in terms of memory and computation.

Crucially, the LMU often outperforms other recurrent neural network architectures in stream processing tasks. This includes Long Short-Term Memories (LSTMs; Hochreiter and Schmidhuber, 1997), Gated Recurrent Units (GRUs; Chung et al., 2014) and Echo State Networks (ESNs; Jaeger et al., 2004). The LMU performance has been verified for synthetic benchmarks (Voelker, 2019, Chapter 6.2; Voelker, Kajić, et al., 2019; Gu et al., 2020), as well as real-world applications such as keyword spotting (Blouw, Malik, et al., 2021), bird song classification (Gupta, Kshirsagar, et al., 2021), and natural language processing (NLP; Chilkuri, 2021; Chilkuri and Eliasmith, 2021). A recent study on NLP tasks by Chilkuri, Hunsberger, et al. (2021) suggests that networks based on a variant of the LMU achieve a ten-times higher data efficiency than Transformers (Vaswani et al., 2017). According to Chilkuri et al., this is comparable to the improvement offered by Transformers over LSTMs.

The goal of this section is to explore in how far the success of the LMU depends on projecting the input signal onto the *Legendre* polynomials, and to further characterise the LMU time and space complexity. Specifically, we test how well our modified Fourier system (cf. Section 4.2.3) fares in place of the LDN system. Given that we are no longer in a biological setting, we furthermore test sliding-window transformations that rely on perfect memory.

4.4.1 The Legendre Memory Unit

The original LMU proposed by Voelker, Kajić, et al. (2019) is depicted in Figure 4.42A. Fundamentally, the idea is to project an input $\mathbf{x}_t \in \mathbb{R}^d$ onto a scalar u_t by means of an encoding vector \mathbf{e}_x .³⁰ This scalar is then fed into a zero-order hold (ZOH) discretised LDN system with input matrix $\tilde{\mathbf{B}}$ and feedback matrix $\tilde{\mathbf{A}}$. Specifically, it holds (e.g., Brogan, 1991, Section 9.8):

$$\mathbf{m}_{t+1} = \tilde{\mathbf{A}}\mathbf{m}_t + \tilde{\mathbf{B}}u_t, \quad \tilde{\mathbf{A}} = \exp\left(\frac{\mathbf{A}}{N}\right), \quad \tilde{\mathbf{B}} = \mathbf{A}^{-1}(\tilde{\mathbf{A}} - \mathbf{I})\mathbf{B}, \quad \text{where } N\Delta t = \theta. \quad (4.31)$$

Finally, the q -dimensional state \mathbf{m}_t is passed through a nonlinear layer, resulting in the output $\mathbf{h}_t \in \mathbb{R}^m$. Note that Voelker et al. include all possible connections between the linear and nonlinear layers in the model. All weight matrices except for $\tilde{\mathbf{A}}$, $\tilde{\mathbf{B}}$ are trained via backpropagation (e.g., Bishop, 2006, Section 5.3). The LTI matrices $\tilde{\mathbf{A}}$, $\tilde{\mathbf{B}}$ are kept fixed.

³⁰Since artificial neural networks use discretised time, all input *signals* \mathfrak{x} become input *sequences* \mathbf{x}_t .

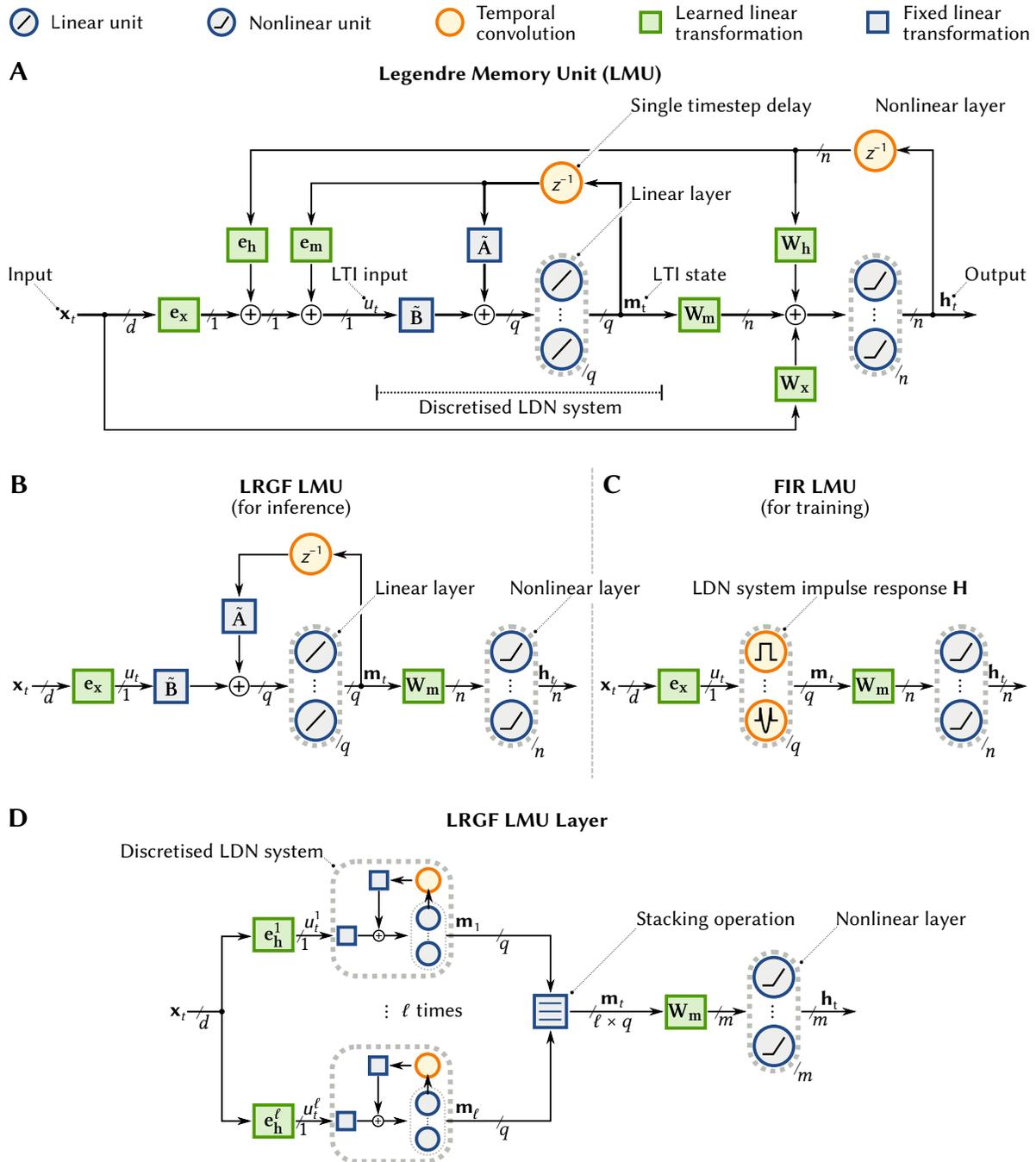


Figure 4.42: Overview of the Legendre Memory Unit. See text for more detail. **(A)** Variant of the LMU network as proposed by Voelker, Kajić, and Eliasmith (2019). The important signal paths are highlighted with bold arrows. **(B, C)** The reduced LRGF and FIR LMU (our acronyms) proposed by Chilkuri and Eliasmith (2021). **(D)** In practice, multiple LDN systems are stacked a single layer.

The LRGF and FIR LMU. Chilkuri (2021, Section 3.1.3) observes that the added connections outside the main signal path have little to no positive impact on the performance of the system. In fact, removing these connections altogether—and thus reducing the number of trainable weights—has a net positive impact on the system performance (Chilkuri and Eliasmith, 2021). This leaves the LDN as the only recurrent component in the network (cf. Figure 4.42B). Tsoi and Back (1997) refer to such networks as “locally recurrent, globally feed-forward” (LRGF).

Since the state-space matrices $\tilde{\mathbf{A}}$, $\tilde{\mathbf{B}}$ are fixed, we can replace the LDN system with a convolution operation (cf. Figure 4.42C; Chilkuri and Eliasmith, 2021). Taking advantage of the fact that the LDN impulse response decays rapidly after $N = \theta\Delta t^{-1}$ samples, we can *approximate* the convolution operation using a *finite* impulse response (FIR) filter of length N .

Let $\mathbf{u}_t = (u_{t-N+1}, \dots, u_t)$ be a slice of the input sequence. We can construct a matrix of q FIR filters $\mathbf{H} \in \mathbb{R}^{q \times N}$, such that $\mathbf{m}_t \approx \mathbf{H}\mathbf{u}_t$ is the state of the LDN system at time t . As pointed out by Chilkuri, the k th column of \mathbf{H} is simply $(\mathbf{H}^T)_k = \tilde{\mathbf{A}}^k \tilde{\mathbf{B}}$. This is equivalent to the mean impulse response of the continuous system during the k th discrete time-step (cf. Stöckel, 2021b):

$$(\mathbf{H}^T)_k = \frac{1}{N} \int_{t_0}^{t_1} \exp(\mathbf{A}\tau) \mathbf{B} \, d\tau = \tilde{\mathbf{A}}^k \tilde{\mathbf{B}}, \quad \text{where } t_0 = \frac{k}{N}, \quad t_1 = \frac{k+1}{N}. \quad (4.32)$$

Replacing the LTI system with a set of FIR filters makes the network purely feed-forward. We thus do not have to rely on backpropagation through time (e.g., Werbos, 1990) for training, and can use the FFT to quickly convolve pre-recorded test and training signals with the q FIR filters. This can vastly improve training throughput on GPUs (Chilkuri and Eliasmith, 2021). Still, once training has completed, we can always switch back to the recurrent implementation—this *may* (see below) be more efficient when performing realtime stream processing.

The idea of using FIR filters (i.e., temporal convolutions) in neural networks is by no means new. In fact, it harkens back to the early days of multi-layer networks (e.g., Waibel et al., 1989; Back and Tsoi, 1991) and has lately been repopularised by Bai, Kolter, and Koltun (2018). The novelty of the feed-forward LMU lies in using *fixed* FIR filters approximating an *orthogonal* sliding window spectrum, as well as exploiting the duality between the recurrent and feed-forward networks for efficient inference and training.

LMU layers and spatiotemporal tuning. When constructing neural networks, we generally stack multiple LMUs into an *LMU layer* (cf. Figure 4.42D). That is, the d -dimensional spatial input \mathbf{x}_t is projected onto ℓ LMUs using encoding vectors $\mathbf{e}_h^1, \dots, \mathbf{e}_h^\ell$. This results in a vector \mathbf{m}_t of ℓq states feeding into a nonlinear layer of n neurons via a weight matrix \mathbf{W}_m .

Importantly, using encoding vectors \mathbf{e}_h^i is only efficient if the number of LMUs is smaller than the number of spatial input dimensions, i.e., $\ell < d$. Otherwise, due to the linearity of convolution, we can simply filter each input dimension separately and implicitly learn the encoding vectors as a part of \mathbf{W}_m . In this case, each row in \mathbf{W}_m is a vector of length dq . This vector *exactly* corresponds to a flattened version of our encoding matrix \mathbf{E}^t (cf. eq. 4.25). Put differently, an LMU layer is a spatiotemporal NEF population with LDN tuning (cf. Section 4.3.3).

Algorithm 4.1: Euler linear time LDN update. The function below takes the current LDN system state $\mathbf{m} = (m_1, \dots, m_q)$, input u_t , system order q , window-width θ , as well as the time-step Δt and computes the state update \mathbf{m}' according to eq. (4.33) in $\mathcal{O}(q)$.

```

1 function EulerLDNUpdate( $\mathbf{m}, u, q, \theta, \Delta t$ )
2    $v \leftarrow \Delta t \theta^{-1}$ ,  $\mu_1 \leftarrow \sum_{i=1}^q m_i$ ,  $m'_1 \leftarrow m_1 - v(\mu_1 - u)$ 
3   for  $i = 2$  to  $q$  do
4      $k \leftarrow i - 2$  if  $i \geq 3$  else 1
5      $\mu_i \leftarrow \mu_k - 2m_{i-1}$ 
6      $m'_i \leftarrow m_i - v(2i - 1)(\mu_i - (-1)^i u)$ 
7   return  $(m'_1, \dots, m'_q)$ 

```

4.4.2 Computational Efficiency of the LDN LTI System

The main advantage of using an order q LTI system to approximate a sliding-window spectrum (as in the LRGF LMU) is memory efficiency in *online* stream-to-stream processing applications. After all, we only need to remember q state dimensions, compared to the last $N = \theta/\Delta t$ input samples when using FIR filters (as in the FIR LMU). Correspondingly, we can operate the network in regimes where $N \gg q$ without requiring large amounts of memory. However, there are some trade-offs to consider with respect to computational efficiency.

FIR filters can be more computational efficient than state-space LTI systems. On the surface, it may seem as if evaluating a discrete LTI system of order q in every time-step should be computationally more efficient than evaluating q FIR filters. After all, computing $\mathbf{m}_{t+1} = \tilde{\mathbf{A}}\mathbf{m}_t + \tilde{\mathbf{B}}u_t$ requires $\mathcal{O}(q^2)$ operations per time-step, while computing $\mathbf{m}_t = \mathbf{H}u_t$, that is, convolving \mathbf{u}_t with a set of FIR filters *online*, requires $\mathcal{O}(qN)$ operations, where $N \geq q$.

However, as pointed out by Gardner (1995), online convolution with q FIR filters can be implemented efficiently in about $34q \log_2(N)$ operations (amortised), albeit at the cost of $\mathcal{O}(N \log(N))$ memory. Still, this means that for large q , using FIR filters can be *computationally* more efficient than evaluating a generic LTI system.

Euler LDN update. Fortunately, the LDN system is not a generic LTI system. In fact, as pointed out by Voelker (2019, Figure 6.6) in the form of a circuit diagram, it is possible to compute the state update in $\mathcal{O}(q)$ operations if we discretise the LDN using an Euler step instead of the ZOH method from eq. (4.31). We get the following update equation and transformation matrix \mathbf{H}' :

$$\mathbf{m}_{t+1} = \mathbf{m}_t + \frac{\mathbf{A}\mathbf{m}_t + \mathbf{B}u_t}{N} = \tilde{\mathbf{A}}'\mathbf{m}_t + \tilde{\mathbf{B}}'u_t, \quad \text{where } N\Delta t = \theta \quad \text{and} \quad ((\mathbf{H}')^T)_k = (\tilde{\mathbf{A}}')^k \tilde{\mathbf{B}}'. \quad (4.33)$$

An algorithm for computing \mathbf{m}_{t+1} in $\mathcal{O}(q)$ time is outlined in Algorithm 4.1; we exploit the fact that (scaling factors aside) two rows i and $i-2$ only differ in one column $i-1$ (cf. Figure 4.22B).³¹

³¹Note that Algorithm 4.1 is for the “original” LDN system, i.e., *not* the rescaled version from eq. (4.24).

As any textbook on numerically solving differential equations will warn (e.g., Press et al., 2007, Chapter 17.1), Euler integration is rarely a good choice. It introduces instabilities and, with a $\mathcal{O}(\Delta t)$ residual, is rather imprecise for the expended computational effort. In contrast, the ZOH scheme is an optimal discretisation of the system impulse response (cf. eq. 4.32).

We characterise the error introduced by Euler discretisation and its asymptotic stability in Figure 4.43. However, note that—in our case—deviations from the continuous system are irrelevant as long as \mathbf{H}' spans an orthogonal basis. We characterise the “quality” of the generated transformation matrix \mathbf{H}' by the sum Σ of its normalised singular values (cf. Figures 4.43B and 4.44); this roughly corresponds to the number of intrinsic orthogonal basis functions.

Independent of the error measure, we observe a quadratic relationship between N and q for maintaining a constant error. In other words, when using the Euler update, increasing q mandates a quadratically higher sampling rate. This asymptotically negates the effect of switching to a faster update equation.

Higher order methods. Choosing higher-order Runge-Kutta methods instead of Euler can substantially reduce errors (Press et al., 2007, Chapter 17.1). For a Runge-Kutta method of order p , the discrete feedback matrix $\tilde{\mathbf{A}}^{\{p\}}$ is a truncated matrix exponential (cf. Whitney, 1969):

$$\tilde{\mathbf{A}}^{\{p\}} = \sum_{k=0}^p \frac{1}{k!} \left(\frac{\mathbf{A}}{N} \right)^k \quad \text{where} \quad \tilde{\mathbf{A}}^{\{1\}} = \mathbf{I} + \frac{\mathbf{A}}{N} = \tilde{\mathbf{A}}', \quad \text{and} \quad \tilde{\mathbf{A}}^{\{\infty\}} = \exp \left(\frac{\mathbf{A}}{N} \right) = \tilde{\mathbf{A}}.$$

Specifically, $\tilde{\mathbf{A}}^{\{2\}}$ corresponds to the midpoint method, and $\tilde{\mathbf{A}}^{\{4\}}$ to the canonical Runge-Kutta integrator. Note that the asymptotic complexity of computing $\tilde{\mathbf{A}}^{\{p\}} \mathbf{m}_t$ is still in $\mathcal{O}(q)$. Runge-Kutta methods consist of p nested Euler steps, all of which can be evaluated using Algorithm 4.1.

We analyse higher order methods in Figure 4.45. Switching to the midpoint integrator makes a much wider range of (N, q) -pairs usable compared to Euler. Further increasing the order to $p = 4$ has no appreciable effect.³² Crucially, using a higher-order integrator does not affect contour line curvature. N must still be in $\mathcal{O}(q^2)$ to maintain a constant basis quality.

Efficient modified Fourier system update. An efficient Euler state update algorithm exists for the modified Fourier LTI system as well. In the continuous case, the feedback matrix is the difference between \mathbf{A}_F , a matrix describing a set of harmonic oscillators, and an information erasure dampening term Γ_F (cf. eq. 4.22). The dampening term Γ_F is an outer product $\mathbf{e}(\theta) \otimes \mathbf{d}(\theta)$, and hence $\Gamma_F \mathbf{m}$ can be computed in $\mathcal{O}(q)$. Moreover, \mathbf{A}_F merely consists of a series of 2×2 block matrices along the diagonal; correspondingly, computing $\mathbf{A}_F \mathbf{m}$ is also in $\mathcal{O}(q)$.

The error characteristics for using the Euler update differ slightly from those of the LDN (cf. Figures 4.46 and 4.47). The system is highly unstable when using Euler (not depicted); however, when using a higher-order Runge-Kutta update, the system generates an expressive basis for a wide range of q and N . Still, N remains slightly superlinear in q (i.e., $N \approx q^{\frac{4}{3}}$).

³²Technically, Euler’s method reaches slightly higher relative Σ above 100% (up to 103%). We believe that this is mostly due to the poor discretisation adding noise to \mathbf{H}' . The practical benefit of this is likely negligible.

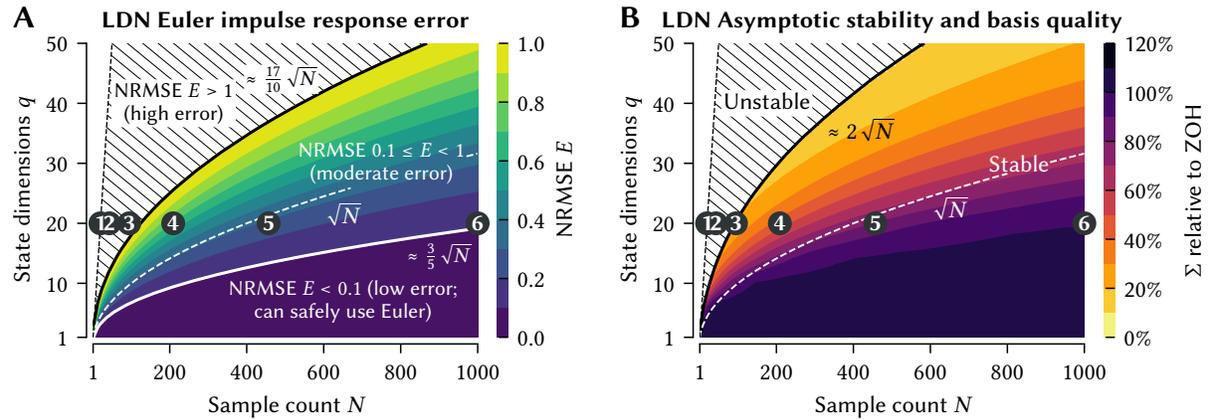


Figure 4.43: Exploring the effect of Euler discretisation on the LDN impulse response and stability for different state-space dimensions q and sample counts $N = \theta\Delta t^{-1}$. Circled numbers correspond to the (N, q) -pairs in Figure 4.44 below. Thin dashed line to the left indicates $N < q$. **(A)** NRMSE between the ZOH and Euler basis transformation matrices \mathbf{H}, \mathbf{H}' (eqs. 4.31, 4.33) and an \mathbf{H}' computed using Euler’s method. **(B)** Asymptotic stability and basis quality. Points above the black line are asymptotically unstable, the Euler system possesses a growth factor $|1 + \lambda_i/N| > 1$. Coloured background is the sum of the (normalised) singular values Σ of \mathbf{H}' divided by the same quantity for \mathbf{H} at this point. A value of 100% indicates that the spanned basis is as expressive as the one obtained with ZOH discretisation.

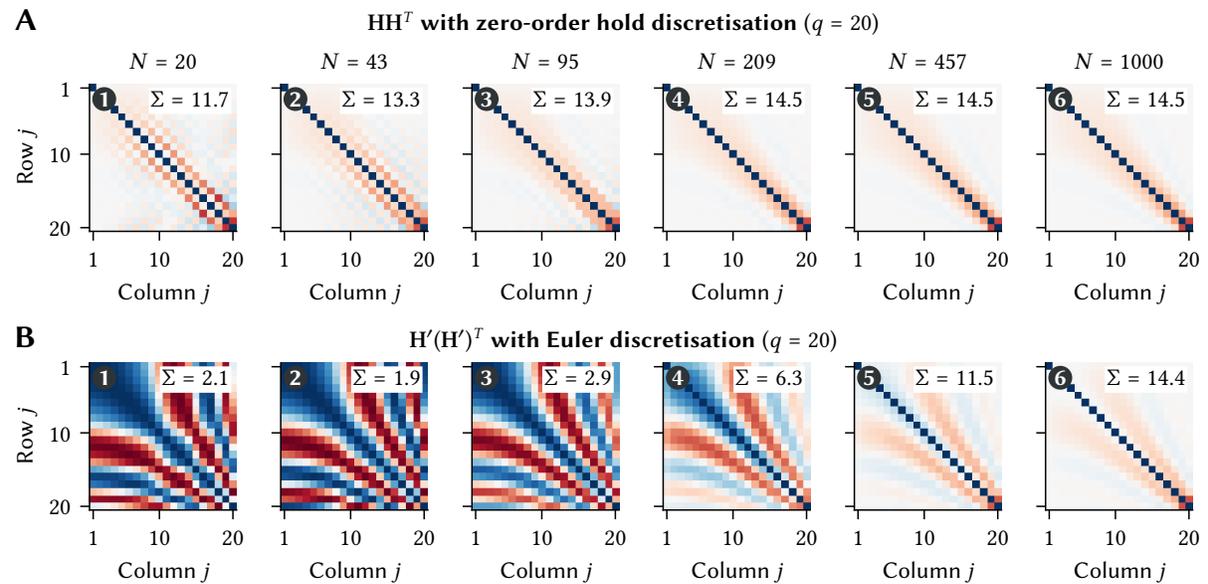


Figure 4.44: Orthogonality of the ZOH and Euler LDN basis transformation matrix for $q = 20$. Each plot is the result of multiplying **(A)** the ZOH based \mathbf{H} or **(B)** Euler based \mathbf{H}' with its own transpose. Dark red cells correspond to an entry of -1 , dark blue cells to 1 . Σ is the sum of the normalised singular values for each \mathbf{H} or \mathbf{H}' ; this roughly indicates the number of “hidden” orthogonal basis functions (optimally $\Sigma = q$). **(A)** The ZOH based basis transformation matrix \mathbf{H} is nearly orthogonal for all N . **(B)** The Euler based \mathbf{H}' requires a twenty times larger N to reach the same degree of orthogonality.

4.4. Applications to Artificial Neural Networks

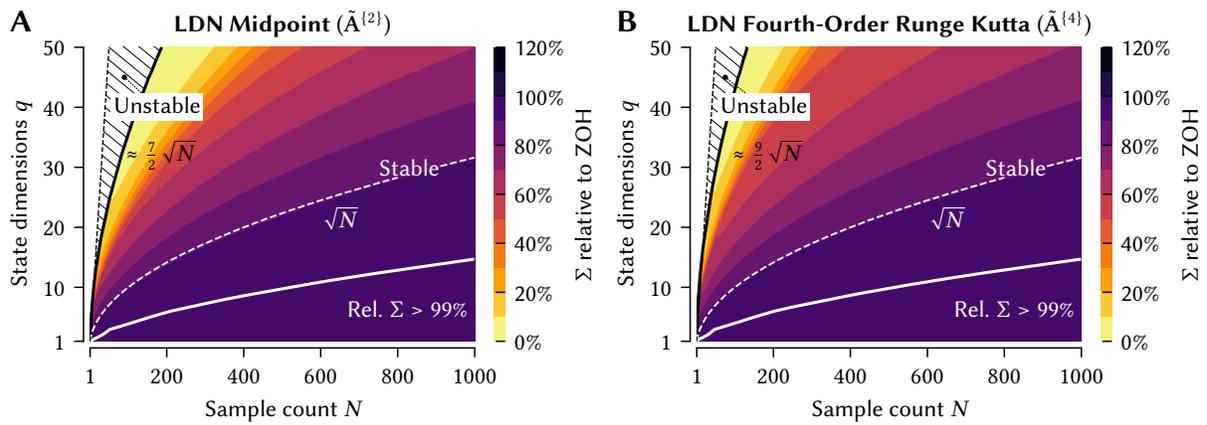


Figure 4.45: Exploring the effect of higher order discretisation on the LDN basis quality and stability. Same plot as Figure 4.43B, but for (A) the midpoint integrator (a second-order Runge-Kutta method) or (B) the fourth-order Runge-Kutta integrator. Both perform substantially better than Euler’s method.

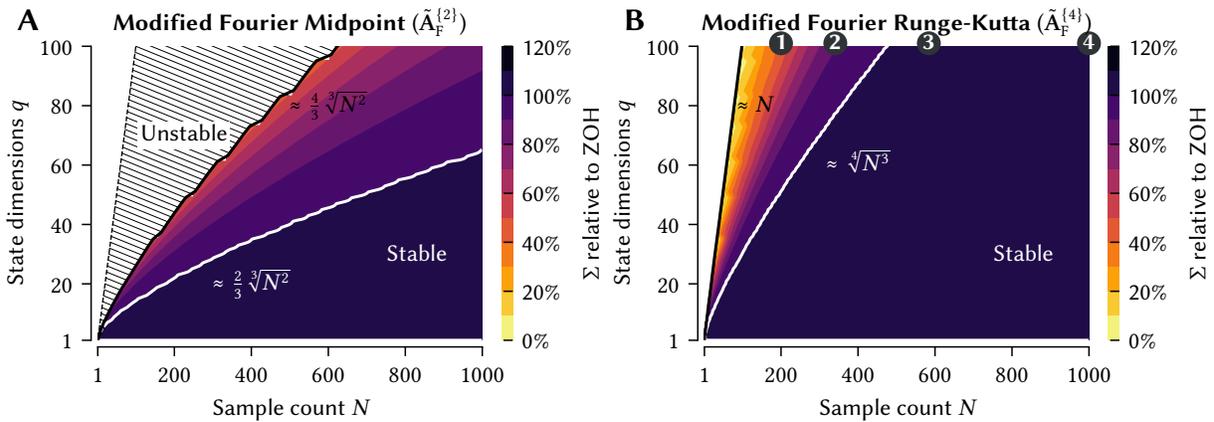


Figure 4.46: Effect of higher order discretisation on the modified Fourier basis. Same analysis as in Figures 4.43 and 4.45, however, for the modified Fourier system. Only odd q are included in the analysis. (A) Compared to the LDN, a large portion of the space is unstable when using the midpoint method. The contour lines no longer describe a strictly quadratic relationship between q and N , and a large portion of the space generates an expressive basis. (B) No asymptotic instabilities are visible when using the fourth-order Runge-Kutta solver; N is *almost* linear in q . Markers correspond to Figure 4.47.

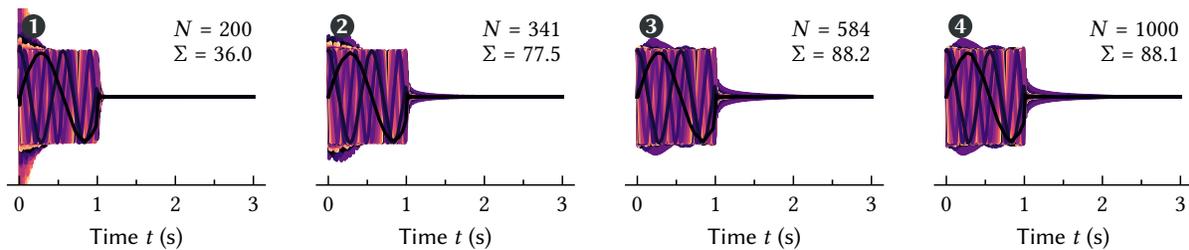


Figure 4.47: Integrating the Modified Fourier Basis using Runge-Kutta. Each line corresponds to the impulse response of one of $q = 101$ state dimensions. The system is stable even for small N to q ratios.

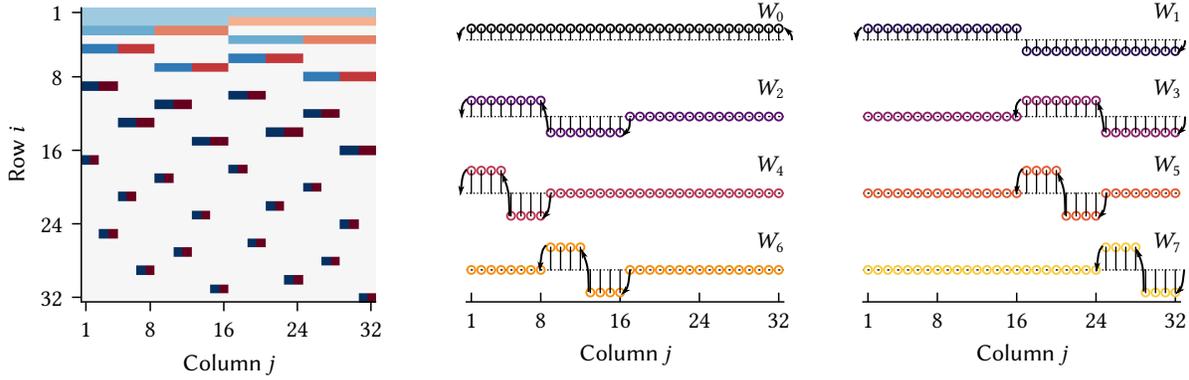


Figure 4.48: Visualisation of the orthonormal discrete Haar wavelets. *Left:* Discrete basis transformation matrix \mathbf{W} for $q = N = 32$. Blue is positive, red negative. *Right:* Visualisation of the first eight rows in \mathbf{W} . Arrows correspond to the transitions mentioned in the text.

4.4.3 Efficient Sliding-Window Spectrum Transformations

The superlinear dependency between N and q for Runge-Kutta state updates implies that we effectively only reach a time complexity between $\mathcal{O}(q^{2.3})$ and $\mathcal{O}(q^3)$ over a window period θ with N samples. Moreover, and independent of the discretisation method, remember that we only *approximate* a sliding-window spectrum (Section 4.2.1) if we use an order q LTI system. This is due to ringing caused by not being able to losslessly reconstruct the history of non-bandlimited inputs from the state \mathbf{m}_t when performing information erasure (eq. 4.20; Figure 4.20).

For discrete sliding-window transformations, these artefacts can be avoided at the cost of recording the input history \mathbf{u}_t . In this case, we no longer need to rely on \mathbf{m}_t to approximate a delayed version of the input, and can use a discrete version of Lemma 4.3 to realise a perfect window. Of course, this requires $\mathcal{O}(N)$ memory and may be infeasible for some applications.

Crucially, for some discrete bases, keeping \mathbf{u}_t in memory allows us to construct a *perfect* sliding-window transformation with $\mathcal{O}(qN)$ operations per window period θ , where N is fully independent of q . Below, we discuss the discrete Haar wavelets and the sliding discrete Fourier transform. We compare these bases to other sliding transformations in the next section.

Haar Wavelets. One particular set of discrete basis functions (Definition 4.7) famous for allowing efficient state updates are the *Haar wavelets* depicted in Figure 4.48 (Haar, 1910). A discrete Haar transformation $\mathbf{W}\mathbf{u}$ with $\mathbf{W} \in \mathbb{R}^{q \times N}$ can be computed in $\mathcal{O}(N)$ (Kaiser, 1998), and the sliding spectrum can be updated in $\mathcal{O}(q)$ per sample.

To see this, assume that we store both our current state \mathbf{m}_t and the last N input samples $\mathbf{u}_t = (u_{t-N+1}, \dots, u_t)$ in memory. Sliding one of the discrete basis functions W_i over \mathbf{u}_t , there are at most three samples in the input history that cross a transition point (arrows in the figure) and thus influence the value of \mathbf{m}_t . Instead of re-convolving with W_i and \mathbf{u}_t in every time-step, we update \mathbf{m}_t by accounting for the transitioning samples.

Sliding Discrete Fourier Transform (SDFT). Similar $\mathcal{O}(q)$ update algorithms can be obtained for the discrete Fourier and cosine basis (cf. Section 4.2.2), namely the sliding discrete Fourier (SDFT; Springer, 1991; Jacobsen and Lyons, 2003) and cosine transformation (SDCT; Kober, 2004). Particularly, the SDFT is given as (using real instead of complex coefficients):

$$\begin{aligned} m_t^{2k-1} &= m_{t-1}^{2k-1} \cos(f_k) - m_{t-1}^{2k} \sin(f_k) - u_{t-N} + u_t, \\ m_t^{2k} &= m_{t-1}^{2k-1} \sin(f_k) + m_{t-1}^{2k} \cos(f_k), \end{aligned} \quad \text{where } f_k = \frac{2\pi k}{N}, \quad (4.34)$$

where m_t^{2k-1} and m_t^{2k} are the real and imaginary coefficients belonging to the frequency term f_k . Note the similarity to a harmonic oscillator LTI system with an additional delayed input, akin to our perfect continuous LTI rectangle window in Lemma 4.3.³³

4.4.4 Experiments

So far, we have seen several alternatives to the LDN system used in the Legendre Memory Unit. As is summarised in Table 4.1, these systems can be implemented in different ways, each with different characteristics in terms of time and space complexity, and different degrees to which they realise an orthogonal function basis (Figure 4.49).

To explore how well these systems actually perform in a neural network context, we repeat two experiments from the original LMU paper (Voelker, Kajić, et al., 2019); specifically, the psMNIST and Mackey-Glass tasks. Our goal is to explore in how far changing the sliding-window transformation affects the performance of the LMU in comparison to the LDN variant.

Our results indicate that the basis choice has a significant, but small effect on the system performance. The power of the LMU architecture stems from using a *fixed* and *orthogonal* temporal projection; the particular shape and quality of this projection are secondary. Practitioners should select a transformation from Table 4.1 that suits their particular constraints.

psMNIST. The MNIST dataset (Lecun et al., 1998) contains 28×28 pixel greyscale images of hand-written digits between zero and nine. The task is to classify the digits; state-of-the-art classification accuracies are above 99% (Baldominos, Saez, and Isasi, 2019).

The permuted sequential MNIST (psMNIST) task has originally been proposed by Le, Jaitly, and Hinton (2015). The idea is to turn the MNIST dataset into a benchmark for the memory capacity of recurrent neural networks. Specifically, each image is treated as a sequence \mathbf{u} of $N = 784$ pixels. Each pixel is fed one-by-one into the network in consecutive timesteps. Once the final sample has been processed, the system must correctly classify the digit. To eliminate spatial correlations in the input, a random but fixed permutation π is applied to the samples (cf. Figure 4.50). The input signal is thus of the form $\mathbf{u} = (u_{\pi(1)}, \dots, u_{\pi(784)})$.

³³Note that, in contrast to the Haar basis, computing the SDFT does not require random access to the signal history \mathbf{u}_t . This drastically reduces the required memory bandwidth.

Table 4.1: Time and space complexity of different sliding transformations. Memory requirements only take state retained between updates into account; we exclude constant matrices and filters. The column “window N ” describes the relationship between N and q (typically $N \geq q$). Batch processing corresponds to computing the sliding window spectrum for every point in time for an entire recorded signal \mathbf{u} of length N in one go. Any sliding-window transformation has a FIR representation and can always fall back to one of the algorithms in the first three rows.

Basis	Algorithm	Online (per sample)		Batch	Window N
		Time	Memory	Time	
●, ○, ▲ Any	Naïve	$\mathcal{O}(qN)$	$\mathcal{O}(N)$	$\mathcal{O}(qN^2)$	$\mathcal{O}(q)$
	FFT ^[1]	/	/	$\mathcal{O}(qN \log(N))$	$\mathcal{O}(q)$
	Gardner ^[2]	$\mathcal{O}(q \log(N))$	$\mathcal{O}(N \log(N))$	/	$\mathcal{O}(q)$
● LDN	ZOH ^[3]	$\mathcal{O}(q^2)$	$\mathcal{O}(q)$	$\mathcal{O}(q^2N)$	$\mathcal{O}(q)$
	Euler ^[4]	$\mathcal{O}(q)$	$\mathcal{O}(q)$	$\mathcal{O}(qN)$	$\approx \mathcal{O}(q^2)$
● Mod. Fourier	ZOH ^[3]	$\mathcal{O}(q^2)$	$\mathcal{O}(q)$	$\mathcal{O}(q^2N)$	$\mathcal{O}(q)$
	Euler ^[4]	$\mathcal{O}(q)$	$\mathcal{O}(q)$	$\mathcal{O}(qN)$	$\approx \mathcal{O}(q^{\frac{4}{3}})$
○ Fourier	SDFT ^[5]	$\mathcal{O}(q)$	$\mathcal{O}(N)$	$\mathcal{O}(qN)$	$\mathcal{O}(q)$
○ Cosine	SDCT ^[6]	$\mathcal{O}(q)$	$\mathcal{O}(N)$	$\mathcal{O}(qN)$	$\mathcal{O}(q)$
○ Haar	FHT ^[7]	$\mathcal{O}(q)$	$\mathcal{O}(N)$	$\mathcal{O}(qN)$	$\mathcal{O}(q)$

● Sliding transformation with a continuous LTI state-space system of order q ; ○ Discrete sliding transformation with a fast update equation; ▲ Sliding transformation with a FIR filter representation. [1] Cooley and Tukey, 1965; [2] Gardner, 1995; [3] See eq. (4.31); [4] See Section 4.4.2; [5] Springer, 1991; Jacobsen and Lyons, 2003; [6] Kober, 2004; [7] Kaiser, 1998.

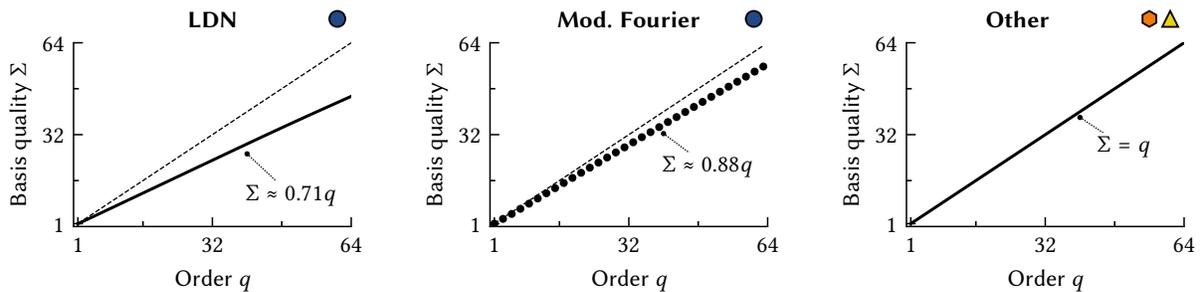


Figure 4.49: Comparing the orthogonality of different methods for generating sliding-window spectra. Plotted is the sum Σ of the normalised singular values of the system transformation matrices \mathbf{H} for different q (at $N = 1000$). When using sliding discrete transformations (○) or simple a set of arbitrary FIR filters (▲), it holds $\Sigma = q$. The quality of the basis generated by the LDN and the modified Fourier system (●) is suboptimal. Note that $\Sigma \approx 0.88q$ for the modified Fourier basis (for odd q); the slope of 88% is close to the percentage to which the individual state dimensions have been slowed down (90%).

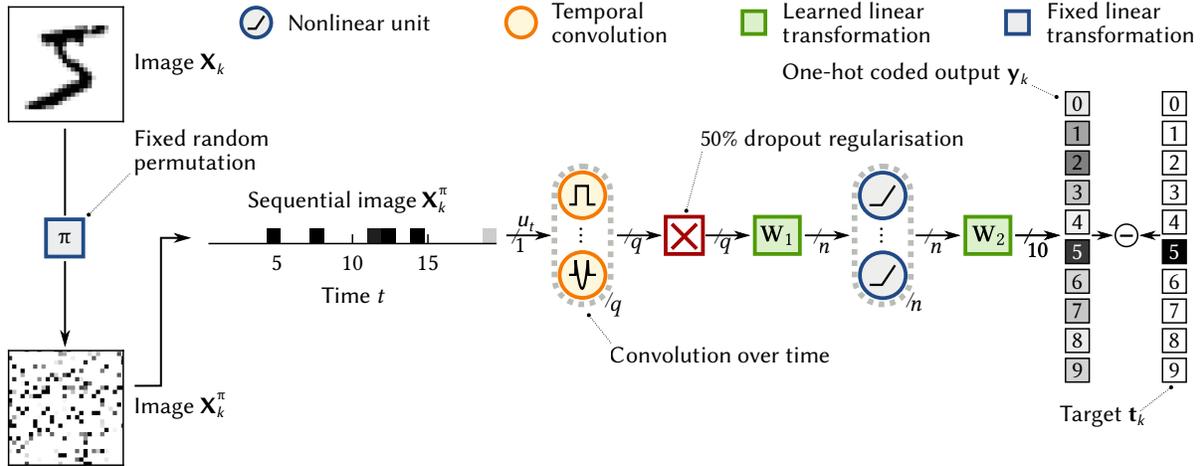


Figure 4.50: Overview of the psMNIST task and our network architecture. Input images are randomly permuted and serialised into a stream of pixels presented over time. After all $N = 784$ pixels have been fed into the network, the network must output the correct one-hot coded classification. In our particular case $q = 468$ and $n = 346$ for a total of 166 000 weights.

Additional constraints. The psMNIST task as described above is slightly under-defined. Specifically, two additional constraints should be met. First, the network should use less state memory than is required to just store the whole image (Voelker, Kajić, et al., 2019). Second, the network should support *serial execution* (Chandar et al., 2019). In other words, the network must produce correct classifications over time, even if multiple input signals are concatenated.

Implementing temporal convolution as FIR filters of length N (cf. Section 4.4.1) intrinsically fulfils the serial execution constraint, but violates the memory constraint. Each filter requires access to the past N samples; we essentially just compute the inner product between each of the q FIR filters and the input. The point of using the LDN or modified Fourier system impulse response is that the same convolution can be performed by an LTI system of order (and thus memory) $q < N$ within the recurrent LRGF LMU network.³⁴ Therefore, the results for our $\mathcal{O}(N)$ memory transformations should be taken with a grain of salt.

Methods. Our network architecture is similar to the setup used by Voelker, Kajić, et al. (2019). We provide an overview of the model in Figure 4.50; the code describing the network can be found in Appendix B.3.3. We apply a set of $q = 468$ FIR filters to the $N = 784$ input samples. The filter output is passed through a 50% dropout layer for regularisation (Hinton et al., 2012), followed by $n = 346$ ReLU neurons that nonlinearly processes the temporal representation. The neural activities are linearly projected onto a one-hot coded output. Note that $q = 468$ is motivated by the amount of memory used by Chandar et al. (2019).

³⁴Note that both the LDN and modified Fourier system impulse response produce ringing artefacts that extend beyond the first N samples (cf. Figure 4.47), that is, there is some interaction between two consecutively presented input images. We ignore this to be consistent with the way the psMNIST task is typically evaluated in the literature. However, we take ringing into account in the Mackey-Glass experiment.

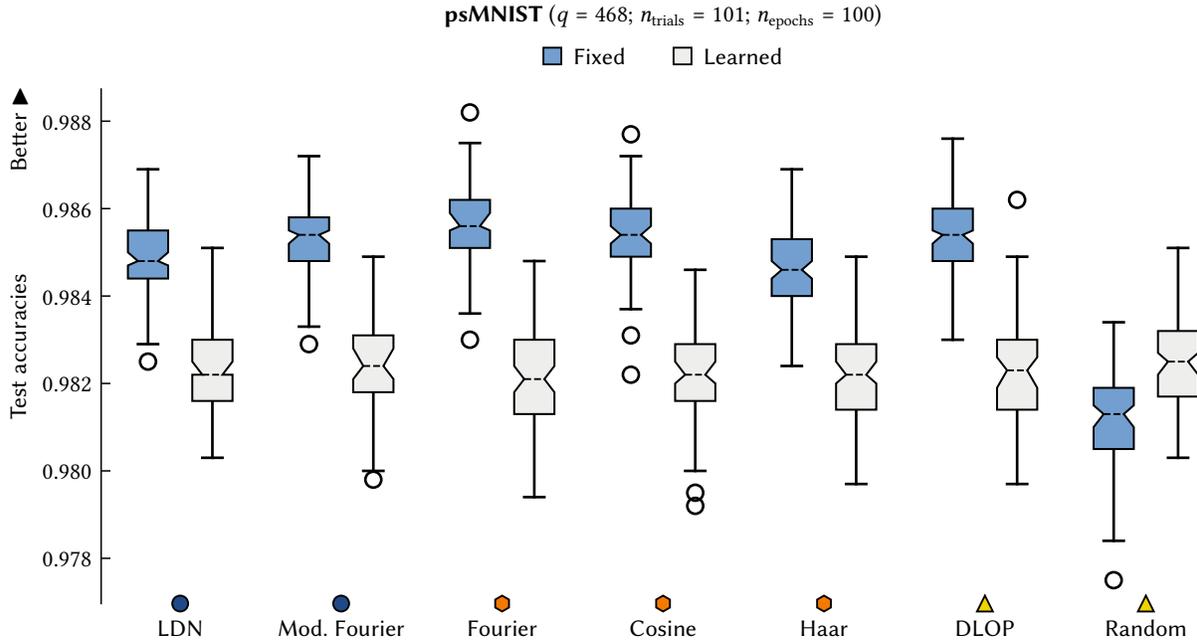


Figure 4.51: Classification accuracies for the psMNIST dataset using different sliding-window transformations. Depicted are standard box-plots over 101 trials for each basis, each with a different random permutation π and weight initialisations. Box corresponds to the first and third quartile; whiskers are the minimum/maximum after outlier rejection; outliers are depicted as circles. Dashed black line is the mean, notches correspond to the bootstrapped 95% confidence interval. Numerical values and significance levels are provided in Table 4.2. Symbols correspond to possible algorithms from Table 4.1.

As is customary, we split the MNIST dataset into 50 000 training and 10 000 validation samples. We use a categorical cross-entropy loss function and an Adam optimiser (Kingma et al., 2015) with default parameters over 100 epochs at a batch size of 100. The reported test errors are computed for the epoch with the smallest validation error. For $q = 468$ the number of trainable parameters is $\approx 166\,000$. Per default, the FIR filters are fixed and initialised with one of the previously discussed discretised basis transformations with window length $N = 786$ (i.e., the number of pixels). The LTI systems are discretised using zero-order hold.

As a point of comparison, we repeat the experiments with randomly initialised FIR filters, as well as enabled learning for the FIR filter matrices. Learning the FIR filters adds another 350 000 parameters and is supposed to test in how far our original transformations are optimal.

Results. Results are depicted in Figure 4.51 and Tables 4.2 and 4.3; learning curves are provided in Figure B.10. The modified Fourier, Fourier, cosine and discrete Legendre (DLOP; Section 4.2.2) basis outperform the LDN and Haar wavelets slightly but significantly. The LDN result is state-of-the-art for the psMNIST task at about 98.5% accuracy (Chilkuri and Eliasmith, 2021). There is no significant difference between initialisations when learning the convolution; enabling learning on the FIR filters always results in the same, substantially worse performance.

Table 4.2: Test accuracies for the psMNIST experiment for $q = 468$. Data over $n = 101$ trials and 100 epochs. Q1 and Q3 are the 25- and 75-percentile, respectively. The best three results are highlighted in each column (darker colours are better).

Basis	■ Fixed convolution				■ Learned convolution			
	Mean	Median	Q1	Q3	Mean	Median	Q1	Q3
● LDN	98.49%	98.48%	98.44%	98.55%	98.23%	98.22%	98.16%	98.30%
● Mod. Fourier	98.53%	98.54%	98.48%	98.58%	98.24%	98.24%	98.18%	98.31%
● Fourier	98.56%	98.56%	98.51%	98.62%	98.21%	98.21%	98.13%	98.30%
● Cosine	98.54%	98.54%	98.49%	98.60%	98.22%	98.22%	98.16%	98.29%
● Haar	98.47%	98.46%	98.40%	98.53%	98.22%	98.22%	98.14%	98.29%
▲ DLOP	98.54%	98.54%	98.48%	98.60%	98.23%	98.23%	98.14%	98.30%
▲ Random	98.11%	98.13%	98.05%	98.19%	98.24%	98.25%	98.17%	98.32%

Table 4.3: Statistical significance of the psMNIST test accuracies. The given significance levels are based on a two-sided Kolmogorov-Smirnov test; $\cdot \hat{=} p < 0.05$, $\cdot\cdot \hat{=} p < 0.01$, $\cdot\cdot\cdot \hat{=} p < 0.001$.

Basis	■ Fixed convolution							■ Learned convolution						
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(1)	(2)	(3)	(4)	(5)	(6)	(7)
● LDN	(1)							
● Mod. Fourier	(2)							
● Fourier	(3)							
● Cosine	(4)							
● Haar	(5)							
▲ DLOP	(6)							
▲ Random	(7)							

Discussion. Learned convolutions performing worse is likely a result of the increased parameter count and overfitting to the training data (cf. Figure B.10). Visualising the learned FIR filters (data not shown), we find that they do not differ substantially from their original initialisation. This suggests that our orthogonal transformations are locally optimal.

Thinking about the FIR filter matrix \mathbf{H} purely in terms of a projection $\mathbf{H}\mathbf{u}$, it is unclear why our systematic bases work better than a random initialisation—after all, this results in an (almost) orthogonal projection as well. There is a history of successfully applying orthogonal basis transformations such as the discrete Cosine or Haar transform to the MNIST dataset for spatial decorrelation (Baldominos et al., 2019). However, these experiments are without pixel permutation; with permutation, spatial decorrelation is an unlikely explanation.

One hint at the worse performance of the random initialisation is our “orthogonality measure” Σ (that is, the sum of the normalised singular values). For the random FIR filters Σ is only about $0.5q$, while we reach higher Σ for our other bases (cf. Figure 4.49).

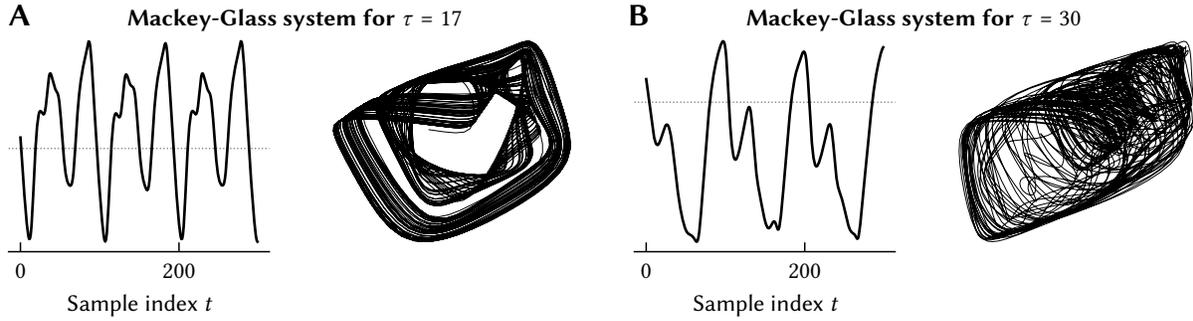


Figure 4.52: Visualisation of the Mackey-Glass system as used in our benchmark task. Left plot shows a short segment of the state evolution, the right plot a 2D delay embedding (delay of 10 samples).

Mackey-Glass. The Mackey-Glass dynamical system (Mackey and Glass, 1977) is a popular benchmark for time-series prediction (cf. Mendel, 2017, Section 4.3.1). The goal is to predict the time-course of the following differential equation (cf. Figure 4.52).

$$\dot{u}(t) = \frac{au(t - \tau)}{1 + u(t - \tau)^{10}} - bu(t) \quad \text{for } t \geq 0,$$

where $a = 0.2$, $b = 1.2$. For $t < 0$ we assume that the state variable $u(t)$ is a Gaussian process with mean $\mu = 1.2$ and standard-deviation $\sigma = 1$. Notably, the system behaves chaotically if $\tau \geq 17$; that is, small perturbations to the initial state can lead to dramatically different state trajectories. This makes predicting the system particularly hard. For our experiment, we construct a network that attempts to predict the time-course of the discretised Mackey-Glass system with $\tau = 30$ for the next fifteen state samples u_{t+1}, \dots, u_{t+15} given the recent state history.

Dataset. For our experiments, we choose $\tau = 30$. As a training dataset we generate 400 Mackey-Glass state trajectories consisting of 10 000 samples at $\Delta t = 1$ using a fourth order Runge-Kutta integrator; the trajectories are normalised to mean zero and standard deviation one. All trajectories are different due to the stochasticity of the initial $u(t)$ for $t < 0$.

We randomly extract 100 input sequences of length 117 followed by a target sequence of length 15 from each trajectory; this results in 40 000 training samples. We similarly generate 10 000 validation and 10 000 test samples. Training, validation, and test samples are all taken from separately generated trajectories. The sample length 117 corresponds to the absolute width of the combined FIR filters in our network; as is illustrated in Figure 4.53B, only the last 37 samples are effectively taken into account for the prediction.

Methods. Our neural network architecture is inspired by Voelker, Kajić, et al. (2019) and depicted in Figure 4.53. The relevant code may be found in Appendix B.3.3.

There are four cascading LMU layers; the last three layers consist of ten stacked sliding-window transformations each. In contrast to the previous experiment, each transformation uses small q with $N = q$. Note that a sliding-window transformation with $q = N$ is lossless, and that the linear readout weights can theoretically decode the state in terms of any other

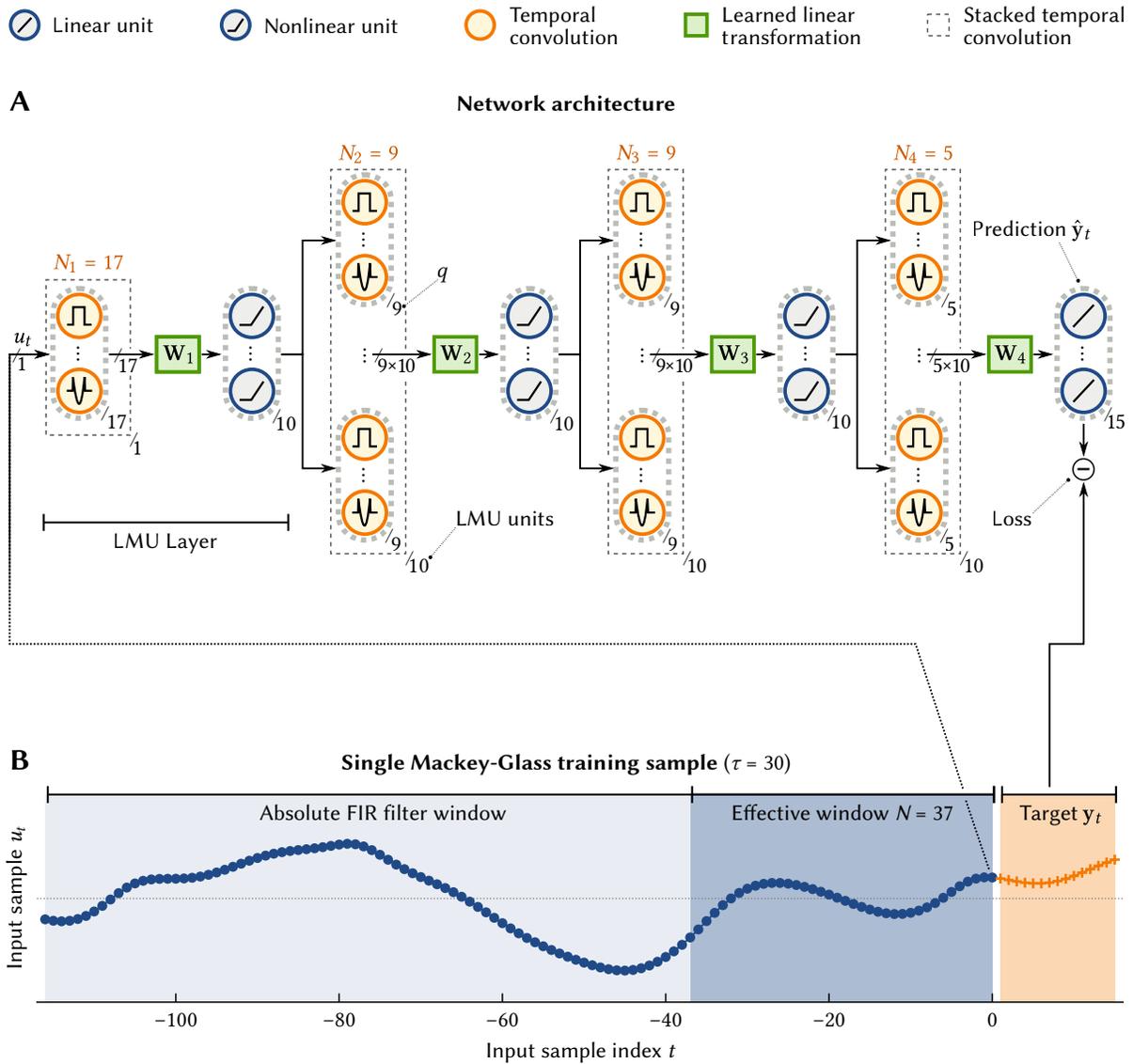


Figure 4.53: Overview of the Mackey-Glass prediction network and dataset. **(A)** Samples u_t fed into the network pass a series of four LMU layers (here depicted as FIR LMUs); the output is a prediction \hat{y}_t of the next fifteen samples u_{t+1}, \dots, u_{t+15} . Every sample fed into the network immediately updates the prediction in the output. When using the LDN or the modified Fourier system as a sliding transformation, the FIR LMUs can of course be replaced with a corresponding recurrent LRGF LMU. **(B)** Exemplary input-target pair for $\tau = 17$. The absolute window of the FIR filters in our network amounts to 117 samples; samples older than this cannot *possibly* affect the output. However, only the last 37 samples *effectively* influence the system. The longer absolute window ensures that we account for ringing artefacts in the LDN and modified Fourier system (cf. Figure B.9).

basis.³⁵ Specifically, the first LMU layer uses $q_1 = N_1 = 17$, the second and third $q_{23} = N_{23} = 9$, and the final layer $q_4 = N_4 = 5$. Assuming that our sliding-transformation implements a perfect rectangle window, this results in the aforementioned *effective* combined window width of 37.³⁶ The absolute window width stems from the impulse response of the LDN and modified Fourier system technically extending beyond N_i . For good measure, we therefore extend the FIR filters to be three times as long as the actual window width.

As in the previous experiment, we compare fixed convolutions, including random initialisations, to a version of the network where the convolutions are initialised in the same way, but then trained during training. Note that we do not train the extended filters. The total number of parameters is 2750; another 476 parameters are added if the convolutions are trained.

We train the network for 100 epochs with a batch size of 100 using a standard Adam optimiser. The final test error is computed for the parameters in the epoch with the smallest validation error. Our training loss is the MSE, we report the NRMSE. Note that our results are not directly comparable to those in the literature due to using a different τ and normalisation of the Mackey-Glass dataset. Our goal is merely to compare different LMU variants.

Results. Results are given in Figure 4.54 and Table 4.4; see Figure B.10 for learning curves. Overall, all bases perform similarly, with the LDN outperforming the Fourier and modified Fourier bases. The modified Fourier basis performs worst. The random and identity initialisations result in by far the highest errors, and the LDN, cosine, and DLOP filters result in the lowest error. Learning the convolution closes the performance-gap between the bases.

Discussion. The cause of the unfavourable performance of the modified Fourier basis is unclear. We can rule out the ringing artefacts (cf. Figure B.9)—the performance gap remains after forcing a perfect rectangle window (see Figure B.11 and Table B.9); however, the superiority of the LDN over the other bases vanishes in this case.

A possible culprit is that the best-performing bases (i.e., the LDN, DLOP, and cosine basis) are aperiodic, and that even the unmodified Fourier basis performs slightly worse than the other bases. Using a sub-optimal Fourier basis may thus further reduce the performance to the level observed here. However, a more detailed analysis is required.

Overall, our results indicate that the basis choice is secondary, as long as it is not random or an identity basis. Using fixed orthogonal temporal convolutions can greatly reduce the number of trainable parameters without sacrificing performance. The particular architecture to choose (e.g., FIR or LRGF LMU), or whether to rely on an order q LTI system instead of an efficient sliding-window transformation depends on the particular application. Limiting constraints are the sampling rate (determining whether a fast Euler update is possible), available memory, as well as latency requirements (i.e., whether to use batch or online processing).

³⁵ $q = N$ is a good use-case for an efficient sliding-window spectrum (Section 4.4.3). This requires $2q$ memory and less computation than the LDN system—we cannot use an efficient Euler update for $q = N$ (Section 4.4.2).

³⁶Specifically, $37 = N_1 + N_2 + N_3 + N_4 - 3$. Each layer “consumes” $N_i - 1$ samples, and one additional sample is required to obtain a single output sample. See also our discussion in Appendix B.3.3.

4.4. Applications to Artificial Neural Networks

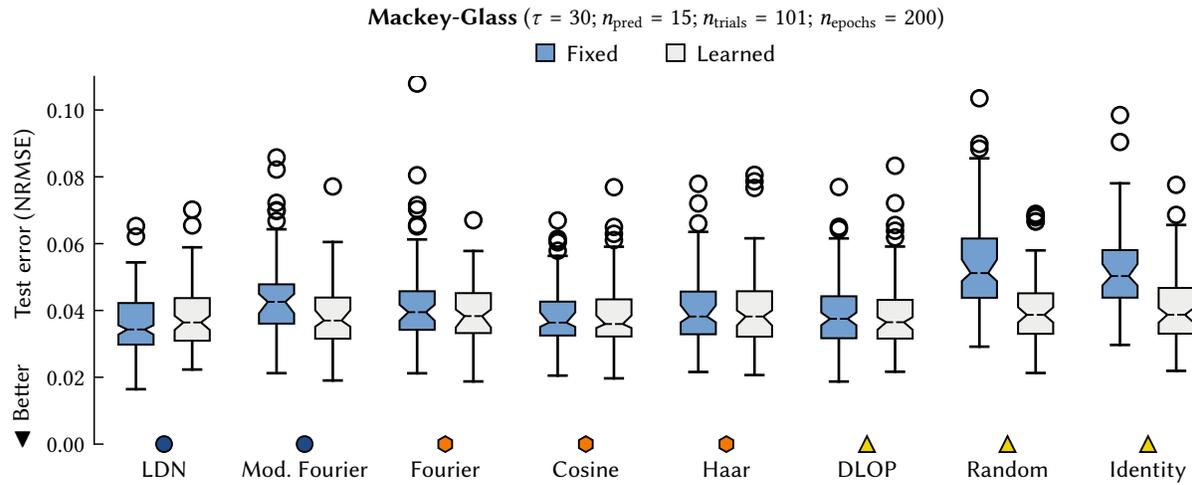


Figure 4.54: Prediction errors for the Mackey-Glass dataset using different sliding-window transformations. See Figure 4.51 for a description of the plot. Numerical values are given in Table 4.4.

Table 4.4: Prediction errors and statistical analysis for the Mackey-Glass dataset. Data over 101 trials after 100 epochs of training. See Tables 4.2 and 4.3 for a description. The given significance levels are based on a two-sided Kolmogorov-Smirnov test; $\hat{\cdot} \hat{=} p < 0.05$, $\hat{\cdot\cdot} \hat{=} p < 0.01$, $\hat{\cdot\cdot\cdot} \hat{=} p < 0.001$.

Basis	Fixed convolution				Learned convolution			
	Mean	Median	Q1	Q3	Mean	Median	Q1	Q3
● LDN	3.63%	3.43%	2.98%	4.22%	3.80%	3.64%	3.09%	4.37%
● Mod. Fourier	4.38%	4.26%	3.61%	4.78%	3.89%	3.70%	3.15%	4.39%
● Fourier	4.21%	3.95%	3.42%	4.58%	3.93%	3.83%	3.32%	4.52%
● Cosine	3.96%	3.64%	3.25%	4.26%	3.80%	3.59%	3.22%	4.33%
● Haar	4.01%	3.82%	3.29%	4.57%	3.95%	3.82%	3.22%	4.58%
▲ DLOP	3.90%	3.75%	3.17%	4.42%	3.86%	3.65%	3.16%	4.32%
▲ Random	5.38%	5.12%	4.38%	6.15%	4.04%	3.87%	3.30%	4.51%
▲ Identity	5.21%	5.03%	4.38%	5.81%	4.04%	3.87%	3.31%	4.67%

Basis	Fixed convolution								Learned convolution							
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
● LDN	(1)								
● Mod. Fourier	(2)								
● Fourier	(3)								
● Cosine	(4)									
● Haar	(5)								
▲ DLOP	(6)								
▲ Random	(7)								
▲ Identity	(8)								

4.5 Conclusion

We proposed a linear model of spatiotemporal tuning as a generalisation of the NEF dynamics principle (cf. Section 2.3.5; Eliasmith and Anderson, 2003, Chapter 8). In particular, we incorporated a model of temporal tuning in visual cortex (cf. Carandini et al., 1999) into the NEF tuning curve equation. This way, we can construct biologically plausible spiking neural networks that approximate spatiotemporal functions. We furthermore pointed out that there is a direct connection between the Legendre Memory Unit (LMU; Voelker, Kajić, et al., 2019), a promising component for stream-to-stream processing in artificial neural networks, and our spatiotemporal NEF populations.

Conceptually, our most important argument is that it is possible to systematically harness diverse temporal tuning as a resource for temporal computation. That is, the pronounced dynamical properties of biological neural networks are an important part of the computation that is being performed. Just as neural nonlinearities and diverse *spatial* tuning can be exploited to compute non-temporal functions, it is possible to exploit the dynamics inherent to biological networks to form temporal bases, from which we can in turn decode functions *through time*. We focused on synaptic filters as a primary source of dynamics, but neural dynamics and even signal propagation delays could similarly be taken into account.

In line with the NEF dynamics principle, we suggested that recurrent connections play a crucial role in forming diverse temporal tuning. While feed-forward connections can recombine existing temporal tuning, they are inherently limited by the fixed filters along the signal path.

Solving for weights that realise desired dynamics through recurrent connections is often accomplished through computationally intensive methods such as backpropagation through time (Werbos, 1990) or, in the context of modelling neurobiological systems, FORCE (Sussillo et al., 2009; Nicola et al., 2017; see Voelker, 2019, Section 2.2.4 for a review). With our approach, we instead rely on the “self-fulfilling prophecy” inherent to the NEF. We assume that each pre-population already possesses its desired tuning, and use this fact to realise the post-population tuning. Curiously, the pre- and post-population are the same for recurrent connections, and solving for weights is a matter of minimising a linear least-squares problem.

Although our new formalisms are in some ways equivalent to the NEF dynamics principle, they offer a new perspective when modelling neurobiological networks. Modellers can specify the temporal tuning of a neuron population independently of its spatial tuning down to individual neurons. We use our least-squares optimisation problem to realise this desired tuning. An example of this is the use of bell-shaped temporal encoders depicted in Figure 4.28.

This way, our work eliminates a limitation of the NEF dynamics principle. While it is possible to directly take empirical neurophysiological data into account when selecting spatial tuning properties (i.e., using the representation principle) and solving for spatial connection weights (i.e., using the transformation principle), the dynamics principle typically—with some exceptions—requires modellers to provide closed-form differential equations. Providing such closed-form equations is further complicated when attempting to realise dynamics in networks

with heterogeneous or higher-order synapses (Voelker and Eliasmith, 2018). Our approach suggests a way to better take empirical data into account and to automate the process of solving for weights, even in networks with complex recurrences (cf. Sections 4.1.4 and 4.3.1).

Finally, we presented a method for constructing low-order LTI systems generating temporal bases using an “information erasure” technique. This approach can be used to derive the LTI system underlying the Legendre Delay Network (LDN; Voelker and Eliasmith, 2018), but can similarly be applied to other bases as well. We suggested a “modified Fourier basis”, that outperforms the LDN system in several benchmark tasks, and when mapped onto our spiking neural networks can be used to realise time-cells in biological network models.

Furthermore, the modified Fourier basis could be particularly attractive for stream processing in artificial neural networks, where, due to an efficient Runge-Kutta state update, we effectively require $\mathcal{O}(q)$ space and $\mathcal{O}(q^{2.3})$ time for processing a window of length θ . This is a substantial improvement over the LDN used in the LMU which we found to require $\mathcal{O}(q^3)$ time.

Future work. A central shortcoming of our work is a lack of mathematical stability guarantees. While we argued that our recurrent networks will realise the desired dynamics *if* they can be realised perfectly, it is unclear what happens in cases where this condition is broken. We observe in practice that our networks are typically well-behaved (i.e., not asymptotically unstable); however, it would be interesting to augment our weight solver with techniques from system identification that guarantee stability (cf. Verhaegen et al., 2007).

Another aspect that we have only considered in the context of our adaptive filter experiment, is using the temporal tuning curve paradigm to realise nonlinear tuning. This is possible using the NEF dynamics principle (Eliasmith and Anderson, 2003, Chapter 8) and also not a limitation of our temporal tuning curve approach. In fact, our general definition of a temporal tuning curve is oblivious to the specific type of dynamics (cf. Definition 4.1, Appendix A.3.2). Rather, the problem is finding a good mathematical formalisation; linear temporal encoders ϵ_i offer an intuitive parametrisation of temporal tuning curves that we would have to give.

Taking intrinsic neural dynamics into account is another aspect of the work presented here that requires further research. While linearly approximating neural dynamics was successful for the simple neuron models tested here, it would be interesting to extend this to more complex neuron models, potentially incorporating work by Duggins (2017). It could similarly be interesting to model transmission delays using temporal tuning, and demonstrate that our methods can capture low-level phenomena such as auditory coincidence detection in the brain stem (Kandel et al., 2012, Chapter 31).

In order to disseminate our methods to researchers modelling neurobiological systems, it would be interesting to integrate our approaches into a software tool such as the Nengo spiking neural network simulator (Bekolay, Bergstra, et al., 2014). An interesting engineering challenge would be to define a convenient API for temporal tuning that ties well into the existing system. In this respect, it would also be beneficial to find ways to more efficiently solve the least-squares problem defined in eq. (4.8) by relying on spectral decompositions.

Chapter 5

An Adaptive Filter Model of the Cerebellum

The CNS [Central Nervous System] needs to be understood at four nearly independent levels of description: (1) that at which the nature of a computation is expressed; (2) that at which the algorithms that implement a computation are characterized; (3) that at which an algorithm is committed to particular mechanisms; and (4) that at which the mechanisms are realized in hardware. In general, the nature of a computation is determined by the problem to be solved, the mechanisms that are used depend upon the available hardware, and the particular algorithms chosen depend on the problem and on the available mechanisms.

— DAVID MARR AND TOMASO POGGIO

From Understanding Computation to Understanding Neural Circuitry (1976)

Material in this chapter from prior publications

This chapter is, with permission from the publisher, in edited and extended from adapted from Stöckel, Stewart, and Eliasmith (2021). In turn, this publication is based on work presented at the Cognitive Science Society Conference (“CogSci”; Stöckel, Stewart, and Eliasmith, 2020a) and at the International Conference on Cognitive Modelling (“ICCM”; Stöckel, Stewart, and Eliasmith, 2020b).

Contributions in this chapter

The author was responsible for conducting the experiments presented in this chapter, contributing the underlying theoretical work (cf. previous chapters), writing, creating all figures, and conducting the literature review. Terrence C. Stewart helped with designing the final network, exploring the parameter space, and writing the paper. Chris Eliasmith supervised this work, provided helpful feedback, and edited the manuscript. The novelty of this work lies in demonstrating that we can use the modelling techniques presented above to construct systems that adhere to detailed neurophysiological constraints. Our results suggest that the Granule-Golgi circuit can, under mild assumptions, generate temporal representations. Furthermore, by systematically varying neurophysiological parameters in our simulation, we find that the parameters observed in nature tend to be near-optimal for the high-level function we are trying to implement.

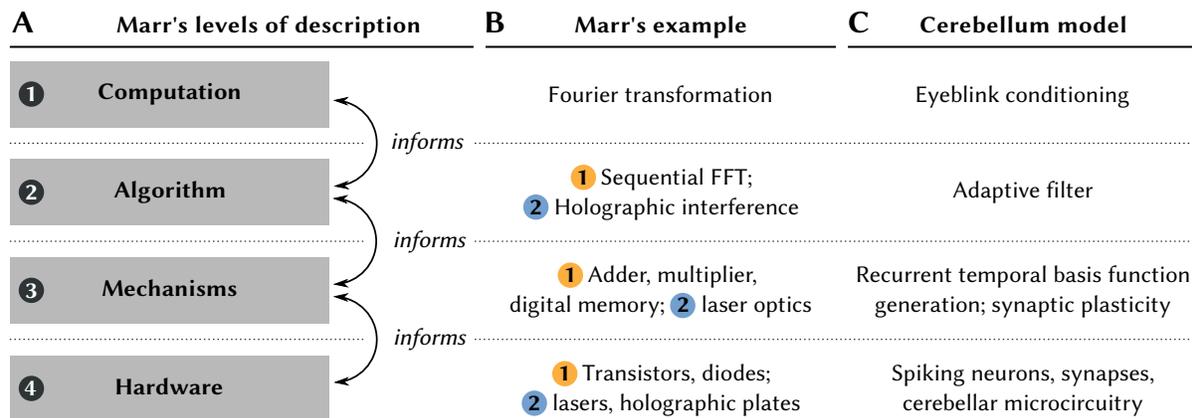


Figure 5.1: Marr and Poggio’s levels of description. **(A, B)** Levels of description and corresponding examples from the original paper (Marr and Poggio, 1976); coloured numbers highlight different realisations of the same computation. **(C)** Levels of description for our model of eyeblink conditioning.

So far, we have focused on mapping abstract mathematical descriptions onto low-level biological mechanisms. For example, we have demonstrated that it is possible to compute a variety of mathematical functions while accounting for biological constraints such as Dale’s principle. Furthermore, we showed that synaptic filters and neural dynamics can be exploited to realise temporal bases. In terms of Marr and Poggio’s famous “levels of description” (quoted on the previous page; cf. Figure 5.1; Marr and Poggio, 1976), we have been concerned with levels three and four, that is, thinking about how algorithms can be “committed” to mechanisms, and how these mechanisms are in turn implemented in a (simulated) neural substrate.

Hence, the primary goal of this chapter is to provide an example of how the methods discussed above can be used to bridge all four levels of abstraction. Specifically, we implement a simple “cognitive” system in the form a model of eyeblink conditioning in the cerebellum—starting from the high-level computation (namely “implementing eyeblink conditioning”) down to the constrained cerebellar microcircuit (cf. Figure 5.1C). Since constructing an intricately detailed model of the entire cerebellum is out of scope for this thesis, we particularly focus on temporal representations in the Granule-Golgi circuit.

Implications for cognitive modelling. Our methodology is remarkably different from the way in which many cognitive scientists and psychologists approach constructing cognitive models. In these disciplines, models tend to be at the computational and algorithmic levels, and seldom take limitations of the neural substrate into account (Eliasmith and Kolbeck, 2015).

Ignoring biological detail can be reasonable depending on the hypothesis that is being explored. Yet, as we discussed in Section 2.3, a closer look at biology may help in two complementary ways. First, we can *validate* hypotheses about cognition by determining whether it is possible to implement a particular algorithm in a constrained biological network. Second, we can *generate* new hypotheses by asking what class of algorithms a network could support.

We believe that cognitive modellers must ultimately embrace a combination of top-down and bottom-up modelling to narrow down the space of possible cognitive science theories and to direct research attention within that space. Central roadblocks to the adaptation of this methodology are the availability of modelling tools such as those discussed in this thesis, disseminating information on how to use these tools in practice, and understanding how low-level detail influences high-level function. Correspondingly, a second goal of this chapter is to study how biological constraints influence NEF networks on a functional level.

Implications for neuroscience. A closed-loop model of eyeblink conditioning in the cerebellum is also interesting from a neuroscience perspective. It is still unclear how exactly the cerebellum manages to learn and reproduce the timings observed in motor-learning tasks such as eyeblink conditioning. While earlier theoretical work suggests that the granule cell activities could span a temporal basis (Medina et al., 2000), this idea has recently come under scrutiny (Johansson et al., 2014). To some degree, this is because recordings of the granule activities are scarce, and the source of the hypothesised temporal tuning is unclear.

Understanding how the cerebellum generates timings has far-reaching implications outside motor learning. Recent evidence—ranging from studies in functional connectivity, neuronal tracing, clinical pathology, to evolutionary physiology—suggests that the same mechanisms are potentially exploited by other cognitive processes as well. That is, the cerebellum may be recruited by various brain regions as a “co-processor” to support brain functions related to higher-order cognition, such as language-based thinking, working memory, perceptual prediction, and tasks requiring precise timings in general (Sullivan, 2010; Buckner, 2013; O’Reilly, Mesulam, and Nobre, 2008; E et al., 2014; Sanger, Yamashita, and Kawato, 2020).

Our experiments suggest that—at least under the constraints we consider—the Granule-Golgi circuit is well-suited to encode temporal information using an approximated sliding-window transformation. The resulting representation can be used to learn delays by modulating synaptic weights in the granular-to-Purkinje projection in accordance with the Marr-Albus hypothesis (Marr, 1969; Albus, 1971; Sanger et al., 2020). These results support earlier simulation work (Rössert, Dean, and Porrill, 2015). We furthermore use our model to generate hypotheses as to why some biological parameters are as observed.

Structure of this chapter. We first review the eyeblink conditioning task and the theorised neurophysiological mechanisms underpinning delay learning. We then discuss five neural network implementations with an increasing number of biological constraints, followed by a series of experiments to explore the impact of biological detail on the high-level function.

Finally, we extend our model to the complete eyeblink conditioning task by incorporating the remaining cerebellar microcircuitry, albeit without incorporating quite as much biological detail as our most detailed circuits considered to this point. We compare the output of this model to empirical data and show that we are able to—at least qualitatively—reproduce the behaviour of experimental animals. While building this model, we developed a software library that implements many of the methods described in previous chapters (cf. Appendix C.2).

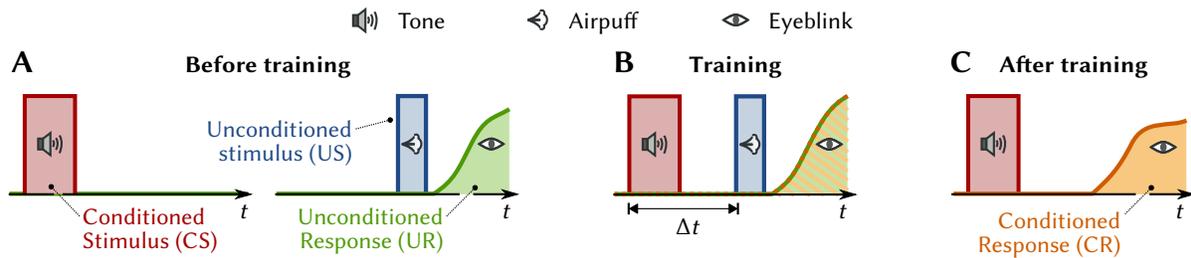


Figure 5.2: Illustration of eyeblink conditioning. **(A)** Before training, the animal shows no reaction to the conditioned stimulus (CS). The animal reacts to the unconditioned stimulus (US), a puff of air into the eye, with an eyeblink. **(B)** During training, both the CS and the US are presented with a fixed delay Δt . **(C)** After training, the animal reacts to the previously neutral conditioned stimulus with an eyeblink. The delay is maintained; the animal begins to close the eye *before* the puff arrives.

5.1 Background

In order to explore the consequences of adding biological details to a neural system, we need to decide—on a computational and algorithmic level—what task the neural system should ideally perform. As we mentioned above, we would like to implement eyeblink conditioning by mapping an LTI system generating a sliding-window transformation (cf. Section 4.2), specifically, the LDN system, onto the recurrent Granule-Golgi circuit. We first provide a review of eyeblink conditioning, followed by an overview of the cerebellar microcircuitry thought to be responsible for this behaviour, and, finally, a discussion of hypothesised mechanisms.

5.1.1 Eyeblink conditioning

Eyeblink conditioning is a form of delay conditioning and a widely studied form of motor learning (e.g., Kandel et al., 2012, Chapter 42, pp. 975-979). The overall experimental setup is illustrated in Figure 5.2. Scientists direct a puff of air, the *unconditioned stimulus* (US), at the eye of an experimental animal. This triggers the eyeblink reflex, the *unconditioned response* (UR). The animal is then repeatedly exposed to the US paired with a neutral, *conditioned stimulus* (CS), for example a short tone. Here, “neutral” refers to the fact that the animal typically shows no strong reaction to this stimulus. The CS precedes the US by a constant time offset Δt .

Over time, the subject forms a *conditioned response* (CR) to the previously neutral CS, maintaining the fixed delay Δt . In other words, the subject will learn to have its eye closed Δt seconds after the tone, even if the puff is absent. In our experiments, we use mouse data from Heiney et al. (2014). However, this experiment in principle works in most mammals, including humans (Cheng et al., 2008). Importantly, the formation of this conditioned response critically depends on the cerebellum. Ablating the cerebellum causes previously learned CRs to be absent (McCormick, Lavond, et al., 1981). This indicates that the cerebellum is not only capable of establishing the connection between the CS and the UR, but also of learning the motor trajectory of the response, including the delay.

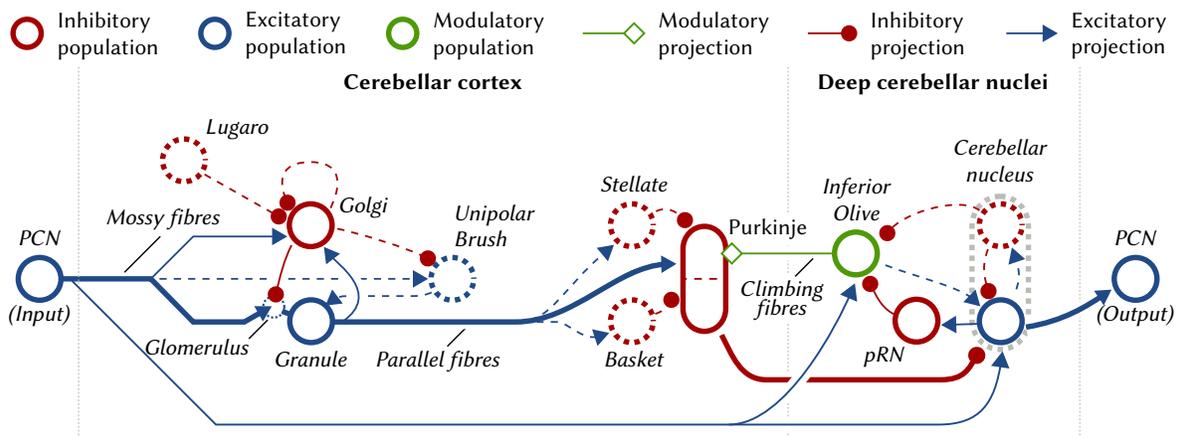


Figure 5.3: Schematic of the cerebellar microcircuit. Dashed projections and populations are not included in our model. Cerebellar nucleus afferents affect both the excitatory and inhibitory sub-populations. The main feed-forward pathway is highlighted in bold. *PCN* = pre-cerebellar nuclei. *pRN* = parvicellular Red Nucleus. Data from Ito (2010) and Llinás (2010).

5.1.2 Cerebellar Microcircuitry

Before we continue to discuss the two prevalent theories on how the cerebellum could learn the conditioned response, it is worthwhile to review the cerebellar microcircuitry depicted in Figure 5.3. Afferent nerves from pre-cerebellar nuclei (PCN; brainstem and cerebral cortex carrying sensory signals) project as “mossy fibres” onto granule cells in the cerebellar cortex. Cerebellar granule cells are tiny and account for the majority of neurons in the mammalian brain. Thus, small numbers of pre-cerebellar neurons connect onto many granule cells.

Granule cells also have interneurons interspersed amongst them, the Golgi cells. These form an inhibitory feedback loop with the granule cells. That is, granule cells excite Golgi cells, and Golgi cells inhibit granule cells (Ito, 2010). Notably, the connections from Golgi and PCN to granule cells are formed through so-called glomeruli. Each granule cell extends to on average four glomeruli and, at each glomerulus, receives input from one pre-cerebellar neuron through a mossy fibre terminal, as well as one or more Golgi cells (Palkovits, Magyar, and Szentágothai, 1972; Jakab and Hámori, 1988; Chadderton, Margrie, and Häusser, 2004). Furthermore, the connectivity between Golgi and granule cells is spatially constrained. Golgi cells only connect to granule cells within a 600 μm diameter (Albus, 1971; D’Angelo et al., 2013). The ratio of granule to Golgi cells is about 400:1 (Korbo et al., 1993).

Granule cell axons, the so called “parallel fibres”, project onto the Purkinje cells, which inhibit neurons in the cerebellar nucleus, and in turn project back onto the brainstem and cerebral cortex (Ito, 2010; Llinás, 2010). So-called “climbing fibres” project from Inferior Olive neurons in the deep cerebellar nucleus onto Purkinje cells. Evidence suggests that activity in the climbing fibres is responsible for modulating the synaptic strength of granule to Purkinje projections. Climbing fibre activity is often interpreted as an “error signal” $\varepsilon(t)$ responsible for driving learning in the cerebellar cortex (Ito, 2010).

5.1.3 *Hypothesised Mechanisms Supporting Delay Learning*

There is no current consensus on what the exact mechanism is that supports delay learning in the cerebellum. Specifically, there are two competing hypotheses: the classic adaptive filter theory and the view that intrinsic properties of the Purkinje are responsible for timing.

Adaptive filter hypothesis. As originally pointed out by Marr (1969) and Albus (1971), the massive divergence (number of post-neurons for a single pre-neuron) and small convergence (number of pre-neurons for a single post-neuron) in the PCN to granule projection suggests that granular cells are tuned to specific patterns of activity in the PCN. Modulating the synaptic weights between the granule and the Purkinje cells using the climbing fibre error signal can be used to recombine the sparse pattern detectors and to decode arbitrary functions.

The adaptive filter hypothesis, originally proposed by Fujita (1982), can be interpreted as extending this idea toward temporal tuning. That is, similarly to what we discussed in the previous chapter, granule cell activity does not only depend on the current PCN activities, but on their past history. Correspondingly, if this temporal tuning is diverse enough, that is, granule cells are sensitive to different time-courses, this will form a suitable temporal basis from which functions over time, such as delays, can be decoded. A possible source for the temporal tuning could be the recurrent granule-to-granule connections that are mediated by the inhibitory Golgi cells. Indeed, Rössert et al. (2015) find that randomly connecting the Golgi and granule cells generates diverse tuning that can be exploited as a temporal basis. Our contribution is to test whether optimal temporal tuning—i.e., an LTI system approximating a sliding-window transformation (cf. Section 4.2)—could be realised in the Granule-Golgi circuit.

Intrinsic properties of the Purkinje cells. A more recent theory is that responses observed in tasks such as eyeblink conditioning inherently rely on intrinsic properties of the Purkinje cells. In other words, the Purkinje cells act as “time cells” (Lusk et al., 2016), and the temporal properties of the granule cells play a lesser role. Instead, climbing fibre input triggers processes within the Purkinje cell and their dendritic structures that are responsible for the formation of a delayed output. Evidence for this comes from Johansson et al. (2014), who find that bypassing the granule cells and directly injecting signals into the parallel fibres still evokes a previously learned delayed response from the cerebellum, albeit a weaker one.

Discussion. We think that these two theories are not inherently contradictory. Both temporal tuning of the granular cells and intrinsic temporal properties of the Purkinje cells could play a role in delay learning—as we discussed in Chapter 4, both the intrinsic dynamics of neurons, and the network dynamics can in principle be harnessed to produce desired dynamics.

The findings in this chapter provide a strong argument that the Granule-Golgi circuit is well-suited for implementing some kind of temporal basis, and as such confirms the results of previous studies (cf. Dean et al., 2010; Rössert et al., 2015). Still, our work only takes a fraction of the available data on cerebellar neurophysiology into account and as such should not be seen as strong evidence for or against either of the two proposed mechanisms.

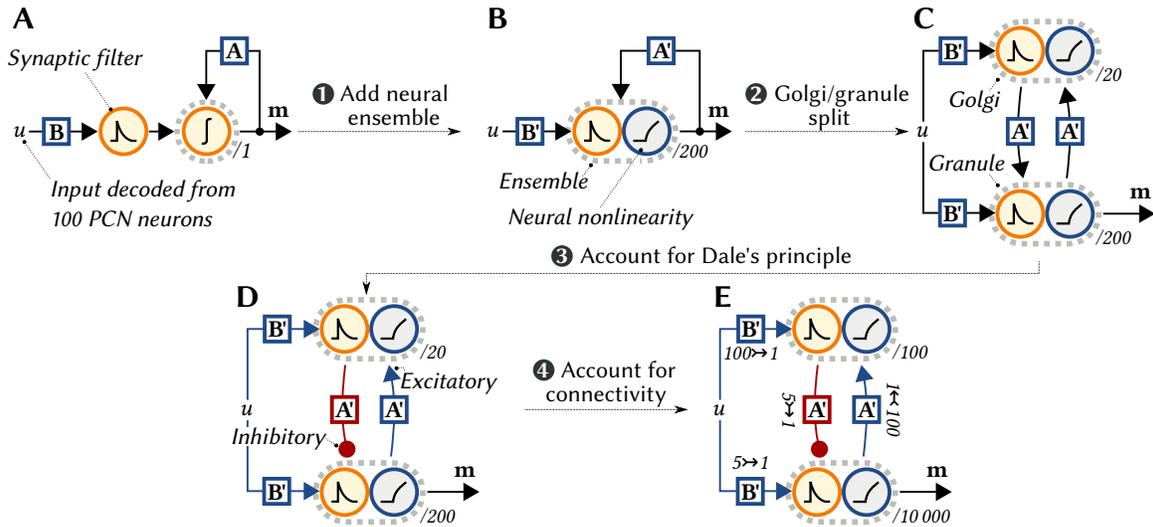


Figure 5.4: Network types used in our experiments. (A) “Direct” implementation with an optimal integrator. (B) Using the synaptic filter of a single population of spiking neurons for temporal integration. (C) Interneuron population in the recurrent path. (D) Same as (C), but accounting for Dale’s principle. (E) Same as (D), but with more detailed biological constraints (see text).

5.2 Modelling the Granule-Golgi Circuit

As we discussed in the introduction, we would like to analyse in how far adding biological detail to a system influences its high-level function. To this end, we focus on one specific part of the cerebellar microcircuit, namely the Granule-Golgi circuit, and test to what degree this circuit is capable of generating a temporal representation at five different levels of biological detail. Below, we first describe these individual circuits, followed by a series of experiments in which we explore the performance of the individual networks.

5.2.1 Levels of Biological Detail

We discuss five models of increasing complexity (cf. Figure 5.4). The first model is merely an abstract implementation of the LDN system, the final model respects spatial connectivity constraints, tuning curves, and, to a degree, neurotransmitter time-constants. In all cases, the input u is received from an NEF ensemble with 100 spiking LIF neurons representing the PCN.

Model A: “Direct” Implementation. For this model, we directly solve the differential equations describing the LDN system (cf. eq. 4.24) by integration. That is, we have a single layer of “neurons” that are pure integrators. Correspondingly, this model focuses on the high-level theory of what the system is computing, and proposes a new computational mechanism for capturing delay learning via the LDN system. We model the PCN and the granule cells as neuron populations, but these are not responsible for generating the dynamics.

Model B: Single Population. In our second stage, we replace the integrators with an ensemble of 200 LIF neurons. As we discussed in Section 2.3.3, these neurons form a distributed representation of the LDN system state \mathbf{m} using a population code (cf. eq. 2.20). Maximum firing rates a_{\max} are between 50 Hz to 100 Hz. We solve for optimal input and recurrent connection weights that result in these desired tuning curves while implementing the equivalent calculation as in *Model A* using the NEF dynamics principle (Section 2.3.5).

Model C: Interneurons. As a next step, we separate the single layer of neurons into two populations corresponding to the Golgi and granule cells, reflecting the actual biology of the cerebellum. This introduces two synaptic filters which need to be taken into account when solving for the connection weights that best approximate the Legendre system; we discussed this in Section 4.1.4. Furthermore, to at least approximate the fact that there are far fewer Golgi cells than granule cells, we use 20 Golgi cells and 200 granule cells.

Model D: Inhibition and Excitation. In the next step, we account for Dale’s principle using the techniques described in Section 3.2.2. We mark the granule cells as purely excitatory, and the Golgi cells as purely inhibitory and decode the bias current from the pre-population. Note that we assume—in contrast to the evidence presented in Figure 2.31—that the granule cells possess intrinsic biases and uniform x -intercepts; we discuss this in more detail below.

Model E: Sparse connectivity and activities. Finally, we constrain the neural connectivity and adjust the PCN tuning. The previous models used all-to-all connections, whereas we now only allow a subset of the connection weights to be non-zero. This applies to both the input to the Granule-Golgi system and to the recurrent connections.

In particular, as is indicated in Figure 5.4E, we account for the granule cell convergence numbers by randomly selecting five PCN and five Golgi cells as pre-neurons—this number is slightly larger than the number reported above, since, as we discuss in more detail in Section 5.2.3, the number of pre-neurons places a strict upper limit on the connectivity. Given this extremely sparse connectivity, we increase the number of neurons in the simulation to 10 000 granule and 100 Golgi cells, which is closer to the ratio observed in nature.

To account for spatially imposed connectivity constraints, we assign a location \mathbf{x} in $[-1, 1]^2$ to each neuron. This represents a location on a small patch of the folded two-dimensional surface of the cerebellar cortex. As is depicted in Figure 5.5, the probability p_{ij} of a post-neuron i to receive a connection from a pre-neuron j is proportional to $\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/\sigma^2)$.¹

The input representation in the PCN was made more temporally sparse by adjusting the x -intercepts (Section 2.3.3). Data reported by Chadderton et al. (2004) indicate that granule cell excitatory event rates—which are driven by PCN activity—drop from an average of 40 s^{-1} with a stimulus being present to only 8.5 s^{-1} when there is no stimulus. We adjusted the PCN tuning curves to match these statistics in the final network (cf. Figure 5.6).

¹See Appendix C.2.3 for an overview of how these sparsity constraints can be specified in NengoBio.

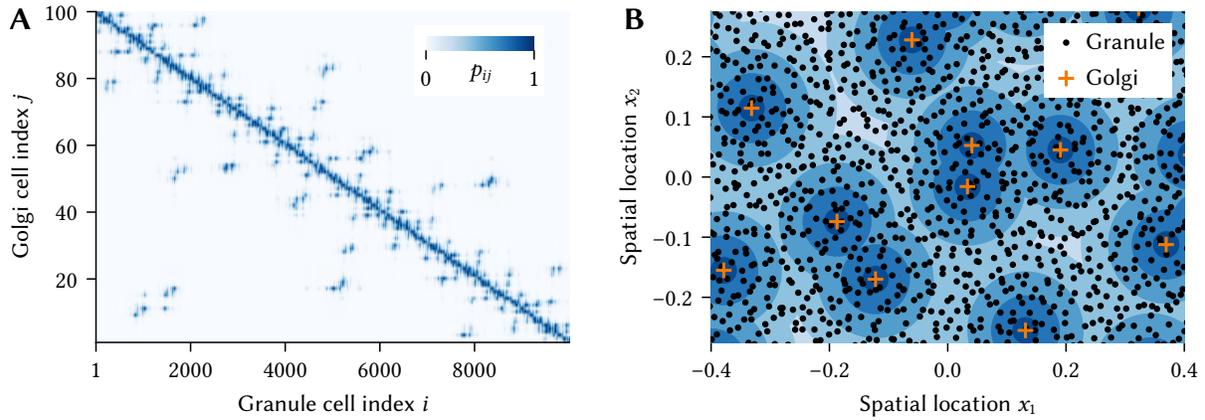


Figure 5.5: Spatial connectivity constraints between the Golgi and granule cells. **(A)** Normalised connection probabilities p_{ij} for $\sigma = 0.25$. Note that the neuron locations are sampled from a Hilbert curve with additive random jitter. Correspondingly, neurons with similar (relative) indices tend to be closer together, resulting in the clearly visible diagonal. **(B)** Spatial organisation of the Golgi and granule cells. The background depicts the cumulative density of the Golgi to granule connection probability for a virtual Granule cell at each location; same colours as in (A).

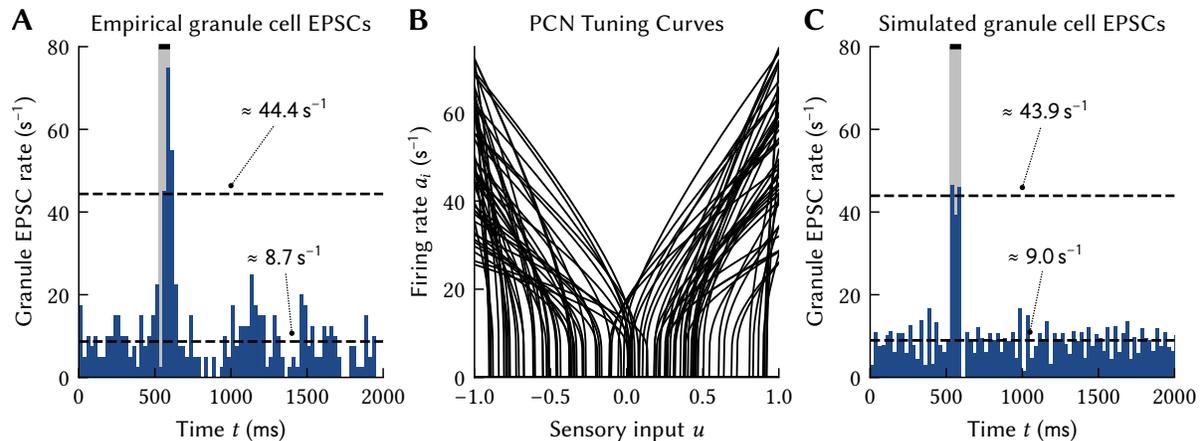


Figure 5.6: Granule cell EPSC distributions and PCN tuning curves. **(A)** Empirical granule cell EPSC rates (cf. Chadderton, Margrie, and Häusser, 2004, Figure 3C; rescaled from counts per bin to rates). Data are the mean over 16 trials. Black bar and shaded background correspond to stimulation of the animal's whiskers. Upper dashed line corresponds to the average spike rate with input (including the first bin after the input), lower dashed line to the average spike rate without input. **(B, C)** In our model, PCN input via the mossy fibres is the only excitatory input to the granule cells. Setting the maximum firing rates to be uniformly distributed in $[25, 75]$ and the x -intercepts to be uniformly sampled from $(-0.15, 0.95)$ qualitatively results in a similar EPSC distribution. Data are the mean over 10 000 cells.

5.2.2 Impact of Biological Detail on Temporal Representations

To explore the effect of adding biological detail to the Granule-Golgi circuit, we first take a look at the temporal representations generated by these systems. We then analyse in how far we are able to decode delays from these representations, as is required for delay conditioning.

Temporal Representations. Figure 5.7 depicts the response of the different Granule-Golgi circuits for a rectangle input. When decoding $\hat{\mathbf{m}}$ using the identity decoder (eq. 2.21), we find that adding detail increases the amount of noise and decreases the similarity with the “direct” implementation. The decoding for the most detailed network, *Model E*, is less noisy due to the large number of granule cells, but does not resemble the direct implementation.

However, the fact that $\hat{\mathbf{m}}$ is not close to the desired \mathbf{m} does not imply that the granule cell *activities* do not form a useful temporal representation. A singular value decomposition of the filtered granule cell activities reveals that the realistic network establishes a temporal representation that resembles the Legendre basis and—judging from the sum of the singular values Σ —is almost as expressive as that of the unconstrained two-population model. A control experiment (Figure 5.7F) with random stable feedback matrices \mathbf{A} demonstrates that solving for the LDN system is significantly better than not.

Interestingly, deactivating intrinsic biases in the granule cells (Figure 5.7G) leads to a substantial decrease in the quality of the generated representation. This implies that if the granule cells indeed form a temporal representation in nature, then granule cell tuning must be diversified using a mechanism that is not captured by our model.

Decoding delays. To test in how far the temporal representations generated by the Granule-Golgi network can actually be used to decode delayed versions of an input signal, we systematically generate two different types of input $u(t)$: periodic pulses of varying width t_{on} and band-limited white noise of bandwidth B . We then determine how accurately the past history $u(t - \theta')$ over the window $\theta = 0.4$ s can be recovered from the granule activity via optimal linear readout weights. We simulate the network for $T = 10$ s.

Figure 5.8 depicts the average reconstruction error for varying inputs (horizontal axis) and delays (vertical axis) over ten trials. An example run of *Model E* (the model with the most biological detail) is shown in Figure 5.9. Overall, the network successfully functions as a method for encoding the input signal over the desired window of time θ . As predicted by the previous experiment, adding biological detail generally decreases the accuracy of the system, although *Model E* outperforms *Model D*. Note that there is a peak in accuracy when decoding data from $\theta' = 60$ ms in the past ($\theta'/\theta = 0.15$). This corresponds to the neurotransmitter time-constant $\tau = 60$ ms we use for all connections in the Granule-Golgi circuit, and which is based on a first-order low-pass fit to the NMDA Granule-Golgi dynamics reported in Dieudonné (1998).²

²Note that the reported time-constants of other connections in the Granule-Golgi circuit are significantly shorter (Kanichay and Silver, 2008). While we could use the techniques from the previous chapter to compute the connections weights for these heterogeneous τ , we assume homogeneous τ for the sake of simplicity.

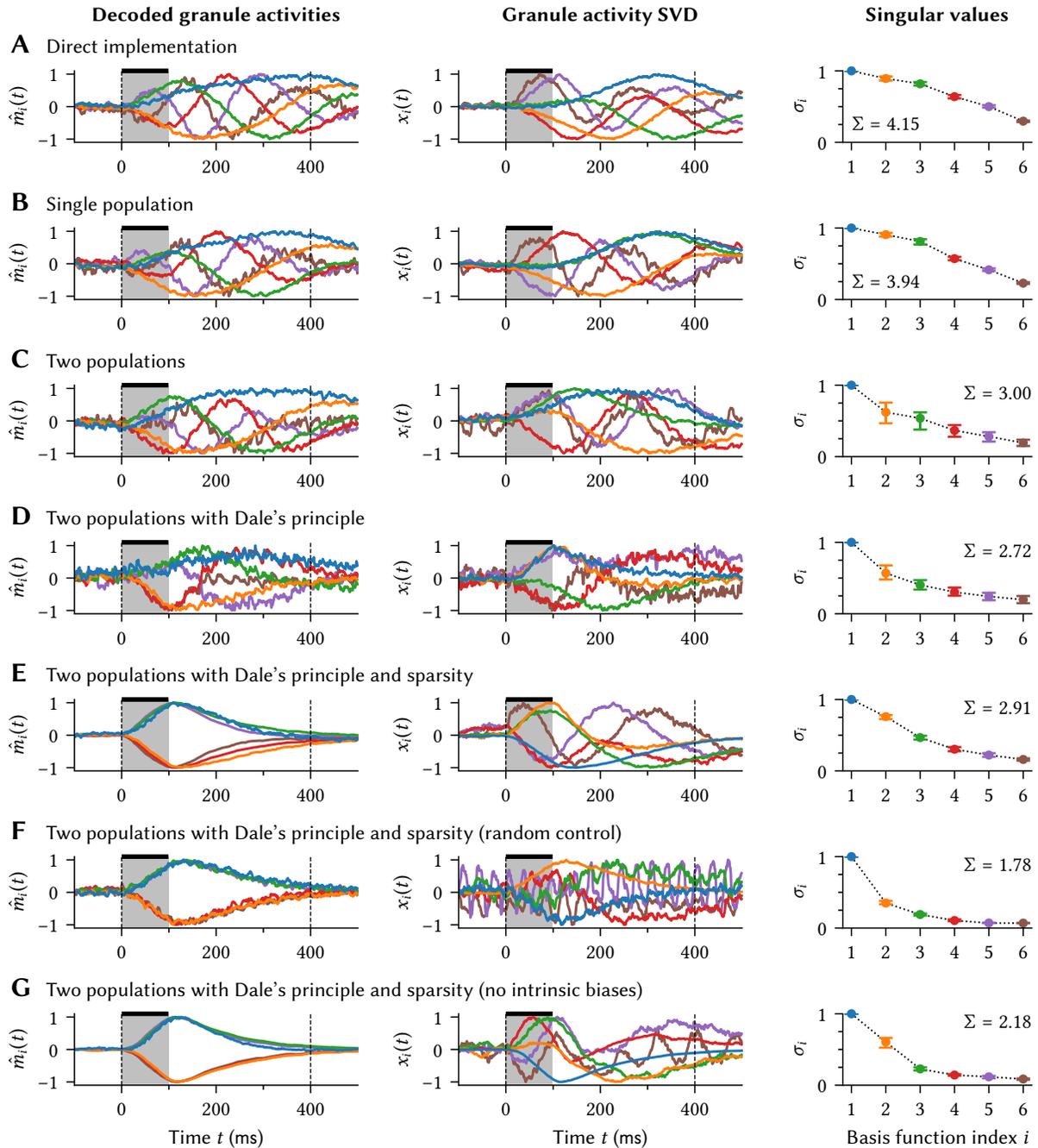


Figure 5.7: Temporal response of the Granule-Golgi networks for a rectangle input (black line). All responses are centred and normalised. *Left:* Example identity decoding of the granule activities. *Centre and right:* Example singular value decomposition (SVD) of the granule cell activities and the median normalised singular values over 100 trials. Error-bars indicate the 25th and 75th percentile. **(A-E)** Data for the individual models discussed in this section. **(F)** Network with a stable random feedback matrix A . **(G)** Disabling intrinsic biases substantially reduces the performance of the network.

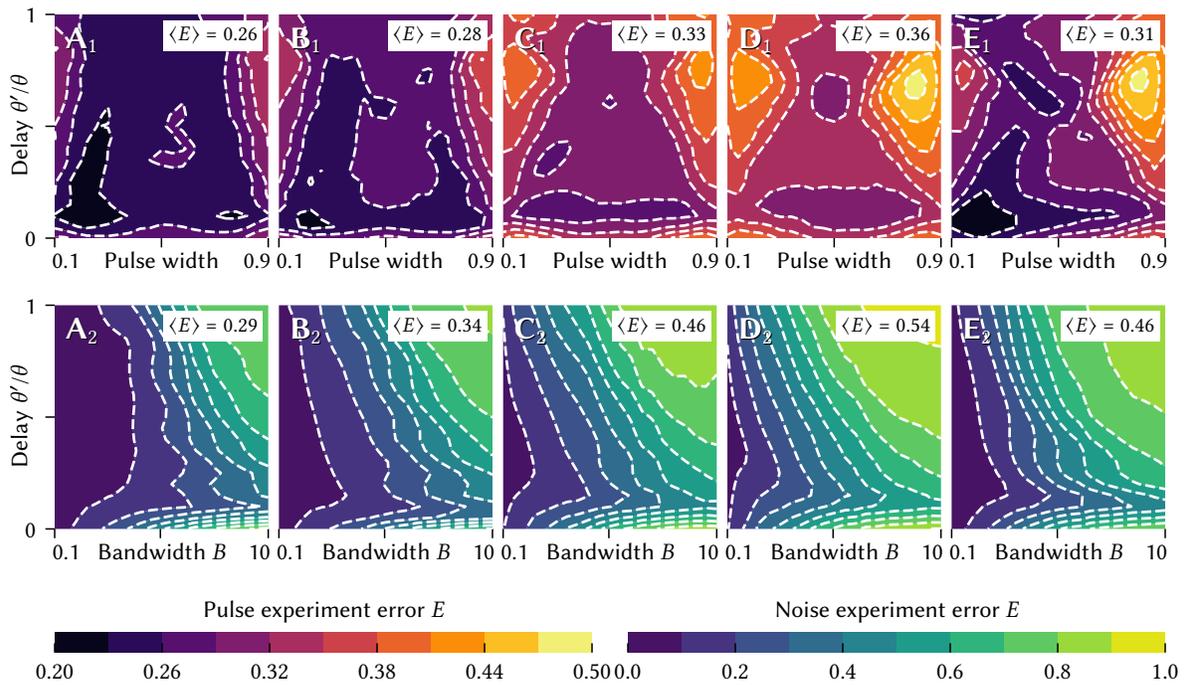


Figure 5.8: Delayed signal reconstruction error for different types of input signals, delays, and network types. All error values are expressed as RMSE divided by the mean RMS of the input signal over 100 trials. Columns correspond to the network types in Figure 5.4. *Top row:* Reconstruction error for rectangle pulse signals of varying width. *Bottom row:* Reconstruction error for a band-limited white noise input signal with varying band-limit.

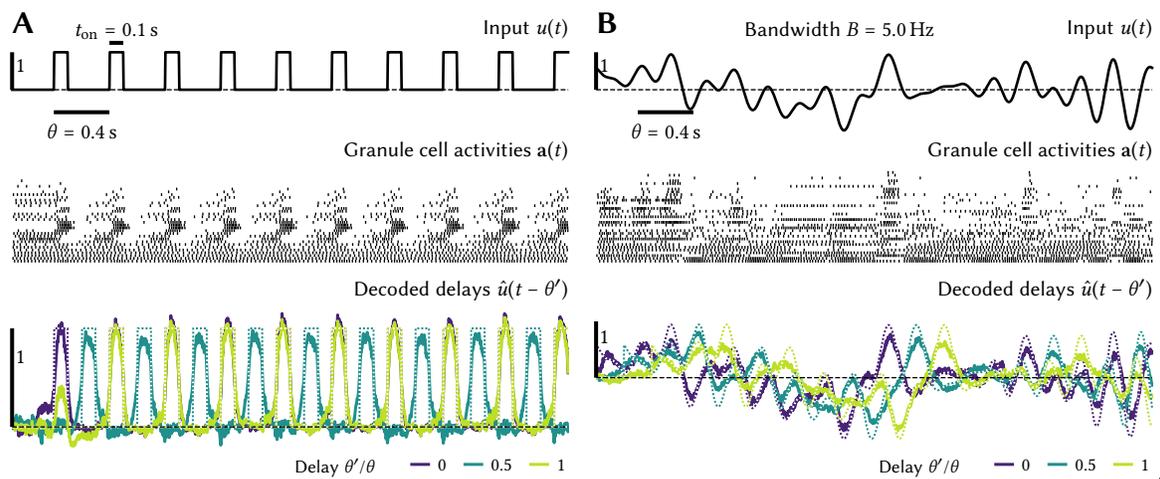


Figure 5.9: Examples showing the delayed input signals decoded from the granule layer in the detailed model (Figure 5.4E). *Top row:* Input signal (rectangle pulses in **A**, white noise in **B**). *Middle row:* Spike raster for 40 randomly selected granule cells. *Bottom row:* Delays decoded from one thousand randomly selected granule cells. Dotted lines correspond to an optimal delay.

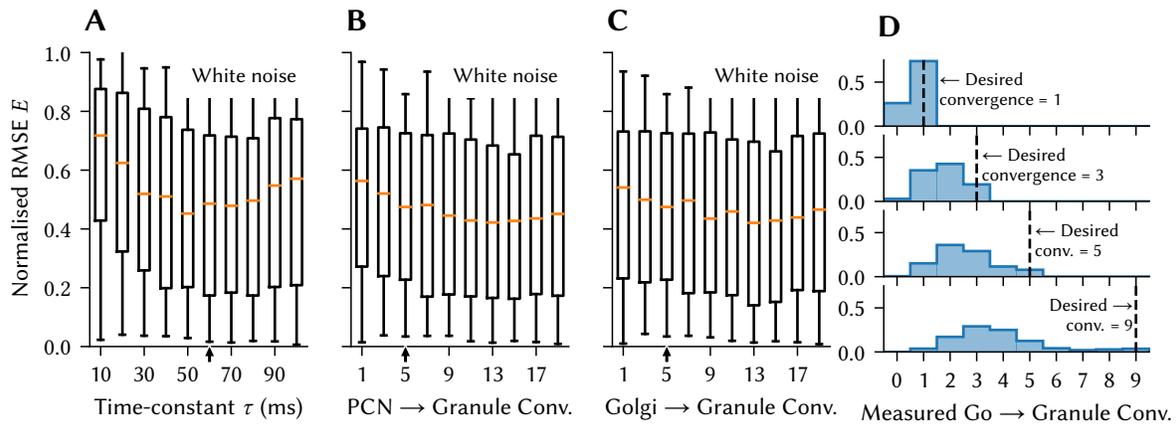


Figure 5.10: Parameter exploration. **(A-C)** Effects of varying parameters on the delay reconstruction error (arrows indicate default parameters). The box plots show the median, lower and upper quartile of all the data depicted in a single plot in Figure 5.8 ($n = 441$ data points). **(D)** Histograms showing the frequency of measured PCN to granule convergences compared to the desired convergences.

5.2.3 Parameter Exploration

Notably, we can use our model to determine what the accuracy would be like if we changed individual parameters, such as the aforementioned synaptic time-constant. This is shown in Figure 5.10. Interestingly, the performance of the system is best for a filter time-constant of 50 ms, which is close to the value we used based on measured Granule-Golgi dynamics.

As discussed above, a striking feature of the cerebellar microcircuitry is the low granule cell convergence. One possible hypothesis is that these numbers are a trade-off between minimising connectivity and the overall performance of the resulting system. In our model, we can test this hypothesis by systematically varying the number of pre-neurons the solver has access to. Results are shown in Figures 5.10B and 5.10C.

For the PCN to Granule convergence (Figure 5.10D), the performance of the system does improve with larger limits, yet plateaus at still relatively small convergence numbers. Importantly, as mentioned above, the desired convergence solely controls the number of *potential* pre-neurons. Since the weight solver may still set a weight to zero, these convergence numbers are upper limits. Measuring the actual convergence in the PCN to granule connections (Figure 5.10D), we see a peak at one to three PCN cells connecting to each granule cell, and this peak is only weakly affected by the desired convergence. Importantly, in nature, PCN cells connect to on average four granule cells (see Palkovits et al., 1972, for the complete statistics), whereas we only measure an average convergence of two in our model (for a desired convergence of five). We can match the observed average in our model by increasing the desired convergence to 13. This slightly increases the performance of the system and coincides with the optimal desired convergence (cf. Figure 5.10B). Nevertheless, since the gain in performance is relatively small, we decided not to alter the desired convergence in our model setup.

Figure 5.10C indicates that the performance of the system improves up to a desired Golgi to granule convergence number of 13, with a measured average convergence of 4.8 in the final network. This is a little higher than what is commonly assumed in the neuroscience literature, although, notably, there is some uncertainty regarding this number. The original study by Jakab et al. (1988) on the physiology of individual glomeruli (the sites receiving Golgi cell axons and granule cell dendrites; cf. Figure 5.3) counts up to 145 Golgi axon synapses in one glomerulus. However, it is unclear whether these synapses originate from a single or different Golgi cells. Most researchers assume at most one Golgi cell connecting to each glomerulus, and, as explained above, each granule cell in turn connecting to on average of four glomeruli (Palkovits et al., 1972). Our model predicts that if the Granule-Golgi circuit were to generate a temporal representation, then multiple Golgi cells sometimes connect to the same glomerulus.

5.3 Extension Toward a Model of Eyeblink Conditioning

Given the above model for the Golgi and granule cells, we can now introduce learning into the model to test the behaviour of the system in an eyeblink conditioning task. Since we are mostly interested in whether it is possible at all to decode functions online from the Granule cell activities, we do not focus on biological detail for the added components beyond the standard NEF techniques discussed in Section 2.3. For example, we do not account for Dale's principle for the added neuron populations and use relatively few spiking neurons at high maximum firing rates (200 s^{-1} to 400 s^{-1}) to represent individual ensembles.

Network description and learning. Our final network architecture is depicted in Figure 5.11. Consistent with the adaptive-filter hypothesis of cerebellar function and as discussed in Section 5.1.2, we assume that the error signal $\varepsilon(t)$ originates from the inferior olive (IO). This signal is the difference between the conditioned response (what the model has learned so far), and the unconditioned response (the motor response produced by the innate eyeblink reflex). We model the eyelid as a first-order dynamical system that receives a velocity command. The reflexive velocity command has been fitted to the UR data in Heiney et al. (2014).

To adjust the connection weights between the granule cells and the Purkinje cells, we use the Prescribed Error Sensitivity (PES) rule defined by MacNeil et al. (2011), a biologically plausible variant of the classic delta learning rule. Let $w_{ij}^{\text{Gr} \rightarrow \text{Pu}}$ be the connection weight between the j th granule cell and the i th Purkinje cell, η a learning rate parameter, $a_j^{\text{Gr}}(t)$ the j th post-synaptic granule cell activity filtered by a low-pass filter with time-constant τ_{learn} , and $e_i^{\text{Pu}} \in \{-1, 1\}$ the encoder of the i th Purkinje cell. The PES learning rule is given as

$$\frac{\partial}{\partial t} w_{ij}^{\text{Gr} \rightarrow \text{Pu}}(t) = -\eta \varepsilon(t) e_i^{\text{Pu}} a_j^{\text{Gr}}(t) = -\eta \sum_k w_{ik}^{\text{IO} \rightarrow \text{Pu}} a_k^{\text{IO}}(t) a_j^{\text{Gr}}(t),$$

where $a_k^{\text{IO}}(t)$ is the activity of the k th IO cell, and $w_{ik}^{\text{IO} \rightarrow \text{Pu}}$ are the synaptic weights between the k th IO cell and the i th Purkinje cell. These weights are the product of the Purkinje cell encoder e_i^{Pu} and a decoding vector \mathbf{d}_k^{IO} that linearly decodes the error signal $\varepsilon(t)$ from IO cell activity.

5.3. Extension Toward a Model of Eyeblink Conditioning

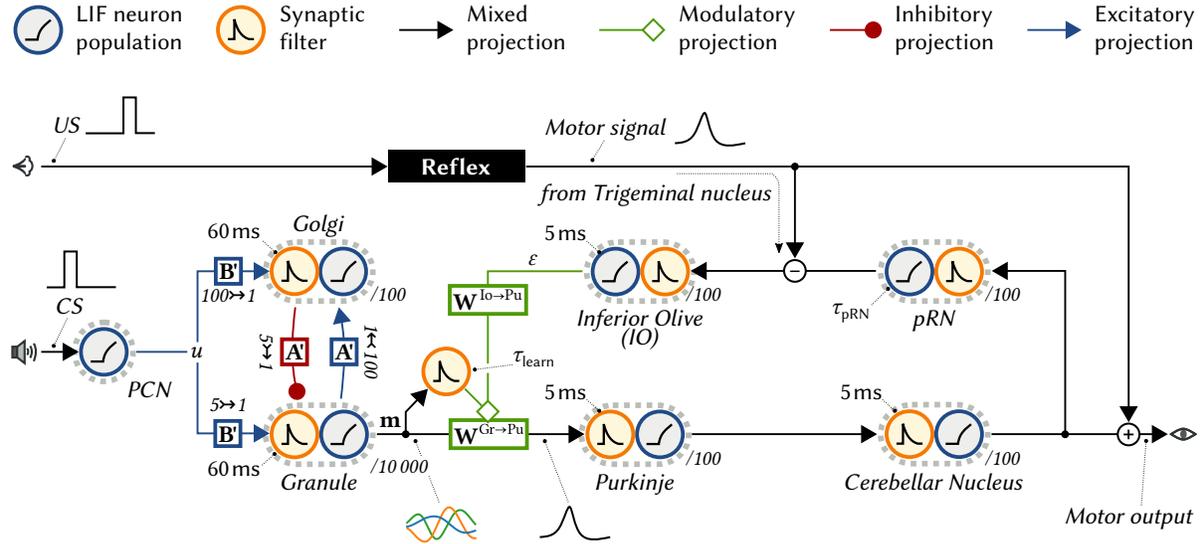


Figure 5.11: Overview of the final eyeblink conditioning network. Note that our model of the Purkinje cells and the deep cerebellar nuclei is less biologically detailed than the Granule-Golgi circuit.

Most parameters were set based on biological data; the synaptic time-constant $\tau = 5$ ms except in the Granule-Golgi microcircuit, as described above. The only free parameters are the learning rate $\eta = 180 \times 10^{-6}$, $\tau_{\text{pRN}} = 100$ ms for the connections involving the pRN, and $\tau_{\text{learn}} = 60$ ms for filtering the granule cell activity in the learning rule. The learning rate was adjusted to match the number of trials typically needed for mice to learn the task (≈ 300 trials). Velocity commands smaller than $v_{\text{th}} = 2 \text{ mm s}^{-1}$ are counted as zero.

Results. Figure 5.12 shows the behaviour of a typical run of the detailed version of our model performing the eyeblink conditioning task over 500 trials. The “tone” (CS) is modelled as a rectangle pulse with $t_{\text{on}} = 100$ ms. The “puff” (US) occurs 250 ms after the CS onset. The model learns to have the eye closed when the puff occurs. Notably, individual instances of the network show slight differences in learning speed, just as individual experimental animals do.

While our model reproduces key features of eyeblink conditioning, its behaviour differs from the experimental data in some aspects. Foremost, our eyelid closure trajectories are quite linear, whereas the animal data shows pronounced curvature. This could point at a deficiency of our motor system model. Furthermore, our model shows far less inter-trial variance compared to experimental animals. We think that the reasons for this are twofold. First, the 10 000 granule cells used in our experiments provide an on average very stable temporal basis from which we can decode the motor control signal. Second, we do not model external systems that might interfere with the cerebellar motor commands, such as signals originating from motor cortex, or the physics of the eyelid itself. Since these processes may be a significant source of inter-trial variance, it is not unsurprising that our model produces a relatively noise-free output. Still, more research in these areas will be required in the future.

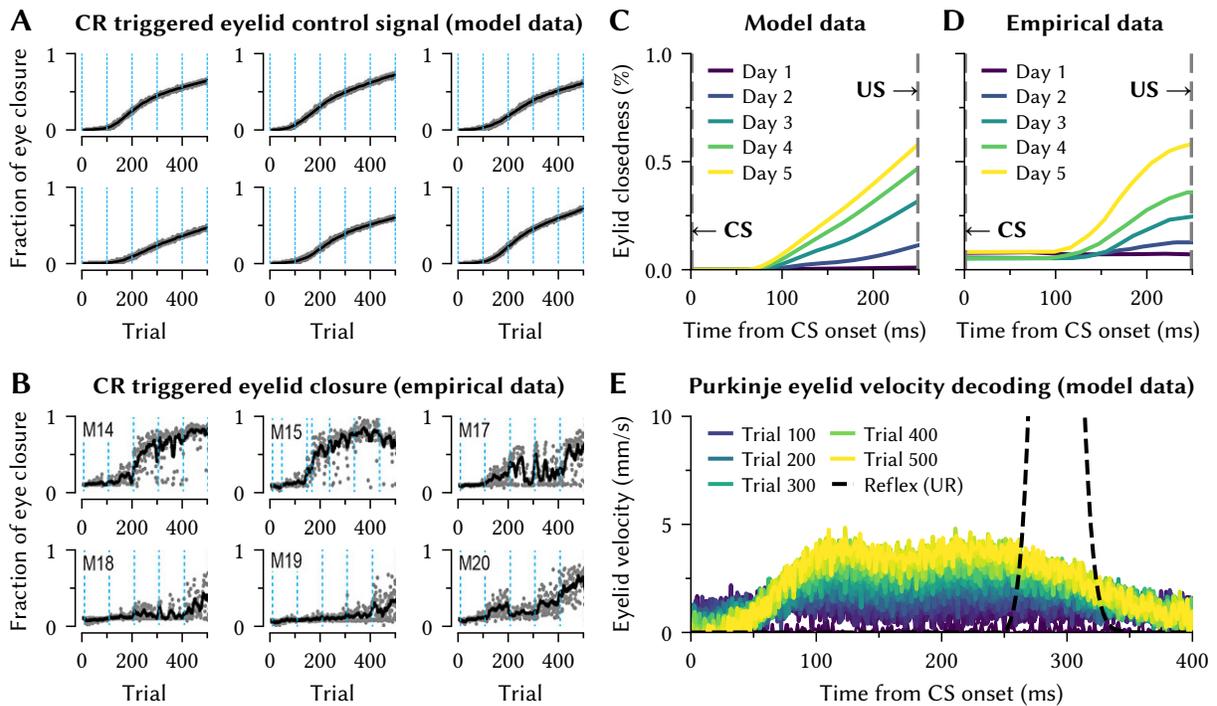


Figure 5.12: Model and experimental data for the eyeblink conditioning task. **(A, B)** Maximum CR triggered eyelid closure over 500 trials for six random instances of the model/experimental animals. Gray dots correspond to the eyelid closure at the time of the US. Black line is the moving average over 11 trials. Blue dotted lines correspond to an experimental day, that is, 100 trials in simulation, and up to 100 trials in the animal experiment. **(C, D)** Eyelid closure trajectory averaged over one experimental day and all six models/animals. **(E)** Eyelid velocity signal decoded at the Purkinje cells compared to the reflex-triggered velocity signal. Data for **(B, D)** adapted from Heiney et al. (2014).

Discussion of the synaptic filter τ_{learn} . The low-pass filter τ_{learn} on the learning connection can be interpreted as summarising temporal filtering corresponding to the mechanisms driving synaptic plasticity (e.g., Kandel et al., 2012, Chapter 66). Functionally, τ_{learn} is required to reproduce the observation that the animal will close the eye *before* the puff occurs as learning progresses. Without the filter, the system learns to close the eye soon *after* the puff; our system tries to exactly re-create the motor pattern produced by the unconditioned reflex—which closes the eye *after* the puff happens.

However, by slowing the passage of information from the Cerebellar Nucleus back to the IO, we are effectively comparing the reflex at one point in time to the generated output from the cerebellum at an *earlier* point in time. This allows the new learned reflex to occur slightly earlier than the unconditioned response, and thus the eye closes before the air puff. We note that this is a situation where adding biological detail (the synaptic filtering) improves the performance of the model. If the model were to learn to *exactly* reproduce the UR given the CS, then the eyeblink would occur *after* the puff of air, which would be a less useful response.

5.4 Discussion

We mapped a high-level, mathematical function onto a brain microcircuit while incorporating biological constraints. Although we were not able to precisely fit the desired Legendre system onto the most detailed Granule-Golgi circuit, the resulting system implicitly produces a good temporal representation, and the quality of this representation critically depends on trying to implement the Legendre system.

The key difference of our approach to existing models of the Granule-Golgi circuit (such as Rössert et al., 2015) is that our modelling techniques are more general with respect to the high-level function that is being mapped onto the underlying circuit. Instead of relying on random connectivity, we specify the high-level function we would like the system to perform.

Predictions. We demonstrated that measurements from our model can be used to generate hypotheses about the kind of electrophysiological data we would expect to find, if this function was indeed realised in the brain. Having access to low-level biological parameters *in silico* furthermore facilitates the exploration of physiological changes that are difficult to achieve experimentally *in vivo*. As discussed above with respect to the synaptic time-constants and convergence numbers, this allows us to investigate why certain parameters are as observed. Furthermore, directly tying the time-constant to the behavioural performance suggests what kind of behavioural effects would be observed if we could shorten the time-constant *in vivo*.

Importantly, our results indicate that the Granule-Golgi circuit could in principle implement a temporal basis function representation. However, as we discussed above, this is under the condition that the Granule cell response curves are diversified by some mechanism not directly captured by our model. Without diverse granule tuning curves the expressiveness of the generated temporal representation is reduced, although not fully unusable. We furthermore predicted that the Granule-Golgi circuit would be better suited for temporal basis function generation if more than one Golgi cell would sometimes connect to a glomerulus.

Future work. While our model of eyeblink conditioning is concerned with a relatively low-level task, the techniques presented here for mapping function onto brain microcircuits are applicable to models of higher-level cognitive function as well, beyond what was already possible with the NEF and Nengo. In particular, it would be interesting to see whether our model of the Granule-Golgi circuit in conjunction with the Purkinje cell's plasticity could serve as a supervised learner for timings in cognitive and perceptual tasks, as suggested by various studies (O'Reilly et al., 2008; E et al., 2014; Sanger et al., 2020).

Future work should focus on incorporating additional biological detail into the model (such as separate biological time-constants for all synapses); specifically, it would be interesting to extend the detail applied to the Granule-Golgi circuit to the remaining portions of the cerebellar microcircuit. Furthermore, it would be interesting to find potential mechanisms for the diversification of the Granule cell tuning curves and to gain a better understanding of why the Legendre system cannot be mapped exactly onto the detailed Granule-Golgi circuit.

Chapter 6

Conclusion

We are all agreed that your theory is crazy. The question that divides us is whether it is crazy enough to have a chance of being correct.

— NILS BOHR
(1958)

Given the sheer complexity of the central nervous system, learning about neuroscience is more often than not a lesson in humility. As a modeller, it seems as if the challenge in describing biological systems lies in deciding what *not* to model; that is, to judge when it is important to choose a detailed description, and when to resort to an abstraction.

The mathematical models presented in this thesis lean towards favouring simplicity over sophistication. This is intentional, but, as we discuss below, also leaves room for future work. For example, we opened this thesis by drawing the reader’s attention to the importance of dynamics in nervous systems. Yet, we barely touched upon some topics typically associated with neural dynamics, such as the oscillatory behaviour classically observed in EEG signals (Lopes da Silva and Gonçalves, 2009; Gerstner and Kistler, 2002, Chapter 8), or complex neuron models with elaborate dynamics and active dendritic trees (Sections 2.1.3 and 2.2.2; Izhikevich, 2007; London et al., 2005).

However, as we discussed in Section 2.3.6, it is important to first consider *why* some biological detail should be integrated into a neurobiological system model. For example, it is conceivable that neural oscillations are the outcome of network level-effects and can be modelled using the NEF as is. Similarly, in the context of more complex neuron models, it is important to ask in how far the dynamics produced by those models *in vitro* or *in silico* are relevant for the dynamics observed in a functioning network (*in situ*).

To illustrate how difficult it can be to decide whether some phenomenon is best modelled on a neural or a network level, consider bursts, that is, multiple action potentials produced by a neuron in quick succession. While burst production is generally ascribed to individual neurons (Figures 2.17C and 2.17D; Kandel et al., 2012, Chapter 2), we can readily observe bursting neurons in NEF networks with laterally connected LIF neurons, assuming that these neurons are tuned to oscillator dynamics (cf. Figure 6.1).

Futhermore, even in cases where bursts have a functional relevance beyond acting as a more robust code—for example the “complex spikes” linked to synaptic plasticity in Purkinje cells in the cerebellum (Kandel et al., 2012, Chapter 42; Richards and Lillicrap, 2019)—it is

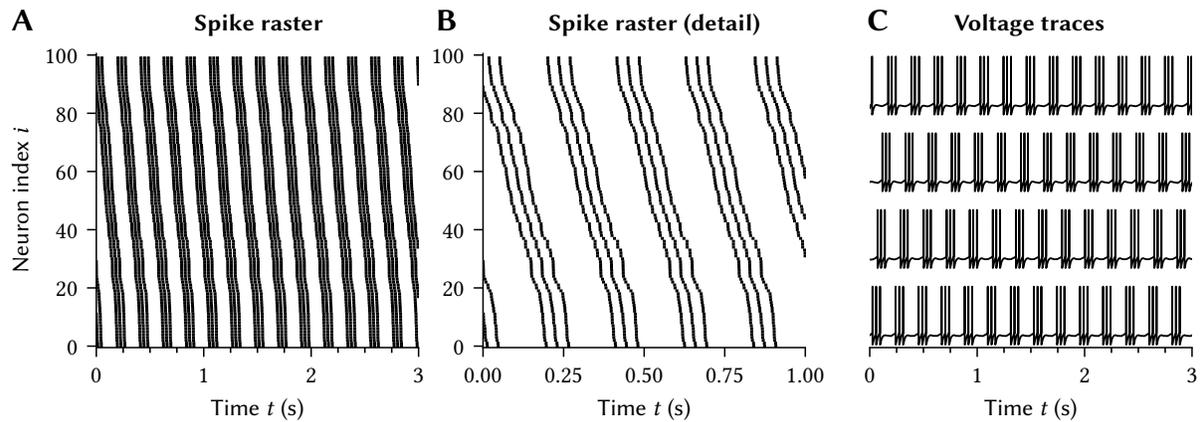


Figure 6.1: NEF networks with oscillator dynamics produce tonically bursting neurons. **(A, B)** Spike raster of a recurrently connected NEF population with 100 neurons implementing a 3.5 Hz oscillator. Neurons are sorted by the angle of their two-dimensional temporal tuning vector e^t . **(C)** Note how the voltage traces of individual neurons resemble tonic bursting (cf. Figure 2.17C). Inspired by Voelker (2019, Figure 3.2, p. 48).

uncertain in how far this detail is important for describing the overall behaviour produced by a brain network. Ultimately, the answer to this question depends on the purpose of the model. A model focusing on Purkinje cell plasticity should of course consider climbing fibre complex spikes, whereas an abstract learning rule may be sufficient in a more phenomenological model of eyeblink conditioning, such as the one discussed in Chapter 5.

This is not to discourage the reader from integrating more complex neuron models into NEF networks. To the contrary, we believe that our temporal tuning paradigm lays the groundwork for accomplishing this, although more research is required to move beyond the simple ALIF example discussed in Section 4.3.5.

Proposed modelling projects. All this being said, the usefulness of the NEF as a modelling technique—including the extensions presented in this thesis—arguably depends on the number of low- and high-level phenomena that can be successfully described. To fathom the limitations of our techniques, we suggest the following modelling projects that could benefit from our work, and that could uncover potential issues. Constructing each of these models is a small research project on its own (similar to Chapter 5), and, while tempting, was out of scope for this thesis.

1. *A model of auditory processing.* A classic example for the relevance of axonal transmission delays in sensory processing are the delay lines involved in extracting interaural time difference in the superior olivary nuclei (Kandel et al., 2012, Chapter 31). By employing our temporal tuning paradigm, we can model neurons that are tuned to delayed versions of the signals originating from the cochlear nerves. These delays can be realised by appropriately choosing filters h_{ij} in eq. (4.8).

Since the neurophysiological properties of neurons involved in this task are well known, including the types of neurotransmitters, this model also benefits from our extensions presented in Section 3.2 regarding Dale’s principle. It would be interesting to see in how far—given a spiking cochlea model (e.g., Zilany and Bruce, 2006)—the generated temporal representations form a good basis for further spatiotemporal processing in auditory cortex. Previous modelling work in this area includes Bekolay (2016).

2. *A model of a patch of early visual cortex.* Our concept of linear temporal encoders $\epsilon_i(t)$ is based on a model of spatiotemporal tuning in layer one of visual cortex (Carandini et al., 1999). It would be interesting to model a small patch of visual cortex using this approach, for example using space-time Gabor functions (Figure 4.4). Previous modelling work in this area includes Hurzook (2012).

A particular challenge for constructing such a model is the high dimensionality of the spatiotemporal representation; this mandates a large neuron population and dense weight matrices. Correspondingly, such a system could benefit from acceleration on neuromorphic hardware. An interesting extension would be to use our two-compartment LIF neurons (Section 3.4) to include an attention mechanism similar to Bobier et al. (2014).

3. *A model of mechanosensory processing.* Pirschel and Kretzberg (2016) observe that mechanoreceptors in the leech (and similarly in other organisms including primates), encode both the stimulus intensity and location of a touch by multiplexing an instantaneous rate code (encoding the intensity; Figure 2.28) and a time-to-first-spike code (encoding the location; cf. Thorpe, Delorme, and Van Rullen, 2001). It would be interesting to see whether our linear temporal encoders are sufficient to produce such a code, and whether our weight optimisation method can solve for linear decoders separating the two quantities. Again, since the leech nervous system is comparably well understood, a model of these phenomena will also benefit from our extensions concerning Dale’s principle.

Simulator software and neuromorphic hardware. As mentioned in Section 4.5, a major remaining task is a better integration of our work into *NengoBio* (cf. Appendix C.2). So far *NengoBio* only supports two-compartment LIF neurons (instead of general n -LIF neurons; Section 3.3), and does not implement temporal tuning; the experiments regarding temporal tuning performed in this thesis were solely implemented using prototype software.

While, as mentioned in Section 4.5, deciding on a user-friendly API for temporal tuning is one challenge, another difficulty lies in the run-time costs associated with simulating networks using our extensions. This includes subthresholded relaxation (Section 3.2.3), user-defined diverse temporal tuning (cf., Figure 4.28), and, in particular, heterogeneous synaptic filters (Section 4.3.1). Remember that typical NEF networks can be simulated efficiently due to the factorisation of the weight matrix into encoder-decoder pairs (Section 2.3.4; Bekolay, Bergstra, et al., 2014) and the ability to collapse the synaptic filter matrix h_{ij} into a single filter (cf. Figure 2.30). This is no longer possible when using the aforementioned extensions.

In the light of exploring our techniques further, it would therefore be interesting to map such networks onto neuromorphic neural network accelerators that are not limited to factorisable weight matrices, and that offer more flexibility regarding diverse synaptic time-constants. Examples of such platforms include Loihi (Davies et al., 2018), BrainScaleS (Schemmel, Brüderle, et al., 2010), and SpiNNaker (Painkras et al., 2013); Nengo backends already exist for Loihi and SpiNNaker (Mundy et al., 2015; Blouw, Choo, et al., 2019). BrainScaleS in particular could be an interesting target because of its energy-efficient analogue computation, per-neuron synaptic filter time-constants, ability to simulate multi-compartment LIF neurons with conductance-based synapses (Section 3.3), and an integrated plasticity processor that could be used to solve for weights on-chip (Friedmann et al., 2017).

Sliding-window transformations and machine learning. In Section 4.2, we presented an “information erasure” technique for constructing LTI systems that generate windowed function bases as their impulse responses. We discovered that one of the systems generated in this manner, the “modified Fourier system”, outperforms the Legendre Delay Network (LDN; Voelker and Eliasmith, 2018) system in a number of benchmark experiments. Furthermore, we found in Section 4.4 that the modified Fourier system can be updated with relatively small errors when performing an efficient $\mathcal{O}(q)$ Runge-Kutta state update.

Future work should further investigate this system. So far, it remains unclear why the modified Fourier system can possibly outperform the LDN system. In theory, the Padé approximants underlying the LDN are an *optimal* approximation of a Laplace-domain delay¹ (Voelker, 2019, Section 6.1.1) and should thus form an optimal sliding-window basis. One culprit may be the sensitivity of the generated bases to our input signals $u(t)$, but our experiments in this regard are inconclusive (Appendix B.3.1).

In general, it remains unclear why systematic (i.e., of increasing frequency content) orthogonal sliding transformations are so effective in a deep-learning context, for example, as part of the Legendre Memory Unit (LMU; Voelker, Kajić, et al., 2019). In our Mackey-Glass experiment (Section 4.4.4; Table 4.4), all tested transformations were mathematically equivalent² in that they did not lose any information ($q = N$)—the learned linear read-out weights could in principle realise any temporal transformation. Still, we find that even after many epochs of training, all systematic bases outperform the identity and random transformations.

Summary of our contributions. To conclude, we presented several extensions to the NEF with the intent to better harness the dynamics inherent in spiking neural networks, and to provide tools for taking a larger number of potential neurophysiological constraints into account when modelling functional spiking neural networks. Such constraints include Dale’s principle (Section 3.2.2), multi-channel neurons with conductance-based synapses (Section 3.3), temporal

¹Note that, in general, there are few guarantees that the Padé approximants actually arrive at an optimal approximation of a function (Press et al., 2007, Section 5.12). However, the Padé approximants are optimal for exponential functions (Borwein, 1983). This includes the Laplace-domain delay $e^{-\theta t}$.

²At least in the version of the experiment with enforced rectangle window (cf. Figure B.11 and Table B.9).

tuning (Section 4.1.2), and spatially constrained networks (Section 5.2.1). Our techniques are based on a linear least squares current-space loss function (Section 3.2.1) that can be enhanced by a subthreshold relaxation term (Section 3.2.3), resulting in a convex quadratic program.

Analysing the dynamics underlying a simple multi-compartment neuron model, we found that networks comprised of two-compartment neurons can compute a wide range of benchmark functions at substantially smaller error than equivalent multi-layer networks (Section 3.4). In particular, one such benchmark function is nonnegative multiplication, also referred to as *gain modulation* in neuroscience (Salinas and Thier, 2000). While neurons with three and more compartments can approximate four-quadrant multiplication, the additional computational power may not outweigh the more complex weight optimisation procedure (Section 3.5).

Temporal tuning can be seen as a generalisation of the NEF dynamics principle that facilitates realising empirically observed spatiotemporal receptive fields, and can account for heterogeneous and higher-order synaptic filters without explicitly requiring access to higher-order differentials (Section 4.1.4). As before, we only rely on a linear least-squares loss (Section 4.1.2). Asking what optimal temporal tuning may look like, we suggested an autoregressive method for solving for LTI systems generating temporal bases with a subsequent *information erasure* step to enforce a rectangle window (Section 4.2.3). Using this technique, we provided a novel derivation of the LDN system (Section 4.2.4), and discovered an alternative *modified Fourier system* that outperforms the LDN in a variety of benchmark tasks.

Realising any of our LTI systems for approximating a sliding-window transformation in a spiking neuron population generates temporal activity patterns that resemble time-cells (Section 4.3.1). These same systems, overall, provide a powerful basis for computing multivariate spatiotemporal functions (Section 4.3.3). This includes online-learning of weights in an adaptive filter context (Section 4.3.4). We found that spatiotemporal NEF populations with full-rank temporal encoding matrices are equivalent to the Legendre Memory Unit (LMU; Voelker, Kajić, et al., 2019). Numerical data suggests that replacing the LDN with our modified Fourier system could result in a computationally more efficient network (Section 4.4.2) that performs similarly to the original LMU (Section 4.4.4).

Finally, we employed the techniques presented in this thesis to construct a model of eyeblink conditioning in the cerebellum. We demonstrated that it is possible to realise the LDN system in the recurrent Granule-Golgi circuit (Section 5.2), and that the resulting temporal representations support learning to decode delays in an eyeblink conditioning task (Section 5.3).

Potential future work includes strengthening the theoretical foundations of our methods, developing user-friendly simulation software that supports our extensions to the NEF, as well as implementing our work on neuromorphic hardware and analysing the modified Fourier system in more detail.

Bibliography

- Abadi, Martín et al. (2016). “TensorFlow: A System for Large-Scale Machine Learning”. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, pp. 265–283. ISBN: 978-1-931971-33-1. URL: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>.
- Abbott, Laurence F. and Frances S. Chance (2005). “Drivers and Modulators from Push-Pull and Balanced Synaptic Input”. In: *Cortical Function: A View from the Thalamus*. Vol. 149. Progress in Brain Research. Elsevier, pp. 147–155. DOI: 10.1016/S0079-6123(05)49011-1.
- Abbott, Laurence F. and Peter Dayan (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Computational Neuroscience. MIT Press. 480 pp. ISBN: 978-0-262-04199-7. URL: <https://mitpress.mit.edu/books/theoretical-neuroscience>.
- Adams, Marvin L. et al. (2012). *Assessing the Reliability of Complex Models: Mathematical and Statistical Foundations of Verification, Validation, and Uncertainty Quantification*. Washington, DC: The National Academies Press. ISBN: 978-0-309-25634-6. DOI: 10.17226/13395.
- Adelson, Edward H. and James R. Bergen (1985). “Spatiotemporal Energy Models for the Perception of Motion”. In: *Journal of the Optical Society of America A, Optics and Image Science* 2.2, pp. 284–299. DOI: 10.1364/JOSAA.2.000284.
- Aksay, Emre, Georgi Gamkrelidze, H. Sebastian Seung, Robert Baker, and David W. Tank (2001). “In Vivo Intracellular Recording and Perturbation of Persistent Activity in a Neural Integrator”. In: *Nature Neuroscience* 4.2, pp. 184–193. ISSN: 1546-1726. DOI: 10.1038/84023.
- Albus, James S. (1971). “A Theory of Cerebellar Function”. In: *Mathematical Biosciences* 10.1, pp. 25–61. ISSN: 0025-5564. DOI: 10.1016/0025-5564(71)90051-4.
- Alemi, Alireza, Christian K Machens, Sophie Deneve, and Jean-Jacques Slotine (2018). “Learning Nonlinear Dynamics in Efficient, Balanced Spiking Networks Using Local Plasticity Rules”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*.
- Ammar, Waleed et al. (2018). “Construction of the Literature Graph in Semantic Scholar”. In: *Proceedings of NAACL-HLT*. NAACL. New Orleans, Louisiana, United States: Association for Computational Linguistics, pp. 84–91.
- Anderson, John R., Michael Matessa, and Christian Lebiere (1997). “ACT-R: A Theory of Higher Level Cognition and Its Relation to Visual Attention”. In: *Human-Computer Interaction* 12.4, pp. 439–462. DOI: 10.1207/s15327051hci1204_5.

Bibliography

- Arfken, George B. and Hans J. Weber (2005). *Mathematical Methods for Physicists*. Sixth Edition. Burlington, MA: Elsevier Academic Press. 1182 pp. ISBN: 0-12-088584-0.
- Avin, Chen, Michael Borokhovich, Yoram Haddad, Erez Kantor, Zvi Lotker, Merav Parter, and David Peleg (2013). “Generalized Perron–Frobenius Theorem for Multiple Choice Matrices, and Applications”. In: *Proceedings of the 2013 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. New Orleans, Louisiana, United States: Society for Industrial and Applied Mathematics, pp. 478–498. ISBN: 978-1-61197-251-1.
- Back, A. D. and A. C. Tsoi (1991). “FIR and IIR Synapses, a New Neural Network Architecture for Time Series Modeling”. In: *Neural Computation* 3.3, pp. 375–385. ISSN: 0899-7667. DOI: 10.1162/neco.1991.3.3.375.
- Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun (2018). *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*.
- Baldominos, Alejandro, Yago Saez, and Pedro Isasi (2019). “A Survey of Handwritten Character Recognition with MNIST and EMNIST”. In: *Applied Sciences* 9.15. ISSN: 2076-3417. DOI: 10.3390/app9153169.
- Bassett, Danielle S and Olaf Sporns (2017). “Network Neuroscience”. In: *Nature Neuroscience* 20.3, pp. 353–364. ISSN: 1546-1726. DOI: 10.1038/nn.4502.
- Bastiaans, M. (1985). “On the Sliding-Window Representation in Digital Signal Processing”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 33.4, pp. 868–873. DOI: 10.1109/TASSP.1985.1164653.
- Bastos, Andre M., W. Martin Usrey, Rick A. Adams, George R. Mangun, Pascal Fries, and Karl J. Friston (2012). “Canonical Microcircuits for Predictive Coding”. In: *Neuron* 76.4, pp. 695–711. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2012.10.038.
- Bear, Mark F., Barry W. Connors, and Michael A. Paradiso (2016). *Neuroscience: Exploring the Brain*. Fourth Edition. Philadelphia, Pennsylvania: Wolters Kluwer. 975 pp. ISBN: 978-0-7817-7817-6.
- Bekolay, Trevor (2016). “Biologically Inspired Methods in Speech Recognition and Synthesis: Closing the Loop”. PhD thesis. University of Waterloo. URL: <https://uwspace.uwaterloo.ca/handle/10012/10269>.
- Bekolay, Trevor, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C. Stewart, Daniel Rasmussen, Xuan Choo, Aaron R. Voelker, and Chris Eliasmith (2014). “Nengo: A Python Tool for Building Large-Scale Functional Brain Models”. In: *Frontiers in Neuroinformatics* 7.48. ISSN: 1662-5196. DOI: 10.3389/fninf.2013.00048.
- Bekolay, Trevor, Mark Laubach, and Chris Eliasmith (2014). “A Spiking Neural Integrator Model of the Adaptive Control of Action by the Medial Prefrontal Cortex”. In: *The Journal of Neuroscience* 34.5, pp. 1892–1902. URL: <http://www.jneurosci.org/content/34/5/1892.short>.

- Beniaguev, David, Idan Segev, and Michael London (2021). “Single Cortical Neurons as Deep Artificial Neural Networks”. In: *Neuron*. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2021.07.002.
- Berkowitz, A. (2009). “Population Coding”. In: *Encyclopedia of Neuroscience*. Ed. by Larry R. Squire. Oxford: Academic Press, pp. 757–764. ISBN: 978-0-08-045046-9. DOI: 10.1016/B978-008045046-9.01970-7.
- Bernstein, Julius (1912). *Elektrobiologie, die Lehre von den elektrischen Vorgängen im Organismus auf moderner Grundlage dargestellt*. Braunschweig: Friedrich Vieweg & Sohn. 282 pp. URL: <https://archive.org/details/elektrobiologied00bern>.
- Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Red. by Michael Jordan, John Kleinberg, and Bernhard Schölkopf. Information Science and Statistics. New York, New York, United States: Springer Science+Business Media. 738 pp. ISBN: 978-0-387-31073-2. URL: <https://www.springer.com/gp/book/9780387310732>.
- Blouw, Peter, Xuan Choo, Eric Hunsberger, and Chris Eliasmith (2018). “Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware”. In: — (2019). “Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware”. In: *Proceedings of the 7th Annual Neuro-Inspired Computational Elements Workshop*, pp. 1–8.
- Blouw, Peter and Chris Eliasmith (2020). “Event-Driven Signal Processing with Neuromorphic Computing Systems”. In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8534–8538. DOI: 10.1109/ICASSP40776.2020.9053043.
- Blouw, Peter, Gurshaant Malik, Benjamin Morcos, Aaron R. Voelker, and Chris Eliasmith (2021). *Hardware Aware Training for Efficient Keyword Spotting on General Purpose and Specialized Hardware*.
- Boahen, Kwabena (2017). “A Neuromorph’s Prospectus”. In: *Computing in Science Engineering* 19.2, pp. 14–28. ISSN: 1521-9615. DOI: 10.1109/MCSE.2017.33.
- Bobier, Bruce, Terrence C. Stewart, and Chris Eliasmith (2014). “A Unifying Mechanistic Model of Selective Attention in Spiking Neurons”. In: *PLoS computational biology* 10.6. Ed. by Wolfgang Einhäuser, e1003577. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1003577.
- Boerlin, Martin and Sophie Denève (2011). “Spike-Based Population Coding and Working Memory”. In: *PLOS Computational Biology* 7.2, pp. 1–18. DOI: 10.1371/journal.pcbi.1001080.
- Boerlin, Martin, Christian K. Machens, and Sophie Denève (2013). “Predictive Coding of Dynamical Variables in Balanced Spiking Networks”. In: *PLOS Computational Biology* 9.11, pp. 1–16. DOI: 10.1371/journal.pcbi.1003258.
- Bollobás, Béla (1998). *Modern Graph Theory*. Graduate Texts in Mathematics 184. New York, New York, United States: Springer Science+Business Media. 394 pp. ISBN: 978-0-387-98488-9.

Bibliography

- Bolon, Philippe (2006). “Two-Dimensional Linear Filtering”. In: *Digital Filters Design for Signal and Image Processing*. John Wiley & Sons, Ltd, pp. 233–260. ISBN: 978-0-470-61206-4. DOI: 10.1002/9780470612064.ch8.
- Borwein, Peter B. (1983). “On Padé and Best Rational Approximation”. In: *Canadian Mathematical Bulletin* 26.1, pp. 50–57. DOI: 10.4153/CMB-1983-009-x.
- Boyd, Stephen and Lieven Vandenberghe (2004). *Convex Optimization*. Cambridge: Cambridge University Press. DOI: 10.1017/CB09780511804441.
- Braitenberg, Valentino and Almut Schüz (1998). *Cortex: Statistics and Geometry of Neuronal Connectivity*. Second thoroughly revised edition. Berlin, Germany: Springer-Verlag Berlin Heidelberg. 249 pp. ISBN: 978-3-662-03735-5. DOI: 10.1007/978-3-662-03733-1.
- (2013). *Anatomy of the Cortex: Statistics and Geometry*. Vol. 18. Springer Science & Business Media.
- Brette, Romain and Wulfram Gerstner (2005). “Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity”. In: *Journal of Neurophysiology* 94.5, pp. 3637–3642.
- Brogan, William L. (1991). *Modern Control Theory*. Third Edition. Upper Saddle River, New Jersey, United States: Prentice-Hall. 653 pp. ISBN: 978-0-13-589763-8.
- Broomhead, David S and David Lowe (1988). *Radial Basis Functions, Multi-Variable Functional Interpolation and Adaptive Networks*. Royal Signals and Radar Establishment Malvern (United Kingdom).
- Buckner, Randy L. (2013). “The Cerebellum and Cognitive Function: 25 Years of Insight from Anatomy and Neuroimaging”. In: *Neuron* 80.3, pp. 807–815. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2013.10.044.
- Byrd, Richard H., Peihuang Lu, Jorge Nocedal, and Ciyou Zhu (1995). “A Limited Memory Algorithm for Bound Constrained Optimization”. In: *SIAM Journal on Scientific Computing* 16.5, pp. 1190–1208. DOI: 10.1137/0916069.
- Calimera, Andrea, Enrico Macii, and Massimo Poncino (2013). “The Human Brain Project and Neuromorphic Computing”. In: *Functional neurology* 28.3, pp. 191–196. ISSN: 1971-3274. DOI: 10.11138/FNeur/2013.28.3.191.
- Camera, Giancarlo La, Alexander Rauch, Hans-R. Lüscher, Walter Senn, and Stefano Fusi (2004). “Minimal Models of Adapted Neuronal Response to In Vivo–Like Input Currents”. In: *Neural Computation* 16.10, pp. 2101–2124. DOI: 10.1162/0899766041732468.
- Capaday, Charles and Carl Van Vreeswijk (2006). “Direct Control of Firing Rate Gain by Dendritic Shunting Inhibition”. In: *Journal of integrative neuroscience* 5.02, pp. 199–222.
- Capocelli, R. M. and L. M. Ricciardi (1971). “Diffusion Approximation and First Passage Time Problem for a Model Neuron”. In: *Kybernetik* 8.6, pp. 214–223. ISSN: 1432-0770. DOI: 10.1007/BF00288750.

- Carandini, Matteo, David J. Heeger, and J. Anthony Movshon (1999). “Linearity and Gain Control in V1 Simple Cells”. In: *Models of Cortical Circuits*. Ed. by Philip S. Ulinski, Edward G. Jones, and Alan Peters. Boston, MA: Springer US, pp. 401–443. ISBN: 978-1-4615-4903-1.
- Chadderton, Paul, Troy W. Margrie, and Michael Häusser (2004). “Integration of Quanta in Cerebellar Granule Cells during Sensory Processing”. In: *Nature* 428.6985, pp. 856–860. ISSN: 1476-4687. DOI: 10.1038/nature02442.
- Chance, Frances S, Laurence F. Abbott, and Alex D Reyes (2002). “Gain Modulation from Background Synaptic Input”. In: *Neuron* 35.4, pp. 773–782. ISSN: 0896-6273. DOI: 10.1016/S0896-6273(02)00820-6.
- Chandar, Sarath, Chinnadhurai Sankar, Eugene Vorontsov, Samira Ebrahimi Kahou, and Yoshua Bengio (2019). “Towards Non-Saturating Recurrent Units for Modelling Long-Term Dependencies”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01, pp. 3280–3287. DOI: 10.1609/aaai.v33i01.33013280.
- Chen, Donghui and Robert J. Plemmons (2009). “Nonnegativity Constraints in Numerical Analysis”. In: *The Birth of Numerical Analysis*. Ed. by Adhemar Bultheel and Ronald Cools. Singapore: World Scientific, pp. 109–139. ISBN: 978-981-283-625-0. DOI: 10.1142/9789812836267_0008.
- Cheney, Ward and David Kincaid (2012). *Numerical Mathematics and Computing*. 7th Edition. Boston, Massachusetts, United States: Brooks/Cole. 677 pp. ISBN: 978-1-133-10371-4.
- Cheng, Dominic T., John F. Disterhoft, John M. Power, Deborah A. Ellis, and John E. Desmond (2008). “Neural Substrates Underlying Human Delay and Trace Eyeblink Conditioning”. In: *Proceedings of the National Academy of Sciences* 105.23, pp. 8108–8113. ISSN: 0027-8424. DOI: 10.1073/pnas.0800374105.
- Chi, H., M. Mascagni, and T. Warnock (2005). “On the Optimal Halton Sequence”. In: *Mathematics and Computers in Simulation* 70.1, pp. 9–21. ISSN: 0378-4754. DOI: 10.1016/j.matcom.2005.03.004.
- Chilkuri, Narsimha R. (2021). “Parallelizing Legendre Memory Unit Training”. MA thesis. Waterloo, Ontario, Canada: UWSpace. 50 pp. URL: <http://hdl.handle.net/10012/17142>.
- Chilkuri, Narsimha R. and Chris Eliasmith (2021). “Parallelizing Legendre Memory Unit Training”. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 1898–1907. URL: <https://proceedings.mlr.press/v139/chilkuri21a.html>.
- Chilkuri, Narsimha R., Eric Hunsberger, Aaron Voelker, Gurshaant Malik, and Chris Eliasmith (2021). *Language Modeling Using LMUs: 10x Better Data Efficiency or Improved Scaling Compared to Transformers*.

Bibliography

- Chollet, François (2017). *Deep Learning with Python*. Shelter Island, New York, United States: Manning. 384 pp. ISBN: 978-1-61729-443-3. URL: <https://www.manning.com/books/deep-learning-with-python>.
- Choo, Xuan (2018). “Spaun 2.0: Extending the World’s Largest Functional Brain Model”. PhD thesis. University of Waterloo / UWSpace. URL: <http://hdl.handle.net/10012/13308>.
- Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*.
- Churchland, Patricia S. and Terrence J. Sejnowski (1992). *The Computational Brain*. Computational Neuroscience Series. Cambridge, Massachusetts, United States: MIT Press. 568 pp. ISBN: 978-0-262-53339-3.
- Cleland, Thomas A. (1996). “Inhibitory Glutamate Receptor Channels”. In: *Molecular Neurobiology* 13.2, pp. 97–136. ISSN: 1559-1182. DOI: 10.1007/BF02740637.
- Cooley, James W. and John W. Tukey (1965). “An Algorithm for the Machine Calculation of Complex Fourier Series”. In: *Mathematics of Computation* 19, pp. 297–301. DOI: 10.1090/S0025-5718-1965-0178586-1.
- D’Angelo, Egidio, Sergio Solinas, Jonathan Mapelli, Daniela Gandolfi, Lisa Mapelli, and Francesca Prestori (2013). “The Cerebellar Golgi Cell and Spatiotemporal Organization of Granular Layer Activity”. In: *Frontiers in Neural Circuits* 7, p. 93. ISSN: 1662-5110. DOI: 10.3389/fncir.2013.00093.
- Davies, M. et al. (2018). “Loihi: A Neuromorphic Manycore Processor with on-Chip Learning”. In: *IEEE Micro* 38.1, pp. 82–99. DOI: 10.1109/MM.2018.112130359.
- De Bot, Kees, Wander Lowie, and Marjolijn Verspoor (2007). “A Dynamic Systems Theory Approach to Second Language Acquisition”. In: *Bilingualism: Language and Cognition* 10.1, pp. 7–21. DOI: 10.1017/S1366728906002732.
- Dean, Paul, John Porrill, Carl-Fredrik Ekerot, and Henrik Jörntell (2010). “The Cerebellar Microcircuit as an Adaptive Filter: Experimental and Computational Evidence”. In: *Nature Reviews Neuroscience* 11.1, pp. 30–43. ISSN: 1471-0048. DOI: 10.1038/nrn2756.
- DeAngelis, G. C., I. Ohzawa, and R. D. Freeman (1993). “Spatiotemporal Organization of Simple-Cell Receptive Fields in the Cat’s Striate Cortex. I. General Characteristics and Postnatal Development”. In: *Journal of Neurophysiology* 69.4, pp. 1091–1117. DOI: 10.1152/jn.1993.69.4.1091.
- Den Brinler, A.C. (1996). “The Generalized Sliding-Window Spectrum”. In: *Proceedings of Third International Symposium on Time-Frequency and Time-Scale Analysis (TFTS-96)*, pp. 425–428. DOI: 10.1109/TFSA.1996.550083.
- Dieudonné, Stéphane (1998). “Submillisecond Kinetics and Low Efficacy of Parallel Fibre-Golgi Cell Synaptic Currents in the Rat Cerebellum”. In: *The Journal of Physiology* 510.3, pp. 845–866. DOI: 10.1111/j.1469-7793.1998.845bj.x.

- Duggins, Peter (2017). “Incorporating Biologically Realistic Neuron Models into the NEF”. MA thesis. University of Waterloo. URL: <https://uwspace.uwaterloo.ca/handle/10012/12393>.
- Duggins, Peter, Terrence C. Stewart, Xuan Choo, and Chris Eliasmith (2017). “Effects of Guanfacine and Phenylephrine on a Spiking Neuron Model of Working Memory”. In: *Topics in Cognitive Science*. ISSN: 1756-8757. DOI: 10.1111/tops.12247.
- E, Keren-Happuch, Shen-Hsing Annabel Chen, Moon-Ho Ringo Ho, and John E Desmond (2014). “A Meta-Analysis of Cerebellar Contributions to Higher Cognition from PET and fMRI Studies”. In: *Human brain mapping* 35.2, pp. 593–615. ISSN: 1097-0193. DOI: 10.1002/hbm.22194.
- Eccles, John C. (1986). “Chemical Transmission and Dale’s Principle”. In: ed. by T. Hökfelt, K. Fuxe, and B. Pernow. Vol. 68. Progress in Brain Research. Elsevier, pp. 3–13. DOI: 10.1016/S0079-6123(08)60227-7.
- Eliasmith, Chris (1996). “The Third Contender: A Critical Examination of the Dynamicist Theory of Cognition”. In: *Philosophical Psychology* 9.4, pp. 441–463. DOI: 10.1080/09515089608573194.
- (2013). *How to Build a Brain: A Neural Architecture for Biological Cognition*. Oxford Series on Cognitive Models and Architectures. New York, New York: Oxford University Press. 456 pp. ISBN: 978-0-19-026212-9.
- Eliasmith, Chris and Charles H. Anderson (2003). *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. Cambridge, Massachusetts: MIT Press. 380 pp. ISBN: 978-0-262-55060-4.
- Eliasmith, Chris, Jan Gosmann, and Xuan-Feng Choo (2016). “BioSpaun: A Large-Scale Behaving Brain Model with Complex Neurons”. In: *ArXiv*.
- Eliasmith, Chris and Carter Kolbeck (2015). “Marr’s Attacks: On Reductionism and Vagueness”. In: *Topics in Cognitive Science*, pp. 1–13. DOI: 10.1111/tops.12133.
- Eliasmith, Chris, Terrence C. Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen (2012). “A Large-Scale Model of the Functioning Brain”. In: *Science* 338, pp. 1202–1205. DOI: 10.1126/science.1225266.
- Ellson, John, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull (2004). “Graphviz and Dynagraph — Static and Dynamic Graph Drawing Tools”. In: *Graph Drawing Software*. Ed. by Michael Jünger and Petra Mutzel. Mathematics and Visualization. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, pp. 127–148. ISBN: 978-3-642-18638-7.
- Enderle, John (2011). “Bioelectric Phenomena”. In: *Introduction to Biomedical Engineering*. Ed. by John Enderle and Joseph Bronzino. 3rd. Academic Press Series in Biomedical Engineering. Burlington, Massachusetts, USA: Academic Press, pp. 747–815. ISBN: 978-0-

Bibliography

- 12-374979-6. URL: <https://www.elsevier.com/books/introduction-to-biomedical-engineering/enderle/978-0-12-374979-6>.
- Fang, Kai-Tai and Yuan Wang (1994). *Number-Theoretic Methods in Statistics*. First edition. Monographs on Statistics and Applied Probability 51. London, UK: Chapman and Hall. 344 pp. ISBN: 978-0-412-46520-8.
- Field, David J., David J. Tolhurst, and Fergus William Campbell (1986). “The Structure and Symmetry of Simple-Cell Receptive-Field Profiles in the Cat’s Visual Cortex”. In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 228.1253, pp. 379–400. DOI: 10.1098/rspb.1986.0060.
- Friedman-Hill, Stacia, Pedro E. Maldonado, and Charles M. Gray (2000). “Dynamics of Striate Cortical Activity in the Alert Macaque: I. Incidence and Stimulus-Dependence of Gamma-Band Neuronal Oscillations”. In: *Cerebral Cortex* 10.11, pp. 1105–1116. ISSN: 1047-3211. DOI: 10.1093/cercor/10.11.1105.
- Friedmann, Simon, Johannes Schemmel, Andreas Grübl, Andreas Hartel, Matthias Hock, and Karlheinz Meier (2017). “Demonstrating Hybrid Learning in a Flexible Neuromorphic Hardware System”. In: *IEEE Transactions on Biomedical Circuits and Systems* 11.1, pp. 128–142. ISSN: 1940-9990. DOI: 10.1109/TBCAS.2016.2579164.
- Fujita, M. (1982). “Adaptive Filter Model of the Cerebellum”. In: *Biological Cybernetics* 45.3, pp. 195–206. ISSN: 1432-0770. DOI: 10.1007/BF00336192.
- Furber, Steve B. (2016). “Large-Scale Neuromorphic Computing Systems”. In: *Journal of Neural Engineering* 13.5, p. 051001. DOI: 10.1088/1741-2560/13/5/051001.
- Furber, Steve B., David R. Lester, Luis Plana, Jim D. Garside, Eustace Painkras, Sally Temple, and Andrew D. Brown (2013). “Overview of the SpiNNaker System Architecture”. In: *Computers, IEEE Transactions on* 62.12, pp. 2454–2467.
- Gabbott, P. L. A. and P. Somogyi (1986). “Quantitative Distribution of GABA-immunoreactive Neurons in the Visual Cortex (Area 17) of the Cat”. In: *Experimental Brain Research* 61.2, pp. 323–331. ISSN: 1432-1106. DOI: 10.1007/BF00239522.
- Gabor, Dennis (1946). “Theory of Communication. Part 1: The Analysis of Information”. In: *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering* 93.26, 429–441(12). ISSN: 0367-7540. URL: <https://digital-library.theiet.org/content/journals/10.1049/ji-3-2.1946.0074>.
- Gallego, Juan A., Matthew G. Perich, Lee E. Miller, and Sara A. Solla (2017). “Neural Manifolds for the Control of Movement”. In: *Neuron* 94.5, pp. 978–984. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2017.05.025.
- Gardner, William G. (1995). “Efficient Convolution without Input-Output Delay”. In: *Journal of the Audio Engineering Society* 43.3, pp. 127–136. URL: <http://www.aes.org/e-lib/browse.cfm?elib=7957>.

- Gayler, Ross (2003). “Vector Symbolic Architectures Answer Jackendoff’s Challenges for Cognitive Neuroscience”. In: *Proceedings of the ICCS/ASCS International Conference on Cognitive Science*. ICCS/ASCS International Conference on Cognitive Science. Sydney, Australia: University of New South Wales, pp. 133–138.
- George, Suma, Sihwan Kim, Sahil Shah, Jennifer Hasler, Michelle Collins, Farhan Adil, Richard Wunderlich, Stephen Nease, and Shubha Ramakrishnan (2016). “A Programmable and Configurable Mixed-Mode FPAA SoC”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.6, pp. 2253–2261. DOI: 10.1109/TVLSI.2015.2504119.
- Gerstner, Wulfram and Werner M. Kistler (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press. ISBN: 978-0-521-89079-3.
- Gerstner, Wulfram, Werner M. Kistler, R. Naud, and L. Paninski (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press. ISBN: 978-1-107-06083-8.
- Gewaltig, Marc-Oliver and Markus Diesmann (2007). “NEST (NEural Simulation Tool)”. In: *Scholarpedia* 2.4, p. 1430.
- Godsil, Chris and Gordon Royle (2001). *Algebraic Graph Theory*. 1st edition. Graduate Texts in Mathematics 207. New York, New York, United States: Springer Science+Business Media. 439 pp. ISBN: 978-0-387-95220-8.
- Goldman, David E. (1943). “Potential, Impedance, and Rectification in Membranes”. In: *Journal of General Physiology* 27.1, pp. 37–60. ISSN: 0022-1295. DOI: 10.1085/jgp.27.1.37.
- Göltz, Julian et al. (2021). “Fast and Energy-Efficient Neuromorphic Deep Learning with First-Spike Times”. In: *Nature Machine Intelligence* 3.9, pp. 823–835. ISSN: 2522-5839. DOI: 10.1038/s42256-021-00388-x.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. Cambridge, Massachusetts, United States: MIT Press. 800 pp. ISBN: 978-0-262-03561-3.
- Gosmann, Jan (2015). *Precise Multiplications with the NEF*. Waterloo, ON: Centre for Theoretical Neuroscience.
- Gosmann, Jan and Chris Eliasmith (2017). “Automatic Optimization of the Computation Graph in the Nengo Neural Network Simulator”. In: *Frontiers in Neuroinformatics* 11, p. 33. ISSN: 1662-5196. DOI: 10.3389/fninf.2017.00033.
- (2021). “CUE: A Unified Spiking Neuron Model of Short-Term and Long-Term Memory”. In: *Psychological Review* 128.1, pp. 104–124. DOI: 10.1037/rev0000250.
- Grossberg, Stephen and Nestor A. Schmajuk (1989). “Neural Dynamics of Adaptive Timing and Temporal Discrimination during Associative Learning”. In: *Neural Networks* 2.2, pp. 79–102. ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90026-9.
- Gu, Albert, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré (2020). “HiPPO: Recurrent Memory with Optimal Polynomial Projections”. In: *Advances in Neural Information Process-*

Bibliography

- ing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., pp. 1474–1487. URL: <https://proceedings.neurips.cc/paper/2020/file/102f0bb6efb3a6128a3c750dd16729be-Paper.pdf>.
- Güçlü, Umut and Marcel A. J. van Gerven (2015). “Deep Neural Networks Reveal a Gradient in the Complexity of Neural Representations across the Ventral Stream”. In: *Journal of Neuroscience* 35.27, pp. 10005–10014. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.5023-14.2015.
- Guennebaud, Gaël, Benoît Jacob, et al. (2010). *Eigen V3*. URL: <http://eigen.tuxfamily.org>.
- Gupta, Anirudh, Yun Wang, and Henry Markram (2000). “Organizing Principles for a Diversity of GABAergic Interneurons and Synapses in the Neocortex”. In: *Science* 287.5451, pp. 273–278. ISSN: 0036-8075. DOI: 10.1126/science.287.5451.273.
- Gupta, Gaurav, Meghana Kshirsagar, Ming Zhong, Shahrzad Gholami, and Juan Lavista Ferres (2021). “Comparing Recurrent Convolutional Neural Networks for Large Scale Bird Species Classification”. In: *Scientific Reports* 11.1, p. 17085. ISSN: 2045-2322. DOI: 10.1038/s41598-021-96446-w.
- Haar, Alfred (1910). “Zur Theorie der orthogonalen Funktionensysteme”. In: *Mathematische Annalen* 69, pp. 331–371. DOI: 10.1007/BF01456326.
- Haykin, Simon (2014). *Adaptive Filter Theory*. 5th international edition. London, United Kingdom: Pearson Education, Limited. ISBN: 978-0-273-76408-3.
- He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.
- Hefferon, Jim (2020). *Linear Algebra*. 525 pp. ISBN: 978-1-944325-03-9. URL: <https://hefferon.net/linearalgebra/>.
- Heiney, Shane A., Margot P. Wohl, Selmaan N. Chettih, Luis I. Ruffolo, and Javier F. Medina (2014). “Cerebellar-Dependent Expression of Motor Learning during Eyeblink Conditioning in Head-Fixed Mice”. In: *Journal of Neuroscience* 34.45, pp. 14845–14853. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.2820-14.2014.
- Hendry, SH and Edward G. Jones (1981). “Sizes and Distributions of Intrinsic Neurons Incorporating Tritiated GABA in Monkey Sensory-Motor Cortex”. In: *Journal of Neuroscience* 1.4, pp. 390–408. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.01-04-00390.1981.
- Herz, Andreas V. M., Tim Gollisch, Christian K. Machens, and Dieter Jaeger (2006). “Modeling Single-Neuron Dynamics and Computations: A Balance of Detail and Abstraction”. In: *Science* 314.5796, pp. 80–85. ISSN: 0036-8075. DOI: 10.1126/science.1127240.
- Hilbert, David (1891). “Über Die Stetige Abbildung Einer Linie Auf Ein Flächenstück”. In: *Mathematische Annalen* 38, pp. 459–460. ISSN: 0025-5831.

- Hinton, Geoffrey E., Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov (2012). *Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors*.
- Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long Short-Term Memory”. In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.
- Hodgkin, A. L. and B. Katz (1949). “The Effect of Sodium Ions on the Electrical Activity of the Giant Axon of the Squid”. In: *The Journal of Physiology* 108.1, pp. 37–77. DOI: 10.1113/jphysiol.1949.sp004310.
- Hodgkin, Alan L. and Andrew F. Huxley (1952). “A Quantitative Description of Membrane Current and Its Application to Conduction and Excitation in Nerve”. In: *The Journal of Physiology* 117.4, pp. 500–544.
- Hokanson, Jeffrey M. and Caleb C. Magruder (2018). *Least Squares Rational Approximation*.
- Holt, Gary R. and Christof Koch (1997). “Shunting Inhibition Does Not Have a Divisive Effect on Firing Rates”. In: *Neural Computation* 9.5, pp. 1001–1013. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.5.1001.
- Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). “Multilayer Feedforward Networks Are Universal Approximators”. In: *Neural Networks* 2.5, pp. 359–366. ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8.
- Howard, Marc W., Christopher J. MacDonald, Zoran Tiganj, Karthik H. Shankar, Qian Du, Michael E. Hasselmo, and Howard Eichenbaum (2014). “A Unified Mathematical Framework for Coding Time, Space, and Sequences in the Hippocampal Region”. In: *Journal of Neuroscience* 34.13, pp. 4692–4707. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.5808-12.2014.
- Howell, William H. (1916). *A Text-Book of Physiology: For Medical Students and Physicians*. London, UK: Philadelphia and London: W. B. Saunders Company. 1043 pp. URL: <https://archive.org/details/textbookofphysio1916howe>.
- Hubbard, John H. (1999). “The Forced Damped Pendulum: Chaos, Complication and Control”. In: *The American Mathematical Monthly* 106.8, pp. 741–758. DOI: 10.1080/00029890.1999.12005113.
- Hubel, D. H. and T. N. Wiesel (1962). “Receptive Fields, Binocular Interaction and Functional Architecture in the Cat’s Visual Cortex”. In: *The Journal of Physiology* 160.1, pp. 106–154. DOI: 10.1113/jphysiol.1962.sp006837.
- Hubel, David H. and Torsten N. Wiesel (1959). “Receptive Fields of Single Neurones in the Cat’s Striate Cortex”. In: *The Journal of physiology* 148.3, pp. 574–591. DOI: 10.1113/jphysiol.1959.sp006308.
- Hunsberger, Eric (2016). *System Identification of Adapting Neurons*. Waterloo, ON: Centre for Theoretical Neuroscience. DOI: 10.13140/RG.2.2.28060.08327.

Bibliography

- Hunsberger, Eric (2018). “Spiking Deep Neural Networks: Engineered and Biological Approaches to Object Recognition”. PhD thesis. University of Waterloo. URL: <http://hdl.handle.net/10012/12819>.
- Hunsberger, Eric and Chris Eliasmith (2015). *Spiking Deep Networks with LIF Neurons*. URL: <http://arxiv.org/abs/1510.08829>.
- (2016). *Training Spiking Deep Networks for Neuromorphic Hardware*. DOI: 10.13140/RG.2.2.10967.06566. URL: <https://arxiv.org/abs/1611.05141>.
- Hunsberger, Eric, Matthew Scott, and Chris Eliasmith (2014). “The Competing Benefits of Noise and Heterogeneity in Neural Coding”. In: *Neural Computation* 26.8. DOI: 10.1162/NECO_a_00621.
- Hurzook, Aziz (2012). “A Mechanistic Model of Motion Processing in the Early Visual System”. MA thesis. Waterloo, Ontario: University of Waterloo. 94 pp. URL: <http://hdl.handle.net/10012/7140>.
- IEEE Standard for Information Technology: Portable Operating System Interface (POSIX®)* (2018). Issue 7. The Open Group Standard Base Specifications. New York, New York, United States: The Open Group and The Institute of Electrical and Electronics Engineers. 3903 pp. ISBN: 978-1-5044-4542-9.
- Ito, Masao (2010). “Cerebellar Cortex”. In: *Handbook of Brain Microcircuits*. Ed. by Gordon Shepherd and Sten Grillner. 1st. Oxford, UK: Oxford University Press, pp. 293–300. ISBN: 978-0-19-538988-3. URL: <https://oxfordmedicine.com/view/10.1093/med/9780195389883.001.0001/med-9780195389883-chapter-028>.
- Izhikevich, Eugene M. (2004). “Which Model to Use for Cortical Spiking Neurons?” In: *IEEE transactions on neural networks* 15.5, pp. 1063–1070.
- (2007). *Dynamical Systems in Neuroscience*. Computational Neuroscience. Cambridge, Massachusetts, USA: MIT Press. 464 pp. ISBN: 978-0-262-51420-0. URL: <https://www.izhikevich.org/publications/dsn.pdf>.
- Jacobsen, E. and R. Lyons (2003). “The Sliding DFT”. In: *IEEE Signal Processing Magazine* 20.2, pp. 74–80. ISSN: 1558-0792. DOI: 10.1109/MSP.2003.1184347.
- Jaeger, Herbert and Harald Haas (2004). “Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication”. In: *Science* 304.5667, pp. 78–80. DOI: 10.1126/science.1091277.
- Jakab, R. L. and J. Hátori (1988). “Quantitative Morphology and Synaptology of Cerebellar Glomeruli in the Rat”. In: *Anatomy and Embryology* 179.1, pp. 81–88. ISSN: 1432-0568. DOI: 10.1007/BF00305102.
- Jech, Thomas (2003). *Set Theory*. The Third Millennium Edition, revised and expanded. Berlin, Germany: Springer. 769 pp. ISBN: 978-3-540-44085-7.

- Johansson, Fredrik, Dan-Anders Jirenhed, Anders Rasmussen, Riccardo Zucca, and Germund Hesslow (2014). “Memory Trace and Timing Mechanism Localized to Cerebellar Purkinje Cells”. In: *Proceedings of the National Academy of Sciences* 111.41, pp. 14930–14934. ISSN: 0027-8424. DOI: 10.1073/pnas.1415371111.
- Jonas, Peter, Guy Major, and Bert Sakmann (1993). “Quantal Components of Unitary EPSCs at the Mossy Fibre Synapse on CA3 Pyramidal Cells of Rat Hippocampus”. In: *The Journal of Physiology* 472.1, pp. 615–663.
- Jones, Peter and Trevor Bekolay (2014). *Neurotransmitter Time Constants (PSCs)*. URL: <http://compneuro.uwaterloo.ca/research/constants-constraints/neurotransmitter-time-constants-pscs.html> (visited on 04/26/2021).
- Kaiser, G. (1998). “The Fast Haar Transform”. In: *IEEE Potentials* 17.2, pp. 34–37. ISSN: 1558-1772. DOI: 10.1109/45.666645.
- Kalman, R. E. (1960). “A New Approach to Linear Filtering and Prediction Problems”. In: *Journal of Basic Engineering* 82.1, pp. 35–45. ISSN: 0021-9223. DOI: 10.1115/1.3662552.
- Kandel, Eric, James H. Schwartz, Thomas M. Jessell, Steven A. Siegelbaum, and A. J. Hudspeth (2012). *Principles of Neural Science*. 5th ed. McGraw-Hill Education.
- Kanichay, Roby T. and R. Angus Silver (2008). “Synaptic and Cellular Properties of the Feed-forward Inhibitory Circuit within the Input Layer of the Cerebellar Cortex”. In: *Journal of Neuroscience* 28.36, pp. 8955–8967. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.5469-07.2008.
- Kanwisher, Nancy and Galit Yovel (2006). “The Fusiform Face Area: A Cortical Region Specialized for the Perception of Faces”. In: *Philosophical transactions of the Royal Society of London. Series B, Biological sciences* 361.1476, pp. 2109–2128. ISSN: 0962-8436. DOI: 10.1098/rstb.2006.1934.
- Kingma, Diederik P. and Jimmy Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. URL: <http://arxiv.org/abs/1412.6980>.
- Kiselev, Mikhail, Alexey Ivanov, and Daniil Ivanov (2020). “Approximating Conductance-Based Synapses by Current-Based Synapses”. In: *Advances in Neural Computation, Machine Learning, and Cognitive Research IV*. Ed. by Boris Kryzhanovskiy, Witali Dunin-Barkowski, Vladimir Redko, and Yury Tiumentsev. Moscow, Russia: Springer International Publishing, pp. 394–402. ISBN: 978-3-030-60577-3.
- Kober, V. (2004). “Fast Algorithms for the Computation of Sliding Discrete Sinusoidal Transforms”. In: *IEEE Transactions on Signal Processing* 52.6, pp. 1704–1710. DOI: 10.1109/TSP.2004.827184.

Bibliography

- Koch, Christof (1999). *Biophysics of Computation: Information Processing in Single Neurons*. Oxford, United Kingdom: Oxford University Press. 562 pp. ISBN: 0-19-510491-9.
- Koch, Christof, Chun-Hui Mo, and William Softky (2002). “Single-Cell Models”. In: *The Handbook of Brain Theory and Neural Networks*. Ed. by Michael A. Arbib. Second Edition. Cambridge, Massachusetts, United States: MIT Press, pp. 1044–1049. ISBN: 0-262-01197-2.
- Koch, Christof and Tomaso Poggio (1992). “Multiplying with Synapses and Neurons”. In: *Single Neuron Computation*. Ed. by Thomas McKenna, Joel Davis, and Steven F. Zornetzer. Neural Networks: Foundations to Applications. San Diego: Academic Press, pp. 315–345. ISBN: 978-0-12-484815-3. DOI: 10.1016/B978-0-12-484815-3.50019-0.
- Koch, Christof, Tomaso Poggio, and V Torre (1983). “Nonlinear Interactions in a Dendritic Tree: Localization, Timing, and Role in Information Processing”. In: *Proceedings of the National Academy of Sciences* 80.9, pp. 2799–2802. ISSN: 0027-8424. DOI: 10.1073/pnas.80.9.2799.
- Komer, Brent (2015). “Biologically Inspired Adaptive Control of Quadcopter Flight”. MA thesis. Waterloo, ON: University of Waterloo. 65 pp. URL: <https://uwspace.uwaterloo.ca/handle/10012/9549>.
- (2020). “Biologically Inspired Spatial Representation”. PhD thesis. Waterloo, ON: University of Waterloo. URL: <https://uwspace.uwaterloo.ca/handle/10012/16430>.
- Komer, Brent and Chris Eliasmith (2016). “A Unified Theoretical Approach for Biological Cognition and Learning”. In: *Current Opinion in Behavioral Sciences* 11, pp. 14–20. ISSN: 2352-1546. DOI: 10.1016/j.cobeha.2016.03.006.
- Korbo, Lise, Birgitte Bo Andersen, Ole Ladefoged, and Arne Møller (1993). “Total Numbers of Various Cell Types in Rat Cerebellar Cortex Estimated Using an Unbiased Stereological Method”. In: *Brain Research* 609.1, pp. 262–268. ISSN: 0006-8993. DOI: 10.1016/0006-8993(93)90881-M.
- Kozlov, M.K., S.P. Tarasov, and L.G. Khachiyan (1980). “The Polynomial Solvability of Convex Quadratic Programming”. In: *USSR Computational Mathematics and Mathematical Physics* 20.5, pp. 223–228. ISSN: 0041-5553. DOI: 10.1016/0041-5553(80)90098-1.
- Kreutz-Delgado, Ken (2015). *Mean Time-to-Fire for the Noisy LIF Neuron-A Detailed Derivation of the Siegert Formula*.
- Lapicque, Louis (1907). “Recherches quantitatives sur l’excitation électrique des nerfs traitée comme une polarisation”. In: *Journal de Physiologie et de Pathologie Generale* 9, pp. 620–635.
- Lawson, Charles L. and Richard J. Hanson (1995). *Solving Least Squares Problems*. Classics Edition. Classics in Applied Mathematics. Philadelphia, Pennsylvania, United States: Society for Industrial and Applied Mathematics. 337 pp. ISBN: 978-0-89871-356-5. URL: <https://epubs.siam.org/doi/book/10.1137/1.9781611971217>.
- Le, Quoc V., Navdeep Jaitly, and Geoffrey E. Hinton (2015). *A Simple Way to Initialize Recurrent Networks of Rectified Linear Units*.

- Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: 10.1109/5.726791.
- LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep Learning”. In: *Nature* 521, p. 436. URL: <https://doi.org/10.1038/nature14539>.
- Levin, Michael (2014). “Molecular Bioelectricity: How Endogenous Voltage Potentials Control Cell Behavior and Instruct Pattern Regulation in Vivo”. In: *Molecular Biology of the Cell* 25.24, pp. 3835–3850. DOI: 10.1091/mbc.e13-12-0708.
- Levy, E. C. (1959). “Complex-Curve Fitting”. In: *IRE Transactions on Automatic Control* AC-4.1, pp. 37–43. DOI: 10.1109/TAC.1959.6429401.
- Llinás, Rodolfo R. (2010). “Olivocerebellar System”. In: *Handbook of Brain Microcircuits*. Ed. by Gordon Shepherd and Sten Grillner. 1st. Oxford, UK: Oxford University Press, pp. 301–308. ISBN: 978-0-19-538988-3. URL: <https://oxfordmedicine.com/view/10.1093/med/9780195389883.001.0001/med-9780195389883-chapter-029>.
- London, Michael and Michael Häusser (2005). “Dendritic Computation”. In: *Annual Review of Neuroscience* 28.1, pp. 503–532. DOI: 10.1146/annurev.neuro.28.061604.135703.
- Lopes da Silva, F H and S I Gonçalves (2009). “Electroencephalography (EEG)”. In: *Encyclopedia of Neuroscience*. Ed. by J C De Munck. Oxford: Academic Press, pp. 849–855. ISBN: 978-0-08-045046-9. DOI: 10.1016/B978-008045046-9.01401-7.
- Lundqvist, Mikael, Pawel Herman, and Earl K. Miller (2018). “Working Memory: Delay Activity, Yes! Persistent Activity? Maybe Not”. In: *Journal of Neuroscience* 38.32, pp. 7013–7019. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.2485-17.2018.
- Lusk, Nicholas A, Elijah A Petter, Christopher J MacDonald, and Warren H Meck (2016). “Cerebellar, Hippocampal, and Striatal Time Cells”. In: *Current Opinion in Behavioral Sciences* 8, pp. 186–192. ISSN: 2352-1546. DOI: 10.1016/j.cobeha.2016.02.020.
- Lütkepohl, Helmut (1997). *Handbook of Matrices*. Chichester, England, United Kingdom: John Wiley & Sons. 320 pp. ISBN: 978-0-471-97015-6.
- Ma, W.J. and A. Pouget (2009). “Population Codes: Theoretic Aspects”. In: *Encyclopedia of Neuroscience*. Ed. by Larry R. Squire. Oxford: Academic Press, pp. 749–755. ISBN: 978-0-08-045046-9. DOI: 10.1016/B978-008045046-9.01401-7.
- MacDonald, Christopher J., Kyle Q. Lepage, Uri T. Eden, and Howard Eichenbaum (2011). “Hippocampal “Time Cells” Bridge the Gap in Memory for Discontiguous Events”. In: *Neuron* 71.4, pp. 737–749. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2011.07.012.
- Mackey, Michael C. and Leon Glass (1977). “Oscillation and Chaos in Physiological Control Systems”. In: *Science* 197.4300, pp. 287–289. DOI: 10.1126/science.267326.

Bibliography

- MacNeil, David and Chris Eliasmith (2011). “Fine-Tuning and the Stability of Recurrent Neural Networks”. In: *PLOS ONE* 6.9, pp. 1–16. DOI: 10.1371/journal.pone.0022885.
- Makhoul, J. (1980). “A Fast Cosine Transform in One and Two Dimensions”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.1, pp. 27–34. DOI: 10.1109/TASSP.1980.1163351.
- Marčelja, S. (1980). “Mathematical Description of the Responses of Simple Cortical Cells”. In: *Journal of the Optical Society of America* 70.11, pp. 1297–1300. DOI: 10.1364/JOSA.70.001297.
- Marr, David (1969). “A Theory of Cerebellar Cortex”. In: *The Journal of physiology* 202.2, pp. 437–470. ISSN: 0022-3751. DOI: 10.1113/jphysiol.1969.sp008820.
- (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. San Francisco, California, United States: W. H. Freeman. 361 pp. ISBN: 978-0-262-51462-0.
- Marr, David and Tomaso Poggio (1976). “From Understanding Computation to Understanding Neural Circuitry”. In: *AI Memo* 357, p. 22. URL: <http://hdl.handle.net/1721.1/5782>.
- McAllister, A. Kimberley (2000). “Cellular and Molecular Mechanisms of Dendrite Growth”. In: *Cerebral Cortex* 10.10, pp. 963–973. ISSN: 1047-3211. DOI: 10.1093/cercor/10.10.963.
- McCormick, David A, David G Lavond, Gregory A Clark, Ronald E Kettner, Christina E Rising, and Richard F Thompson (1981). “The Engram Found? Role of the Cerebellum in Classical Conditioning of Nictitating Membrane and Eyelid Responses”. In: *Bulletin of the Psychonomic Society* 18.3, pp. 103–105.
- McCormick, David A. (2014). “Membrane Potential and Action Potential”. In: *From Molecules to Networks*. Ed. by John H. Byrne, Ruth Heidelberger, and M. Neal Waxham. Third Edition. Boston: Academic Press, pp. 351–376. ISBN: 978-0-12-397179-1. DOI: 10.1016/B978-0-12-397179-1.00012-9.
- McCormick, David A., Yousheng Shu, and Yuguo Yu (2007). “Hodgkin and Huxley Model — Still Standing?” In: *Nature* 445.7123, E1–E2. ISSN: 1476-4687. DOI: 10.1038/nature05523.
- McCulloch, Warren S. and Walter Pitts (1943). “A Logical Calculus of the Ideas Immanent in Nervous Activity”. In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.
- McLean, Judith and Larry A. Palmer (1989). “Contribution of Linear Spatiotemporal Receptive Field Structure to Velocity Selectivity of Simple Cells in Area 17 of Cat”. In: *Vision Research* 29.6, pp. 675–679. ISSN: 0042-6989. DOI: 10.1016/0042-6989(89)90029-1.
- Mead, Carver (1990). “Neuromorphic Electronic Systems”. In: *Proceedings of the IEEE* 78.10, pp. 1629–1636. DOI: 10.1109/5.58356.
- Medina, Javier F. and Michael D. Mauk (2000). “Computer Simulation of Cerebellar Information Processing”. In: *Nature Neuroscience* 3.11, pp. 1205–1211. ISSN: 1546-1726. DOI: 10.1038/81486.

- Mel, Bartlett W. (1994). “Information Processing in Dendritic Trees”. In: *Neural Computation* 6.6, pp. 1031–1085. ISSN: 0899-7667. DOI: 10.1162/neco.1994.6.6.1031.
- Mendel, Jerry M. (2017). *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions*. 2nd Edition. Cham, Switzerland: Springer International Publishing AG. 701 pp. ISBN: 978-3-319-51369-0. DOI: 10.1007/978-3-319-51370-6.
- Meriney, Stephen D. and Erika E. Fanselow (2019). *Synaptic Transmission*. London, UK: Academic Press. 516 pp. ISBN: 978-0-12-815320-8. DOI: 10.1016/C2017-0-02762-3.
- Meunier, Claude and Idan Segev (2002). “Playing the Devil’s Advocate: Is the Hodgkin–Huxley Model Useful?” In: *Trends in Neurosciences* 25.11, pp. 558–563. ISSN: 0166-2236. DOI: 10.1016/S0166-2236(02)02278-6.
- Minsky, Marvin and Seymour A Papert (1987). *Perceptrons: An Introduction to Computational Geometry*. Expanded Edition. Cambridge, Massachusetts: MIT Press. 308 pp. ISBN: 978-0-262-63111-2.
- Moorhouse, A.J. (2016). “Membrane Potential: Concepts”. In: *Encyclopedia of Cell Biology*. Ed. by Ralph A. Bradshaw and Philip D. Stahl. Waltham: Academic Press, pp. 218–236. ISBN: 978-0-12-394796-3. DOI: 10.1016/B978-0-12-394447-4.10027-6.
- Mundy, Andrew, James Knight, Terrence C. Stewart, and Steve B. Furber (2015). “An Efficient SpiNNaker Implementation of the Neural Engineering Framework”. In: *IJCNN*.
- Muthukrishnan, Nikesh, Farhad Maleki, Katie Ovens, Caroline Reinhold, Behzad Forghani, and Reza Forghani (2020). “Brief History of Artificial Intelligence”. In: *Neuroimaging Clinics of North America* 30.4, pp. 393–399. ISSN: 1052-5149. DOI: 10.1016/j.nic.2020.07.004.
- Neckar, A., S. Fok, B. V. Benjamin, Terrence C. Stewart, N. N. Oza, Aaron R. Voelker, Chris Elia-smith, R. Manohar, and Kwabena Boahen (2019). “Braindrop: A Mixed-Signal Neuromorphic Architecture With a Dynamical Systems-Based Programming Model”. In: *Proceedings of the IEEE* 107.1, pp. 144–164. ISSN: 0018-9219. DOI: 10.1109/JPROC.2018.2881432.
- Nernst, Walther (1888). “Zur Kinetik der in Lösung befindlichen Körper”. In: *Zeitschrift für physikalische Chemie* 2U.1, pp. 613–637. DOI: 10.1515/zpch-1888-0274.
- Neuman, C. P. and D. I. Schonbach (1974). “Discrete (Legendre) Orthogonal Polynomials—a Survey”. In: *International Journal for Numerical Methods in Engineering* 8.4, pp. 743–770. DOI: 10.1002/nme.1620080406.
- Nicola, Wilten and Claudia Clopath (2017). “Supervised Learning in Spiking Neural Networks with FORCE Training”. In: *Nature Communications* 8.1, p. 2208. ISSN: 2041-1723. DOI: 10.1038/s41467-017-01827-3.
- Nocedal, Jorge and Stephen J. Wright (2006). *Numerical Optimization*. Red. by Thomas V. Mikosch, Stephen M. Robinson, and Sidney I. Resnick. Second Edition. Springer Series in Operations Research and Financial Engineering. New York, New York, United States: Springer Science+Business Media. 664 pp. ISBN: 978-0-387-30303-1.

Bibliography

- Nowakowski, Richard S., Nancy L. Hayes, and M. David Egger (1992). “Competitive Interactions during Dendritic Growth: A Simple Stochastic Growth Algorithm”. In: *Brain Research* 576.1, pp. 152–156. ISSN: 0006-8993. DOI: 10.1016/0006-8993(92)90622-G.
- O’Reilly, Jill X., M. Marsel Mesulam, and Anna Christina Nobre (2008). “The Cerebellum Predicts the Timing of Perceptual Events”. In: *Journal of Neuroscience* 28.9, pp. 2252–2260. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.2742-07.2008.
- Okun, Michael, Pierre Yger, Stephan L. Marguet, Florian Gerard-Mercier, Andrea Benucci, Stefan Katzner, Laura Busse, Matteo Carandini, and Kenneth D. Harris (2012). “Population Rate Dynamics and Multineuron Firing Patterns in Sensory Cortex”. In: *Journal of Neuroscience* 32.48, pp. 17108–17119. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.1831-12.2012.
- Olsen, Shawn R., Dante S. Bortone, Hillel Adesnik, and Massimo Scanziani (2012). “Gain Control by Layer Six in Cortical Circuits of Vision”. In: *Nature* 483.7387, pp. 47–52. ISSN: 1476-4687. DOI: 10.1038/nature10835.
- Oppenheim, Alan V. and Ronald W. Schaffer (2009). *Discrete-Time Signal Processing*. Red. by Alan V. Oppenheim. Third Edition. Prentice Hall Signal Processing Series. London, United Kingdom: Prentice-Hall Inc. 1108 pp. ISBN: 978-0-13-198842-2.
- Painkras, Eustace, Luis Plana, Jim Garside, Sally Temple, Francesco Galluppi, Cameron Patterson, David R. Lester, Andrew D. Brown, and Steve B. Furber (2013). “SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation”. In: *Solid-State Circuits, IEEE Journal of* 48.8, pp. 1943–1953.
- Palkovits, Miklós, Pál Magyar, and János Szentágothai (1972). “Quantitative Histological Analysis of the Cerebellar Cortex in the Cat. IV. Mossy Fiber-Purkinje Cell Numerical Transfer”. In: *Brain Research* 45.1, pp. 15–29. ISSN: 0006-8993. DOI: 10.1016/0006-8993(72)90213-2.
- Parisien, Christopher, Charles H. Anderson, and Chris Eliasmith (2008). “Solving the Problem of Negative Synaptic Weights in Cortical Models”. In: *Neural Computation* 20, pp. 1473–1494.
- Pastalkova, Eva, Vladimir Itskov, Asohan Amarasingham, and György Buzsáki (2008). “Internally Generated Cell Assembly Sequences in the Rat Hippocampus”. In: *Science* 321.5894, pp. 1322–1327. ISSN: 0036-8075. DOI: 10.1126/science.1159775.
- Pintelon, R., P. Guillaume, Y. Rolain, J. Schoukens, and H. Van Hamme (1994). “Parametric Identification of Transfer Functions in the Frequency Domain—a Survey”. In: *IEEE Transactions on Automatic Control* 39.11, pp. 2245–2260. DOI: 10.1109/9.333769.
- Pirschel, Friederice and Jutta Kretzberg (2016). “Multiplexed Population Coding of Stimulus Properties by Leech Mechanosensory Cells”. In: *Journal of Neuroscience* 36.13, pp. 3636–3647. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.1753-15.2016.
- Poirazi, Panayiota, Terrence Brannon, and Bartlett W. Mel (2003). “Pyramidal Neuron as Two-Layer Neural Network”. In: *Neuron* 37.6, pp. 989–999. ISSN: 0896-6273. DOI: 10.1016/S0896-6273(03)00149-1.

- Polsky, Alon, Bartlett W Mel, and Jackie Schiller (2004). “Computational Subunits in Thin Dendrites of Pyramidal Cells”. In: *Nature Neuroscience* 7, p. 621. URL: <https://doi.org/10.1038/nn1253>.
- Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery (2007). *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3rd ed. New York, NY, USA: Cambridge University Press. ISBN: 978-0-521-88068-8.
- Purves, Dale, George J. Augustine, David Fitzpatrick, William C. Hall, Anthony-Samuel LaMantia, Richard D. Mooney, Michael L. Platt, and Leonard E. White, eds. (2017). *Neuroscience*. Sixth Edition. Cary, North Carolina, USA: Oxford University Press USA. 960 pp. ISBN: 978-1-60535-380-7.
- Ramón y Cajal, Santiago (1894). *Les nouvelles idées sur la structure du système nerveux: chez l'homme et chez les vertébrés*. Trans. by L Azoulay. Paris, France: C. Reinwald & Cie. 200 pp. URL: <https://archive.org/details/lesnouvellesid00ram>.
- (1909). *Histologie du système nerveux de l'homme & des vertébrés*. Trans. by L Azoulay. Vol. 1. 2 vols. Paris, France: A. Maloine. 986 pp. URL: https://archive.org/details/b2129592x_0001.
- Reichert, Heinrich (2000). *Neurobiologie*. zweite neubearbeitete und erweiterte Auflage. Stuttgart, Germany: Thieme. VII, 250p. ISBN: 978-3-13-745302-4.
- Richards, Blake A and Timothy P Lillicrap (2019). “Dendritic Solutions to the Credit Assignment Problem”. In: *Current Opinion in Neurobiology* 54, pp. 28–36. ISSN: 0959-4388. DOI: 10.1016/j.conb.2018.08.003.
- Richards, Blake A., Timothy P. Lillicrap, et al. (2019). “A Deep Learning Framework for Neuroscience”. In: *Nature Neuroscience* 22.11, pp. 1761–1770. ISSN: 1546-1726. DOI: 10.1038/s41593-019-0520-2.
- Rockafellar, R. Tyrrell (1993). “Lagrange Multipliers and Optimality”. In: *SIAM Review* 35.2, pp. 183–238. DOI: 10.1137/1035044.
- Rosenblatt, Frank (1958). “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.” In: *Psychological review* 65.6, p. 386.
- Rössert, Christian, Paul Dean, and John Porrill (2015). “At the Edge of Chaos: How Cerebellar Granular Layer Network Dynamics Can Provide the Basis for Temporal Filters”. In: *PLOS Computational Biology* 11.10, pp. 1–28. DOI: 10.1371/journal.pcbi.1004515.
- Roth, Arnd and Mark C. W. van Rossum (2009). “Modeling Synapses”. In: *Computational Modeling Methods for Neuroscientists*. Ed. by Erik De Schutter. The MIT Press, pp. 139–159.
- Sagara, Setsuo and Zhen-Yu Zhao (1989). “Recursive Identification of Transfer Function Matrix in Continuous Systems via Linear Integral Filter”. In: *International Journal of Control* 50.2, pp. 457–477. DOI: 10.1080/00207178908953377.

Bibliography

- Salinas, Emilio and LF Abbott (1994). “Vector Reconstruction from Firing Rates”. In: *Journal of computational neuroscience* 1.1-2, pp. 89–107.
- Salinas, Emilio and Peter Thier (2000). “Gain Modulation: A Major Computational Principle of the Central Nervous System”. In: *Neuron* 27.1, pp. 15–21. ISSN: 0896-6273. DOI: 10.1016/S0896-6273(00)00004-0.
- Sanathanan, Cok and Judith Koerner (1963). “Transfer Function Synthesis as a Ratio of Two Complex Polynomials”. In: *IEEE Transactions on Automatic Control* 8.1, pp. 56–58. DOI: 10.1109/TAC.1963.1105517.
- Sanger, Terence D., Okito Yamashita, and Mitsuo Kawato (2020). “Expansion Coding and Computation in the Cerebellum: 50 Years after the Marr–Albus Codon Theory”. In: *The Journal of Physiology* 598.5, pp. 913–928. DOI: 10.1113/JP278745.
- Schemmel, J., D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner (2010). “A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling”. In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 1947–1950. DOI: 10.1109/ISCAS.2010.5536970.
- Schemmel, Johannes, Laura Kriener, Paul Müller, and Karlheinz Meier (2017). “An Accelerated Analog Neuromorphic Hardware System Emulating NMDA- and Calcium-Based Non-Linear Dendrites”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2217–2226. DOI: 10.1109/IJCNN.2017.7966124.
- Seamans, Jeremy K., Natalia A. Gorelova, and Charles R. Yang (1997). “Contributions of Voltage-Gated Ca²⁺ Channels in the Proximal versus Distal Dendrites to Synaptic Integration in Prefrontal Cortical Neurons”. In: *Journal of Neuroscience* 17.15, pp. 5936–5948. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.17-15-05936.1997.
- Sejnowski, Terrence J, Patricia S Churchland, and J Anthony Movshon (2014). “Putting Big Data to Good Use in Neuroscience”. In: *Nature Neuroscience* 17.11, pp. 1440–1441. ISSN: 1546-1726. DOI: 10.1038/nn.3839.
- Shannon, C. E. (1949). “Communication in the Presence of Noise”. In: *Proceedings of the IRE* 37, pp. 10–21.
- Shepherd, Gordon and Steph Grillner, eds. (2012). *Handbook of Brain Microcircuits*. Oxford, UK. ISBN: 978-0-19-538988-3. URL: <http://oxfordmedicine.com/view/10.1093/med/9780195389883.001.0001/med-9780195389883>.
- Slawski, Martin and Matthias Hein (2013). “Non-Negative Least Squares for High-Dimensional Linear Models: Consistency and Sparse Recovery without Regularization”. In: *Electronic Journal of Statistics* 7 (none), pp. 3004–3056. DOI: 10.1214/13-EJS868.
- Sobolevsky, Alexander I., Michael P. Rosconi, and Eric Gouaux (2009). “X-Ray Structure, Symmetry and Mechanism of an AMPA-subtype Glutamate Receptor”. In: *Nature* 462.7274, pp. 745–756. ISSN: 1476-4687. DOI: 10.1038/nature08624.

- Spielman, Daniel (2012). “Spectral Graph Theory”. In: *Combinatorial Scientific Computing*. Ed. by Uwe Naumann and Olaf Schenk. Boca Raton, Florida, United States: CRC Press, pp. 495–524. ISBN: 978-1-4398-2735-2.
- Springer, Tom (1991). “Sliding FFT Computes Frequency Spectra in Real Time”. In: *Electronic Circuits, Systems and Standards*. Ed. by Ian Hickman. Newnes, pp. 40–46. ISBN: 978-0-7506-0068-2. DOI: 10.1016/B978-0-7506-0068-2.50016-X.
- Squire, William (1971). “A Simple Integral Method for System Identification”. In: *Mathematical Biosciences* 10.1, pp. 145–148. ISSN: 0025-5564. DOI: 10.1016/0025-5564(71)90055-1.
- Stellato, Bartolomeo, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd (2020). “OSQP: An Operator Splitting Solver for Quadratic Programs”. In: *Mathematical Programming Computation* 12.4, pp. 637–672. DOI: 10.1007/s12532-020-00179-2.
- Stevens, Charles F. (2000). “Models Are Common; Good Theories Are Scarce”. In: *Nature Neuroscience* 3.11, pp. 1177–1177. ISSN: 1546-1726. DOI: 10.1038/81451.
- Stewart, Terrence C., Trevor Bekolay, and Chris Eliasmith (2012). “Learning to Select Actions with Spiking Neurons in the Basal Ganglia”. In: *Frontiers in Decision Neuroscience* 6. DOI: 10.3389/fnins.2012.00002.
- Stimberg, Marcel, Romain Brette, and Dan FM Goodman (2019). “Brian 2, an Intuitive and Efficient Neural Simulator”. In: *eLife* 8. Ed. by Frances K Skinner, Ronald L Calabrese, Frances K Skinner, Fleur Zeldenrust, and Richard C Gerkin, e47314. ISSN: 2050-084X. DOI: 10.7554/eLife.47314.
- Stöckel, Andreas (2020). *Assorted Notes on Radial Basis Functions*. Waterloo, ON: Centre for Theoretical Neuroscience. DOI: 10.13140/RG.2.2.27177.62563/1.
- (2021a). *Constructing Dampened LTI Systems Generating Polynomial Bases*. URL: <https://arxiv.org/abs/2103.00051>.
- (2021b). *Discrete Function Bases and Convolutional Neural Networks*. URL: <https://arxiv.org/abs/2103.05609>.
- Stöckel, Andreas and Chris Eliasmith (2021). “Passive Nonlinear Dendritic Interactions as a Computational Resource in Spiking Neural Networks”. In: *Neural Computation* 33.1, pp. 96–128. DOI: 10.1162/neco_a_01338.
- Stöckel, Andreas, Terrence C. Stewart, and Chris Eliasmith (2020a). “A Biologically Plausible Spiking Neural Model of Eyeblink Conditioning in the Cerebellum”. In: *42nd Annual Meeting of the Cognitive Science Society*. Toronto, ON: Cognitive Science Society, pp. 1614–1620.
- (2020b). “Connecting Biological Detail with Neural Computation: Application to the Cerebellar Granule-Golgi Microcircuit”. In: *18th Annual Meeting of the International Conference on Cognitive Modelling*. Toronto, ON: Society for Mathematical Psychology.
- (2021). “Connecting Biological Detail with Neural Computation: Application to the Cerebellar Granule-Golgi Microcircuit”. In: *Topics in Cognitive Science* 13.3, pp. 515–533. DOI: 10.1111/tops.12536.

Bibliography

- Stöckel, Andreas, Aaron R. Voelker, and Chris Eliasmith (2017). *Point Neurons with Conductance-Based Synapses in the Neural Engineering Framework*. URL: <https://arxiv.org/abs/1710.07659>.
- (2018). “Nonlinear Synaptic Interaction as a Computational Resource in the Neural Engineering Framework”. In: *Cosyne Abstracts*. Denver USA. URL: http://cosyne.org/cosyne18/Cosyne2018_program_book.pdf.
- Strata, Piergiorgio and Robin Harvey (1999). “Dale’s Principle”. In: *Brain Research Bulletin* 50.5, pp. 349–350. ISSN: 0361-9230. DOI: 10.1016/S0361-9230(99)00100-8.
- Stringer, Carsen, Marius Pachitariu, Nicholas Steinmetz, Matteo Carandini, and Kenneth D. Harris (2019). “High-Dimensional Geometry of Population Responses in Visual Cortex”. In: *Nature* 571.7765, pp. 361–365. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1346-5.
- Strogatz, Steven H. (1994). *Nonlinear Dynamics and Chaos*. Studies in Nonlinearity. Westview Press. 498 pp. ISBN: 978-0-7382-0453-6.
- Stroustrup, Bjarne (2013). *The C++ Programming Language*. Fourth Edition. Upper Saddle River, New Jersey, United States: Pearson Education, Inc. 1346 pp. ISBN: 978-0-321-56384-2.
- Stuart, Greg and Nelson Spruston (1998). “Determinants of Voltage Attenuation in Neocortical Pyramidal Neuron Dendrites”. In: *Journal of Neuroscience* 18.10, pp. 3501–3510. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.18-10-03501.1998.
- Sullivan, Edith V (2010). “Cognitive Functions of the Cerebellum”. In: *Neuropsychology review* 20.3, pp. 227–228. ISSN: 1573-6660. DOI: 10.1007/s11065-010-9144-8.
- Sun, Ju (2016). “When Are Nonconvex Optimization Problems Not Scary?” PhD thesis. New York, New York, United States: Columbia University. 235 pp. DOI: 10.7916/D8251J7H.
- Sussillo, David and L.F. Abbott (2009). “Generating Coherent Patterns of Activity from Chaotic Neural Networks”. In: *Neuron* 63.4, pp. 544–557. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2009.07.018.
- Sutskever, Ilya, James Martens, George Dahl, and Geoffrey Hinton (2013). “On the Importance of Initialization and Momentum in Deep Learning”. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, pp. 1139–1147. URL: <http://proceedings.mlr.press/v28/sutskever13.html>.
- Thalmeier, Dominik, Marvin Uhlmann, Hilbert J. Kappen, and Raoul-Martin Memmesheimer (2016). “Learning Universal Computations with Spikes”. In: *PLOS Computational Biology* 12.6, pp. 1–29. DOI: 10.1371/journal.pcbi.1004895.
- Thorpe, Simon, Arnaud Delorme, and Rufin Van Rullen (2001). “Spike-Based Strategies for Rapid Processing”. In: *Neural Networks* 14.6, pp. 715–725. ISSN: 0893-6080. DOI: 10.1016/S0893-6080(01)00083-1.

- Tiganj, Zoran, Min Whan Jung, Jieun Kim, and Marc W Howard (2016). “Sequential Firing Codes for Time in Rodent Medial Prefrontal Cortex”. In: *Cerebral Cortex* 27.12, pp. 5663–5671. ISSN: 1047-3211. DOI: 10.1093/cercor/bhw336.
- Traub, Roger D. and Richard Miles (1991). *Neuronal Networks of the Hippocampus*. Cambridge, United Kingdom: Cambridge University Press. 281 pp. ISBN: 0-521-36481-7.
- Treves, Alessandro (1993). “Mean-Field Analysis of Neuronal Spike Dynamics”. In: 4.3, pp. 259–284. DOI: 10.1088/0954-898x_4_3_002.
- Tripp, Bryan (2009). “A Search For Principles of Basal Ganglia Function”. PhD thesis. University of Waterloo. URL: <http://uwspace.uwaterloo.ca/handle/10012/4179>.
- Tripp, Bryan and Chris Eliasmith (2007). “Neural Populations Can Induce Reliable Postsynaptic Currents without Observable Spike Rate Changes or Precise Spike Timing”. In: *Cerebral Cortex* 17, pp. 1830–1840. URL: <http://cercor.oxfordjournals.org/cgi/reprint/bhl092?>
- (2010). “Population Models of Temporal Differentiation”. In: *Neural Computation* 22.3, pp. 621–659. DOI: 10.1162/neco.2009.02-09-970.
- (2016). “Function Approximation in Inhibitory Networks”. In: *Neural Networks* 77, pp. 95–106. DOI: 10.1016/j.neunet.2016.01.010.
- Troy, John B. (2009). “Retinal Ganglion Cells: Receptive Fields”. In: *Encyclopedia of Neuroscience*. Ed. by Larry R. Squire. Oxford: Academic Press, pp. 219–223. ISBN: 978-0-08-045046-9. DOI: 10.1016/B978-008045046-9.00898-6.
- Tsoi, Ah Chung and Andrew Back (1997). “Discrete Time Recurrent Neural Network Architectures: A Unifying Review”. In: *Neurocomputing* 15.3, pp. 183–223. ISSN: 0925-2312. DOI: 10.1016/S0925-2312(97)00161-6.
- Vandenberghe, Lieven (2010). “The CVXOPT Linear and Quadratic Cone Program Solvers”. In: *Online: <http://cvxopt.org/documentation/coneprog.pdf>*.
- VandenBos, Gary R., ed. (2015). *APA Dictionary of Psychology*. Second Edition. Washington, DC, USA: American Psychological Association. 1204 pp. ISBN: 978-1-4338-1944-5. URL: <http://dx.doi.org/10.1037/14646-000>.
- Van Gelder, Tim (1998). “The Dynamical Hypothesis in Cognitive Science”. In: *Behavioral and Brain Sciences* 21.5, pp. 615–628. DOI: 10.1017/S0140525X98001733.
- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc. URL: <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>.

Bibliography

- Verhaegen, Michel and Vincent Verdult (2007). *Filtering and System Identification: A Least Squares Approach*. Cambridge, United Kingdom: Cambridge University Press. 405 pp. ISBN: 978-0-521-87512-7. URL: <https://www.cambridge.org/9780521875127>.
- Verkhratsky, Alexej and Christian Steinhäuser (2000). “Ion Channels in Glial Cells”. In: *Brain Research Reviews* 32.2, pp. 380–412. ISSN: 0165-0173. DOI: 10.1016/S0165-0173(99)00093-4.
- Voelker, Aaron R. (2019). “Dynamical Systems in Spiking Neuromorphic Hardware”. PhD thesis. Waterloo, ON: University of Waterloo. URL: <http://hdl.handle.net/10012/14625>.
- Voelker, Aaron R. and Chris Eliasmith (2018). “Improving Spiking Dynamical Networks: Accurate Delays, Higher-Order Synapses, and Time Cells”. In: *Neural Computation* 30.3, pp. 569–609. DOI: 10.1162/neco_a_01046.
- (2021). “Programming Neuromorphics Using the Neural Engineering Framework”. In: *Handbook of Neuroengineering*. Ed. by Nitish V. Thakor. Singapore: Springer Singapore, pp. 1–43. ISBN: 978-981-15-2848-4. DOI: 10.1007/978-981-15-2848-4_15-1.
- Voelker, Aaron R., Ivana Kajić, and Chris Eliasmith (2019). “Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks”. In: *Advances in Neural Information Processing Systems*.
- Von Neumann, John (1958). *The Computer and the Brain*. Silliman Memorial Lectures. New Haven, Colorado, United States: Yale University Press. URL: <https://archive.org/details/computerbrain00vonn>.
- Von Bartheld, Christopher S., Jami Bahney, and Suzanaerculano-Houzel (2016). “The Search for True Numbers of Neurons and Glial Cells in the Human Brain: A Review of 150 Years of Cell Counting”. In: *Journal of Comparative Neurology* 524.18, pp. 3865–3895. DOI: 10.1002/cne.24040.
- Vu, Eric T, Sunhee C Lee, and Franklin B Krasne (1993). “The Mechanism of Tonic Inhibition of Crayfish Escape Behavior: Distal Inhibition and Its Functional Significance”. In: *Journal of Neuroscience* 13.10, pp. 4379–4393.
- Waibel, A., T. Hanazawa, G. Hinton, K. Shikano, and K.J. Lang (1989). “Phoneme Recognition Using Time-Delay Neural Networks”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.3, pp. 328–339. DOI: 10.1109/29.21701.
- Watson, Andrew B. and Albert J. Ahumada Jr. (1983). *A Look at Motion in the Frequency Domain*. NASA Technical Memorandum 84352. Moffett Field, California, United States: Ames Research Center.
- Werbos, P.J. (1990). “Backpropagation through Time: What It Does and How to Do It”. In: *Proceedings of the IEEE* 78.10, pp. 1550–1560. DOI: 10.1109/5.58337.

- Weste, Neil H. E. and David Money Harris (2011). *CMOS VLSI Design: A Circuits and Systems Perspective*. Fourth Edition. Boston, Massachusetts, United States: Addison-Wesley. 838 pp. ISBN: 978-0-321-54774-3.
- Whitfield, Alan H. and N. Messali (1987). “Integral-Equation Approach to System Identification”. In: *International Journal of Control* 45.4, pp. 1431–1445. DOI: 10.1080/00207178708933819.
- Whitney, D.E. (1969). “More about Similarities between Runge-Kutta and Matrix Exponential Methods for Evaluating Transient Response”. In: *Proceedings of the IEEE* 57.11, pp. 2053–2054. ISSN: 1558-2256. DOI: 10.1109/PROC.1969.7453.
- Widrow, B. and M. E. Hoff (1960). “Adaptive Switching Circuits”. In: *1960 IRE WESCON Convention Record (Pt. 4)*. Western Electronic Show and Convention (WESCON), pp. 96–104.
- Wiener, Norbert (1948). *Cybernetics: Or Control and Communication in the Animal and the Machine*. First american edition. New York: John Wiley & Sons. 194 pp. ISBN: 978-0-262-23107-7.
- (1949). *Extrapolation, Interpolation, and Smoothing of Stationary Time Series: With Engineering Applications*. Cambridge, Massachusetts, United States: MIT Press. 163 pp. ISBN: 978-0-262-25719-0. DOI: 10.7551/mitpress/2946.001.0001.
- Williams, Stephen R. and Greg J. Stuart (2002). “Dependence of EPSP Efficacy on Synapse Location in Neocortical Pyramidal Neurons”. In: *Science* 295.5561, pp. 1907–1910. ISSN: 0036-8075. DOI: 10.1126/science.1067903.
- Yamins, Daniel L. K. and James J. DiCarlo (2016). “Using Goal-Driven Deep Learning Models to Understand Sensory Cortex”. In: *Nature Neuroscience* 19.3, pp. 356–365. ISSN: 1546-1726. DOI: 10.1038/nn.4244.
- Yan, Yexin et al. (2021). “Comparing Loihi with a SpiNNaker 2 Prototype on Low-Latency Keyword Spotting and Adaptive Robotic Control”. In: *Neuromorphic Computing and Engineering*. DOI: 10.1088/2634-4386/abf150.
- Yavuz, Esin, James Turner, and Thomas Nowotny (2016). “GeNN: A Code Generation Framework for Accelerated Brain Simulations”. In: *Scientific Reports* 6.
- Young, Nicholas (1988). *An Introduction to Hilbert Space*. Cambridge Mathematical Textbooks. Cambridge, United Kingdom: Cambridge University Press. 239 pp. ISBN: 978-0-521-33717-5. DOI: 10.1017/CB09781139172011.
- Yuste, Rafael (2015). “From the Neuron Doctrine to Neural Networks”. In: *Nature Reviews Neuroscience* 16, p. 487. DOI: 10.1038/nrn3962.
- Zhang, Lin, Aaron Carpenter, Berkehan Ciftcioglu, Alok Garg, Michael Huang, and Hui Wu (2008). “Injection-Locked Clocking: A Low-Power Clock Distribution Scheme for High-Performance Microprocessors”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 16.9, pp. 1251–1256. DOI: 10.1109/TVLSI.2008.2000976.

Bibliography

- Zilany, Muhammad S. A. and Ian C. Bruce (2006). “Modeling Auditory-Nerve Responses for High Sound Pressure Levels in the Normal and Impaired Auditory Periphery”. In: *The Journal of the Acoustical Society of America* 120.3, pp. 1446–1466. DOI: 10.1121/1.2225512.
- Zou, Hui and Trevor Hastie (2005). “Regularization and Variable Selection via the Elastic Net”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.2, pp. 301–320. DOI: 10.1111/j.1467-9868.2005.00503.x.

Appendix A

Mathematical Detail

A.1 Proofs for Chapter 3

In this section, we provide proofs for the claims made in Chapter 3, where we discussed theoretical aspects of neural networks in general, and dendritic computation in particular.

A.1.1 Proof of Theorem 3.1

In Section 3.1.1 we claimed that there are always functions f that cannot be approximated using an additive network with a fixed nonlinearity σ . We prove this by constructing an f that is constant along the sides of a rectangle, but maps onto a different value on the inside.

Theorem 3.1. *Let $\ell > 1$, $\mathbb{X} \subset \mathbb{R}^\ell$ and $\mathbb{Y} \subset \mathbb{R}$ be compact sets of full dimensionality, and σ, f, f_i be continuous. For any fixed $\sigma : \mathbb{R} \rightarrow \mathbb{Y}$, there always exist $f : \mathbb{X} \rightarrow \mathbb{Y}$ such that there is no $f_1, \dots, f_\ell : \mathbb{R} \rightarrow \mathbb{R}$ with $f(x_1, \dots, x_\ell) = \sigma(f_1(x_1) + \dots + f_\ell(x_\ell))$ for all $(x_1, \dots, x_\ell) \in \mathbb{X}$.*

Proof. Since \mathbb{X} is a compact set with dimensionality $\dim(\mathbb{X}) = \ell$, there must be a rectangular subset of the following form $[a_0, a_1] \times [b_0, b_1] \times \{c_1\} \times \dots \times \{c_{\ell-2}\} \subseteq \mathbb{X}$. Now, consider an f with

$$\begin{aligned} f(a_0, x_2, c_1, \dots, c_{\ell-2}) &= d \quad \text{for } x_1 \in [a_0, a_1], \\ \text{and } f(x_1, b_0, c_1, \dots, c_{\ell-2}) &= d \quad \text{for } x_2 \in [b_0, b_1], \\ \text{and } f(x_1, x_2, c_1, \dots, c_{\ell-2}) &\neq d \quad \text{for } x_1 \in (a_0, a_1] \text{ and } x_2 \in (b_0, b_1]. \end{aligned}$$

Since σ is continuous, the functions $f_1(x_1), f_2(x_2)$ must be constant for all $x_1 \in [a_0, a_1]$ and $x_2 \in [b_0, b_1]$, otherwise the first two conditions cannot be met. However, with f_1 and f_2 being constant over the indicated intervals, the second condition cannot be met. Hence, f is a function that cannot be computed using additive networks, independent of σ . \square

A.1.2 Proof of Theorem 3.2

Minsky et al. (1987, Chapter 2) note that the Perceptron cannot compute XOR. This notion can be extended to the more general “additive networks” discussed in Section 3.1, as well as a weaker notion of the XOR problem (Definition 3.1). In particular, we claimed the following:

Theorem 3.2. *Let σ be monotonic. Then, an additive network of the form $\varphi(x_1, x_2) = \sigma(f_1(x_1) + f_2(x_2))$ cannot solve the weak XOR problem.*

Appendix A. Mathematical Detail

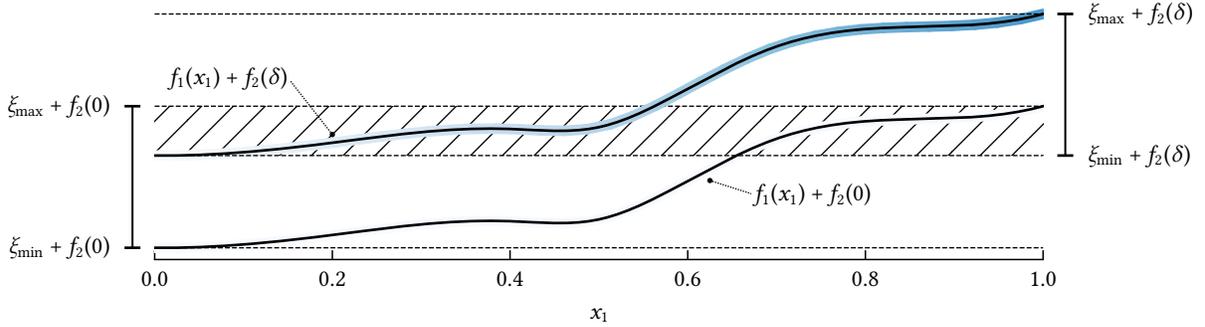


Figure A.1: Visualisation of the proof of Theorem 3.3. We can find a δ such that the intervals spanned by $f_1(x_1) + f_2(0)$ and $f_1(x_2) + f_2(\delta)$ overlap (hatched region). This should not be possible, given that $\sigma(f(x_1) + f(0)) = 0$ and $\sigma(f_1(x_1) + f_2(\delta)) \neq 0$. Coloured background corresponds to the desired product $x_1 x_2 = \sigma(f_1(x_1) + f_2(x_2))$. This colour should be the same along the horizontals of this diagram.

Proof. Suppose $\varphi(x_1, x_2)$ could solve the weak XOR problem. Substitute $a'_0 = f_1(a_0)$, $a'_1 = f_1(a_1)$, $b'_0 = f_2(b_0)$, $b'_1 = f_2(b_1)$, where a_0, a_1, b_0, b_1 are from Definition 3.1. Further expanding this definition by substituting in our additive network $\varphi(x_1, x_2)$ we obtain

$$\begin{aligned} & (\sigma(a'_0 + b'_0) < \sigma(a'_0 + b'_1)) \wedge (\sigma(a'_1 + b'_1) < \sigma(a'_1 + b'_0)) \\ & \wedge (\sigma(a'_0 + b'_0) < \sigma(a'_1 + b'_0)) \wedge (\sigma(a'_1 + b'_1) < \sigma(a'_0 + b'_1)) . \end{aligned}$$

Assume without loss of generality that σ is monotonically increasing. Then, the first line implies $(b'_0 < b'_1) \wedge (b'_0 > b'_1)$ and the second line $(a'_0 < a'_1) \wedge (a'_0 > a'_1)$. \neq \square

A.1.3 Proof of Theorem 3.3

Additive networks are capable of solving the weak XOR problem if we allow a nonmonotonic neural nonlinearity σ . However, as we already saw in Theorem 3.1, being able to freely choose σ does not mean that we can universally approximate any function. An example we gave in Theorem 3.3 is that of multiplication. While this is merely a special case of the kind of function used to prove Theorem 3.1, we provide an alternative proof below.

Theorem 3.3. *There are no continuous, real-valued functions f_1, f_2, σ such that $\sigma(f_1(x_1) + f_2(x_2)) = x_1 x_2$ for all $(x_1, x_2) \in [0, 1]^2$.*

Proof. Assume that continuous f_1, f_2, σ exist such that $\sigma(f_1(x_1) + f_2(x_2)) = x_1 x_2$. Let $\xi_{\min} \neq \xi_{\max}$ be the extrema of f_1 over $[0, 1]$. These extrema exist because f_1 is continuous on a compact set; hence, by the Heine-Cantor theorem, f_1 is uniformly continuous and not unbounded.

Consider $x_2 = 0$ and $x_1 \in [0, 1]$. The expression $f_1(x_1) + f_2(0)$ covers the interval $[\xi_{\min} + f_2(0), \xi_{\max} + f_2(0)]$. Since f_2 is continuous, there must be, according to the epsilon-delta definition of continuity, a $\delta \neq 0$ such that $f_2(\delta) - f_2(0) < \xi_{\max} - \xi_{\min}$. Hence, as illustrated in Figure A.1,

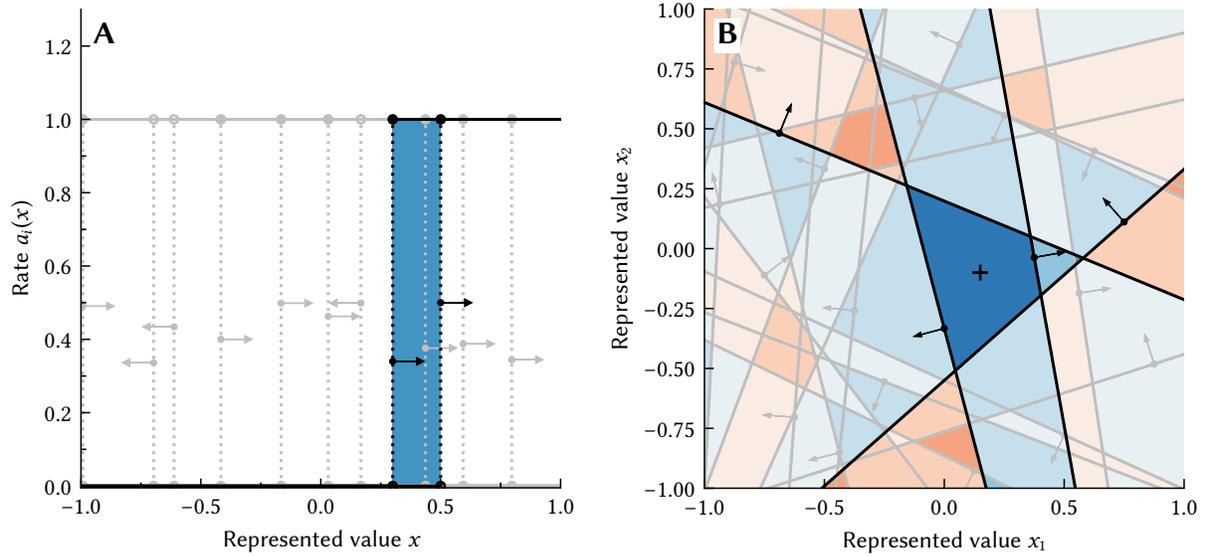


Figure A.2: Illustration of Theorem 3.4. **(A)** 1D ensemble with step activation function. Lines are tuning curves, black arrows the encoder. For $n \rightarrow \infty$ neurons, every point on the x -axis is enclosed by two neighbouring tuning-curves. Subtracting and scaling these two tuning curves appropriately, we can assign a value $f(x)$ to all $x \in [0, 1]$. For example, the two highlighted tuning curves, enclose a small region on the x -axis (blue highlight). **(B)** 2D ensemble with step activation. Lines are the intercept of an individual neuron, arrows the encoder. The tuning curves can be linearly combined such that any convex region in the space is assigned a relatively large value. The depicted weighting encloses the region marked by the cross. Blue corresponds to positive, red to negative decoded values. The highlighted intercepts correspond to the neurons with the largest weights.

the following two sets are not disjoint

$$\{f_1(x_1) + f_2(0) \mid x_1 \in (0, 1]\} \cap \{f_1(x_1) + f_2(\delta) \mid x_1 \in (0, 1]\} \neq \emptyset.$$

But, according to our assumption, it should also hold that

$$\begin{aligned} \sigma(f_1(x_1) + f_2(0)) &= x_1 \cdot 0 = 0 \text{ for all } x_1 \in (0, 1] \\ \text{and } \sigma(f_1(x_1) + f_2(\delta)) &= x_1 \cdot \delta \neq 0 \text{ for all } x_1 \in (0, 1]. \end{aligned}$$

However, since the values passed to σ overlap in both cases, this is a contradiction. $\not\equiv$ □

A.1.4 Discussion of Theorem 3.4

Theorem 3.4 claims that a single NEF ensemble, i.e., a multi-layer network with a single hidden layer, is a universal function approximator. More precisely, we stated the following:

Theorem 3.4. Let $\ell \geq 1$, and $f : \mathbb{B}^\ell \rightarrow \mathbb{R}$ be a continuous function mapping from the ℓ -dimensional unit ball onto \mathbb{R} . Furthermore, let $\sigma(\xi) = \max\{0, \xi\}$, \mathbf{e}_i be sampled uniformly from

the unit-sphere \mathbb{S}^ℓ , and α_i and β_i be sampled such that $-\beta_i/\alpha_i$ is uniformly distributed between $[-1, 1]$ and $\alpha_i + \beta_i$ is uniform over $(0, 1]$. There exist $d_i \in \mathbb{R}$ such that

$$f(\mathbf{x}) = \lim_{n \rightarrow \infty} \sum_{i=1}^n d_i \sigma(\alpha_i \langle \mathbf{e}_i, \mathbf{x} \rangle + \beta_i) \quad \text{for all } \mathbf{x} \in \mathbb{B}^\ell. \quad (3.2)$$

This universality of two-layer networks of this form is a well known fact, and has, in a more general form, first been proved by Hornik et al. (1989). In the terminology of the authors, the above equation describes a Σ -network, and σ is a “squashing function”. A notable difference to our theorem is that Hornik et al., define $\alpha_i \langle \mathbf{e}_i, \mathbf{x} \rangle + \beta_i$ to be all possible affine functions mapping from \mathbb{R}^ℓ onto \mathbb{R} , while we limit the affine functions to those required to cover the unit-ball \mathbb{B}^ℓ .

Instead of providing a complete proof, we would like to suggest geometrically why this theorem holds. Consider the case where σ is a step-function, i.e., $\sigma(\xi) = 1$ if $\xi > 0$, otherwise $\sigma(\xi) = 0$. In the case of a one-dimensional network, and as is depicted in Figure A.2A, it is apparent that as the number of neurons n goes to infinity, we can enclose any point on the x -axis with arbitrary precision by subtracting two appropriate neuron tuning curves with equal encoder. By scaling the two tuning curves, we can hence assign a value $f(x)$ to each represented value x .

Similarly, for higher-dimensional representations, we can linearly combine tuning curves such that any convex region enclosed by the neural intercepts is assigned an arbitrary value $f(\mathbf{x})$, while all other regions are assigned values close to zero (cf. Figure A.2B). As we increase the number of neurons, and if the conditions stated in the theorem are met, the enclosed regions converge to individual points. We can approximate any function $f(\mathbf{x})$ by scaling and summing the individual decodings.

A.1.5 Proof of Theorem 3.6

Back in Section 3.3.2 we claimed that the feedback matrix of the n -LIF dynamical system is stable and does not possess any oscillatory behaviour. To prove this, we first show the following lemma concerning the Laplacian quadratic form (cf. Spielman, 2012, Section 18.3.2):

Lemma A.1. *Let \mathbf{L} be the Laplacian of a positively weighted connected graph with weights c_{ij} and edges E . Then $\mathbf{x} = \alpha \mathbf{1}$ with $\alpha \in \mathbb{R}$ is the only root (besides $\mathbf{x} = \mathbf{0}$) of the Laplacian quadratic form*

$$\mathbf{x}^T \mathbf{L} \mathbf{x} = \sum_{(i,j) \in E} c_{ij} (x_i - x_j)^2.$$

Proof. Because of all c_{ij} being positive, the quadratic form is clearly nonnegative. In other words, \mathbf{L} is positive semidefinite and all its eigenvalues are nonnegative. Since \mathbf{L} is symmetric, its eigen-decomposition is $\mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^T$, where \mathbf{Q} is orthogonal. Let $\mathbf{y} = \mathbf{Q}^T \mathbf{x}$. It holds

$$\mathbf{x}^T \mathbf{L} \mathbf{x} = \mathbf{y}^T \mathbf{\Lambda} \mathbf{y} = \sum_{i=1}^n y_i^2 \lambda_i = 0 \iff \mathbf{\Lambda} \mathbf{y} = \mathbf{0} \text{ since } \lambda_i \geq 0.$$

Since \mathbf{Q} is orthogonal, it must be full-rank. This implies that its null-space only contains the zero-vector. Correspondingly, $\mathbf{x}^T \mathbf{Q} \mathbf{A} \mathbf{Q}^T \mathbf{x} = 0$ exactly if $\mathbf{Q} \mathbf{A} \mathbf{Q}^T \mathbf{x} = \mathbf{L} \mathbf{x} = \mathbf{0}$, or, put differently, \mathbf{x} is in the null-space of \mathbf{L} .

By Theorem 13.1.1 in Godsil and Royle (2001, Chapter 13), the multiplicity of the eigenvalue zero in the Laplacian \mathbf{L} is one, with eigenvector $\mathbf{1}$. Hence, the null-space of \mathbf{L} is $\alpha \mathbf{1}$. \square

Equipped with this Lemma, we can now show the original theorem.

Theorem 3.6. *Consider an n -LIF neuron with at least one static conductance-based channel and any input vector \mathbf{g} with nonnegative entries for conductance-based input channels. In this case, the feedback matrix $\mathbf{A}[\mathbf{g}] \text{diag}(\mathbf{C}_m)^{-1}$ is negative definite for any constant input vector \mathbf{g} .*

Proof. The feedback matrix $\mathbf{A}[\mathbf{g}] \text{diag}(\mathbf{C}_m)^{-1}$ has the following form:

$$\mathbf{A}[\mathbf{g}] \text{diag}(\mathbf{C}_m)^{-1} = -(\mathbf{L} + \text{diag}(\mathbf{a}' + \mathbf{A}' \mathbf{g})) \text{diag}(\mathbf{C}_m)^{-1} = -(\mathbf{L} + \text{diag}(\boldsymbol{\epsilon})) \text{diag}(\mathbf{C}_m)^{-1},$$

where \mathbf{L} is the graph Laplacian, \mathbf{C}_m is a positive vector of membrane capacitances, and $\boldsymbol{\epsilon}$ is, by the conditions specified in the theorem (i.e., there being at least one conductance-based channel), a nonnegative vector with $\boldsymbol{\epsilon} \neq \mathbf{0}$. Eliminating the minus sign, the following must hold to ensure negative definiteness:

$$\mathbf{x}^T \mathbf{L}' \mathbf{x} = \mathbf{x}^T (\mathbf{L} + \text{diag}(\boldsymbol{\epsilon})) \text{diag}(\mathbf{C}_m)^{-1} \mathbf{x} > 0 \quad \text{for all } \mathbf{x} \neq \mathbf{0}.$$

Note that multiplication with $\text{diag}(\mathbf{C}_m)^{-1}$ (i.e., scaling each row of the Laplacian by the inverse of the corresponding C_m) turns the undirected graph into a directed graph. However, we can still expand this expression using the Laplacian quadratic form (cf. Spielman, 2012, Section 18.3.5). Again, let E be the set of edges and c_{ij} the connection weights. We get

$$\begin{aligned} \mathbf{x}^T \mathbf{L}' \mathbf{x} &= \mathbf{x}^T (\mathbf{L} + \text{diag}(\boldsymbol{\epsilon})) \text{diag}(\mathbf{C}_m)^{-1} \mathbf{x} = \mathbf{x}^T \mathbf{L} \text{diag}(\mathbf{C}_m)^{-1} \mathbf{x} + \mathbf{x}^T \text{diag}(\boldsymbol{\epsilon}) \text{diag}(\mathbf{C}_m)^{-1} \mathbf{x} \\ &= \sum_{(i,j) \in E} c_{ij} C_{m,i}^{-1} (x_i - x_j)^2 + \sum_{i=1}^n \epsilon_i C_{m,i}^{-1} x_i^2. \end{aligned}$$

This expression is obviously nonnegative due to c_{ij} , $C_{m,i}^{-1}$ being positive and ϵ_i being nonnegative. However, to show strict positivity, and correspondingly, negative definiteness of our original expression, consider the conditions under which $\mathbf{x}^T \mathbf{L}' \mathbf{x}$ could be zero:

- (i) The sum over $c_{ij} C_{m,i}^{-1} (x_i - x_j)^2$ is zero exactly if there are no edges in the graph (since the graph is connected, this is only possible for $n = 1$), or, under the conditions listed in Lemma A.1, that is $\mathbf{x} = \alpha \mathbf{1}$, or, put differently, $x_i = x_j$ for all i, j .
- (ii) The sum over $\epsilon_i C_{m,i}^{-1} x_i^2$ is zero exactly if $\epsilon_i \neq 0$ implies that $x_i = 0$.

Appendix A. Mathematical Detail

Crucially, Lemma A.1 still applies in (i) despite the multiplication with $C_{m,i}^{-1}$. This is because $C_{m,i}^{-1}$ and c_{ij} are strictly positive, and we know that the only way for $c_{ij}(x_i - x_j)^2$ to evaluate to zero is that all summands evaluate to zero.

To complete the proof, first consider $n = 1$, as suggested by (i). In this case, since $\mathbf{x} \neq \mathbf{0}$ it follows $x_1 \neq 0$. Furthermore $\varepsilon_1 > 0$. Hence, $\varepsilon_i x_i^2 > 0$ and $\mathbf{xL}'\mathbf{x}^T$ is strictly positive.

Now assume that $n > 1$. Since $\mathbf{x} \neq \mathbf{0}$, and at least one ε_i is non-zero, this, by (ii), implies that the corresponding $x_i = 0$. By (i), $\mathbf{xL}'\mathbf{x}^T$ can only evaluate to zero if $\mathbf{x} = \mathbf{0}$, which we excluded. Hence, $\mathbf{xL}'\mathbf{x}^T > 0$ for all $\mathbf{x} \neq \mathbf{0}$ and \mathbf{L}' is positive definite. This implies that all eigenvalues of \mathbf{L}' are non-zero and that \mathbf{L}' is non-singular. Our original claim follows trivially. \square

A.1.6 Proof of Theorem 3.7

We stated in Section 3.3.4 that the dendritic nonlinearity H of an n -LIF neuron only contains product terms between the input channels of different compartments, but not between input channels of the same compartment. In particular, we claimed the following:

Theorem 3.7. *Consider an n -LIF neuron with ℓ branches, where each compartment is connected to k unique conductance- and k unique current-based input channels. We denote these inputs as g_j^i and J_j^i , where i and j are the compartment and input channel indices, respectively. Furthermore, arrange the compartment indices such that $i_{m-1} + 1, \dots, i_m$ belong to the same branch, where m with $1 \leq m \leq \ell$ is the branch index. Then, the somatic current model H has the following form*

$$H_0(g_1^1, \dots, g_k^1, J_1^1, \dots, J_k^1) + \frac{H_1^B(g_1^2, \dots, g_k^1, J_1^2, \dots, J_k^1)}{H_1^A(g_1^2, \dots, g_k^1)} + \dots + \frac{H_\ell^B(g_1^{i_{\ell-1}+1}, \dots, g_k^{i_\ell}, J_1^{i_{\ell-1}+1}, \dots, J_k^{i_\ell})}{H_\ell^A(g_1^{i_{\ell-1}+1}, \dots, g_k^{i_\ell})},$$

where (A) H_0 is an affine function over all inputs injected into the somatic compartment, (B) the H_m^A are nonnegative affine functions of product terms between conductance-based inputs belonging to different dendritic compartments, and (C) the H_m^B are affine functions of product terms between conductance-based inputs belonging to different dendritic compartments, and at most one dendritic current-based input per product term.

Proof. We first prove claim (A), followed by both (B) and (C). However, to do so, we first need to lay out the problem at hand in more detail. First, note that all ℓ branches of the neuron, including the soma itself, form block matrices $\tilde{C}_m[\mathbf{g}]$ within the reduced system matrix $\tilde{\mathbf{A}}[\mathbf{g}]$. In other words, the reduced system matrix and its inverse are given as

$$\tilde{\mathbf{A}}[\mathbf{g}] = \begin{bmatrix} 1 & 0 & \dots & \dots & 0 \\ 0 & \tilde{C}_1[\mathbf{g}] & \dots & \dots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & \tilde{C}_\ell[\mathbf{g}] \end{bmatrix} \iff \tilde{\mathbf{A}}[\mathbf{g}]^{-1} = \begin{bmatrix} 1 & 0 & \dots & \dots & 0 \\ 0 & \tilde{C}_1[\mathbf{g}]^{-1} & \dots & \dots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & \tilde{C}_\ell[\mathbf{g}]^{-1} \end{bmatrix}, \quad (\text{A.1})$$

To simplify writing out the block matrix $\tilde{\mathbf{C}}_\ell[\mathbf{g}]$, let $r = i_{m-1} + 1$ be the index of the first compartment in each branch, and $s = i_m$ be the index of the last compartment in each branch. Each block matrix is of the form

$$\tilde{\mathbf{C}}_m[\mathbf{g}] = \begin{bmatrix} d_r[\mathbf{g}] & -c_{r,r+1} & \dots & -c_{r,s} \\ -c_{r+1,r} & \dots & \dots & \vdots \\ \vdots & \dots & \dots & -c_{s-1,r} \\ -c_{s,r} & \dots & -c_{s,r-1} & d_s[\mathbf{g}] \end{bmatrix},$$

where, by construction of $\tilde{\mathbf{A}}[\mathbf{g}]$, each diagonal element $d_i[\mathbf{g}]$ is an affine function in the conductance-based input channels. At the same time, each entry in the input vector $(\tilde{\mathbf{b}}[\mathbf{g}])_i$ is an affine function in both the conductance- and current-based input channels targeting this channel. We have

$$d_i[\mathbf{g}] = \tilde{a}_0^i + \sum_{j=1}^k \tilde{a}_j^i g_j^i, \quad (\tilde{\mathbf{b}}[\mathbf{g}])_i = \tilde{b}_0^i + \sum_{j=1}^k \tilde{b}_j^i g_j^i + \sum_{k=1}^k \tilde{c}_j^i J_j^i,$$

where $\tilde{a}_j^i, \tilde{b}_j^i, \tilde{c}_j^i$ are the corresponding system matrix entries. Finally, recall eq. (3.27), i.e., the mapping between the input \mathbf{g} and the predicted somatic current:

$$H(\mathbf{g}) \approx \sum_{i=1}^n \tilde{c}_i(\tilde{v}_i^{\text{eq}} - \bar{v}), \quad \text{where } \tilde{\mathbf{v}}^{\text{eq}} = -\tilde{\mathbf{A}}[\mathbf{g}]^{-1} \tilde{\mathbf{b}}[\mathbf{g}]. \quad (\text{A.2})$$

Claim (A). The first claim (i.e., the structure of the affine function H_0) trivially follows from multiplying the inverse of the first ‘‘somatic block’’ of the reduced system matrix $\tilde{\mathbf{A}}[\mathbf{g}]$ (i.e., the identity) with the first entry of the reduced input vector $(\tilde{\mathbf{b}}[\mathbf{g}])_1$. This results in first entry of $\tilde{\mathbf{v}}^{\text{eq}}$; further combining this with eq. (A.2) yields an affine function $H_0(g_1^1, \dots, g_k^1, J_1^1, \dots, J_k^1)$.

Claims (B) and (C). The last two claims, i.e., the somatic current model H being a sum of rational functions with the given structural constraints, can be obtained by systematically inverting the individual block matrices $\tilde{\mathbf{C}}_m[\mathbf{g}]$.

In particular, note that the sum-of-rational-functions structure directly follows from plugging $\tilde{\mathbf{A}}[\mathbf{g}]^{-1}$ from eq. (A.1) into eq. (A.2). As we will become apparent in a moment, the product between the inverted block matrix $\tilde{\mathbf{C}}_m[\mathbf{g}]$ for branch m and the corresponding section of the input vector $\tilde{\mathbf{b}}[\mathbf{g}]$ form the individual rational functions. By construction, this rational function can only depend on the input channels targeting the branch m . That is, each sum term m can only depend on the variables $g_1^r, \dots, g_k^s, J_1^r, \dots, J_k^s$, where r, s are as defined above.

To see that the product between the inverted block matrix and the input vector forms a rational function, remember that the inverse of a matrix can be written as its adjoint scaled by the inverse of its determinant (e.g., Hefferon, 2020, Theorem 1.9, p. 366):

$$\tilde{\mathbf{C}}_m[\mathbf{g}]^{-1} = \frac{1}{\det(\tilde{\mathbf{C}}_m[\mathbf{g}])} \text{adj}(\tilde{\mathbf{C}}_m[\mathbf{g}]).$$

Appendix A. Mathematical Detail

Crucially, the determinant fully determines the denominator H_m^A of each rational function. The division by the determinant is the only division in each product between the inverted block-matrix $\tilde{C}_m[\mathbf{g}]^{-1}$ and the corresponding portion of the input vector $\tilde{\mathbf{b}}[\mathbf{g}]$.

In addition to the properties listed above, claim (B) states that H_m^A only depends on the conductance-based inputs, and that it only contains products between inputs targeting different compartments. The first statement follows from $\tilde{A}[\mathbf{g}]$ only containing conductance-based inputs by construction. The second statement follows from the permutation expansion of the determinant (e.g., Hefferon, 2020, Section 4.I.3, p. 337), we have

$$H_m^A(g_1^I, \dots, g_k^J) = \det(\tilde{C}_m[\mathbf{g}]) = \sum_{\pi \in \mathbb{P}} \text{sgn}(\pi) \prod_{i=1}^n (\tilde{C}_m[\mathbf{g}])_{i, \pi(i)+1}, \quad \text{where } \mathbb{P} = S(1, \dots, n-1).$$

Here, $S(X)$ denotes the set of permutations of the set X , and $\text{sgn}(\pi) \in \{-1, 1\}$ is the ‘‘signum’’ of the permutation π . Importantly, each product in the determinant only contains each diagonal element $d_i[\mathbf{g}]$ exactly once and there are no product-terms between inputs targeting the same compartment i .

Finally, claim (C) states that the numerators H_m^B only contains product terms between inputs targeting different compartments, and that each product term features at most one current-based input. The second statement is a result of $\tilde{A}[\mathbf{g}]$ only depending on conductance-based inputs, and all current-based inputs only influencing $\tilde{\mathbf{b}}[\mathbf{g}]$. Multiplications with current-based inputs are a result of the final matrix-vector product between $\tilde{A}^{-1}[\mathbf{g}]^{-1}$ and the input matrix $\tilde{\mathbf{b}}[\mathbf{g}]$ and there can be only one current-based input per product term.

The first statement follows from the structure of adjoint. The numerator H_m^B is determined by the matrix product between the adjoint of $\tilde{C}_m[\mathbf{g}]$ and the corresponding portion of $\tilde{\mathbf{b}}[\mathbf{g}]$. Notably, the adjoint of a matrix A is a matrix of determinants of A where the k th row and the ℓ th column have been deleted:

$$(\text{adj}(\tilde{C}_m[\mathbf{g}]))_{k\ell} = \sum_{\pi \in \mathbb{P}} \text{sgn}(\pi) \prod_{\substack{i=1 \\ i \neq k}}^n (\tilde{C}_m[\mathbf{g}])_{i, \pi(i)+1}, \quad \text{where } \mathbb{P} = S(1, \dots, \ell-1, \ell+1, n-1).$$

Hence, the entry $(\text{adj}(\tilde{C}_m[\mathbf{g}]))_{k\ell}$ neither contains the term $d_k[\mathbf{g}]$, nor $d_\ell[\mathbf{g}]$, and the product-terms formed by computing the matrix-vector product between $\text{adj}(\tilde{C}_m[\mathbf{g}])$ and $\tilde{\mathbf{b}}[\mathbf{g}]$ contain each input channel at most once. This concludes our proof. \square

A.1.7 Proof of Theorem 3.8

We noted in Section 3.3.4 that the two-compartment LIF nonlinearity cannot be used to solve the weak XOR problem. More precisely, we stated the following:

Theorem 3.8. *Let $g_E(x_1, x_2) = g_E^1(x_1) + g_E^2(x_2)$ and $g_I(x_1, x_2) = g_I^1(x_1) + g_I^2(x_2)$ be nonnegative functions. Furthermore, let $a_0 > 0$ and $a_1, a_2, b_0, b_1, b_2 \geq 0$. The two-compartment LIF nonlinearity*

$$\varphi(x_1, x_2) = H(g_E(x_1, x_2), g_I(x_1, x_2)) = \frac{b_0 + b_1 g_E(x_1, x_2) - b_2 g_I(x_1, x_2)}{a_0 + a_1 g_E(x_1, x_2) + a_2 g_I(x_1, x_2)}$$

cannot be used to solve the weak XOR problem (see Definition 3.1).

The proof for this is analogous to the proof that additive networks cannot solve the XOR problem (cf. Appendix A.1.2). Particularly, we rely on the denominator being strictly positive.

Proof. For $b_0 \neq 0$, H as given in the above theorem can be reparametrised to H' as follows

$$H(g_E, g_I) = H' \left(\frac{b_1 g_E}{|b_0|}, \frac{b_2 g_I}{|b_0|} \right) = H'(x, y) = \frac{\pm 1 + x - y}{c_0 + c_1 x + c_2 y},$$

where $c_0 > 0$ and $c_1, c_2, x, y \geq 0$. Assume that $\varphi(x, y) = H'(x, y)$ can solve the weak XOR problem. Since the denominator in the above nonlinearity is strictly positive, we can safely cross-multiply with the denominator across the inequalities and apply the above definition

$$\begin{aligned} & \left(0 < x_0 y_0 c_1 + x_0 y_0 c_2 - x_0 y_1 c_1 - x_0 y_1 c_2 + y_0 c_0 \pm y_0 c_2 - y_1 c_0 \mp y_1 c_2 \right) \\ & \wedge \left(0 < -x_1 y_0 c_1 - x_1 y_0 c_2 + x_1 y_1 c_1 + x_1 y_1 c_2 - y_0 c_0 \mp y_0 c_2 + y_1 c_0 \pm y_1 c_2 \right) \\ & \wedge \left(0 < -x_0 y_0 c_1 - x_0 y_0 c_2 + x_1 y_0 c_1 + x_1 y_0 c_2 - x_0 c_0 \pm x_0 c_1 + x_1 c_0 \mp x_1 c_1 \right) \\ & \wedge \left(0 < x_0 y_1 c_1 + x_0 y_1 c_2 - x_1 y_1 c_1 - x_1 y_1 c_2 + x_0 c_0 \mp x_0 c_1 - x_1 c_0 \pm x_1 c_1 \right). \end{aligned}$$

This can be simplified to

$$\begin{aligned} & \left(0 < ((c_1 + c_2)x_0 \pm c_2 + c_0)(y_0 - y_1) \right) \wedge \left(0 < -((c_1 + c_2)x_1 \pm c_2 + c_0)(y_0 - y_1) \right) \\ & \wedge \left(0 < -((c_1 + c_2)y_0 \mp c_1 + c_0)(x_0 - x_1) \right) \wedge \left(0 < ((c_1 + c_2)y_1 \mp c_1 + c_0)(x_0 - x_1) \right). \end{aligned}$$

Due to the nonnegativity constraints either the first line implies $(y_0 - y_1 > 0) \wedge (y_0 - y_1 < 0)$ (for the “+” branch of the “ \pm ”), or the second line implies $(x_0 - x_1 < 0) \wedge (x_0 - x_1 > 0)$ (for the “+” branch of the “ \mp ”), which is a contradiction. The argument for $b_0 = 0$ is similar. Thus, the theorem holds. In contrast to the previous proof no contradiction can be derived for both lines at the same time. In other words, there are valid parameters c_0, c_1, c_2 for which there exist x_0, y_0, x_1, y_1 such that two of the four inequalities hold. \square

A.2 Conditioning the n -LIF system

We mentioned in Section 3.3.1 that the reduced n -LIF system is notoriously ill-conditioned. In fact, the weight and parameter optimisation methods presented in Section 3.5 do not work properly without preconditioning the reduced system. We first describe our preconditioning procedure, followed by an example that highlights the importance of conditioning.

A.2.1 Suggested Preconditioning Procedure

We suggest the following series of transformations as a preconditioning step. Apart from the final scaling step, this transformation does not alter the current predicted by H in any way.

Step 1. Offset all voltages such that $\bar{v} = 0$. This is accomplished by updating $\tilde{\mathbf{b}}'$ and $\tilde{\mathbf{B}}'$:

$$\tilde{\mathbf{b}}' \leftarrow \tilde{\mathbf{b}}' - (\tilde{\mathbf{L}}' + \text{diag}(\tilde{\mathbf{a}}'))\bar{\mathbf{v}}, \quad \tilde{\mathbf{B}}' \leftarrow \tilde{\mathbf{B}}' - \tilde{\mathbf{A}}'\bar{\mathbf{v}}.$$

This rule follows from applying an offset \mathbf{o} to the equilibrium state of the n -LIF system:

$$-\tilde{\mathbf{A}}[\mathbf{g}]^{-1}\tilde{\mathbf{b}}[\mathbf{g}] + \mathbf{o} = -\tilde{\mathbf{A}}[\mathbf{g}]^{-1}(\tilde{\mathbf{b}}[\mathbf{g}] - \tilde{\mathbf{A}}[\mathbf{g}]\mathbf{o}).$$

Expanding $\tilde{\mathbf{A}}[\mathbf{g}]\mathbf{o}$ with $\mathbf{o} = -\bar{\mathbf{v}}$ yields the above equations.

Step 2. Scale all voltages such that $\tilde{c}_i \in \{1, 0\}$ or $\tilde{a}'_i = 1$. Together with $\bar{\mathbf{v}} = \mathbf{0}$, this implies that the equilibrium state of each compartment connected to the soma directly expresses the current flowing into the soma. To this end, we first assemble a positive vector of scaling factors $\boldsymbol{\alpha} \in \mathbb{R}^n$ and update $\tilde{\mathbf{L}}'$, $\tilde{\mathbf{a}}'$, $\tilde{\mathbf{A}}'$, $\tilde{\mathbf{c}}$ as follows:

$$\alpha_i = \begin{cases} \tilde{c}_i & \text{if } \tilde{c}_i > 0, \\ \tilde{a}'_i & \text{if } \tilde{a}'_i > 0 \text{ and } \tilde{c}_i = 0, \\ 1 & \text{otherwise,} \end{cases} \quad \begin{aligned} \tilde{\mathbf{L}} &\leftarrow \tilde{\mathbf{L}} \text{diag}(\boldsymbol{\alpha})^{-1}, & \tilde{\mathbf{a}}' &\leftarrow \text{diag}(\boldsymbol{\alpha})^{-1}\tilde{\mathbf{a}}', \\ \tilde{\mathbf{A}}' &\leftarrow \text{diag}(\boldsymbol{\alpha})^{-1}\tilde{\mathbf{A}}', & \tilde{\mathbf{c}} &\leftarrow \text{diag}(\boldsymbol{\alpha})^{-1}\tilde{\mathbf{c}}. \end{aligned}$$

This update rule follows from scaling the equilibrium state of the n -LIF system:

$$-\text{diag}(\boldsymbol{\alpha})\tilde{\mathbf{A}}[\mathbf{g}]^{-1}\tilde{\mathbf{b}}[\mathbf{g}] = -(\tilde{\mathbf{A}}[\mathbf{g}] \text{diag}(\boldsymbol{\alpha})^{-1})^{-1}\tilde{\mathbf{b}}[\mathbf{g}].$$

Again, expanding $\tilde{\mathbf{A}}[\mathbf{g}] \text{diag}(\boldsymbol{\alpha})^{-1}$ yields the first three update equations. Scaling $\tilde{\mathbf{c}}$ is necessary to preserve the correct output current. In general, if $\bar{\mathbf{v}} \neq \mathbf{0}$, then $\bar{\mathbf{v}}$ must be multiplied by $\text{diag}(\boldsymbol{\alpha})$.

Step 3. Scale the system input and output. Inputs to the n -LIF system are typically in the microsiemens (μS) or nanoampere (nA) range, while outputs are usually single-digit nanoampere values. We hence suggest scaling all inputs by $\alpha_{\text{in}} = 10^6$ (before passing them into the system) and all outputs by $\alpha_{\text{out}} = 10^9$. Given that $\bar{v} = 0$, we can adapt the system as follows:

$$\tilde{\mathbf{A}}' \leftarrow \tilde{\mathbf{A}}' \frac{1}{\alpha_{\text{in}}}, \quad \tilde{\mathbf{b}}' \leftarrow \tilde{\mathbf{b}}' \alpha_{\text{out}}, \quad \tilde{\mathbf{B}}' \leftarrow \tilde{\mathbf{B}}' \frac{\alpha_{\text{out}}}{\alpha_{\text{in}}}.$$

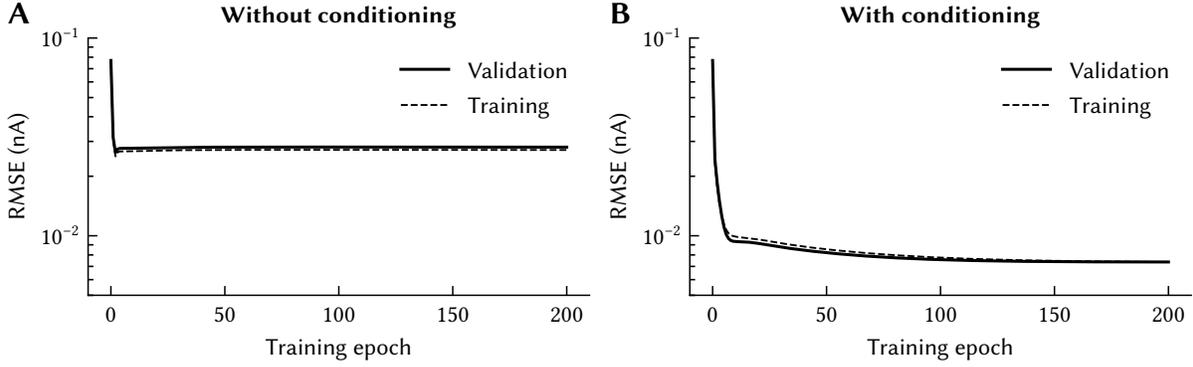


Figure A.3: Impact of conditioning on gradient-based model parameter optimisation. Example of parameter optimisation for the neuron model described below. Data for $N = 1000$ training and test samples $\mathbf{g}_k \in [0 \mu\text{S}, 1 \mu\text{S}]^4$. **(A)** is without conditioning, **(B)** is with conditioning. Learning rates α are as large as possible without introducing major instabilities; $\alpha = 3 \times 10^{-7}$ in (A), $\alpha = 10^{-2}$ in (B).

A.2.2 Impact of Conditioning in Practice

Consider the three-compartment neuron depicted in Figure 3.22D. Using the parameters in Table B.1 and coupling conductances $c_{12} = 50 \text{ nS}$, $c_{23} = 200 \text{ nS}$, we get the following system:

$$\tilde{\mathbf{a}}' = \begin{bmatrix} 1 \\ 100 \times 10^{-9} \\ 50 \times 10^{-9} \end{bmatrix}, \quad \tilde{\mathbf{A}}' = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}, \quad \tilde{\mathbf{b}}' = \begin{bmatrix} -57.5 \times 10^{-3} \\ -6.125 \times 10^{-9} \\ -3.25 \times 10^{-9} \end{bmatrix}, \quad \tilde{\mathbf{c}} = \begin{bmatrix} 1 \\ 50 \times 10^{-9} \\ 0 \end{bmatrix},$$

$$\tilde{\mathbf{B}}' = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 20 \times 10^{-3} & -75 \times 10^{-3} \\ 20 \times 10^{-3} & -75 \times 10^{-3} & 0 & 0 \end{bmatrix}, \quad \tilde{\mathbf{L}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 200 \times 10^{-9} & -200 \times 10^{-9} \\ 0 & -200 \times 10^{-9} & 200 \times 10^{-9} \end{bmatrix}.$$

The condition number κ of $\tilde{\mathbf{A}}[\mathbf{0}] = \tilde{\mathbf{L}}$ is approximately $\kappa \approx 10^7$. As pointed out by Cheney and Kincaid (2012, Section 8.4, p. 406) this implies that we lose about seven digits of precision when computing $\tilde{\mathbf{A}}[\mathbf{g}]^{-1}\tilde{\mathbf{b}}$. In contrast, the conditioned system has a condition number of $\kappa \approx 10^1$:

$$\tilde{\mathbf{a}}' = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, \quad \tilde{\mathbf{A}}' = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 20 & 20 \\ 20 & 20 & 0 & 0 \end{bmatrix}, \quad \tilde{\mathbf{b}}' = \begin{bmatrix} 0 \\ -0.375 \\ -0.375 \end{bmatrix}, \quad \tilde{\mathbf{c}} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix},$$

$$\tilde{\mathbf{B}}' = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 57.5 & -17.5 \\ 57.5 & -17.5 & 0 & 0 \end{bmatrix}, \quad \tilde{\mathbf{L}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & -4 \\ 0 & -4 & 4 \end{bmatrix}.$$

Conditioning significantly impacts the weight and parameter optimisation schemes discussed in Section 3.5. For example, the parameter gradients from eqs. (3.41) to (3.44) vary by ten orders of magnitude in the unconditioned system. This can make gradient-based optimisation less efficient, as illustrated in Figure A.3.

A.3 Proofs and Mathematical Detail for Chapter 4

This section contains proofs for the mathematical claims made throughout Chapter 4. Specifically, we prove that our least-squares loss generalises the NEF dynamics principle, that our notion of a “delay re-encoder” works as intended, and provide the complete derivation of the LDN system using the delay re-encoder.

A.3.1 Proof for Theorem 4.1

We claimed in Section 4.1.3 that solving our least-squares loss function in linear networks with homogeneous first-order low-pass filters results in exactly the same solution as using the NEF dynamics principle. More precisely, we claimed the following:

Theorem 4.1. *Let $\mathbf{A} \in \mathbb{R}^{n \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times m}$ (w.l.o.g. $m = 1$) represent an LTI system, \mathbf{e}_i be the impulse response of its i th state dimension, and $h(t)$ be the impulse response of a first-order low-pass:*

$$\dot{\mathbf{m}}(t) = \mathbf{A}\mathbf{m}(t) + \mathbf{B}u(t), \quad \mathbf{e}_i(t) = \left(\exp(\mathbf{A}t)\mathbf{B} \right)_i, \quad h(t) = \tau^{-1} e^{-t\tau^{-1}}.$$

Let $\mathbf{x}_k : \mathbb{R}^- \rightarrow \mathbb{R}$ be one of $N \geq n + 1$ arbitrary (with some mild constraints; see proof) input signals. If all \mathbf{e}_i and \mathbf{x}_k are pairwise linearly independent, the following least-squares loss has a unique minimum at $(b'_i, a'_{i,1}, \dots, a'_{i,n}) = (\mathbf{B}', \mathbf{A}')_i$ for all $i \in \{1, \dots, n\}$, where $\mathbf{A}' = \tau\mathbf{A} + \mathbf{I}$, $\mathbf{B}' = \tau\mathbf{B}$:

$$E = \sum_{k=1}^N \left(\int_0^\infty \mathbf{e}_i(\tau)\mathbf{x}_k(-\tau) - b'_i h(\tau)\mathbf{x}_k(-\tau) - \sum_{j=1}^n a'_{ij} h(\tau) \int_0^\infty \mathbf{x}_k(-\tau - \tau')\mathbf{e}_j(\tau') d\tau' d\tau \right)^2.$$

Proof. We prove this theorem in two steps: (1) the solution to the least-squares loss is indeed unique, and (2) we can achieve a zero error with the solution mentioned above; therefore, this solution must be the global optimum. Note that the first part of our proof is less rigorous; we pay less attention to some edge-cases that could technically occur.

(1) *The solution to the least-squares loss is unique.* We have a linear least-squares loss over $n + 1$ variables and $N \geq n + 1$ samples. For a unique solution, the design matrix $\mathbf{P} \in \mathbb{R}^{N \times n+1}$ must be of full column rank:

$$\mathbf{P} = \begin{bmatrix} (h * \mathbf{x}_1)(0) & (h * (\mathbf{e}_1 * \mathbf{x}_1))(0) & \dots & (h * (\mathbf{e}_n * \mathbf{x}_1))(0) \\ (h * \mathbf{x}_2)(0) & (h * (\mathbf{e}_1 * \mathbf{x}_2))(0) & \dots & (h * (\mathbf{e}_n * \mathbf{x}_2))(0) \\ \vdots & \vdots & \ddots & \vdots \\ (h * \mathbf{x}_N)(0) & (h * (\mathbf{e}_1 * \mathbf{x}_N))(0) & \dots & (h * (\mathbf{e}_n * \mathbf{x}_N))(0) \end{bmatrix}$$

where “*” denotes convolution. Because all functions \mathbf{e}_i and \mathbf{x}_k are pairwise linearly independent, \mathbf{P} is usually of full column rank; filtering each function with the same low-pass filter h does not change this. One possible exception would be selecting pathological \mathbf{x}_k that are orthogonal to all \mathbf{e}_k ; random \mathbf{x}_k are unlikely to have this property.

(2) Using the matrices \mathbf{A}' and \mathbf{B}' as a solution results in a zero error. Instead of proving this for the input sequences \mathbf{r}_k , we instead prove this property for any \mathbf{r} . In particular, for a zero error in the least-squares loss, the following equality must hold for all $i \in \{1, \dots, n\}$.

$$\int_0^\infty \mathbf{e}_i(\tau)\mathbf{r}(-\tau)d\tau \stackrel{!}{=} \int_0^\infty b'_i h(\tau)\mathbf{r}_k(-\tau) + \sum_{j=1}^n a'_{ij}h(\tau) \int_0^\infty \mathbf{r}_k(-\tau - \tau')\mathbf{e}_j(\tau') d\tau' d\tau.$$

Rewriting the integrals as convolutions evaluated at zero and rearranging, we obtain

$$0 \stackrel{!}{=} (\mathbf{e}_i * \mathbf{r})(0) - b'_i(h * \mathbf{r})(0) - \sum_{j=1}^n a'_{ij}(h * (\mathbf{e}_n * \mathbf{r}))(0).$$

Expressing the convolution operations vectorially over all i in the Laplace domain (evaluating the convolution everywhere, and not just at zero), we get

$$0 \stackrel{!}{=} E(s)X(s) - \mathbf{B}'H(s)X(s) - \mathbf{A}'H(s)E(s)X(s). \quad (\text{A.3})$$

Per definition, $E(s)$ is the impulse response of the LTI system defined by \mathbf{A} , \mathbf{B} . Let $M(s)$ be the Laplace transform of the linear system state $\mathbf{m}(t)$. We have the following quantities:

$$M(s) = \frac{1}{s}\mathbf{A}M(s) + \mathbf{B}X(s), \quad E(s) = \frac{M(s)}{X(s)} = \frac{1}{s} \frac{\mathbf{A}M(s) + \mathbf{B}X(s)}{X(s)}.$$

Furthermore, recall that the first-order low-pass filter $H(s)$ is given as

$$H(s) = \frac{1}{\tau s + 1}.$$

Substituting these definitions along with $\mathbf{A}' = \tau\mathbf{A} + \mathbf{I}$ and $\mathbf{B}' = \tau\mathbf{B}$ into eq. (A.3) yields

$$\begin{aligned} & E(s)X(s) - \mathbf{B}'H(s)X(s) - \mathbf{A}'H(s)E(s)X(s) \\ &= \frac{1}{s}(\mathbf{A}M(s) + \mathbf{B}X(s)) - \mathbf{B}'H(s)X(s) - \frac{1}{s}\mathbf{A}'H(s)(\mathbf{A}M(s) + \mathbf{B}X(s)) \\ &= \frac{1}{s}(\mathbf{A}M(s) + \mathbf{B}X(s)) - \frac{1}{\tau s + 1}\mathbf{B}'X(s) - \frac{1}{(\tau s + 1)s}\mathbf{A}'(\mathbf{A}M(s) + \mathbf{B}X(s)) \\ &= \frac{1}{s}(\mathbf{A}M(s) + \mathbf{B}X(s)) - \frac{1}{\tau s + 1}\tau\mathbf{B}X(s) - \frac{1}{(\tau s + 1)s}(\tau\mathbf{A} + \mathbf{I})(\mathbf{A}M(s) + \mathbf{B}X(s)) \\ &= \frac{(\mathbf{A}M(s) + \mathbf{B}X(s) - sM(s))\mathbf{A}\tau}{(\tau s + 1)s} \\ &= \frac{(\mathbf{A}M(s) + \mathbf{B}X(s) - \mathbf{A}M(s) - \mathbf{B}X(s))\mathbf{A}\tau}{(\tau s + 1)s} \\ &= 0. \end{aligned} \quad \square$$

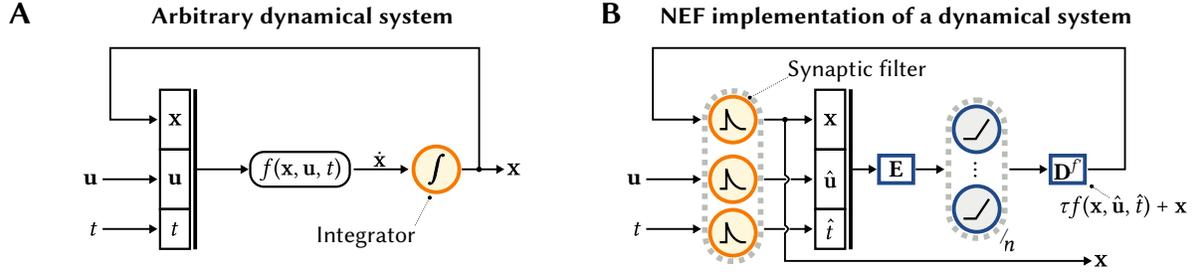


Figure A.4: Implementing arbitrary nonlinear dynamics in the NEF. **(A)** Diagram illustrating integrating an arbitrary dynamical system. The state x , input u and time t are passed through f to obtain the differential \dot{x} . The integral of \dot{x} is equal to x . **(B)** Implementing the same system in the NEF. All quantities are passed through a first-order low-pass filter with time-constant τ . The output of the filter operation is then represented in a population of neurons using an encoder matrix E in a population of n neurons. We can then decode the feedback function f' compensating for the synaptic filter. The state x is exactly the result of passing the result of f' through the low-pass filter.

A.3.2 Temporal Tuning and Nonlinear Dynamics

We noted that temporal tuning is a generalisation of the NEF dynamics principle. That is, we can—in addition to accounting for heterogeneous and higher-order filters, and taking empirical data into account—use temporal tuning curves to implement any dynamical system that could be implemented using the dynamics principle. This includes nonlinear dynamical systems.

Nonlinear dynamical systems in the NEF. Theorem 4.1 states that, under some mild conditions, our optimisation problem in eq. (4.7) perfectly realises any LTI system of the form $\dot{\mathbf{m}}(t) = \mathbf{A}\mathbf{m}(t) + \mathbf{B}\mathbf{u}(t)$ in a recurrently connected linear neuron population, given that the connecting synapses are heterogeneous first-order low-pass filters. In particular, the resulting weight matrices are $\mathbf{A}' = \tau\mathbf{A} + \mathbf{I}$, and $\mathbf{B}' = \tau\mathbf{B}$. These matrices implement the desired LTI system while compensating for the low-pass filter acting as a leaky integrator (cf. Figure 2.29).

While this is an important observation, the NEF dynamics principle is more general than that. As we noted in Section 2.3.5, the above substitution is a special case of a more general transformation that can be used to implement any dynamical system of the form $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t), t)$ across a low-pass filter with time-constant τ (cf. Figure A.4). We have:

$$f'(\mathbf{x}(t), \hat{\mathbf{u}}(t), \hat{t}(t)) = \tau f(\mathbf{x}(t), \hat{\mathbf{u}}(t), \hat{t}(t)) + \mathbf{x}(t). \quad (\text{A.4})$$

Here, $\hat{\mathbf{u}}(t)$ and $\hat{t}(t)$ are low-pass filtered versions of the input and the time t . Note that this is a *perfect* transformation. As long as we are able to realise f' without error, the resulting system will perfectly produce the desired dynamics described by $f(\mathbf{x}(t), \mathbf{u}(t), t)$. Of course, to realise an arbitrary f' that is nonlinear over $\mathbf{x}(t)$, $\hat{\mathbf{u}}(t)$, $\hat{t}(t)$, all these variables must be represented in a single neuron population with diverse tuning (cf. Figure A.4B).

Nonlinear temporal tuning curves. To realise nonlinear dynamical systems using the temporal tuning paradigm, we need to make use of the general concept of a temporal tuning curve from Definition 4.1; *linear* temporal tuning curves (Definition 4.2) are restricted to LTI systems. For the sake of simplicity, we only consider time-invariant systems, that is $f(\mathbf{x}, \mathbf{u})$.¹

The temporal tuning curve of a neuron tuned to an n -dimensional system $f(\mathbf{x}, \mathbf{u})$ is

$$a_i(\mathbf{u}) = G \left[\alpha_i \langle \mathbf{e}_i, (\mathbf{x}(0, \mathbf{u}), (h * \mathbf{u})(0)) \rangle + \beta_i \right], \quad \text{where} \quad \mathbf{x}(t, \mathbf{u}) = \int_{-\infty}^t f(\mathbf{x}(t', \mathbf{u}), \mathbf{u}(t')) dt'.$$

Here, a_i is the activity of the i th neuron given an m -dimensional input signal $\mathbf{u} : \mathbb{R}^- \rightarrow \mathbb{R}^m$, and \mathbf{e}_i is an encoding vector uniformly sampled from the hypersphere \mathbb{S}^{m+n} . This tuning curve exactly describes the activity of a neuron in Figure A.4B in response to an input signal \mathbf{u} .

With this temporal tuning, in the limit of an infinite number of neurons $m \rightarrow \infty$ and samples $N \rightarrow \infty$, and w.l.o.g. $J_i(\xi) = \xi$, our optimisation problem from eq. (4.7) becomes

$$E = \sum_{k=1}^N \left[\langle \mathbf{e}_i, (\mathbf{x}(0, \mathbf{u}_k), (h * \mathbf{u}_k)(0)) \rangle - \sum_{j=1}^m w_{ij} \int_0^\infty h(\tau) a_j(\mathbf{u}_k * \delta_\tau) d\tau - \sum_{j=1}^m w'_{ij} (h * \mathbf{u}_k)(0) \right]^2,$$

where the w_{ij} are the feedback weights, and the w'_{ij} are the input weights. Obviously, the optimal input weights w'_{ij} are merely a block of the encoder matrix \mathbf{E} . This leaves us with determining the feedback weights w_{ij} . In particular, note that it must hold $E = 0$ if we can find a decoding matrix $(\mathbf{D})_{ij} = d_{ij}$ such that for all k and all $i \in \{1, \dots, n\}$

$$(\mathbf{x}(0, \mathbf{u}_k))_i = \sum_{j=1}^m d_{ij} \int_0^\infty h(\tau) a_j(\mathbf{u}_k * \delta_\tau) d\tau = \sum_{j=1}^m \int_0^\infty h(\tau) d_{ij} a_j(\mathbf{u}_k * \delta_\tau) d\tau.$$

Since $m, N \rightarrow \infty$, and our neuron population represents the current state $\mathbf{x} = \mathbf{x}(0, \mathbf{u}_k)$ and the low-pass filtered input $\hat{\mathbf{u}}$, we can further simplify this to:

$$(\mathbf{x})_i = \int_0^\infty h(\tau) f'_i(\mathbf{x}(-\tau), \hat{\mathbf{u}}(-\tau)) d\tau.$$

In other words, the optimal solution is a function f' that takes \mathbf{x} , the filtered $\hat{\mathbf{u}}$, and, if being passed through a first-order low-pass filter h over time, returns the current state \mathbf{x} . Our f' from above *exactly* fulfills this property (cf. Figure A.4B); hence, by the same argument as in our proof for Theorem 4.1 (and under the same mild conditions), the optimisation problem *must* solve for connection weights that realise f' .

¹The same considerations of course still hold if we supply an external time variable, for example as part of \mathbf{u} . Technically, neither the temporal tuning approach, nor the NEF are truly capable of realising time-variant systems. The universal approximator properties of two-layer networks only hold for compact domains \mathbb{X} (cf. Theorem 3.4); as soon as time is represented, \mathbb{X} is no longer compact. In practice, t could be compressed into a vector representation using Spatial Semantic Pointers (Komer, 2020).

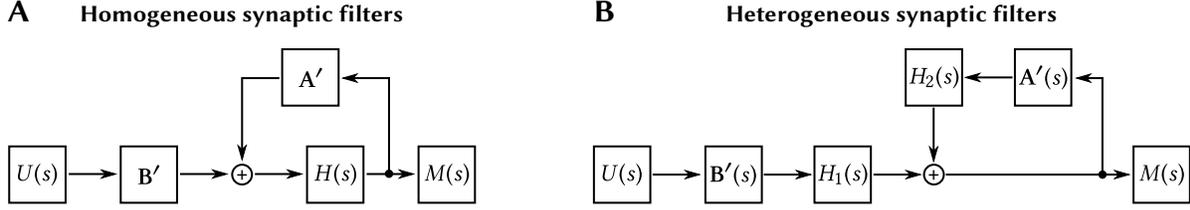


Figure A.5: Block-diagram of a recurrent networks with homogeneous and heterogeneous filters in the Laplace domain. $U(s)$ is the input signal, $M(s)$ is the state vector with the desired LTI dynamics $sM(s) = \mathbf{A}M(s) + \mathbf{B}U(s)$. **(A)** Recurrent network used in the derivation of the NEF dynamics principle. The input and feedback pass through the same filter $H(s)$. This filter takes the role of the integrator in a standard LTI system; \mathbf{A}' and \mathbf{B}' compensate for the lack of an integrator to realise the desired dynamics. **(B)** For heterogeneous $H_1(s)$, $H_2(s)$, both $\mathbf{A}'(s)$ and $\mathbf{B}'(s)$ are time-dependent filters themselves.

A.3.3 Heterogeneous Time Constants Require Access to the Derivative

We mentioned in Section 4.1.4 that compensating for the dynamics of a network with heterogeneous time-constants requires access to the derivative of the input and feedback signal. To get some intuition for this, recall the derivation of the original NEF dynamics principle (cf. Section 2.3.5; Eliasmith and Anderson, 2003, Chapter 8). Using the homogeneous network illustrated in Figure A.5A

$$M(s) = \frac{1}{s}(\mathbf{A}M(s) + \mathbf{B}U(s)), \quad (\text{A.5})$$

$$\text{and } M(s) = \mathbf{B}'H(s)U(s) + \mathbf{A}'H(s)M(s) = \frac{1}{\tau s + 1}(\mathbf{B}'U(s) + \mathbf{A}'M(s)), \quad (\text{A.6})$$

where the first line are our desired dynamics, and the second line are the actual network dynamics. Multiplying both sides of eq. (A.6) with $\tau s + 1$ and rearranging the equation into the form of eq. (A.5) and solving for \mathbf{A}' , \mathbf{B}' yields:

$$M(s) = \frac{1}{s} \left(\frac{\mathbf{A}' - \mathbf{I}}{\tau} M(s) + \frac{\mathbf{B}'}{\tau} U(s) \right) \quad \Rightarrow \quad \mathbf{A}' = \tau \mathbf{A} + \mathbf{I}, \quad \mathbf{B}' = \tau \mathbf{B}.$$

We can now repeat the same procedure for the network with heterogeneous synaptic filters depicted in Figure A.5B; however, we will obtain filters $\mathbf{A}'(s)$, $\mathbf{B}'(s)$. Assuming that $H_1(s)$ and $H_2(s)$ are first-order low-pass filters with time-constants τ_1 , τ_2 , we have

$$M(s) = \mathbf{B}'(s)H_1(s)U(s) + \mathbf{A}'(s)H_2(s)M(s) = \mathbf{B}'(s)\frac{1}{\tau_1 s + 1}U(s) + \mathbf{A}'(s)\frac{1}{\tau_2 s + 1}M(s).$$

To make sure that our filters $\mathbf{A}'(s)$ and $\mathbf{B}'(s)$ are causal (i.e., the order of the numerator is smaller or equal to the order of the denominator), we must first rearrange the equation, for example after multiplying both sides with $(\tau_1 s + 1)(\tau_2 s + 1)$. We obtain

$$M(s) = \frac{1}{s} \left(\frac{\mathbf{A}'(s)(\tau_1 s + 1) - \mathbf{I}((\tau_1 + \tau_2)s - 1)}{s\tau_1\tau_2} M(s) + \frac{\mathbf{B}'(s)(s\tau_2 + 1)}{s\tau_1\tau_2} U(s) \right).$$

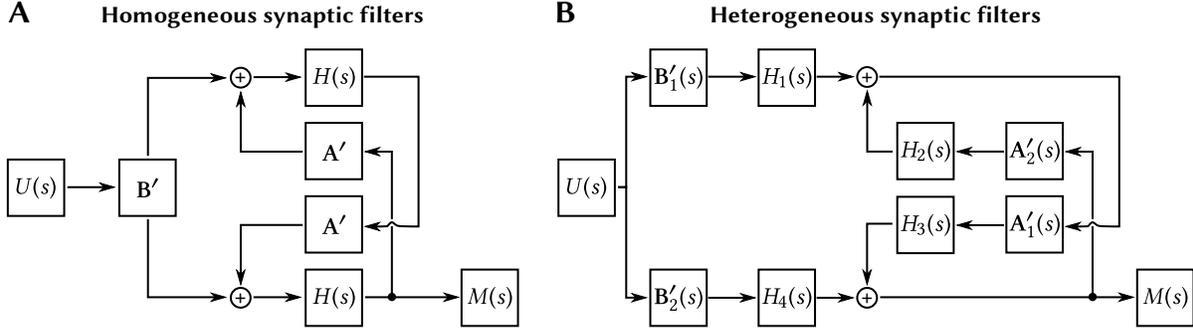


Figure A.6: Block-diagram of the dynamics of a recurrent network with an interneuron population. As in Figure A.5, $U(s)$ is the input signal, $M(s)$ is the state vector with desired dynamics $sM(s) = AM(s) + BU(s)$. **(A)** If all synaptic filters in the feedback and inhibitory path are the same, then there is a simple time-invariant solution for matrices A' , B' that realise the dynamical system. **(B)** For heterogeneous filters, there is no time-invariant solution for the input and feedback matrices. See Section 4.1.4 for a numerical approximation.

Solving for $A'(s)$ and $B'(s)$ results in two first-order high-pass filters:

$$A'(s) = \frac{(A\tau_1\tau_2 + \tau_1 + \tau_2)s + 1}{\tau_1s + 1}, \quad B'(s) = \frac{B\tau_1\tau_2s}{\tau_2s + 1}.$$

Curiously, this solution does not coincide with the original NEF solution for $\tau_1 = \tau_2$; this is due to the problem of finding $A'(s)$ and $B'(s)$ being underconstrained. However, and without providing a proof, for $\tau_1 \neq \tau_2$, s will always be in the numerator of $A'(s)$ and $B'(s)$; correspondingly, these filters require access to the derivative of their input.

A.3.4 Homogeneous Recurrent Networks with Interneurons

We claimed in Section 4.1.4 that the time-invariant matrices $A' = \tau A + I$, $B' = \tau B$ can be used to realise any desired LTI system in a recurrent neural network with an intermediate population and homogeneous synaptic filters (cf. Figure A.6A). We discuss why this is the case below—we first provide an intuitive solution, and then prove the result more rigorously. Note that, similarly to the argument made in the previous section, finding time-invariant A' , B' is generally not possible for recurrent networks with heterogeneous filters (cf. Figure A.6B).

Intuitive explanation. Observe that if we were able to “undo” the filtering performed by the synaptic filter of the upper interneuron population, we would have a “normal” recurrent neuron population. In other words, we would, from the perspective of the “primary” neuron like interneurons to behave like a pass-through. Using the notation from Figure A.6B, and ignoring the input through B'_1 , we need a filter $A'_1(s)$ that reverses the synaptic filter $H_1(s) = H(s)$:

$$M(s) = H(s)A'_1(s)M(s) = H(s)\frac{1}{H(s)}M(s) = \frac{1}{\tau s + 1}(1 + \tau s)M(s).$$

Appendix A. Mathematical Detail

To implement $\mathbf{A}'_1(s)M(s) = (1 + \tau s)M(s) = M(s) + \tau sM(s)$, we need access to $M(s)$ and its differential $sM(s)$. Fortunately, the differential is, by definition, just given as

$$sM(s) = \mathbf{A}M(s) + \mathbf{B}U(s).$$

Hence, the total input of the upper synaptic filter $H_1(s)$ is

$$(1 + \tau s)M(s) = M(s) + \tau sM(s) = M(s) + \tau \mathbf{A}M(s) + \tau \mathbf{B}U(s) = \underbrace{(\tau \mathbf{A} + \mathbf{I})}_{\mathbf{A}'_1} M(s) + \underbrace{\tau \mathbf{B}}_{\mathbf{B}'_1} U(s)$$

Since these $\mathbf{A}'_1, \mathbf{B}'_1$ eliminate the upper synaptic filter, we can now treat \mathbf{A}'_2 and \mathbf{B}'_2 as implementing a “normal” recurrent NEF network. Hence $\mathbf{A}'_1 = \mathbf{A}'_2 = \mathbf{A}' = \tau \mathbf{A} + \mathbf{I}$, $\mathbf{B}'_1 = \mathbf{B}'_2 = \mathbf{B}' = \tau \mathbf{B}$.

Proof. For homogeneous filters, the dynamics of the network depicted in Figure A.6B are

$$M(s) = H(s)\mathbf{B}'_2U(s) + H^2(s)\mathbf{A}'_1\mathbf{B}'_1U(s) + H^2(s)\mathbf{A}'_1\mathbf{A}'_2(s)M(s)$$

Multiplying both sides with s and substituting in our alleged solution for $\mathbf{A}'_1, \mathbf{A}'_2, \mathbf{B}'_1, \mathbf{B}'_2$:

$$\begin{aligned} sM(s) &= \tau sH(s)\mathbf{B}U(s) + sH^2(s)(\tau(\tau \mathbf{A} + \mathbf{I})\mathbf{B}U(s) + s(\tau \mathbf{A} + \mathbf{I})^2M(s)) \\ &= \frac{\tau s\mathbf{B}U(s)}{\tau s + 1} + \frac{\tau s(\tau \mathbf{A} + \mathbf{I})\mathbf{B}U(s) + s(\tau \mathbf{A} + \mathbf{I})^2M(s)}{(1 + \tau s)^2} \end{aligned}$$

Combining both fractions and continuing to expand:

$$\begin{aligned} &= \frac{\tau s\mathbf{B}U(s)(1 + \tau s) + \tau s(\tau \mathbf{A} + \mathbf{I})\mathbf{B}U(s) + s(\tau \mathbf{A} + \mathbf{I})^2M(s)}{(1 + \tau s)^2} \\ &= \frac{\tau s\mathbf{B}U(s) + \tau^2 s^2\mathbf{B}U(s) + \tau^2 s\mathbf{A}\mathbf{B}U(s) + \tau s\mathbf{B}U(s) + \tau^2 s\mathbf{A}^2M(s) + 2\tau s\mathbf{A}M(s) + sM(s)}{(1 + \tau s)^2} \\ &= \frac{sM(s) + 2\tau s\mathbf{A}M(s) + 2\tau s\mathbf{B}U(s) + \tau^2 s\mathbf{A}\mathbf{B}U(s) + \tau^2 s\mathbf{A}^2M(s) + \tau^2 s^2\mathbf{B}U(s)}{(1 + \tau s)^2} \end{aligned}$$

Reordering the coefficients and substituting in our desired dynamics $sM(s)$ with $\mathbf{A}M(s) + \mathbf{B}U(s)$:

$$\begin{aligned} &= \frac{\mathbf{A}M(s) + \mathbf{B}U(s) + 2\tau s\mathbf{A}M(s) + 2\tau s\mathbf{B}U(s) + \tau^2 s\mathbf{A}\mathbf{B}U(s) + \tau^2 s\mathbf{A}^2M(s) + \tau^2 s^2\mathbf{B}U(s)}{(1 + \tau s)^2} \\ &= \frac{\mathbf{A}M(s) + \mathbf{B}U(s) + 2\tau s\mathbf{A}M(s) + 2\tau s\mathbf{B}U(s) + \tau^2 s\mathbf{A}(\mathbf{B}U(s) + \mathbf{A}M(s)) + \tau^2 s^2\mathbf{B}U(s)}{(1 + \tau s)^2} \\ &= \frac{\mathbf{A}M(s) + \mathbf{B}U(s) + 2\tau s\mathbf{A}M(s) + 2\tau s\mathbf{B}U(s) + \tau^2 s^2\mathbf{A}M(s) + \tau^2 s^2\mathbf{B}U(s)}{(1 + \tau s)^2} \\ &= \frac{(\mathbf{A}M(s) + \mathbf{B}U(s))(1 + 2\tau s + \tau^2 s^2)}{(1 + \tau s)^2} \\ &= \frac{(\mathbf{A}M(s) + \mathbf{B}U(s))(1 + \tau s)^2}{(1 + \tau s)^2} \\ &= \mathbf{A}M(s) + \mathbf{B}U(s) \end{aligned}$$

Correspondingly, the given solution is consistent with the network exactly producing the desired dynamics. \square

A.3.5 Proof of Lemma 4.2

In Section 4.2.4, we discussed a LTI system with an impulse response that perfectly traces out the Legendre polynomials. In particular, we claimed the following:

Lemma 4.2. *The impulse response of the linear time-invariant system $\theta \dot{\mathbf{m}}(t) = \mathbf{A}' \mathbf{m}(t) + \mathbf{B}' u(t)$ with $\mathbf{m} \in \mathbb{R}^q$, $\mathbf{A}' \in \mathbb{R}^{q \times q}$ and $\mathbf{B}' \in \mathbb{R}^{q \times 1}$*

$$(\mathbf{A}')_{ij} = (4j - 2) \begin{cases} 0 & \text{if } i \leq j \text{ or } i + j \text{ is even,} \\ 4j - 2 & \text{if } i > j \text{ and } i + j \text{ is odd,} \end{cases} \quad (\mathbf{B}')_i = (-1)^{i+1},$$

are the first q shifted Legendre polynomials $\tilde{P}_n(t\theta^{-1})$ scaled to $\tilde{P}_n(1) = 1$ over $t \in [0, \theta]$.

Proof. As pointed out by Gu et al. (2020, Appendix B.1.1), and originally described in Arfken et al. (2005, Chapter 12.2, p. 751), the standard Legendre polynomials (eq. 4.14) fulfil the following recurrence relation with respect to their derivative:

$$\frac{d}{dt} (P_{n+1}(t) - P_{n-1}(t)) = (2n + 1)P_n(t).$$

Substituting in $\tilde{P}_n(t\theta^{-1}) = P_n((2t - 1)\theta^{-1})$ and rearranging we get

$$\begin{aligned} \frac{d}{dt} \theta \tilde{P}_{n+1}(t\theta^{-1}) &= (4n + 2)\tilde{P}_n(t\theta^{-1}) + \frac{d}{dx} \tilde{P}_{n-1}(t\theta^{-1}) \\ &= (4n + 2)\tilde{P}_n(t\theta^{-1}) + (4(n - 2) + 2)\tilde{P}_{n-2}(t\theta^{-1}) + \dots \end{aligned}$$

This recurrence relation terminates with \tilde{P}_0 or \tilde{P}_1 depending on whether n is even or odd. Crucially, this recurrence relation implies that the differential of the n th Legendre polynomial can be expressed as a linear combination of the preceding Legendre polynomials. Let $\mathbf{m}(t) = (\tilde{P}_0(t\theta^{-1}), \dots, \tilde{P}_{q-1}(t\theta^{-1}))$. We can now write the above equation as a vector-matrix product

$$\frac{d}{dt} \theta \mathbf{m}(t) = \mathbf{A}' \mathbf{m}(t),$$

where \mathbf{A} is as defined in lemma 4.2. The vector $\mathbf{B}' = (\tilde{P}_0(0), \dots, \tilde{P}_{q-1}(0))$ defines the initial value of each state dimension in response to a Dirac pulse $u(t) = \delta(t)$. \square

A.3.6 Proof of Lemma 4.3

Recall Lemma 4.3, which stated that we can construct a perfect rectangle window if we have access to a delayed input $u(t - \theta)$.

Lemma 4.3. *Let \mathbf{A} , \mathbf{B} describe an LTI system and let $u(t)$ be some input signal. The impulse response of the following modified system is unchanged compared to the original LTI system for $0 \leq t < \theta$ but zero for all $t \geq \theta$*

$$\dot{\mathbf{m}}(t) = \mathbf{A}\mathbf{m}(t) + \mathbf{B}u(t) - \mathbf{e}(\theta)u(t - \theta), \quad \text{where } \mathbf{e}(\theta) = \exp(\mathbf{A}\theta)\mathbf{B}.$$

Appendix A. Mathematical Detail

Proof. Consider the impulse response, i.e., $u(t) = \delta(t)$, where $\delta(t)$ is a Dirac pulse. We show (1) that the system state is unchanged for $t < \theta$, and (2), that the impulse response is exactly zero for $t > \theta$.

(1) The differential is unchanged for $t < \theta$ as $\delta(t - \theta) = 0$ for all $t \neq \theta$; hence the impulse response of the system is $\mathbf{m}(t) = \exp(\mathbf{A}t)\mathbf{B}$ for $0 \leq t < \theta$.

(2) At $t = \theta$, according to the definition of the Dirac pulse, we subtract $\mathbf{e}(\theta) = \exp(\mathbf{A}\theta)\mathbf{B}$ from the system state $\mathbf{m}(\theta)$, exactly when $\mathbf{m}(\theta)$ is equal to $\mathbf{e}(\theta)$. The resulting \mathbf{m} is zero and remains zero as $u(t) = \delta(t) = 0$ for $t \neq 0$. \square

A.3.7 Proof of Lemma 4.4

In Lemma 4.4 we claimed that, given a delay θ' , the delay decoder (cf. Definition 4.8) for the Legendre polynomials has the following form:

Lemma 4.4. *For the shifted Legendre polynomials $\tilde{P}_i(t\theta^{-1})$ with $\tilde{P}_i(1) = 1$, the delay decoder is*

$$d_i(\theta') = \frac{2i+1}{\theta} \tilde{P}_i(\theta'\theta^{-1}).$$

Proof. Let $\mathbf{e}_i(t) = \tilde{P}_i(t\theta^{-1})$ be the shifted Legendre polynomials with scale $\tilde{P}_i(1) = 1$. Combining the proposed delay decoder $\mathbf{d}(\theta')$ with the delay decoder definition yields

$$\begin{aligned} \langle \mathbf{d}(\theta'), \mathbf{m}(\theta) \rangle &= \sum_{i=0}^{q-1} \frac{2i+1}{\theta} \tilde{P}_i(\theta'\theta^{-1}) \int_0^\theta \tilde{P}_i(\tau\theta^{-1}) \sum_{j=0}^{q-1} \chi_j \tilde{P}_j((\theta - \tau)\theta^{-1}) d\tau \\ &= \sum_{i=0}^{q-1} \sum_{m=0}^{q-1} \chi_j \frac{2i+1}{\theta} \tilde{P}_i(\theta'\theta^{-1}) \int_0^\theta \tilde{P}_i(\tau\theta^{-1}) \tilde{P}_j((\theta - \tau)\theta^{-1}) d\tau. \end{aligned}$$

The integral can be simplified using two properties of the shifted Legendre polynomials:

$$\int_0^1 \tilde{P}_i(\tau) \tilde{P}_j(\tau) d\tau = \frac{\delta_{ij}}{2i+1}, \quad (\text{Orthogonality})$$

$$\tilde{P}_i(t) = (-1)^i \tilde{P}_i(1-t), \quad (\text{Parity})$$

where δ_{ij} is the Kronecker delta. Continuing the above set of equations we get

$$\begin{aligned} \langle \mathbf{d}(\theta'), \mathbf{m}(\theta) \rangle &= \sum_{i=0}^{q-1} \sum_{j=0}^{q-1} \chi_j \frac{2i+1}{\theta} \tilde{P}_i(\theta'\theta^{-1}) (-1)^i \frac{\theta \delta_{ij}}{2i+1} \\ &= \sum_{i=0}^{q-1} \chi_i (-1)^i \tilde{P}_i(\theta'\theta^{-1}) = \hat{u}(\theta - \theta'). \end{aligned}$$

Therefore, the given delay decoder is a valid delay decoder for the Legendre polynomials. \square

Appendix B

Additional Data

B.1 Quantitative Analysis of Publications Related to the NEF

We claimed in Section 2.3 that the Neural Engineering Framework enjoys success as a framework for modelling neurobiological systems, and a technique for programming neuromorphic hardware. Unfortunately, it is challenging to track modelling techniques in the scientific literature. These methods are often secondary to the subject of a publication and thus likely mentioned only in the main body, and not the freely available title and abstract.

For the purpose of obtaining a rough estimate of the prevalence of the NEF in the research community, we performed a superficial survey of the scientific literature based on the Semantic Scholar Open Research Corpus (S2 ORC).¹ The S2 ORC is an index of 191 million scientific publications, including title, abstract, authorship, and citation graphs (Ammar et al., 2018).

As a first query, we track citations of key NEF publications (specifically Eliasmith and Anderson, 2003; Eliasmith, 2013; Bekolay, Bergstra, et al., 2014) over time and classify these publications according to the affiliation of the first author. Results for this are depicted in Figure B.1A. An increased interest in the NEF can be observed after 2012; likely due to Eliasmith, Stewart, et al. (2012). In total, well over 700 external publications cite key work related to the NEF. Of course, citations do not reliably indicate the use of the cited technique.

Correspondingly, we also provide a very conservative estimate where we only select publications that either contain the word “Nengo” or the phrase “Neural Engineering Framework” in the title or abstract. After manual clean-up, this results in 138 matching publications, to which we assign a topic (neuromorphics, modelling neurobiological systems, or theory).

Results are depicted in Figures B.1B and B.1C. Even with this more conservative estimate, about 70 external publications directly mention the NEF or Nengo in their title or abstract. Roughly half of the publications since 2012 stem from external researchers. About half of the publications are either related to neuromorphics and neurobiological modelling, whereas the other half falls into the category of theoretical neuroscience, robotics, or software.

Of course, these data are lower bounds; mentions of the NEF or Nengo in the body of the publication are not captured by this methodology. As of writing (May 2021), a Google Scholar search for “Neural Engineering Framework” counts about 700 publications; Nengo neural returns about 640 results. Unfortunately, there is no official way to access the underlying data.

¹See <https://www.semanticscholar.org/>; our analysis is based on the database dated April 1st 2021.

Appendix B. Additional Data

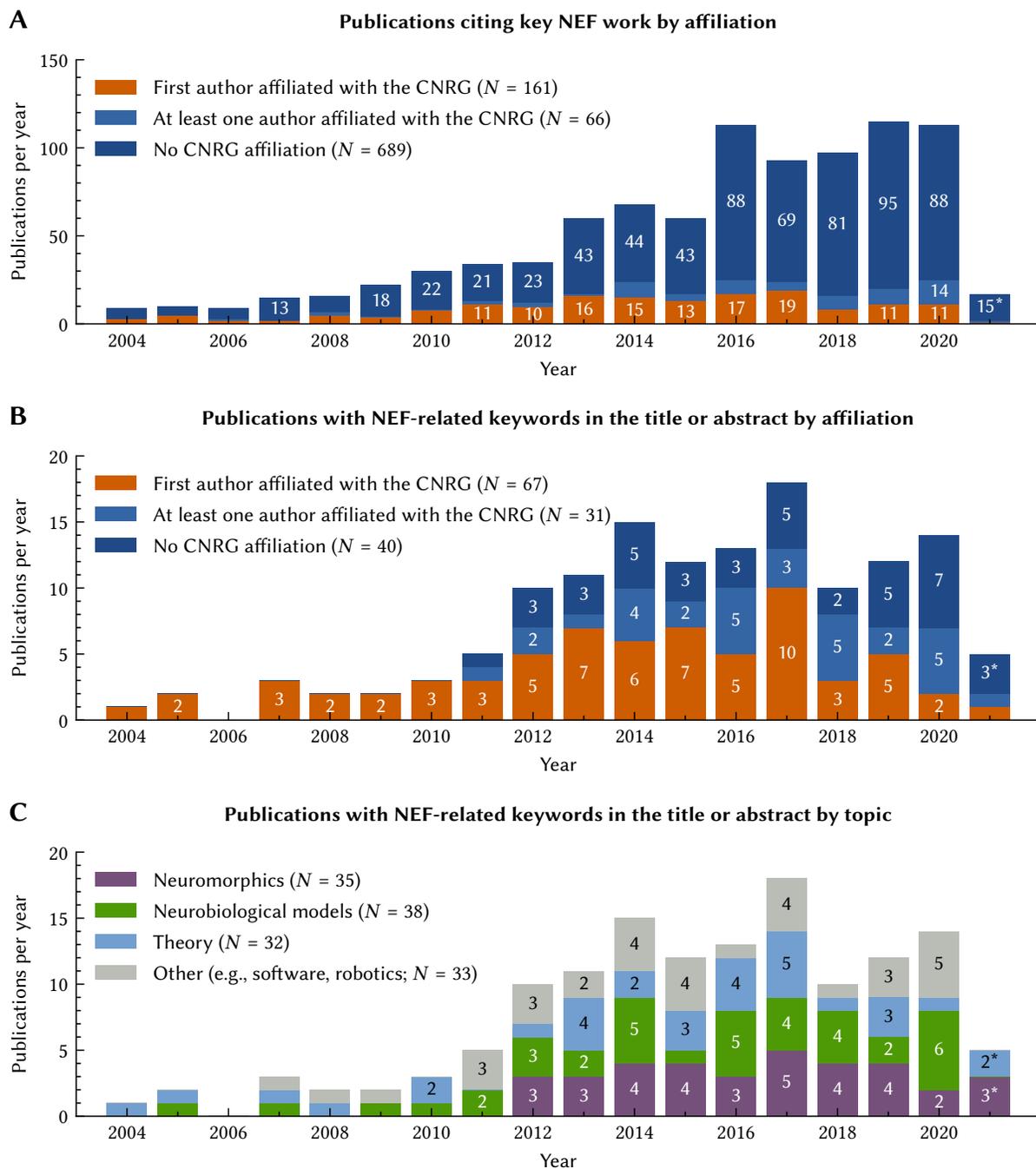


Figure B.1: Publications related to the NEF in the S2 ORC dataset. **(A)** Citations of key NEF publications (see text) by affiliation. Affiliation is based on the presence of past and present members of the Computational Neuroscience Research Group (CNRG) at the University of Waterloo in the author list. **(B)** Counts based on the presence of the word “Nengo” or the phrase “Neural Engineering Framework” in the title or abstract with manual clean-up for false-positive matches. Split by affiliation as in **(A)**. **(C)** Same data as in **(B)**, but instead categorised according to the topic. Categories are based on keyword-matching in the title with manual clean-up. (*) Data for 2021 are incomplete.

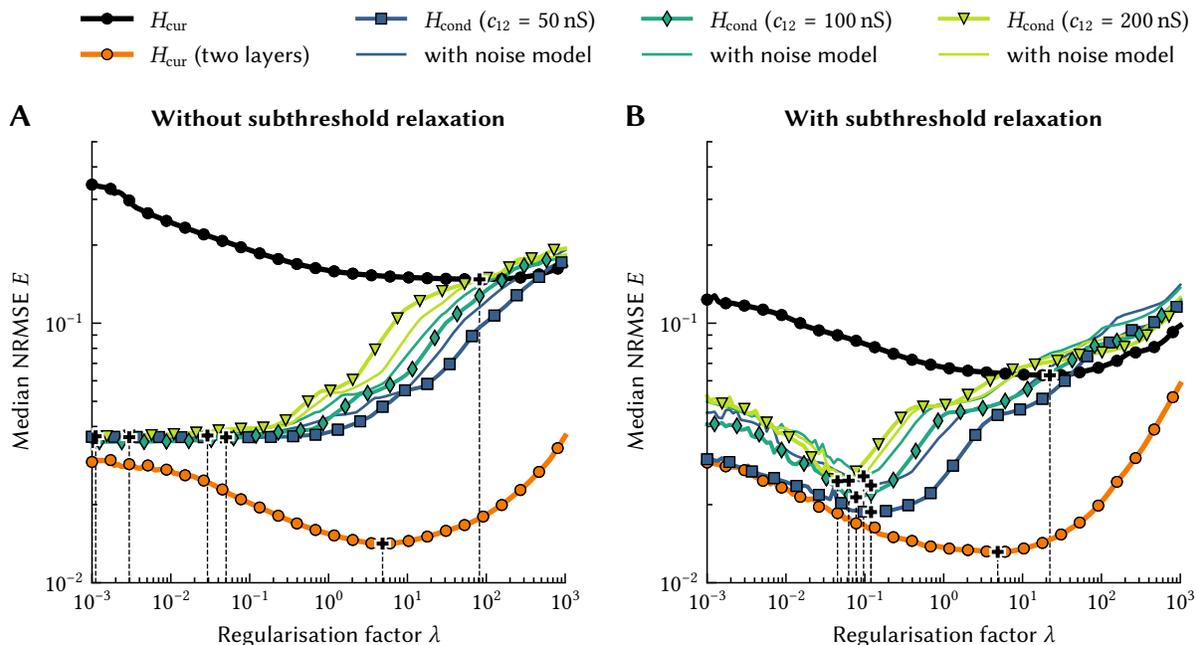


Figure B.2: Regularisation sweep for the theoretical analysis of the two-compartment LIF nonlinearity (Section 3.4.4). Depicted lines are the median error over $N = 128$ experiments for random 2D-functions with spatial filter $\rho^{-1} = 0.5$. **(A)** Is without subthreshold relaxation, **(B)** with subthreshold relaxation.

B.2 Additional Data for Chapter 3

This section contains additional tables and figures for the experiments conducted in Chapter 3. Table B.1 lists neuron parameters used throughout Sections 3.4 and 3.5 unless explicitly stated differently. The model parameters derived in Section 3.4.2 are given in Table B.2; functions used in the two-compartment LIF network experiment are given in Table B.3. Figure B.5 depicts rate approximation errors for more complex n -LIF neurons without calibration.

Extended results for our two-compartment LIF network experiments are provided in Table B.4. All error values correspond to the NRMSE (relative to the RMS of the target function). The error E_{model} corresponds to the static decoding errors assuming the dendritic model is accurate; the error E_{net} is the decoding error after passing the input through the entire network. The given errors are the mean and standard deviation over $N = 100$ experiments. Also refer to Table 3.3 for a more detailed legend.

B.2.1 Regularisation Factor and Pre-Filter Sweep

The regularisation factors λ used in the theoretical analysis of the two-compartment LIF nonlinearity (Section 3.4.4) were determined by performing a parameter sweep with a spatial low-pass filter coefficient of $\rho^{-1} = 0.5$. Results of this sweep are depicted in Figure B.2.

Table B.1: Neuron and synaptic parameters for the two-compartment LIF experiments in Section 3.4.

Parameter	Symbol	Value	Parameter	Symbol	Value
<i>Superthreshold dynamics</i>			<i>Reversal potentials</i>		
Threshold potential	v_{th}	-50 mV	Exc. reversal potential	E_E	20 mV
Spike potential	v_{spike}	20 mV	Resting potential	E_L	-65 mV
Refractory period	τ_{ref}	2 ms	Inh. reversal potential	E_I	-75 mV
Spike period	τ_{spike}	1 ms			
Reset potential	v_{reset}	-65 mV			
Relaxation target threshold	J_{th}	0.56 nA			
<i>Synaptic time-constants</i>			<i>Conductances</i>		
Exc. synapse time-constant	$\tau_{\text{syn,E}}$	5 ms	Coupling conductance	c_{12}	100 nS
Inh. synapse time-constant	$\tau_{\text{syn,I}}$	10 ms	Leak conductances	$g_{L,1}, g_{L,2}$	50 nS
Membrane capacitances	$C_{m,1}, C_{m,2}$	1 nF			

Table B.2: Fitted two-compartment LIF dendritic nonlinearity model parameters $b_0, b_1, b_2, a_0, a_1, a_2$ for Section 3.4.2. J_{max} and J_{min} are the predicted absolute maximum/minimum somatic current (cf. eq. 3.29). The parameters α, β characterise the rectified linear unit used as neuron nonlinearity $G[J]$ in the spike noise experiment.

	Unit	Before fitting			After fitting model parameters					
		Theoretical estimate (i)			No spike noise (ii)			With spike noise (iii)		
		c_{12}			c_{12}			c_{12}		
		50 nS	100 nS	200 nS	50 nS	100 nS	200 nS	50 nS	100 nS	200 nS
b_0	μS	-4.8	-4.8	-4.8	-19.5	-18.8	-17.1	-26.3	-20.7	-17.1
b_1		1000.0	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0
b_2		-225.8	-225.8	-225.8	-425.5	-376.0	-352.3	-487.5	-368.3	-307.6
a_0	mV^{-1}	25.8	19.4	16.1	15.7	9.0	8.3	5.9	4.2	5.3
a_1	nA^{-1}	258.1	129.0	64.5	296.4	193.0	113.3	350.7	260.6	185.1
a_2	nA^{-1}	258.1	129.0	64.5	132.2	41.6	18.6	26.8	7.0	17.1
J_{max}	nA	3.9	7.8	15.5	3.4	5.2	8.8	2.9	3.8	5.4
J_{min}	nA	-0.9	-1.8	-3.5	-3.2	-9.0	-18.9	-1.8	-52.3	-18.0
α	$\text{s}^{-1} \text{nA}^{-1}$	/	/	/	/	/	/	51.3	51.5	51.3
β	s^{-1}	/	/	/	/	/	/	-22.8	-25.1	-26.5

Table B.3: Functions analysed in the n -LIF network experiments (cf. Sections 3.4.5 and 3.5.5). The intervals $[0, 1]^2$ and $[-1, 1]^2$ correspond to the function domain; in any case, functions are re-scaled such that they map onto the interval $[-1, 1]$ for $(x_1, x_2) \in [-1, 1]^2$. Red is negative, blue positive.

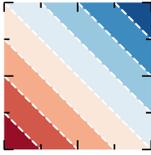
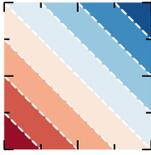
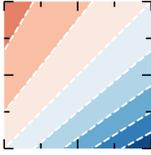
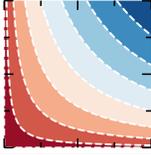
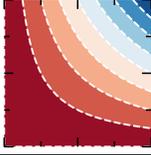
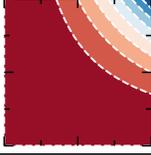
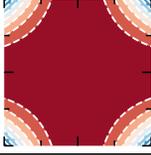
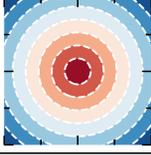
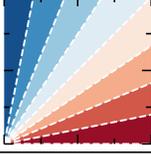
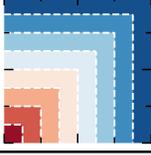
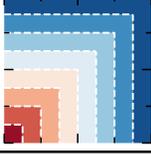
Name	Function $f(x_1, x_2)$	RMS		Contour	
		$[0, 1]^2$	$[-1, 1]^2$	$[0, 1]^2$	$[-1, 1]^2$
Addition	$\frac{1}{2}(x_1 + x_2)$	0.4082	0.4082		
Controlled shunting	$\frac{1 + x_1}{2(1 + x_2)}$	0.2953	/		/
Multiplication with square root	$\sqrt{x_1 x_2}$	0.4714	/		/
Multiplication	$x_1 x_2$	0.6667	0.3333		
Squared multiplication	$x_1^2 x_2^2$	0.8459	0.8459		
Norm	$\sqrt{\frac{1}{2}(x_1^2 + x_2^2)}$	0.4111	0.4111		
Arctan	$\frac{1}{\pi} \arctan\left(\frac{x_2}{x_1}\right)$	0.5293	0.5745		
Maximum	$\max\{x_1, x_2\}$	0.5774	0.5774		

Table B.4: Complete results for the benchmark experiment (see text for a description). † With subthreshold relaxation.

Experiment setup	Target Functions								
	$x_1 + x_2$	$x_1 \times x_2$	$\sqrt{x_1 \times x_2}$	$(x_1 \times x_2)^2$	$x_1/(1 + x_2)$	$\ (x_1, x_2)\ $	$\text{atan}(x_1, x_2)$	$\max(x_1, x_2)$	
standard	E_{model}	2.0 ± 0.3%	24.8 ± 0.9%	24.2 ± 1.2%	21.6 ± 0.6%	14.7 ± 0.7%	18.7 ± 0.5%	20.4 ± 0.8%	36.0 ± 0.9%
	E_{net}	5.7 ± 0.2%	25.8 ± 1.2%	24.9 ± 1.8%	23.1 ± 0.7%	15.0 ± 0.7%	18.6 ± 0.6%	17.8 ± 1.0%	34.5 ± 1.1%
	E_{net}	2.1 ± 0.3%	13.9 ± 1.0%	12.5 ± 1.3%	14.6 ± 0.8%	9.1 ± 0.9%	9.8 ± 0.9%	13.8 ± 1.1%	23.0 ± 1.4%
standard†	E_{model}	5.8 ± 0.3%	15.5 ± 1.2%	14.7 ± 1.7%	16.2 ± 0.9%	10.1 ± 0.8%	10.8 ± 0.9%	12.4 ± 1.0%	22.6 ± 1.4%
	E_{net}	2.3 ± 0.3%	2.4 ± 0.3%	7.7 ± 0.7%	3.0 ± 0.5%	3.4 ± 0.5%	2.8 ± 0.3%	11.9 ± 1.3%	4.1 ± 0.7%
	E_{net}	9.9 ± 0.4%	9.3 ± 0.3%	13.1 ± 0.9%	9.6 ± 0.5%	9.3 ± 0.7%	9.7 ± 0.4%	13.4 ± 0.9%	10.2 ± 0.5%
two layers	E_{model}	2.2 ± 0.3%	2.2 ± 0.3%	7.2 ± 0.7%	2.6 ± 0.5%	3.2 ± 0.5%	2.7 ± 0.3%	10.1 ± 1.3%	3.7 ± 0.6%
	E_{net}	9.9 ± 0.4%	9.2 ± 0.3%	12.6 ± 0.9%	9.3 ± 0.5%	9.2 ± 0.6%	9.7 ± 0.4%	12.5 ± 0.8%	10.0 ± 0.4%
	E_{net}	2.2 ± 0.3%	2.2 ± 0.3%	7.2 ± 0.7%	2.6 ± 0.5%	3.2 ± 0.5%	2.7 ± 0.3%	10.1 ± 1.3%	3.7 ± 0.6%
two layers†	E_{model}	9.9 ± 0.4%	9.2 ± 0.3%	12.6 ± 0.9%	9.3 ± 0.5%	9.2 ± 0.6%	9.7 ± 0.4%	12.5 ± 0.8%	10.0 ± 0.4%
	E_{net}	4.0 ± 0.6%	4.7 ± 1.1%	8.0 ± 1.3%	6.7 ± 1.3%	3.1 ± 0.6%	5.9 ± 0.6%	9.5 ± 1.7%	18.7 ± 2.0%
	E_{net}	5.7 ± 0.6%	6.6 ± 1.3%	9.9 ± 1.7%	8.4 ± 1.3%	4.7 ± 0.5%	7.4 ± 0.7%	9.1 ± 2.1%	18.8 ± 2.0%
standard	E_{model}	2.7 ± 0.4%	3.0 ± 0.4%	6.1 ± 0.7%	3.6 ± 0.8%	3.0 ± 0.6%	3.9 ± 0.5%	8.7 ± 1.6%	12.8 ± 1.4%
	E_{net}	4.3 ± 0.3%	4.4 ± 0.6%	7.1 ± 0.9%	5.1 ± 0.9%	4.5 ± 0.4%	5.3 ± 0.5%	7.2 ± 1.8%	12.9 ± 1.5%
	E_{net}	1.7 ± 0.3%	4.9 ± 1.4%	7.6 ± 1.6%	6.8 ± 1.1%	2.2 ± 0.6%	4.2 ± 0.4%	8.2 ± 1.5%	19.8 ± 2.0%
noise model	E_{model}	7.9 ± 0.9%	5.8 ± 1.2%	6.8 ± 1.4%	8.0 ± 1.3%	11.2 ± 1.7%	9.4 ± 0.8%	7.9 ± 1.3%	18.0 ± 2.0%
	E_{net}	1.8 ± 0.3%	1.8 ± 0.3%	5.5 ± 0.7%	2.1 ± 0.7%	2.5 ± 0.6%	2.7 ± 0.4%	7.9 ± 1.6%	10.4 ± 1.2%
	E_{net}	8.2 ± 0.9%	4.7 ± 0.4%	6.0 ± 0.7%	3.9 ± 0.6%	11.4 ± 1.7%	8.2 ± 0.9%	7.5 ± 1.5%	9.9 ± 1.3%
noise model†	E_{model}	2.1 ± 0.3%	5.2 ± 1.2%	6.9 ± 1.2%	8.3 ± 1.6%	2.7 ± 0.5%	4.0 ± 0.4%	9.3 ± 1.4%	19.2 ± 2.2%
	E_{net}	5.0 ± 0.3%	7.7 ± 1.5%	9.7 ± 2.0%	10.4 ± 1.7%	6.3 ± 0.6%	6.9 ± 0.6%	8.8 ± 1.7%	20.0 ± 2.2%
	E_{net}	2.0 ± 0.3%	3.2 ± 0.4%	5.7 ± 0.7%	5.0 ± 1.1%	2.8 ± 0.6%	3.2 ± 0.5%	9.0 ± 1.7%	13.7 ± 1.5%
standard†	E_{model}	4.7 ± 0.3%	5.4 ± 0.6%	7.7 ± 1.0%	7.2 ± 1.2%	6.1 ± 0.6%	6.0 ± 0.6%	8.3 ± 1.8%	14.9 ± 1.8%
	E_{net}	1.6 ± 0.3%	4.4 ± 1.3%	7.0 ± 1.4%	6.8 ± 1.3%	2.1 ± 0.6%	3.7 ± 0.4%	8.2 ± 1.4%	19.1 ± 2.0%
	E_{net}	9.8 ± 1.1%	7.9 ± 1.0%	7.2 ± 1.3%	11.1 ± 1.2%	15.0 ± 2.2%	12.0 ± 1.2%	11.3 ± 1.2%	18.8 ± 2.0%
noise model	E_{model}	3.5 ± 0.4%	6.2 ± 0.6%	7.7 ± 0.7%	8.4 ± 0.8%	6.0 ± 0.7%	7.0 ± 0.6%	10.8 ± 1.4%	19.1 ± 1.2%
	E_{net}	11.2 ± 0.9%	6.9 ± 0.4%	7.6 ± 0.6%	8.8 ± 1.0%	14.8 ± 1.8%	12.1 ± 0.9%	8.1 ± 1.0%	16.3 ± 0.9%
	E_{net}	1.8 ± 0.3%	5.7 ± 1.2%	7.0 ± 1.2%	9.5 ± 1.9%	2.7 ± 0.6%	3.8 ± 0.4%	10.1 ± 1.3%	19.8 ± 2.3%
standard	E_{model}	6.8 ± 0.5%	9.6 ± 1.7%	11.5 ± 2.8%	12.8 ± 2.0%	8.2 ± 0.8%	8.2 ± 0.8%	11.6 ± 2.5%	21.8 ± 2.6%
	E_{net}	1.8 ± 0.3%	3.6 ± 0.5%	5.9 ± 0.7%	5.7 ± 1.2%	2.7 ± 0.6%	3.1 ± 0.4%	9.5 ± 1.9%	14.0 ± 1.7%
	E_{net}	6.5 ± 0.4%	7.3 ± 0.8%	10.5 ± 1.8%	9.6 ± 1.4%	8.0 ± 0.7%	7.3 ± 0.7%	12.2 ± 3.1%	18.2 ± 2.2%
noise model	E_{model}	2.4 ± 0.3%	11.3 ± 1.5%	9.8 ± 1.2%	14.0 ± 1.4%	5.2 ± 0.7%	6.7 ± 0.8%	14.3 ± 1.2%	25.5 ± 1.9%
	E_{net}	9.7 ± 1.3%	14.4 ± 1.3%	9.4 ± 0.8%	17.3 ± 1.7%	14.4 ± 2.6%	12.8 ± 1.5%	11.6 ± 1.1%	23.8 ± 1.5%
	E_{net}	2.9 ± 0.4%	8.6 ± 0.7%	8.9 ± 0.8%	11.0 ± 0.7%	6.6 ± 0.7%	7.3 ± 0.6%	12.0 ± 1.2%	21.1 ± 1.2%
noise model†	E_{model}	10.4 ± 1.0%	9.1 ± 0.5%	8.3 ± 0.8%	11.5 ± 0.9%	14.9 ± 2.0%	12.1 ± 1.0%	9.3 ± 0.9%	18.5 ± 0.9%
	E_{net}	2.9 ± 0.4%	8.6 ± 0.7%	8.9 ± 0.8%	11.0 ± 0.7%	6.6 ± 0.7%	7.3 ± 0.6%	12.0 ± 1.2%	21.1 ± 1.2%
	E_{net}	10.4 ± 1.0%	9.1 ± 0.5%	8.3 ± 0.8%	11.5 ± 0.9%	14.9 ± 2.0%	12.1 ± 1.0%	9.3 ± 0.9%	18.5 ± 0.9%

Similarly, the regularisation factors used in the network experiment (Section 3.4.5) were determined independently for each network setup by sweeping over the regularisation factor λ , and choosing the λ that minimises the network error E_{net} when computing multiplication. We combine this sweep with the “pre-filter experiment” described below; results can be found in Figures B.3 and B.4. The final parameters used in Section 3.4.5 are listed in Table B.5.

The purpose of the pre-filter experiment is the following. As mentioned in the main text, the dendritic compartment in the two-compartment model can be interpreted as an additional first order low-pass filter with time-constant $C_m g_L^{-1}$ (cf. Table B.1). This low-pass filter reduces the amplitude of high-frequency transients in the input signal and may thus contribute to the reduction of the final approximation error E_{net} .

To explore the effects of low-pass filters in the network, we perform an additional experiment in which we introduced a first-order low-pass filter (the “pre-filter”) located at the input of the target population (for single-layer networks) or the input of the intermediate population (in the case of the two-layer network). By optimising both the regularisation factor λ and the filter time-constant τ , each network setup has the chance to include an additional, optimal low-pass filter. This rules out that the positive effects of the two-compartment neuron are primarily due to the low-pass filter behaviour of the dendritic compartment. Notably, this low-pass filter is not included in the filter-chain for the target signal used to compute the estimation error. Hence, the chosen filter time-constant must balance between suppressing noise in the input and not dampening high-frequency components in the target signal.

As depicted in Figures B.3 and B.4, the pre-filter does not result in any appreciable improvement in error for single layer networks with standard LIF neurons. For both the two-layer LIF network, and the two-compartment LIF neurons without noise model, the error is minimised for a filter coefficient of about 10 ms to 20 ms. As one would expect, when using the two-compartment LIF noise model, adding a pre-filter is more detrimental than useful.

Function approximation errors are given in (cf. Table B.6). Again, we provide the mean and standard deviation of the model and network errors for $N = 100$ samples; see above for a complete description of the table. The pre-filter reduces the error by only about 1-2% across all network setups, which is far less than the observed reduction in error when using the two-compartment model compared to the standard LIF neuron model. We conclude that, while additional low-pass filters in the network can help to reduce the overall error, the reduction in error is too small to be an explanation for the smaller errors produced by the two-compartment model.

Appendix B. Additional Data

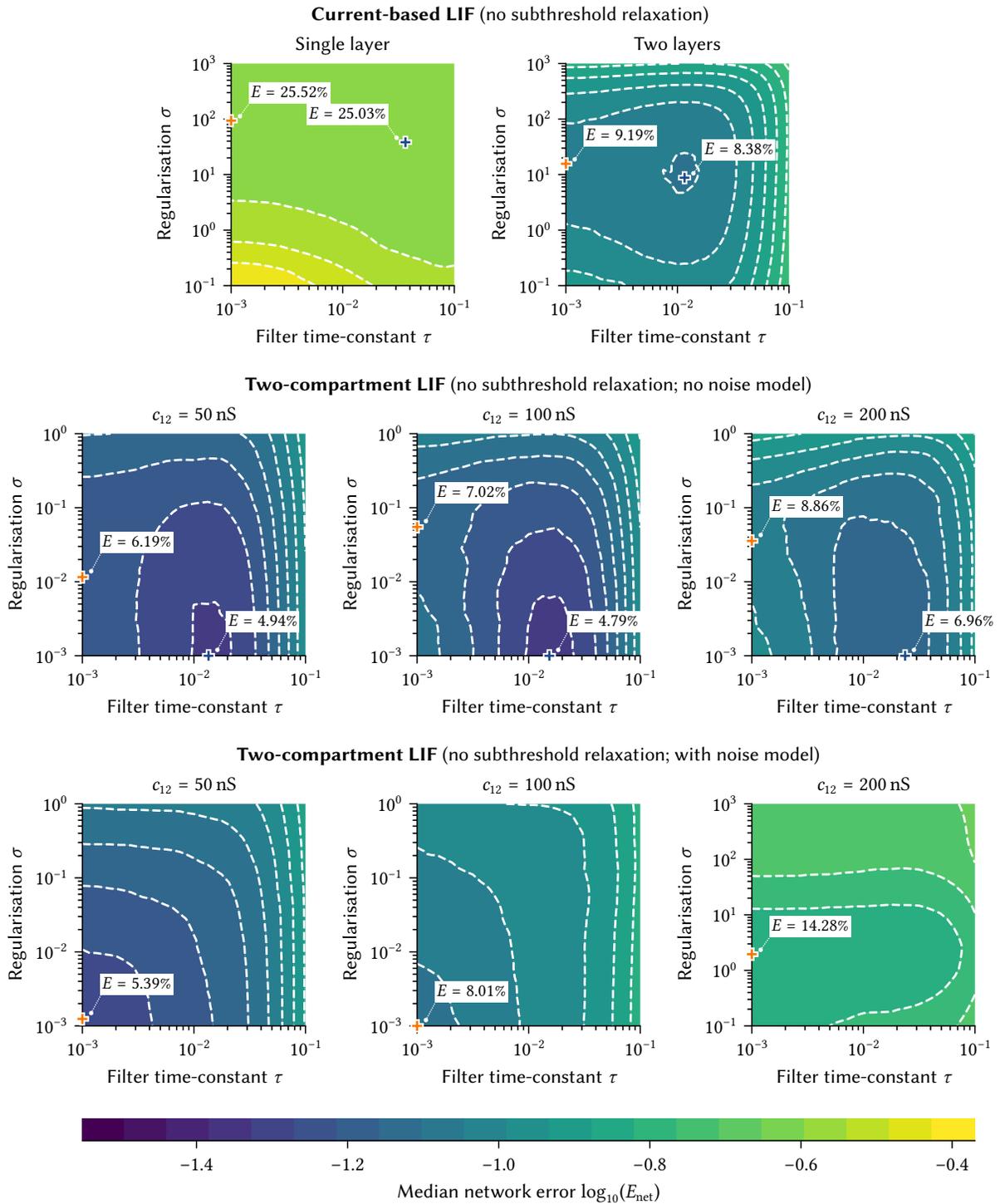


Figure B.3: Regularisation factor and pre-filter time-constant sweeps without subthreshold relaxation. Contour plots are based on the median network error E_{net} over 32 samples on a 32×33 grid. Blue and orange crosses indicate the point with minimum error with and without pre-filter, respectively.

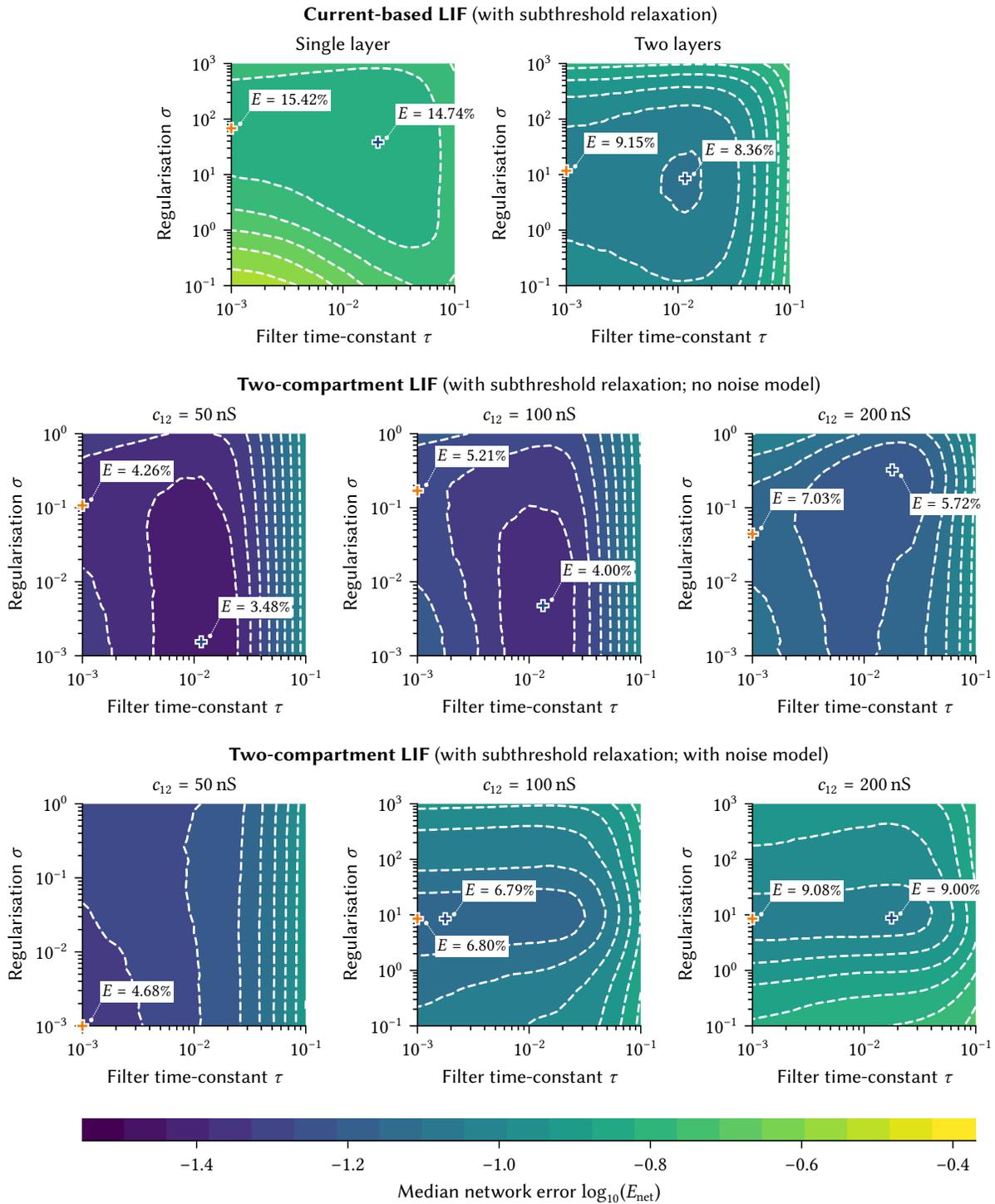


Figure B.4: Regularisation factor and pre-filter time-constant sweep with subthreshold relaxation. Contour plots are based on the median network error E_{net} over 32 samples on a 32×33 grid. Blue and orange crosses indicate the point with minimum error with and without pre-filter, respectively.

Table B.5: Regularisation factors and pre-filter time-constants for the two-compartment LIF network experiments. See text for a description. †With subthreshold relaxation.

Experiment setup		No pre-filter		With pre-filter		
		σ	E_{net}	σ	τ	E_{net}
LIF	standard	9.4×10^1	25.5%	3.8×10^1	36.5 ms	25.0%
	standard†	6.8×10^1	15.4%	3.8×10^1	20.6 ms	14.7%
	two layers	1.6×10^1	9.2%	8.8×10^0	11.6 ms	8.4%
	two layers†	1.2×10^1	9.1%	8.5×10^0	11.6 ms	8.4%
Two comp. LIF $c_{12} = 50 \text{ nS}$	standard	1.2×10^{-2}	6.2%	1.0×10^{-3}	13.5 ms	4.9%
	standard†	1.1×10^{-1}	4.3%	1.5×10^{-3}	11.6 ms	3.5%
	noise model	1.2×10^{-3}	5.4%	1.2×10^{-3}	1.0 ms	5.4%
	noise model†	1.0×10^{-3}	4.7%	1.0×10^{-3}	1.0 ms	4.7%
Two comp. LIF $c_{12} = 100 \text{ nS}$	standard	5.5×10^{-2}	7.0%	1.0×10^{-3}	15.3 ms	4.8%
	standard†	1.7×10^{-1}	5.2%	4.8×10^{-3}	13.3 ms	4.0%
	noise model	1.0×10^{-3}	8.0%	1.0×10^{-3}	1.0 ms	8.0%
	noise model†	8.5×10^0	6.8%	8.5×10^0	1.8 ms	6.8%
Two comp. LIF $c_{12} = 200 \text{ nS}$	standard	3.6×10^{-2}	8.9%	1.0×10^{-3}	23.7 ms	7.0%
	standard†	4.4×10^{-2}	7.0%	3.2×10^{-1}	17.9 ms	5.7%
	noise model	2.0×10^0	14.3%	2.0×10^0	1.0 ms	14.3%
	noise model†	8.5×10^0	9.1%	8.8×10^0	17.6 ms	9.0%

Table B.6: Complete results for the two-compartment LIF benchmark experiment with pre-filter. †With subthreshold relaxation.

Experiment setup		Target Functions							
		$x_1 + x_2$	$x_1 \times x_2$	$\sqrt{x_1 \times x_2}$	$(x_1 \times x_2)^2$	$x_1/(1 + x_2)$	$\ (x_1, x_2)\ $	$\text{atan}(x_1, x_2)$	$\max(x_1, x_2)$
standard	E_{model}	$1.9 \pm 0.3\%$	$25.0 \pm 0.9\%$	$24.2 \pm 1.2\%$	$21.8 \pm 0.6\%$	$14.8 \pm 0.7\%$	$18.8 \pm 0.5\%$	$20.4 \pm 0.8\%$	$36.3 \pm 1.0\%$
	E_{net}	$8.6 \pm 0.1\%$	$25.2 \pm 1.2\%$	$24.1 \pm 1.8\%$	$22.9 \pm 0.7\%$	$15.4 \pm 0.7\%$	$20.8 \pm 0.6\%$	$19.4 \pm 0.9\%$	$35.2 \pm 1.1\%$
standard†	E_{model}	$2.0 \pm 0.3\%$	$13.9 \pm 1.0\%$	$12.5 \pm 1.3\%$	$14.6 \pm 0.8\%$	$9.1 \pm 0.9\%$	$9.8 \pm 0.9\%$	$13.7 \pm 1.2\%$	$23.0 \pm 1.4\%$
	E_{net}	$6.3 \pm 0.2\%$	$14.9 \pm 1.2\%$	$13.9 \pm 1.7\%$	$15.8 \pm 0.9\%$	$9.9 \pm 0.8\%$	$11.7 \pm 0.8\%$	$12.6 \pm 1.0\%$	$22.6 \pm 1.4\%$
two layers	E_{model}	$2.2 \pm 0.3\%$	$2.2 \pm 0.2\%$	$7.4 \pm 0.7\%$	$2.8 \pm 0.4\%$	$3.2 \pm 0.5\%$	$2.7 \pm 0.3\%$	$11.4 \pm 1.3\%$	$3.8 \pm 0.7\%$
	E_{net}	$8.7 \pm 0.3\%$	$8.4 \pm 0.3\%$	$12.1 \pm 0.8\%$	$8.8 \pm 0.5\%$	$7.8 \pm 0.6\%$	$8.6 \pm 0.3\%$	$12.1 \pm 0.9\%$	$9.1 \pm 0.5\%$
two layers†	E_{model}	$2.2 \pm 0.3\%$	$2.1 \pm 0.3\%$	$7.1 \pm 0.7\%$	$2.5 \pm 0.5\%$	$3.2 \pm 0.5\%$	$2.6 \pm 0.3\%$	$9.8 \pm 1.3\%$	$3.5 \pm 0.6\%$
	E_{net}	$8.7 \pm 0.3\%$	$8.4 \pm 0.3\%$	$11.6 \pm 0.8\%$	$8.5 \pm 0.4\%$	$7.7 \pm 0.6\%$	$8.7 \pm 0.4\%$	$11.1 \pm 0.8\%$	$9.0 \pm 0.4\%$
standard	E_{model}	$4.0 \pm 0.6\%$	$4.4 \pm 1.1\%$	$8.0 \pm 1.3\%$	$6.4 \pm 1.2\%$	$3.1 \pm 0.6\%$	$5.9 \pm 0.6\%$	$9.4 \pm 1.7\%$	$18.6 \pm 2.0\%$
	E_{net}	$6.1 \pm 0.5\%$	$5.1 \pm 1.1\%$	$8.3 \pm 1.3\%$	$7.2 \pm 1.4\%$	$4.7 \pm 0.5\%$	$7.9 \pm 0.6\%$	$7.7 \pm 1.7\%$	$17.2 \pm 1.9\%$
standard†	E_{model}	$2.7 \pm 0.4\%$	$2.7 \pm 0.4\%$	$6.1 \pm 0.7\%$	$3.1 \pm 0.7\%$	$2.8 \pm 0.6\%$	$3.7 \pm 0.5\%$	$8.4 \pm 1.5\%$	$11.9 \pm 1.4\%$
	E_{net}	$4.9 \pm 0.3\%$	$3.6 \pm 0.5\%$	$6.4 \pm 0.7\%$	$4.0 \pm 0.8\%$	$4.1 \pm 0.5\%$	$5.9 \pm 0.5\%$	$6.3 \pm 1.3\%$	$11.1 \pm 1.3\%$
noise model	E_{model}	$1.7 \pm 0.3\%$	$4.9 \pm 1.4\%$	$7.6 \pm 1.6\%$	$6.8 \pm 1.1\%$	$2.2 \pm 0.6\%$	$4.2 \pm 0.4\%$	$8.2 \pm 1.5\%$	$19.8 \pm 2.0\%$
	E_{net}	$8.0 \pm 0.9\%$	$5.8 \pm 1.2\%$	$6.8 \pm 1.4\%$	$8.0 \pm 1.2\%$	$11.3 \pm 1.7\%$	$9.5 \pm 0.8\%$	$7.9 \pm 1.2\%$	$18.0 \pm 2.0\%$
noise model†	E_{model}	$1.8 \pm 0.3\%$	$1.8 \pm 0.3\%$	$5.5 \pm 0.7\%$	$2.1 \pm 0.7\%$	$2.5 \pm 0.6\%$	$2.7 \pm 0.4\%$	$7.9 \pm 1.6\%$	$10.4 \pm 1.2\%$
	E_{net}	$8.3 \pm 0.9\%$	$4.7 \pm 0.4\%$	$6.0 \pm 0.7\%$	$4.0 \pm 0.6\%$	$11.5 \pm 1.7\%$	$8.3 \pm 0.9\%$	$7.5 \pm 1.4\%$	$9.9 \pm 1.2\%$
standard	E_{model}	$2.0 \pm 0.3\%$	$4.1 \pm 1.1\%$	$6.8 \pm 1.3\%$	$6.7 \pm 1.5\%$	$2.6 \pm 0.5\%$	$3.9 \pm 0.5\%$	$8.4 \pm 1.3\%$	$18.3 \pm 2.1\%$
	E_{net}	$5.8 \pm 0.3\%$	$5.0 \pm 1.1\%$	$7.2 \pm 1.2\%$	$7.4 \pm 1.7\%$	$6.6 \pm 1.1\%$	$8.4 \pm 0.8\%$	$7.6 \pm 1.1\%$	$17.2 \pm 2.0\%$
standard†	E_{model}	$1.9 \pm 0.3\%$	$2.5 \pm 0.4\%$	$5.7 \pm 0.7\%$	$3.6 \pm 0.8\%$	$2.6 \pm 0.6\%$	$2.7 \pm 0.4\%$	$8.3 \pm 1.7\%$	$11.8 \pm 1.5\%$
	E_{net}	$5.4 \pm 0.3\%$	$4.2 \pm 0.5\%$	$6.5 \pm 0.7\%$	$4.5 \pm 1.0\%$	$5.8 \pm 1.1\%$	$7.1 \pm 0.7\%$	$7.2 \pm 1.1\%$	$11.4 \pm 1.4\%$
noise model	E_{model}	$1.6 \pm 0.3\%$	$4.4 \pm 1.3\%$	$7.0 \pm 1.4\%$	$6.8 \pm 1.3\%$	$2.1 \pm 0.6\%$	$3.7 \pm 0.4\%$	$8.2 \pm 1.4\%$	$19.1 \pm 2.0\%$
	E_{net}	$9.9 \pm 1.1\%$	$8.1 \pm 0.9\%$	$7.2 \pm 1.2\%$	$11.2 \pm 1.1\%$	$15.1 \pm 2.2\%$	$12.1 \pm 1.2\%$	$11.4 \pm 1.2\%$	$18.8 \pm 2.0\%$
noise model†	E_{model}	$3.5 \pm 0.4\%$	$6.2 \pm 0.6\%$	$7.7 \pm 0.7\%$	$8.4 \pm 0.8\%$	$6.0 \pm 0.7\%$	$7.0 \pm 0.6\%$	$10.8 \pm 1.4\%$	$19.1 \pm 1.2\%$
	E_{net}	$11.4 \pm 0.9\%$	$6.9 \pm 0.4\%$	$7.7 \pm 0.6\%$	$8.9 \pm 1.0\%$	$15.0 \pm 1.8\%$	$12.4 \pm 0.9\%$	$8.2 \pm 1.0\%$	$16.3 \pm 0.9\%$
standard	E_{model}	$1.8 \pm 0.3\%$	$4.3 \pm 1.1\%$	$6.7 \pm 1.3\%$	$7.4 \pm 1.7\%$	$2.5 \pm 0.6\%$	$3.6 \pm 0.4\%$	$8.7 \pm 1.4\%$	$18.5 \pm 2.2\%$
	E_{net}	$7.8 \pm 0.2\%$	$7.2 \pm 0.9\%$	$7.7 \pm 1.0\%$	$9.9 \pm 1.5\%$	$9.5 \pm 1.8\%$	$11.2 \pm 1.3\%$	$10.2 \pm 1.4\%$	$18.5 \pm 1.8\%$
standard†	E_{model}	$1.9 \pm 0.3\%$	$5.4 \pm 0.7\%$	$6.4 \pm 0.8\%$	$8.4 \pm 1.2\%$	$3.4 \pm 0.5\%$	$4.2 \pm 0.5\%$	$10.4 \pm 1.7\%$	$17.0 \pm 1.7\%$
	E_{net}	$6.9 \pm 0.2\%$	$5.9 \pm 0.7\%$	$7.2 \pm 0.8\%$	$8.7 \pm 1.5\%$	$9.2 \pm 1.3\%$	$10.2 \pm 0.9\%$	$9.1 \pm 1.2\%$	$16.0 \pm 1.5\%$
noise model	E_{model}	$2.4 \pm 0.3\%$	$11.3 \pm 1.5\%$	$9.8 \pm 1.2\%$	$14.0 \pm 1.4\%$	$5.2 \pm 0.7\%$	$6.7 \pm 0.8\%$	$14.3 \pm 1.2\%$	$25.5 \pm 1.9\%$
	E_{net}	$9.8 \pm 1.3\%$	$14.5 \pm 1.3\%$	$9.4 \pm 0.8\%$	$17.3 \pm 1.7\%$	$14.4 \pm 2.6\%$	$12.9 \pm 1.5\%$	$11.6 \pm 1.1\%$	$23.8 \pm 1.5\%$
noise model†	E_{model}	$2.9 \pm 0.4\%$	$8.6 \pm 0.7\%$	$8.9 \pm 0.8\%$	$11.1 \pm 0.7\%$	$6.6 \pm 0.7\%$	$7.4 \pm 0.6\%$	$12.0 \pm 1.2\%$	$21.2 \pm 1.2\%$
	E_{net}	$13.1 \pm 1.3\%$	$9.1 \pm 0.5\%$	$9.4 \pm 0.9\%$	$11.6 \pm 0.9\%$	$16.7 \pm 2.5\%$	$15.0 \pm 1.3\%$	$10.3 \pm 0.8\%$	$19.0 \pm 1.0\%$

Appendix B. Additional Data

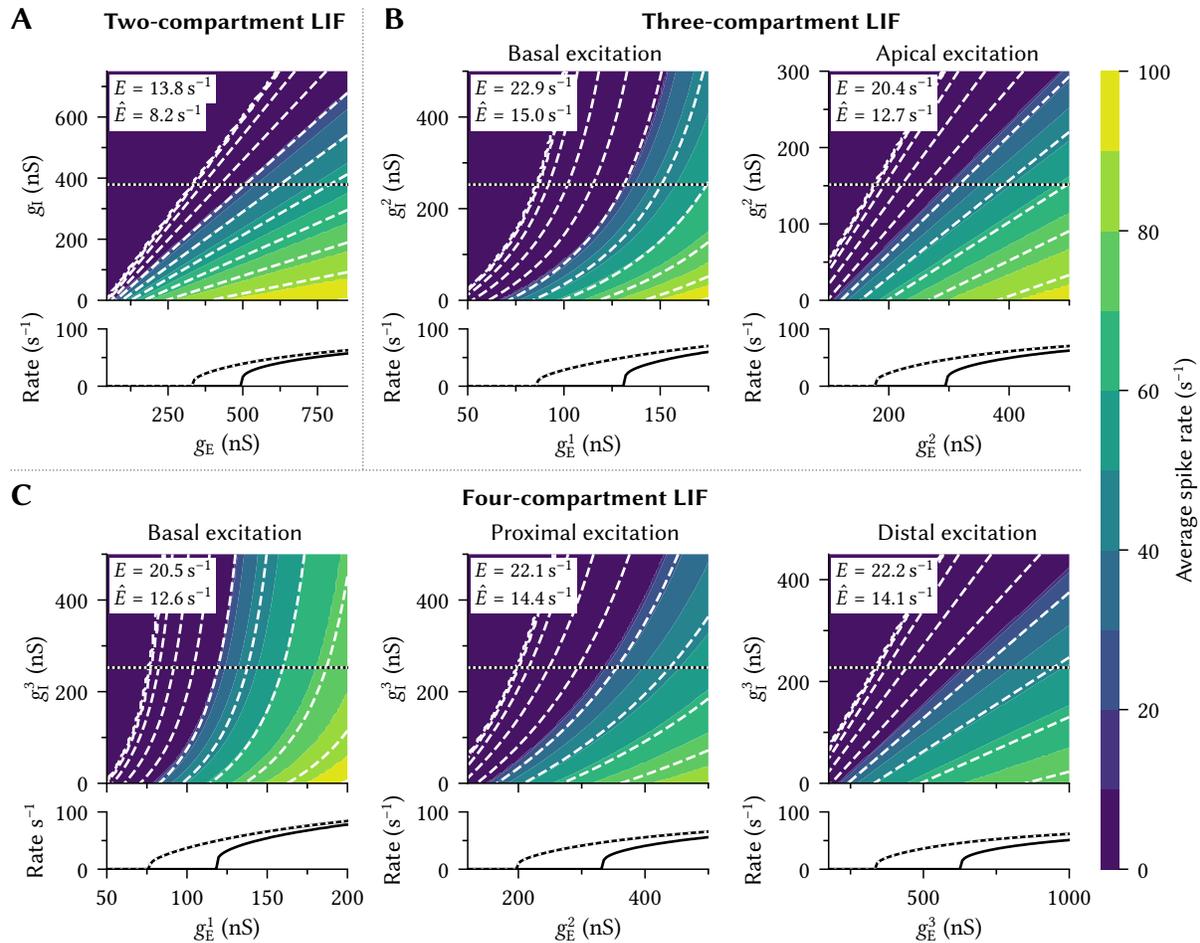


Figure B.5: Uncalibrated rate predictions for different n -LIF neurons. Same data as in Figure 3.39, but without calibration of the neuron. Calibration reduces the prediction errors by a factor of two to four.

Table B.7: Calibrated and conditioned reduced n -LIF system matrices for the experiments in Section 3.5. Letters correspond to the neurons depicted in Figure 3.22: (B) one-compartment with conductance channels, (C) two, (D) three, and (E) four compartments.

Model	$\tilde{\mathbf{a}}'$	$\tilde{\mathbf{A}}'$	$\tilde{\mathbf{b}}'$	$\tilde{\mathbf{B}}'$	$\tilde{\mathbf{L}}$	$\tilde{\mathbf{c}}$
Before calibration						
(B)	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 57.5 & -17.5 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$
(C)	$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 20 & 20 \end{bmatrix}$	$\begin{bmatrix} 0 \\ -0.375 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 57.5 & -17.5 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$
(D)	$\begin{bmatrix} 1 \\ 1.5 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 10 & 10 & 0 \\ 0 & 0 & 20 \end{bmatrix}$	$\begin{bmatrix} 0 \\ -0.375 \\ -0.375 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 57.5 & -17.5 & 0 \\ 0 & 0 & 57.5 & -17.5 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & -4 \\ 0 & -2 & 4 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$
(E)	$\begin{bmatrix} 1 \\ 1.5 \\ 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 10 & 10 & 0 & 0 \\ 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 20 \end{bmatrix}$	$\begin{bmatrix} -0.375 \\ -0.375 \\ -0.375 \\ -0.375 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 57.5 & -17.5 & 0 & 0 \\ 0 & 0 & 57.5 & -17.5 \\ 0 & 0 & 0 & 57.5 & -17.5 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & -4 & 0 \\ 0 & -2 & 14 & -10 \\ 0 & 0 & -10 & 10 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$
After calibration						
(B)	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -0.82 \\ -2.67 \end{bmatrix}$	$\begin{bmatrix} 64.65 & -23.08 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$
(C)	$\begin{bmatrix} 1 \\ 0.59 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 10.50 & 2.90 \end{bmatrix}$	$\begin{bmatrix} -2.67 \\ 0.79 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 55.97 & -11.67 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$
(D)	$\begin{bmatrix} 1 \\ 0.82 \\ 1.40 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 9.75 & 4.94 & 0 \\ 0 & 0 & 19.07 & 16.87 \end{bmatrix}$	$\begin{bmatrix} -0.53 \\ -0.91 \\ 0.10 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 57.63 & -24.93 & 0 \\ 0 & 0 & 55.42 & -16.06 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & -4 \\ 0 & -2 & 4 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$
(E)	$\begin{bmatrix} 1 \\ 0.68 \\ 2.69 \\ 0.54 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 10.20 & 4.66 & 0 & 0 \\ 0 & 0 & 18.35 & 15.12 \\ 0 & 0 & 0 & 15.11 & 10.83 \end{bmatrix}$	$\begin{bmatrix} -0.45 \\ -1.95 \\ -0.20 \\ 2.40 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 58.34 & -26.18 & 0 & 0 \\ 0 & 0 & 56.47 & -15.81 \\ 0 & 0 & 0 & 51.06 & -11.89 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & -4 & 0 \\ 0 & -2 & 14 & -10 \\ 0 & 0 & -10 & 10 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$

Appendix B. Additional Data

Table B.8: Detailed function approximation errors for spiking networks using various n -LIF neurons. Top half of the table contains the static function approximation errors E_{model} . These are the network errors obtained assuming that there is no spike noise and that our rate approximations are correct. See Table 3.4 for more details. Bottom half contains the minimum network errors E_{net} over 100 trials.

Function	Domain	Neuron				
		LIF		n -LIF		
		standard	two layers	$n = 2$	$n = 3$	$n = 4$
Static function approximation errors						
<i>Standard parameters</i> ($\lambda = 10^{-3}$; $\xi_0 \in [-0.95, 0.95]$; with Dale's principle, $p_{\text{inh}} = 30\%$)						
$x_1 + x_2$	$[-1, 1]^2$	$2.1 \pm 0.6\%$	$2.2 \pm 0.4\%$	$2.7 \pm 0.6\%$	$2.2 \pm 0.6\%$	$2.2 \pm 0.6\%$
$x_1/(1 + x_2)$	$[0, 1]^2$	$8.9 \pm 0.8\%$	$3.3 \pm 0.4\%$	$3.8 \pm 1.0\%$	$3.1 \pm 1.0\%$	$3.2 \pm 1.0\%$
$\sqrt{x_1 \times x_2}$	$[0, 1]^2$	$12.4 \pm 1.1\%$	$7.5 \pm 0.5\%$	$6.1 \pm 0.8\%$	$5.1 \pm 0.8\%$	$5.1 \pm 0.8\%$
$x_1 \times x_2$	$[0, 1]^2$	$14.6 \pm 0.9\%$	$2.3 \pm 0.3\%$	$2.8 \pm 0.5\%$	$2.4 \pm 0.4\%$	$2.4 \pm 0.4\%$
$x_1 \times x_2$	$[-1, 1]^2$	$107.1 \pm 1.8\%$	$5.7 \pm 0.5\%$	$77.2 \pm 3.6\%$	$39.0 \pm 4.6\%$	$38.2 \pm 5.0\%$
$(x_1 \times x_2)^2$	$[-1, 1]^2$	$15.3 \pm 1.6\%$	$8.3 \pm 1.0\%$	$16.4 \pm 1.8\%$	$8.7 \pm 1.9\%$	$8.2 \pm 1.8\%$
$\ (x_1, x_2)\ $	$[-1, 1]^2$	$9.0 \pm 0.6\%$	$5.7 \pm 0.8\%$	$6.6 \pm 0.5\%$	$4.0 \pm 0.6\%$	$4.2 \pm 0.6\%$
$\text{atan}(x_1, x_2)$	$[-1, 1]^2$	$35.2 \pm 2.4\%$	$29.2 \pm 3.3\%$	$15.3 \pm 4.3\%$	$13.1 \pm 4.9\%$	$13.3 \pm 4.4\%$
$\max(x_1, x_2)$	$[-1, 1]^2$	$22.3 \pm 1.1\%$	$3.4 \pm 0.3\%$	$8.0 \pm 0.5\%$	$5.4 \pm 0.5\%$	$5.7 \pm 0.6\%$
<i>Adapted parameters</i> ($\lambda = 10^{-6}$; $\xi_0 \in [-0.95, 0]$; no Dale's principle)						
$x_1 + x_2$	$[-1, 1]^2$	$3.4 \pm 0.4\%$	$3.5 \pm 0.3\%$	$3.9 \pm 0.4\%$	$3.4 \pm 0.4\%$	$3.4 \pm 0.4\%$
$x_1 \times x_2$	$[0, 1]^2$	$20.4 \pm 0.9\%$	$4.7 \pm 0.7\%$	$5.4 \pm 0.6\%$	$5.0 \pm 0.7\%$	$5.0 \pm 0.7\%$
$x_1 \times x_2$	$[-1, 1]^2$	$109.7 \pm 2.6\%$	$6.5 \pm 0.3\%$	$71.6 \pm 4.9\%$	$26.1 \pm 5.0\%$	$24.1 \pm 4.0\%$
Minimum network errors						
<i>Standard parameters</i> ($\lambda = 10^{-3}$; $\xi_0 \in [-0.95, 0.95]$; with Dale's principle, $p_{\text{inh}} = 30\%$)						
$x_1 + x_2$	$[-1, 1]^2$	3.8%	8.4%	3.3%	4.4%	5.9%
$x_1/(1 + x_2)$	$[0, 1]^2$	8.0%	8.4%	3.9%	5.6%	6.3%
$\sqrt{x_1 \times x_2}$	$[0, 1]^2$	12.6%	10.2%	4.5%	5.3%	6.7%
$x_1 \times x_2$	$[0, 1]^2$	15.1%	7.5%	3.7%	3.9%	5.9%
$x_1 \times x_2$	$[-1, 1]^2$	100.0%	11.5%	75.1%	31.3%	29.2%
$(x_1 \times x_2)^2$	$[-1, 1]^2$	15.8%	16.8%	17.0%	10.2%	10.4%
$\ (x_1, x_2)\ $	$[-1, 1]^2$	10.2%	12.6%	5.9%	6.2%	8.6%
$\text{atan}(x_1, x_2)$	$[-1, 1]^2$	37.0%	33.7%	15.0%	17.0%	18.0%
$\max(x_1, x_2)$	$[-1, 1]^2$	21.8%	8.1%	6.7%	6.4%	7.3%
<i>Adapted parameters</i> ($\lambda = 10^{-6}$; $\xi_0 \in [-0.95, 0]$; no Dale's principle)						
$x_1 + x_2$	$[-1, 1]^2$	4.0%	7.1%	4.0%	20.3%	22.8%
$x_1 \times x_2$	$[0, 1]^2$	21.3%	9.0%	4.6%	9.3%	9.8%
$x_1 \times x_2$	$[-1, 1]^2$	99.1%	10.3%	65.0%	14.7%	14.9%

B.3 Additional Data for Chapter 4

B.3.1 Comparison Between the LDN and Modified Fourier Systems

Voelker (2019, Section 6.1.1) notes that the LDN system optimally approximates a time-delay for low-frequency inputs generated by an LTI system of order $p + q = 2q - 1$. However, for our experiments in Section 4.2.5, we used low-pass filtered white noise, which technically consists of infinitely many (albeit severely dampened) frequency components. We found that LTI systems generating a modified Fourier basis can outperform the LDN in this scenario.

As is depicted in Figures B.6 and B.7, we test whether switching to a strictly band-limited inputs $u(t)$ impacts the performance of these two systems. Indeed, using strictly band-limited signals substantially reduces the errors for both the modified Fourier system and the LDN. However, the modified Fourier system still significantly outperforms the LDN.

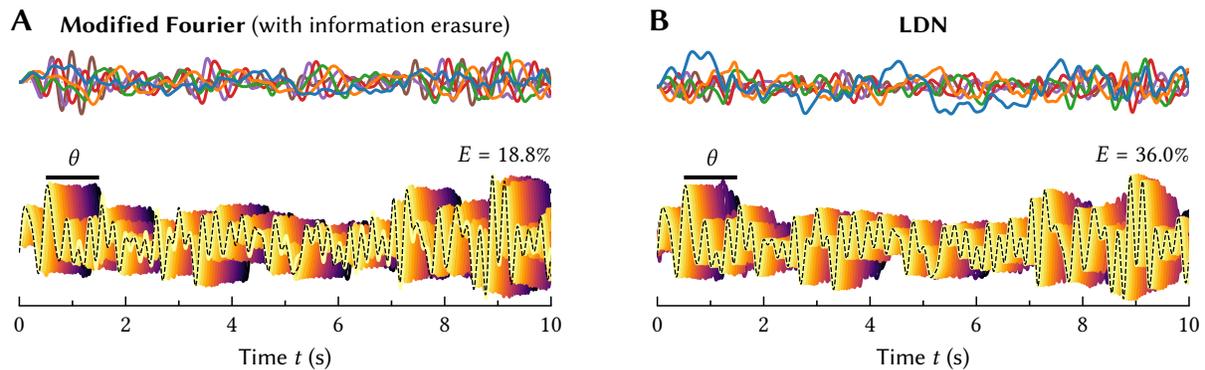


Figure B.6: Example delay decodings for strictly band-limited input. Same figure as Figure 4.23, but with an input $u(t)$ that is strictly band-limited to 5 Hz. The number of state dimensions is $q = 11$.

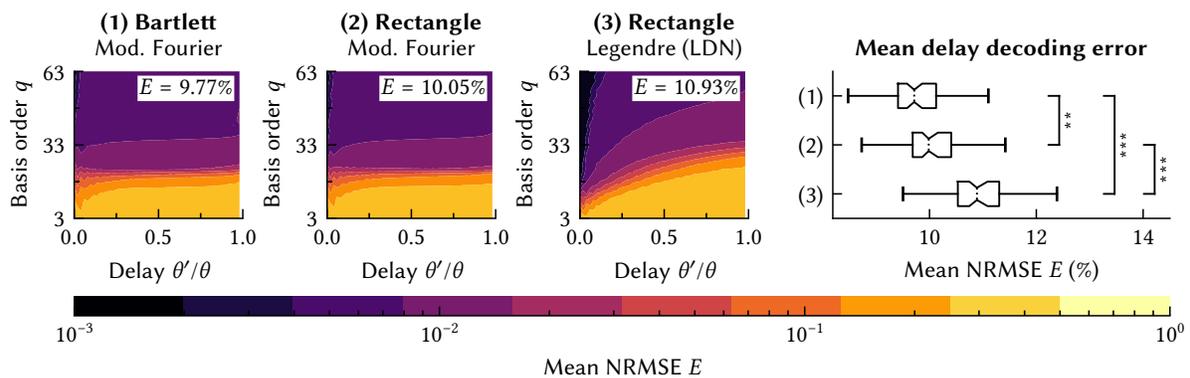


Figure B.7: Comparing the modified Fourier and the LDN systems for band-limited inputs $u(t)$. The analysis is as in Figures 4.24 and 4.25, but for a strictly band-limited white noise signal with a maximum frequency of 10 Hz over $T = 10$ s. Note that adding more basis functions beyond $q = 20$ barely improves the accuracy of the Fourier-based system. Two stars indicate $p < 0.01$, three stars $p < 0.001$.

B.3.2 Realising Spatiotemporal Networks Without Temporal Tuning Curves

We mentioned in Section 4.3.3 that it is possible to implement spatiotemporal neuron populations in the NEF without explicitly relying on temporal tuning curves. Essentially, we merely need to realise the same LTI system d times in the population—flattening the encoder matrices \mathbf{E}_i^t then results in the corresponding NEF encoding vectors \mathbf{e}_i .

Let d be the number of spatial dimensions, q the number of temporal dimensions, and \mathbf{A}, \mathbf{B} be the state-space matrices generating the desired temporal basis. Applying the NEF dynamics principle, we need to realise the following linear input and feedback transformations:

$$\mathbf{B}' = \tau \begin{pmatrix} \mathbf{B} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \mathbf{B} \end{pmatrix}, \quad \mathbf{A}' = \tau \begin{pmatrix} \mathbf{A} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \mathbf{A} \end{pmatrix} + \mathbf{I},$$

where τ is the time-constant of the first-order synaptic filter, $\mathbf{B}' \in \mathbb{R}^{dq \times d}$, and $\mathbf{A}' \in \mathbb{R}^{dq \times dq}$.

We repeat our delayed multiplication experiment from Section 4.3.3 using this technique (see Figure B.8). We find that errors are slightly higher compared to using our temporal tuning curve approach. This is mostly due to using unbiased samples \mathbf{x}_k when solving for connection weights in a (relatively) high-dimensional space. We typically use representative input signals when minimising the temporal tuning curve loss eq. (4.8). This biases the weight solver towards actually observed portions of the activity space (cf. Section 4.3.1). Attempting to sample \mathbf{x}_k from this biased distribution amounts to the same computation as solving eq. (4.8) directly.

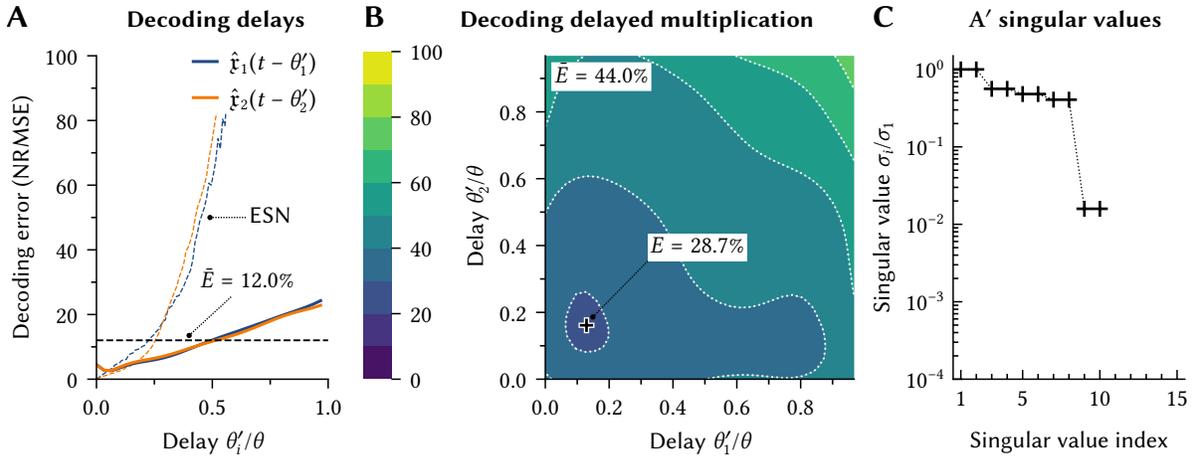


Figure B.8: Results of a delay decoding experiment in a spatiotemporal NEF network. Same plots as in Figure 4.37. **(A, B)** Both the mean delay decoding and mean delayed multiplication errors are slightly larger compared to our temporal tuning curve implementation. **(C)** The singular values of the matrix \mathbf{A}' are close those of the recurrent weight matrix \mathbf{W}_{rec} .

B.3.3 Code and Data for the Machine Learning Experiments

Below, we provide the source code describing the neural network architectures used in Section 4.4.4. The code relies on TensorFlow 2.5 (Abadi et al., 2016) and uses its Keras API (Chollet, 2017) to describe the network. The `TemporalBasisTrafo` layer is our own work.²

psMNIST. We use the following code to describe the network performing the psMNIST task:

```
q = 468; N = 28 * 28; n = 346; H = mk_basis(q, N)
model = tf.keras.models.Sequential([
    tf.keras.layers.Reshape((N, 1)),           # (N, 1)
    TemporalBasisTrafo(H, units=1),           # (1, q)
    tf.keras.layers.Dropout(0.5),            # (1, q)
    tf.keras.layers.Dense(n, activation="relu"), # (1, M)
    tf.keras.layers.Dense(10, use_bias=False), # (1, 10)
    tf.keras.layers.Reshape((10,))          # (10)
])
```

Comments (Nt, Nd) indicate the output dimensions of each layer. Here, Nt is the number of temporal samples; Nd is the number of spatial dimensions. The `TemporalBasisTrafo` layer consumes $N - 1$ temporal dimensions and converts them into q spatial dimensions, where N and q are determined from the filter matrix $\mathbf{H} \in \mathbb{R}^{q \times N}$. The function `mk_basis` generates a set of q FIR filters of length N corresponding to one of our basis transformation methods.³

Mackey-Glass. The network used for solving the Mackey-Glass task consists of four LMU layers. We extend the filter matrix \mathbf{H} to be $W_m = 3$ times as long as the window width N (cf. Figure B.9). This is reflected in the call to the `mk_ext_basis` function and ensures that the ringing artefacts from the LDN and the modified Fourier basis are taken into account. The FIR filters for the “perfect” sliding-window spectra are padded with zeros; this merely introduces superfluous multiplications, but has no further effect on the computation.

```
Wm = 3 # Window multiplier; the \FIR filter is this times as long as the window
N_units0, N_units1, N_units2, N_units3 = 1, 10, 10, 10
N_wnd0, N_wnd1, N_wnd2, N_wnd3 = N_wnds = (17 * Wm, 9 * Wm, 9 * Wm, 5 * Wm)
N_wnd = N_wnd0 + N_wnd1 + N_wnd2 + N_wnd3 - 3
q0, q1, q2, q3 = 17, 9, 9, 5
H0, H1 = mk_ext_basis(q0, N_wnd0 // Wm, Wm), mk_ext_basis(q1, N_wnd1 // Wm, Wm)
H2, H3 = mk_ext_basis(q2, N_wnd2 // Wm, Wm), mk_ext_basis(q3, N_wnd3 // Wm, Wm)
model = tf.keras.models.Sequential([
    tf.keras.layers.Reshape((N_wnd, 1)),
```

²See https://github.com/astoeckel/temporal_basis_transformation_network for more information.

³We provide a handy library that generates different basis transformation matrices at https://github.com/astoeckel/dlop_ldn_function_bases. See Stöckel (2021b) for more information.

Appendix B. Additional Data

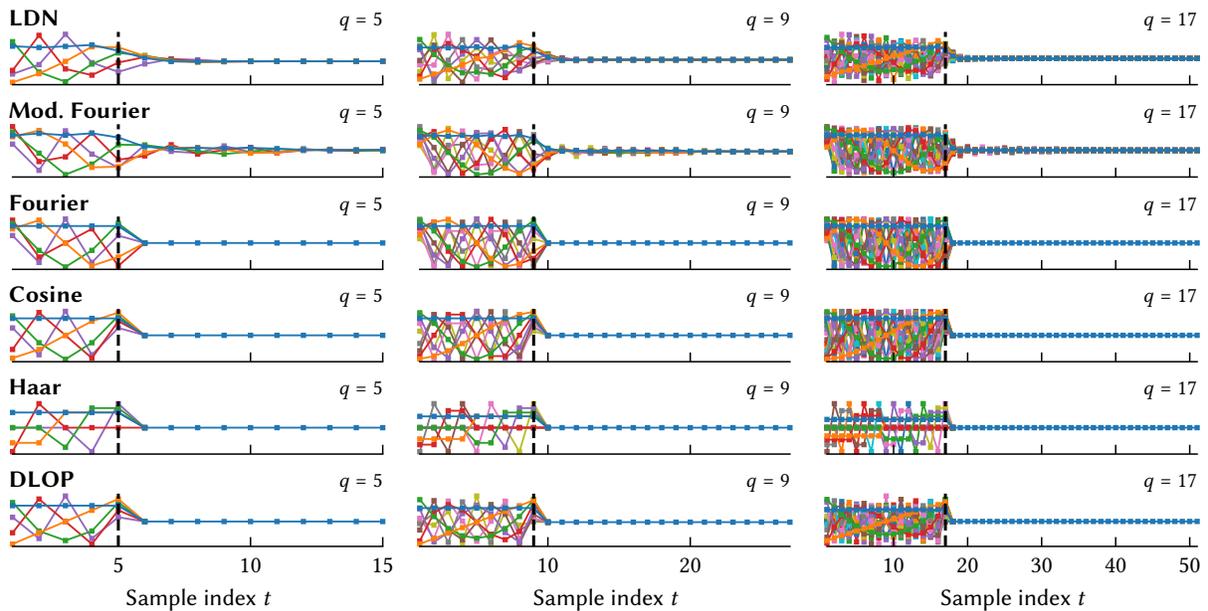


Figure B.9: FIR filters used in the Mackey-Glass experiment. Each filter is three times as long as q . The impulse response of the LDN system and the modified Fourier basis extends beyond q .

```

# (N_wnd0 + N_wnd1 + N_wnd2 + N_wnd3 - 3, 1)
TemporalBasisTrafo(H0, n_units=N_units0),
# (N_wnd1 + N_wnd2 + N_wnd3 - 2, q0 * N_units0)
tf.keras.layers.Dense(N_units1, activation="relu"),
# (N_wnd1 + N_wnd2 + N_wnd3 - 2, N_units1)
TemporalBasisTrafo(H1, n_units=N_units1),
# (N_wnd2 + N_wnd3 - 1, q1 * N_units1)
tf.keras.layers.Dense(N_units2, activation="relu"),
# (N_wnd2 + N_wnd3 - 1, N_units2)
TemporalBasisTrafo(H2, n_units=N_units2),
# (N_wnd3, q2 * N_units2)
tf.keras.layers.Dense(N_units3, activation="relu"),
# (N_wnd3, N_units3)
TemporalBasisTrafo(H3, n_units=N_units3),
# (1, q3 * N_units3)
tf.keras.layers.Dense(N_pred, use_bias=False), # (1, N_pred)
tf.keras.layers.Reshape((N_pred,)) # (N_pred)
1)

```

Comments (N_t , N_d) indicate the output dimensions of each layer, where N_t is the number of temporal, and N_d the number of spatial dimensions. Each `TemporalBasisTrafo` consumes $N_wndX - 1$ temporal dimensions, and replaces them by $qX * N_unitsX$ spatial dimensions.

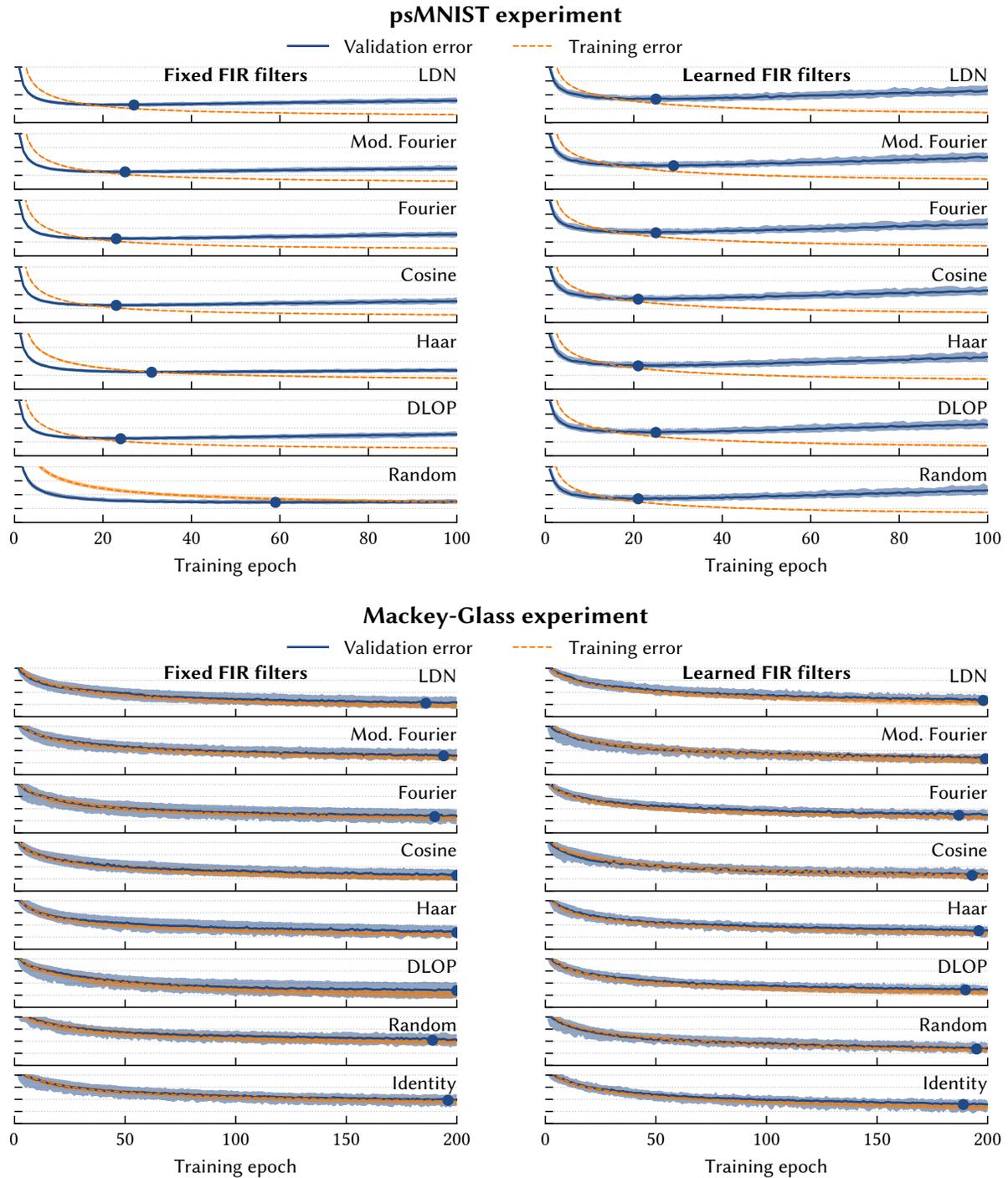


Figure B.10: Learning curves for the psMNIST and Mackey-Glass experiments. y -axis is the value of the loss function (categorical cross-entropy and logarithmic MSE, respectively; scale is the same across all plots). Lines are the median over 101 trials; shaded areas are the 10th and 90th percentiles. Blue circle indicates the minimum median validation error.

Appendix B. Additional Data

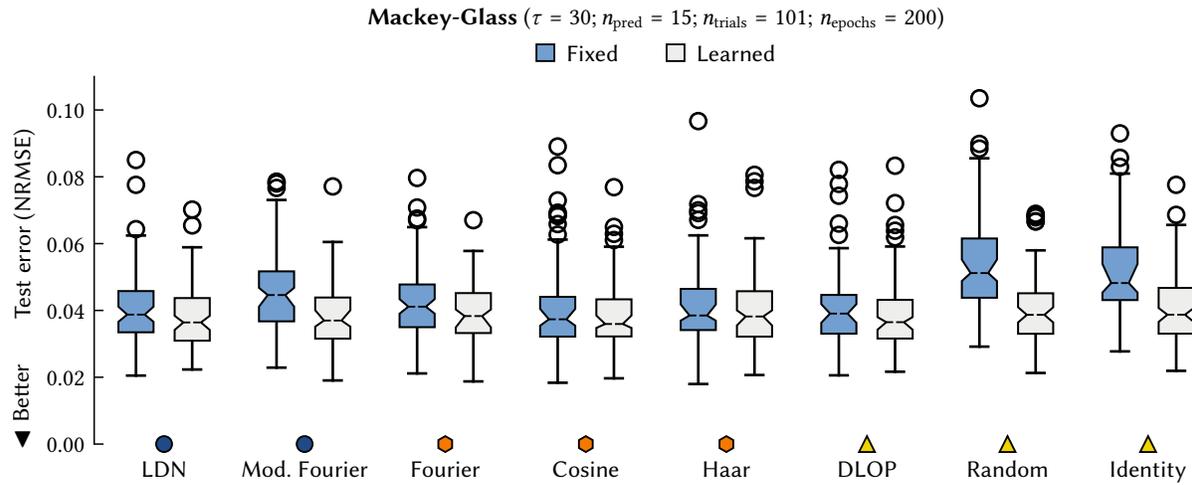


Figure B.11: Prediction errors for the Mackey-Glass dataset with perfect rectangle windows. See Figure 4.51 for a description of the plot. Numerical values are given in Table B.9.

Table B.9: Prediction errors and statistical analysis for the Mackey-Glass dataset with perfect rectangle window. Data over 101 trials after 100 epochs of training. See Tables 4.2 and 4.3 for a more detailed description. The given significance levels are based on a two-sided Kolmogorov-Smirnov test; $\cdot \hat{=} p < 0.05$, $\cdot\cdot \hat{=} p < 0.01$, $\cdot\cdot\cdot \hat{=} p < 0.001$.

	■ Fixed convolution				■ Learned convolution			
● LDN	4.02%	3.88%	3.35%	4.58%	3.80%	3.64%	3.09%	4.37%
● Mod. Fourier	4.56%	4.46%	3.67%	5.17%	3.89%	3.70%	3.15%	4.39%
● Fourier	4.22%	4.11%	3.50%	4.78%	3.93%	3.83%	3.32%	4.52%
● Cosine	4.02%	3.73%	3.22%	4.41%	3.80%	3.59%	3.22%	4.33%
● Haar	4.12%	3.85%	3.41%	4.65%	3.95%	3.82%	3.22%	4.58%
▲ DLOP	4.04%	3.90%	3.30%	4.46%	3.86%	3.65%	3.16%	4.32%
▲ Random	5.38%	5.12%	4.38%	6.15%	4.04%	3.87%	3.30%	4.51%
▲ Identity	5.10%	4.82%	4.31%	5.89%	4.04%	3.87%	3.31%	4.67%

	■ Fixed convolution								■ Learned convolution							
Basis	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
● LDN (1)									
● Mod. Fourier (2)								
● Fourier (3)														
● Cosine (4)										
● Haar (5)														
▲ DLOP (6)									
▲ Random (7)										
▲ Identity (8)										

Appendix C

Software

We developed several software libraries to conduct the research in this thesis, most notably *libnlif* and *NengoBio*. Our weight and parameter solvers, as well as simulators for the n -LIF neurons discussed in Chapter 3 are implemented in *libnlif*.¹ The second library, *NengoBio* is an extension of the *Nengo* neural network simulation package (Bekolay, Bergstra, et al., 2014). Specifically, *NengoBio* implements the extensions to the NEF outlined in Section 3.2 and interfaces with *libnlif* for two compartment weight solving and simulation. We use *NengoBio* as a part of our cerebellum model (cf. Chapter 5 and Stöckel, Stewart, et al., 2021).

Although these libraries are less interesting from a scientific perspective, we would still like to provide a quick overview of the software architecture and give some usage examples.

¹Note that *libnlif* supersedes *libbioneuronqp* in a mostly backwards compatible way. For legacy reasons, the source code of the experiments in Section 3.4 still relies on *libbioneuronqp*, as well as an older version of the code that eventually became *libnlif* (initially published as supplementary material for Stöckel and Eliasmith, 2021).

Software availability

Most of the software required to reproduce the experiments in this document is distributed as part of the thesis source code repository, see https://github.com/astoeckel/phd_thesis.

Operating system images facilitating the execution this code are available at <https://osf.io/y64xu/>. Recent versions of *libnlif* and *NengoBio* can be found at <https://github.com/astoeckel/libnlif> and <https://github.com/astoeckel/nengo-bio>, respectively.

Authorship and licensing

The software discussed in this chapter was primarily developed by the author. Exceptions include dependencies such as *eigen* and *osqp*, as well as some smaller code snippets. Authorship is explicitly noted in the individual files and directories. Where applicable, the author's code is made available under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. Please visit <https://www.gnu.org/licenses/> to obtain a copy of the license.

C.1 An n -LIF Weight Solver and Simulator: `libnlif`

`libnlif` is a hybrid Python and C++ library that facilitates working with n -LIF neurons, specifically implementing the techniques discussed in Section 3.3 onward. This includes providing a simple API for describing and simulating n -LIF neurons, as well as an implementation of the various synaptic weight and parameter optimisers from Sections 3.4 and 3.5.

C.1.1 Describing n -LIF Neurons

The user-facing API for describing neuron models is heavily inspired by Nengo. For example, a standard LIF neuron with current-based input (cf. Figure 3.22A) can be constructed as follows:

```
import nlif

with nlif.Neuron() as lif: # Create a new neuron description
    with nlif.Soma(v_th=-50e-3, tau_ref=2e-3, tau_spike=1e-3, C_m=1e-9) as soma:
        gL = nlif.CondChan(g=50e-9, E_rev=-80e-3) # Static leak channel
        J = nlif.CurChan(mul=1.0)                # Current-based input channel

lif_assm = two_comp_lif.assemble() # Assemble an immutable representation
```

Similarly, the following code describes the two-compartment LIF neuron (cf. Figure 3.22C):

```
with nlif.Neuron() as two_comp_lif:
    with nlif.Soma(v_th=-50e-3, tau_ref=2e-3, tau_spike=1e-3, C_m=1e-9) as soma:
        gL = nlif.CondChan(g=50e-9, E_rev=-65e-3) # Static leak channel
    with nlif.Compartment(C_m=1e-9) as dendrites:
        gL = nlif.CondChan(g=50e-9, E_rev=-65e-3) # Static leak channel
        gE = nlif.CondChan(E_rev=0e-3)           # Excitatory input channel
        gI = nlif.CondChan(E_rev=-75e-3)        # Inhibitory input channel
        nlif.Connection(soma, dendrites, g_c=50e-9)
    two_comp_lif_assm = two_comp_lif.assemble()
```

This overall pattern extends to arbitrary connectivity graphs. Individual objects and the graph structure are validated for adherence to the n -LIF constraints (Section 3.3.1) in the `assemble` method. The Python “with” statement is used to establish an object hierarchy; this is accomplished by overriding the `__enter__` and `__exit__` functions and tracking the current parent object in a thread-local stack. Objects are automatically labelled according to their local variable name whenever a “with”-scope is left.

Calling the `to_svg` method on the assembled neuron object (or simply evaluating the object in a Jupyter cell) generates an annotated “ball-and-stick” representation of the neuron (similar to those depicted in Figure 3.22) using `GraphViz` (Ellson et al., 2004).

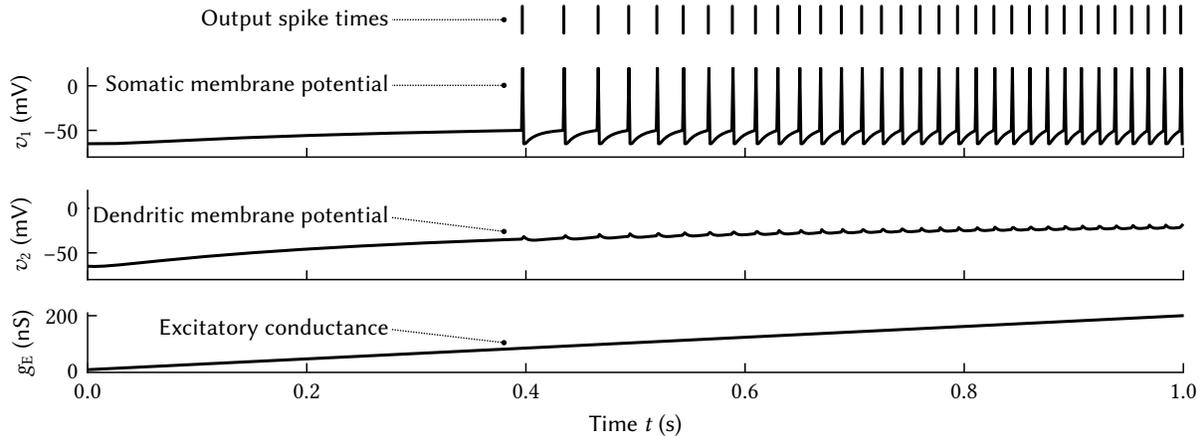


Figure C.1: Output of the *libnlif* two-compartment LIF simulation code example. Output spikes are available as individual spike times at sub-sample resolution or as a discretised sum of Dirac deltas.

C.1.2 Simulating n -LIF Neurons

The assembled n -LIF object holds the system matrices A' , B' , a' , b' , L (Section 3.3.1). These matrices can be accessed through the `A`, `B`, `a_const`, `b_const` and `L` properties and are used by *libnlif* to simulate the dynamics of the neuron. Specifically, the neural dynamics can be simulated using the `nlif.Simulator` object and one of the available `simulate` methods:

```
dt, ss, T = 1e-4, 10, 1.0      # Simulation time-step, sub-sampling and end-time
ts = np.arange(0, T, ss * dt) # Sample points
gEs = np.linspace(0.0, 200e-9, len(ts)) # Sampled input conductances
with nlif.Simulator(two_comp_lif_assm, dt=dt, ss=ss,
                    record_voltages=True, record_spike_times=True) as sim:
    res = sim.simulate({      # Also: simulate_poisson, simulate_filtered
        gE: gEs,              # Sampled input
        gI: 10e-9})          # Constant input
plt.plot(ts, res.v)         # Discretised output pulses are stored in res.out
```

The output of this code is depicted in Appendix C.1.2. Other variants of the “simulate” method include `simulate_poisson`, which generates artificial Poisson-distributed spike noise (e.g., Abbott and Dayan, 2001, Section 1.4) and treats the given inputs as expectation values, as well as `simulate_filtered`, which filters the input using a first-order low-pass.

Automatic simulator generation and compilation. Instantiating the `Simulator` class transparently generates a dynamically linked library that implements a specialised simulator. The library is based on a templated (Stroustrup, 2013, Chapter 23) C++ simulator (cf. `simulator.hpp`); substituting in the specific system matrices and simulator options and compile-time maximises run-time performance. The code is compiled using a custom C++ build system (see `cmodule.py`)

Appendix C. Software

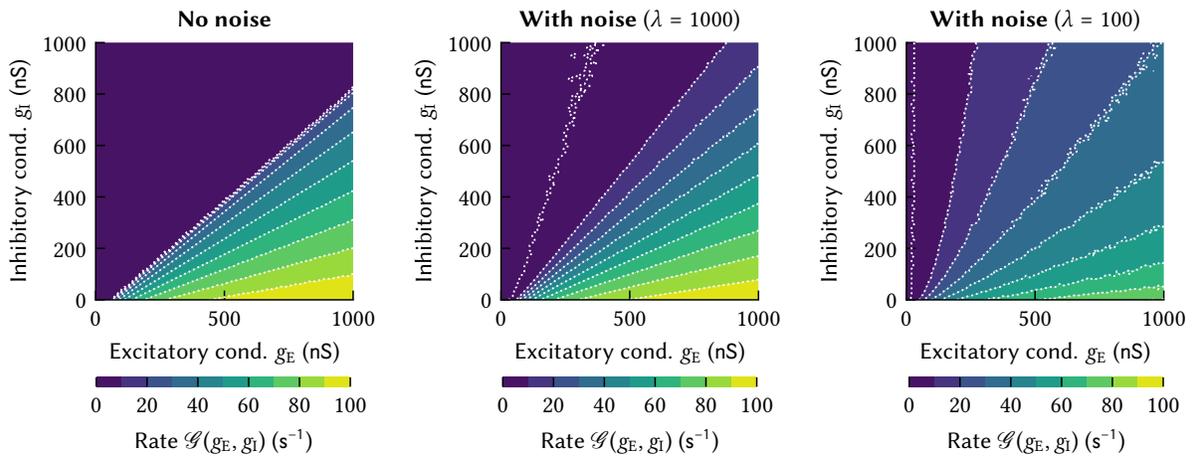


Figure C.2: Output of the *liblif* two-compartment LIF `rate_empirical` rate example. The specified λ corresponds to the rate of the Poisson spike source; the low-pass filter time-constant τ is set to 5 ms.

that calls out to the GNU Compiler Collection² and caches the generated object files according to their input file hashes. Reentrancy of the build system is ensured by employing an idempotency mechanism based on temporary target files and the atomic “rename” filesystem operation (*IEEE Standard for Information Technology: Portable Operating System Interface (POSIX®) 2018*, pp. 1816-1820). The compiled library is loaded via the Python `cypes` foreign function interface; the C++ code directly operates on the memory regions that back the Numpy arrays.

Internally, the simulator uses the Eigen linear algebra library (Guennebaud, Jacob, et al., 2010). The dynamical system is integrated under a zero-order hold assumption by solving for the system state at time $t + \Delta t$ according to the closed-form matrix exponential form eq. (3.23).

Input sweeps. The assembled neuron object provides convenience functions for sampling the multivariate response curves $\mathcal{G}(\mathbf{g})$ (eq. 3.18). The following code simulates the neuron for 100 s and estimates the average firing rate by taking the inverse of the median inter-spike interval. The optional noise, rate, and tau parameters specify Poisson-distributed spike-noise.

```
gEs, gIs = np.linspace(0, 1e-6, 20), np.linspace(0, 1e-6, 20)
gEss, gIss = np.meshgrid(gEs, gIs) # Generate a dense sample grid
rates = two_comp_lif_asm.rate_empirical( # Simulate the neuron at each sample
    {gE: gEss, gI: gIss}, T=100.0, noise=True, rate=10000, tau=5e-3)
```

Evaluation of the sample points is distributed across all available processor cores. The output of the above code-snippet (with a 100×100 grid) is visualised in Figure C.2 for different rates λ .

²See <https://gcc.gnu.org/> for more information.

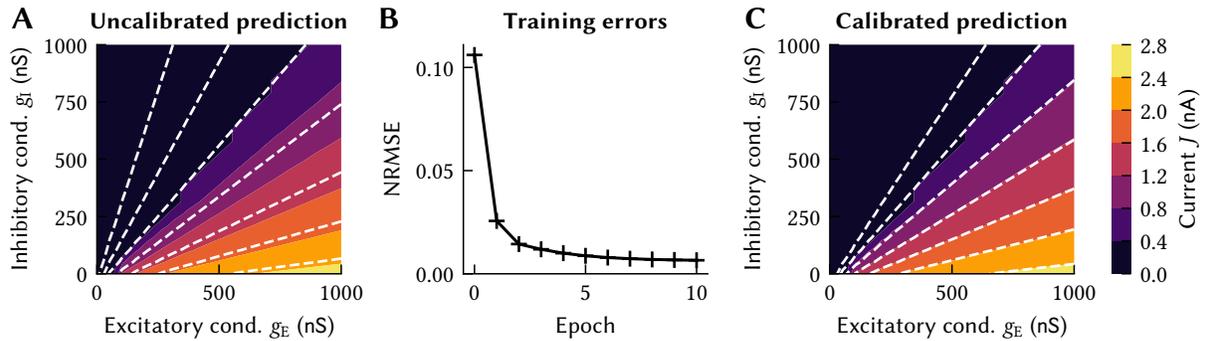


Figure C.3: Two-compartment LIF parameter optimisation using *libnlif*. Filled contours are the empirical error measurements, dashed white lines the predicted somatic currents.

C.1.3 Predicting Somatic Currents and Solving for Parameters and Weights

The assembled neuron object can also predict somatic currents according to eq. (3.27). To this end, we must first compute the reduced system matrices $\tilde{\mathbf{A}}'$, $\tilde{\mathbf{B}}'$, $\tilde{\mathbf{a}}'$, $\tilde{\mathbf{b}}'$, $\tilde{\mathbf{L}}$, $\tilde{\mathbf{c}}$:

```
sys = two_comp_lif_assm.reduced_system(v_som=None) # v_som = 0.5*(v_reset+v_th)
```

The reduced system matrices are stored in the A, B, a_const, b_const, L and c properties of the sys object. Currents can be predicted as follows:

```
i_som_pred = two_comp_lif_assm.i_som({gE: gEss, gI: gIss}, reduced_system=sys)
```

Estimating model parameters. As depicted in Figure C.3A, `i_som_pred` is not very accurate without further calibration. The `nlif.parameter_optimisation` package implements the optimisation methods discussed in Section 3.5.1, including our soft trust-region based SQP:

```
gs_train = two_comp_lif_assm.canonicalise_input({gE: gEss, gI: gIss})
Js_train = two_comp_lif_assm.lif_rate_inv(rates) # Invert the response curve
valid = rates > 12.5 # Discard subthreshold samples
sys = sys.condition() # Condition the reduced system
sys_opt, errs_train = nlif.parameter_optimisation.optimise_trust_region(
    sys, gs_train=gs_train[valid], Js_train=Js_train[valid], N_epochs=10)
```

The optimised system is stored in `sys_opt` and can be passed to the above `i_som` method to obtain improved current predictions (Figures C.3B and C.3C). Just like the n -LIF simulator, the trust-region optimiser relies on a dynamically compiled C++ library that in turn makes use of the Eigen and OSQP libraries (Stellato et al., 2020; see `nlif_solver_parameters.cpp`).

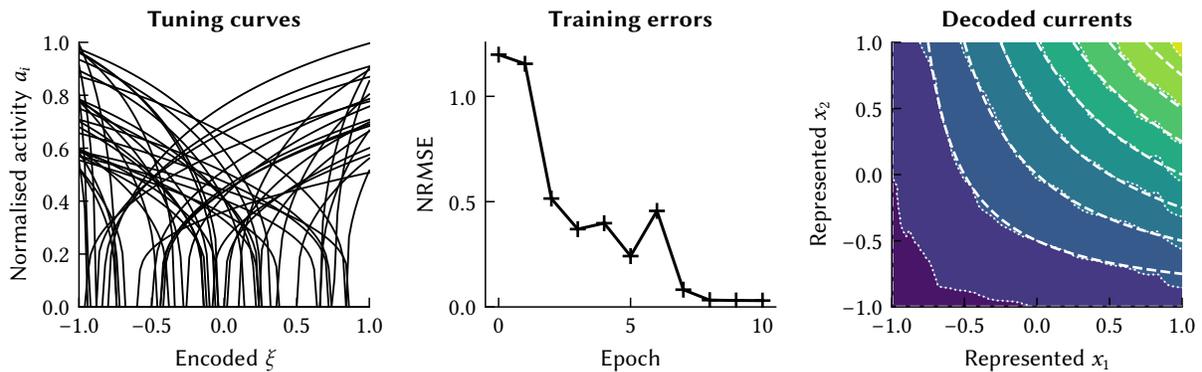


Figure C.4: Nonnegative multiplication in two-compartment LIF neurons using *libnlif*. Coloured contours and dotted lines show the decoded currents; dashed lines are the target.

Solving for synaptic weights. The synaptic weight solvers discussed in Section 3.5.3 are provided in the `nlif.weight_optimisation` package. For example, to find synaptic weights that approximate nonnegative multiplication we can use the following code:

```
# Sample the represented space
xs, ys = np.linspace(-1, 1, 101), np.linspace(-1, 1, 101)
xss, yss = np.meshgrid(xs, ys)
Xs = np.array((xss.flatten(), yss.flatten())).T

# Obtain the target current function and pre-activities
Js = 0.5 * (1.0 + Xs[:, 0]) * (1.0 + Xs[:, 1]) * 1e-9
As = [...] # (N_smpls x m_pre); use the NEF tuning curve equations here

# Perform the actual training
idcs = np.random.randint(0, Xs.shape[0], 256) # Training samples
W_mask = np.ones((2, As.shape[1]), dtype=bool) # All-to-all connectivity
W, errs_train = nlif.weight_optimisation.optimise_trust_region(sys_opt,
    As_train=As[idcs], Js_train=Js[idcs], W_mask=W_mask, N_epochs=10)

# Compute the decoded currents
Js_dec = two_comp_lif_assm.i_som(As @ W.T, reduced_system=sys_opt)
```

Again, the optimiser is implemented in a C++ library (see `nlif_solver_weights.cpp`). Since the generated quadratic programs tend to be moderately large (on the order of thousands of variables, depending on the number of pre-neurons and samples), the code makes extensive use of sparse matrices. To prevent heap reallocations while populating the matrices, the code pre-calculates the number of non-zero entries per column and sequentially writes the matrix coefficients to memory. The low-level `nlif.Solver` class wraps the C++ code more directly and supports concurrent optimisation across multiple post-neurons.

C.2 More Biologically Plausible Nengo Models: NengoBio

NengoBio is an add-on to the Nengo neural network simulator package (Bekolay, Bergstra, et al., 2014) that implements the extensions to the NEF discussed in Section 3.2. At its core, NengoBio adds support for multi-channel neurons and for optimising weights in current-space. We accomplish this by hooking into Nengo’s build system and dynamically rewriting the operator graph (see Gosmann and Eliasmith, 2017 for a description of the operator graph).

NengoBio can account for Dale’s principle, provide special syntactic sugar for interneuron populations, enforce sparsity constraints, and support dendritic computation with two-compartment LIF neurons. We discuss these features in more detail in the following subsections.

C.2.1 Accounting for Dale’s Principle

Dale’s principle can be enforced by replacing Nengo’s original Ensemble and Connection objects with those provided by NengoBio and setting the `p_exc` or `p_inh` properties when constructing the pre-population. The following code-snippet demonstrates this

```
import numpy as np; import nengo; import nengo_bio as bio

with nengo.Network() as model:
    # Construct the input and two NengoBio ensembles
    input = nengo.Node(lambda t: np.sin(2.0 * np.pi * t))
    ens1 = bio.Ensemble(n_neurons=100, dimensions=1, p_exc=1.0)
    ens2 = bio.Ensemble(n_neurons=100, dimensions=1)

    # Nengo connections can target bio.Ensemble objects
    nengo.Connection(input, ens1)

    # NengoBio connections can only be between bio.Ensemble objects. bio.Decode
    # (default) forces decoding bias currents; bio.JBias implies intrinsic biases.
    bio.Connection(ens1, ens2, bias_mode=bio.Decode)
```

This is implemented using the NNLS method discussed in Section 3.2.2. Values between zero and one for `p_exc` sets the probability with which a neuron is marked as excitatory. Not setting `p_exc` disables Dale’s principle for outgoing connections.

NengoBio hooks into the Nengo GUI visualiser³ to highlight purely excitatory or inhibitory connections; with some minor modifications (i.e., repeating the imports and “with” block), the code-snippets in this section can be copied into the visualiser for exploration.

³See https://github.com/nengo/nengo_gui for more details.

C.2.2 Inhibitory Interneuron Populations and Communication Channels

In Section 3.2.2, we discussed two interesting network architectures that benefit from current-space weight solving: networks with inhibitory interneuron populations and purely inhibitory communication channels. Both network types can be easily implemented in NengoBio.

Inhibitory interneurons. In biology, inhibitory input to a neuron population is often routed through inhibitory interneurons. This results in the network architecture depicted in Figure 3.16. We suggested constructing such networks by assuming that both the interneurons and the excitatory pre-population represent the same value x . NengoBio provides special syntax for this: specifying a set $\{\text{ens}_1, \dots, \text{ens}_N\}$ as a pre-population in a connection results in these ensembles being treated as a virtual pre-population that represents a common value:

```
# Create the three ensembles
ens_exc = bio.Ensemble(n_neurons=100, dimensions=1, p_exc=1.0)
ens_inh = bio.Ensemble(n_neurons=100, dimensions=1, p_inh=1.0)
ens_tar = bio.Ensemble(n_neurons=100, dimensions=1)

# Setup connections, treat {ens_exc, ens_inh} as a single population
nengo.Connection(input, ens_exc)
bio.Connection(ens_exc, ens_inh)
bio.Connection({ens_exc, ens_inh}, ens_tar, function=lambda x: x**2)
```

Purely inhibitory communication channels. We can construct purely inhibitory communication channels—and even compute functions along these channels—as long as the target population receives *some* other form of excitatory input. As suggested in Equation (3.14), this can be accomplished by ignoring the excitatory pre-population in the represented space, but using the pre-activities to solve for excitatory input.

```
# Random input for the excitatory ensemble; should not influence the computation
input_noise = nengo.Node(nengo.processes.WhiteSignal(high=5.0, period=10.0))

# Setup connections, treat (ens_inh, ens_exc) as a single population
nengo.Connection(input_noise, ens_exc)
nengo.Connection(input, ens_inh)
bio.Connection((ens_inh, ens_exc), ens_tar, function=lambda x: x[0]**2)
```

Note that we listed the two pre-populations as a tuple $(\text{ens}_1, \dots, \text{ens}_N)$. This instructs NengoBio to form a virtual pre-population where the represented values of the populations are stacked. Here, the two pre-populations represent one-dimensional quantities—correspondingly, the post-population receives a two-dimensional value. In the above code, we simply ignore the second dimension that originates from the excitatory pre-neuron.

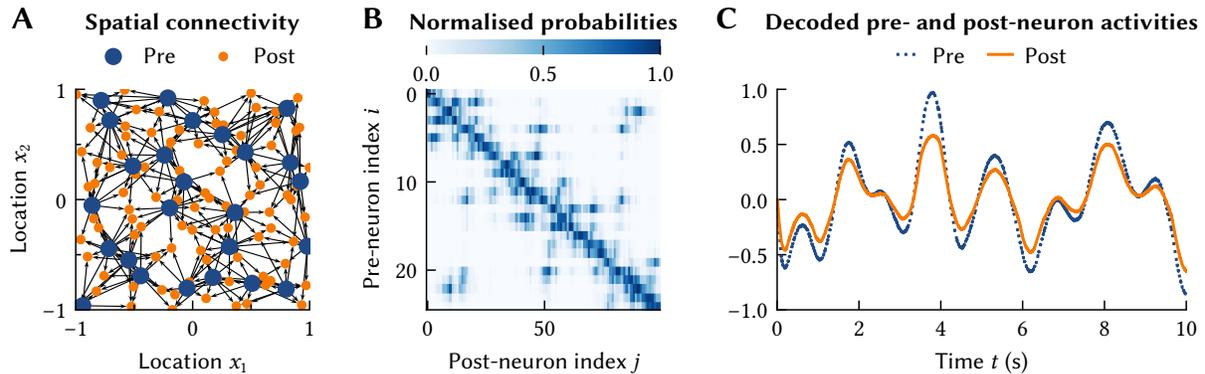


Figure C.5: Spatial connectivity constraints in NengoBio. **(A)** Location of each neuron and connections. **(B)** Normalised connection probabilities. **(C)** Testing the communication channel.

C.2.3 Sparsity Constraints

As we mentioned in Chapter 5, neurobiological microcircuits are often characterised in terms of their convergence and divergence numbers. NengoBio can account for these constraints.

Specifying convergence and/or divergence. Connectivity constraints can be specified by passing a Connectivity object to a connection. For example, the following code-snippet establishes random connectivity that takes the given convergence numbers into account:

```
ens_src = bio.Ensemble(n_neurons=100, dimensions=1)
ens_tar = bio.Ensemble(n_neurons=100, dimensions=1)
bio.Connection(ens_src, ens_tar, # Can alternatively pass "divergence" (or both)
               connectivity=bio.ConstrainedConnectivity(convergence=5))
```

Spatially constrained connectivity. NengoBio also supports spatially constrained connectivity. Each ensemble can be assigned an array or distribution of locations in n -dimensional space; this location information is then used to compute connection probabilities:

```
ens_src = bio.Ensemble(n_neurons=25, dimensions=1,
                      locations=bio.NeuralSheetDist(dimensions=2))
ens_tar = bio.Ensemble(n_neurons=100, dimensions=1,
                      locations=bio.NeuralSheetDist(dimensions=2))
bio.Connection(ens_src, ens_tar, connectivity=
               bio.SpatiallyConstrainedConnectivity(convergence=5, sigma=0.25))
```

Here, `NeuralSheetDist` is a random distribution that arranges neurons along a Hilbert curve to ensure approximate equidistance and that neurons with similar indices are close together in space. The probability matrix and final connectivity of this network are depicted in Figure C.5.

C.2.4 Dendritic Computation

Dendritic computation in NengoBio relies on the tuple-syntax for specifying a virtual pre-population with stacked represented values (Appendix C.2.2). As we discussed in detail in Chapter 3, we can only compute additive functions in the pre-populations with standard current-based LIF neurons. In NengoBio, simply setting the neuron type of an ensemble to the two-compartment LIF neuron enables the computation of nonlinear multivariate functions:

```
# Input nodes and pre-populations with 30% inhibitory neurons
inp_x1, inp_x2 = nengo.Node(size_in=1), nengo.Node(size_in=1)
ens_x1 = bio.Ensemble(n_neurons=100, dimensions=1, p_inh=0.3)
ens_x2 = bio.Ensemble(n_neurons=100, dimensions=1, p_inh=0.3)
nengo.Connection(inp_x1, ens_x1)
nengo.Connection(inp_x2, ens_x2)

# Create a population of two-compartment LIF neurons, use lower maximum rates
# because of the current limit imposed by the conductance-based synapses
ens_tar = bio.Ensemble(n_neurons=100, dimensions=1,
                       neuron_type=bio.neurons.TwoCompLIF(),
                       max_rates=nengo.dists.Uniform(75, 100))

# Compute nonnegative multiplication. Map the input from [-1, 1]^2 onto [0, 1]
# and the output from [0, 1] onto [-1, 1]. Use the quadratic programming solver
# with subthreshold relaxation to improve performance.
bio.Connection((ens_x1, ens_x2), ens_tar,
               function=lambda x: 0.5 * (x[0] + 1.0) * (x[1] + 1.0) - 1.0,
               solver=bio.solvers.QPSolver(relax=True))
```

When using the two-compartment LIF neuron, NengoBio automatically calibrates the two-compartment neuron parameters using the method presented in Section 3.4.1. In particular, NengoBio performs a series of neuron simulations to determine the excitatory and inhibitory conductance range that produces the specified maximum firing rates.

The `QPSolver` class uses the non-sequential quadratic program discussed in Section 3.4.3 to compute synaptic weights. An implementation of this non-sequential weight solver is provided by *libnlif* in addition to the sequential soft trust-region based algorithms.

As of writing, NengoBio does *not* support n -LIF neurons with more than two compartments. Given our implementation of the parameter and weight solver implementations in *libnlif*, this is not a technical challenge per se, but instead hinges on finding a good API that allows users to specify possible synaptic sites. For example, some pre-populations may only connect to basal compartments, whereas others only target apical compartments. It is unclear how users would specify this on an abstract level.