Compression and Analysis of Pre-trained Language Model with Neural Slimming

by

Ruizhou Xu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical & Computer Engineering

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this 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.

Abstract

Neural networks are powerful solutions to help with decision making and solve complex problems in recent years. In the domain of natural language processing, BERT and its variants significantly outperform other network structures. It learns general linguistic knowledge from a large corpus in the pre-training stage and utilizes that to solve downstream tasks. However, the size of this kind of model is enormous and causes the issue of overparameterization. This makes the model deployment on small edge devices less scalable and flexible.

In this thesis, we study how to compress the BERT model in a structured pruning manner. We proposed the neural slimming technique to assess the importance of each neuron and designed the cost function and pruning strategy to remove neurons that make zero or less contribution to the prediction. After getting fine-tuned on the downstream tasks, the model can learn a more compact structure, and we name it SlimBERT.

We tested our method on 7 GLUE tasks and used only 10% of the original parameters to recover 94% of the original accuracy. It also reduced the run-time memory and increased the inference speed at the same time. Compared to knowledge distillation methods and other structured pruning methods, the proposed approach achieved better performance under different metrics with the same compression ratio. Moreover, our method also improved the interpretability of BERT. By analyzing neurons with a significant contribution, we can observe that BERT utilizes different components and subnetworks according to different tasks.

Acknowledgements

First, I would like to thank my supervisor Dr. Fakhri Karray for his supervision during my graduate studies. His continuous guidance and assistance inspire me to achieve this. Also, I would like to thank Dr. Olga Vechtomova for giving me the opportunity to work with her in the last year of my Bachelor's degree. That motivates me to continue my research in the NLP domain. Besides, thank you Dr. Ali Elkamel for reviewing my thesis and providing valuable feedback.

I would like to give special thanks to Yuyu for all your love and company every day. I am truly blessed to have you standing by my side during the challenges of graduate school. Many thanks to my cat Lingxi for all the fun she brings to me.

Finally, I would like to thank my family for their encouragement and support throughout all these years.

Dedication

This is dedicated to my grandparents, Meier Shi and Dinggeng Xu who have been a source of strength, knowledge, and motivation for me since my childhood.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Since BERT was introduced in 2018, it has made the pretrain-fine-tune paradigm the most well known and powerful technology in the natural language processing (NLP) community [18]. Its variants, such as BioBERT and Legal-BERT, successfully leverage domain-specific knowledge to surpass human ability in industrial applications [8, 35]. Compared to previous language models, BERT-based models perform much better on a variety of tasks including question answering, machine translation, text summarization, etc. Despite their success, there are still a number of notable problems with their use.

One of the greatest, but understudied challenges is overparameterization and its consequences. BERT-based models usually contain millions of parameters. Table 1.1 shows the parameter number of BERT and its variants. The original BERT-Base model contains 110 million parameters and occupies at least 420 megabytes of space. Also, BERT-Large and

Model	Number of parameters in million	Size
BERT-Base [18]	110M	420Mb
BERT-Large	340M	1.25Gb
XLNet-Base [89]	About 110M	445Mb
Roberta-Base [40]	125M	478Mb
Roberta-Large	355M	1.33Gb

Table 1.1: BERT-based model size and number of parameters

Roberta-Large take more than 1 gigabytes of memory and 340 million parameters. The massive number of parameters raises concerns about different aspects.

First, the training of large pre-trained models has significantly increased financial and environmental costs in recent years [66]. Strubell et al. quantified the electricity usage and carbon emission of recent NLP models [70]. The carbon dioxide released by training BERT on 64 V100 GPUs is reported to be almost equivalent to one passenger traveling from New York to San Francisco by air. Moreover, the high financial cost of these computation resources became an obstacle for some students to get involved in the BERT research.

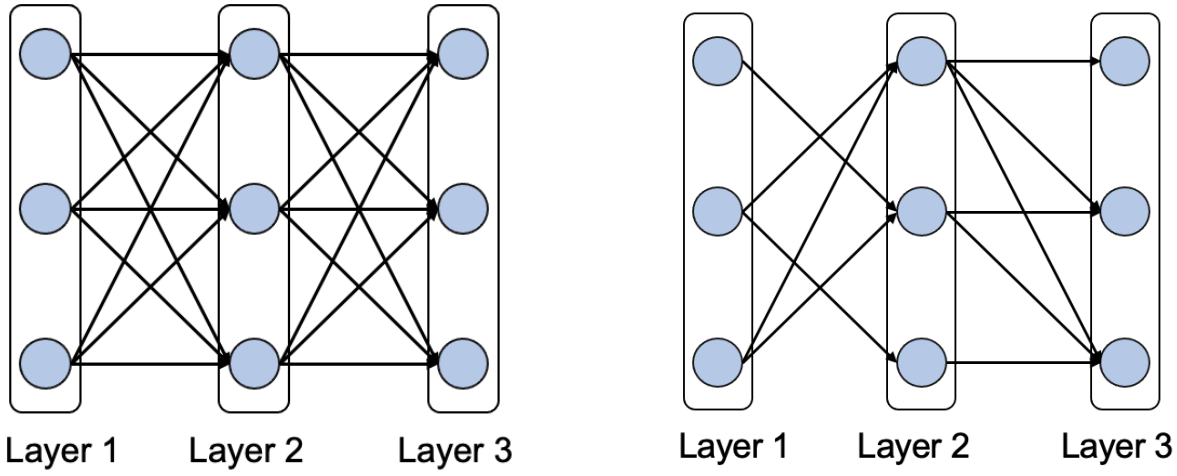
Second, the large size of the parameters and structures makes it more difficult for researchers to reproduce and compare BERT-based models. The low replicability is harmful to its own development and enhancement. Aßenmacher et al. argued that there was a lack of a fair comparison of large pre-trained language models [2]. They noted that current benchmarks were insufficient to rank models without considering other factors such as parameter size or resource consumption. Since a larger model extracts representations with better quality and more suitable architecture also improving the learning capability, changing both of them concurrently mixes up their individual influence on model performance.

Third, giant language models require a vast amount of computation power and storage space. This makes them incompatible with small edge devices or resource-restricted devices. Furthermore, they do not make good use of parameters, and a large portion of them are unnecessary in down-stream tasks. Several researchers have reported that some of the model components are not essential for prediction. Voita et al. investigated the influence of pruning attention heads [81]. They used layer-wise relevance propagation to evaluate the contribution of individual heads in translation tasks and found that only a small subset of heads is important. Furthermore, Dalvi et al. studied general redundancy and task-specific redundancy, and their experiments showed that 85% representation level neurons were redundant [16].

To address these problems, we focus on identifying the unimportant components in the BERT model for downstream tasks and pruning them to reduce the overall size of the model. Meanwhile, we want to maintain the original performance.

1.2 Problem Statement

We aim to answer the following question: How to compress BERT as much as possible but make the loss on performance as minor as possible at the same time? There are different



(a) (b)

Figure 1.1: Model pruning of a 3-layer neural network. (a) Original model (b) Pruned model

kinds of model compression methods, such as quantization, knowledge distillation, matrix decomposition, etc. In this work, we focus on one method, model pruning, which is a well-defined task in the model compression domain.

Here is an example in Figure 1.1. The original model is a fully connected 3-layer neural network and has three neurons in each layer. Each connection between two neurons represents one learnable parameter. By pruning the model, we cut off some of the connections between the neurons, and each neuron will not take the information from all neurons in the previous layer. As a result, our pruned model in Figure 1.1b has a smaller number of parameters.

Given the benefits of model pruning, this leads to another question. How to choose which neurons or components to prune? More specifically, how can we quantify the contribution of an individual neuron to the final result? These two questions are technically challenging for BERT-based models. Unlike white-box models such as linear regression and decision tree that can easily tell the feature importance and decision boundary, BERT-based models are black-box models, which means that they only take the input and then produce the output. The high-dimensional representation they extracted is not understandable to humans [80]. To answer these questions, it is important to find a way to determine the relationship between the model's components and the prediction.

In brief, we want to find the unnecessary neurons for different downstream tasks and prune them to reduce the size of the model.

1.3 Contribution

To summarize, we make the following contribution in this thesis:

- We propose neural slimming, a simple and effective compression method for BERT. Unlike previous research that uses post hoc methods to estimate the contribution of individual neurons, our method can compute their importance and identify the unimportant ones while fine-tuning the model, and no more extra work is needed afterwards.
- We perform structured pruning on redundant neurons in the embedding layer, self-attention layers, and feed-forward layers. The resulting model is named SlimBERT, and it has a smaller number of parameters, less amount of run-time memory, and a faster inference speed.
- We also design two tuning strategies, prune-after-tune and prune-then-tune, based on the proposed method. The first is more suitable to minimize the model size. For the latter, the model can be pruned with any desired compression ratio and has higher accuracy.
- We run extensive experiments on 7 datasets across four kinds of natural language inference tasks. The compression capability remarkably outperforms previous structured pruning methods and performs competitively with the state-of-the-art unstructured pruning method. We also analyze the pattern of the important neurons and components in different tasks.
- We release our model implementation and training hyperparameters for reproducibility.

Chapter 2

Background and Related Work

2.1 Neural Networks

Over the past ten years, the neural network has been used to assist humans in many fields, including image processing, biomedical engineering, autonomous driving, and so on. The success of this type of model is derived from the ability to learn the input data and generalize well on unseen data. It is made up of a group of neurons that are also called the perceptron.

The classical perceptron algorithm was first proposed by Rosenblatt in 1958, and the motivation was to simulate how the human brain stores and processes information [59]. In 1969, Minsky and Papert presented the proof of this algorithm and demonstrated how it learned and solved linear separation problems [45]. Figure 2.1 shows the structure of a single neuron in neural network models. It takes x as the input data. w and b denote the learnable parameters weights and bias correspondingly. The computation of z uses the following formula:

$$z = x_1 * w_1 + x_2 * w_2 + ... + x_n * w_n + b_1. \tag{2.1}$$

It sums up all the weighted input data and the bias term. For the perceptron algorithm, z is the output of the model and the output label can be determined using a threshold such as

$$Label\ of\ (x_1, x_2, ..., x_n) = \begin{cases} 0, & \text{if } z > 0 \\ 1, & \text{otherwise} \end{cases} \tag{2.2}$$

For neural networks, the activation function f is applied to z which yields the output of the neuron and adds non-linearity to the model [19, 84, 21]. Therefore, it makes the model

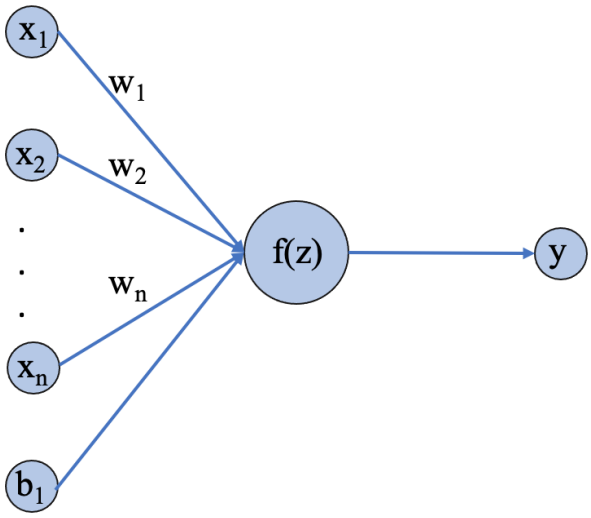


Figure 2.1: A single neuron in the neuron network

can classify the data points that are linearly unseparable. The derivative of the activation function is used when training the model which will be discussed later. Therefore, this function should also be differentiable.

One of the most popular activation functions is the sigmoid function. A sigmoid function is a bounded function that maps the value to the range of 0 to 1, so it is more suitable for the output layer to predict the probability. The following is its definition;

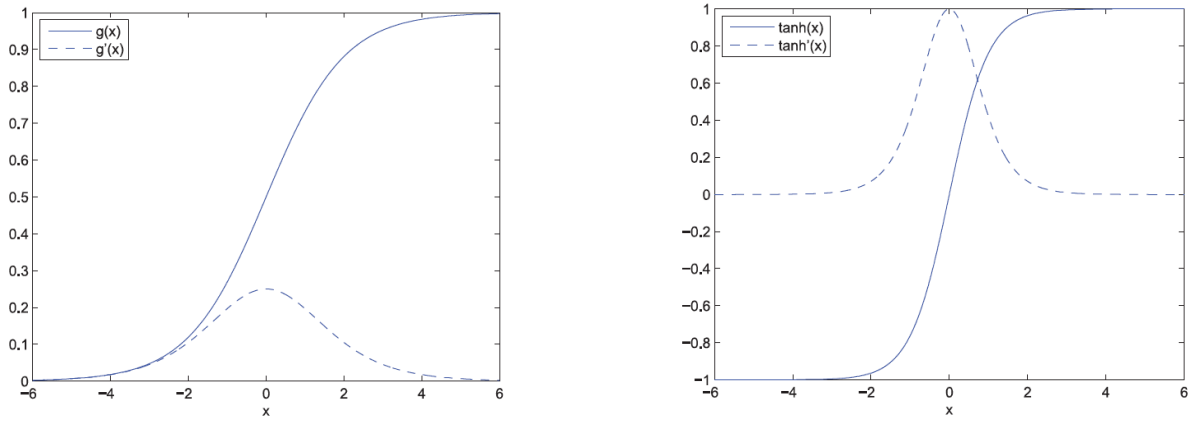
$$Sigmoid(x) = \frac{1}{1 + e^x} \tag{2.3}$$

The curve of the sigmoid function is shown in Figure2.2a. The curve is steep when the input value is close to zero, so there is a larger gradient. As a result, this function tends to push the output value to zero or one, and the model can make a more distinct classification.

Another powerful activation function is the hyperbolic tangent function; the definition is as follows.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.4}$$

The shape of the *tanh* function is similar to that of the sigmoid, but the range starts from -1. One major difference between this function and the sigmoid is their derivatives. *tanh* has a much higher derivative for values close to 0, which makes the model converge more quickly [76]. In other words, it needs less iteration to decrease the loss to the local or global minimum value.



(a) Sigmoid (b) *tanh*

Figure 2.2: Curve of activation functions and their derivative (a) Sigmoid Function (b) *tanh* Function [19]

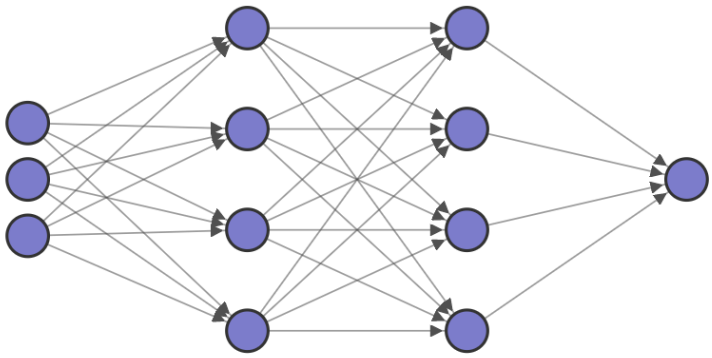


Figure 2.3: Example of a fully connected neural network

The traditional deep neural network is also called a fully connected neural network (FCNN). Figure2.3 gives an example of an FCNN with 4 layers. It consists of a sequence of fully connected layers, which means that each neuron in the previous layer is connected to the next layer. Each network has at least one input layer and one output layer. When making the prediction, all the data follow the direction of the edge and flow from the input layer to the output layer, which is called forward propagation. The layers between the input layer and the output layer are called hidden layers. This sample model has two hidden layers, each of them contains 4 neurons. The number of neurons and layers is not limited, and making the model deeper and wider generally improves the accuracy [55]. However, too deep models may strictly fit the training data, but not perform well in the testing data, and this phenomenon is called overfitting[7].

The main property of a fully connected neural network is that it does not make any assumption about the relationship between input features. That is to say, this type of model can be easily used on any kind of problem. It is very flexible and easy to implement. However, this property also restricts the performance of the model on some domain-specific tasks, such as computer vision and language understanding, which have a more specific relationship between the features. Hence, it is more suitable to approximate the underlying mapping function from input to output.

2.1.1 Training and Optimization

Training is the procedure that updates the parameters and allows the model to identify the hidden pattern between input and output. There are two main components of model training, one is the loss function, and the other is the optimization algorithm. The first one computes the difference between the model's prediction and the desired output. It indicates the learning progress of the model.

For regression problem where the output is a continuous value, the most common loss function is mean squared squared error (MSE), and its definition is as follows:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - y_i')^2}{n} \tag{2.5}$$

It measures the average of squared difference between the predicted value y_i and the actual one y_i'. The loss increases exponentially as their difference rises, so it speeds up the training process using its mathematical property.

With regard to classification problem, the cross entropy loss (CE Loss) is frequently

used. Its formular is written as:

$$CELoss = -\sum_{c=1}^{n} y'_c \log y_c \tag{2.6}$$

In this equation, y'_c equals to 1 if class c is the actual label and 0 otherwise. For classification problem, the output is a probability falls in range 0 to 1. This function pushes the probability of expected class towards 1 and unexpected ones to 0. It efficiently minimizes the distance between predicted distribution and desired distribution [28].

In order to minimize the loss function, an appropriate optimization method should be used to update the model's parameters. For neural networks, gradient descent has been the most common choice for optimization. While the vanilla gradient descent is slow and has a large memory requirement, mini-batch gradient descent is a better approach [60, 31]. It takes n training data points, computes the gradients, and updates the parameters θ with learning rate α:

$$\theta = \theta - \alpha * \nabla_\theta f(\theta, x^{i:i+n}, y^{i:i+n}) \tag{2.7}$$

Sebastian described that it decreases the variance when updating parameters which makes the convergence more stable and the way that measures gradient according to a small group of data is very efficient [31].

Moreover, there are methods to further accelerate the gradient descent algorithm. Qian add an extra momentum term to the original function [53]. It helps to increase the step size of updating parameters if it is going towards the right direction. Furthermore, Kingma et al. proposed the Adam optimizer, which combines the benefits of maintaining a learning rate for each parameter and adapting each learning rate to the magnitude of the recent gradient [32]. It has been the most common approach since introduced and it is especially preferred by sizeable models with a lot of layers.

2.1.2 Recurrent Neuron Network

Although fully connected network can be applied to a lot of tasks, it's input size is always fixed. For NLP tasks, the length of input sentences or documents are uncertain, and it is required that the model can take input with a variable length. FC network is not a proper choice in this situation. To address this problem and still use the approximation ability of the neural network, the recurrent neural network (RNN) is introduced, which is able to take a sequence of input with any length [29, 61].

Figure2.4 shows the general architecture of a vanilla recurrent neural network. In this model, x_t, y_t and h_t are the input feature and hidden state at time stamp t. $W_a b$ is the

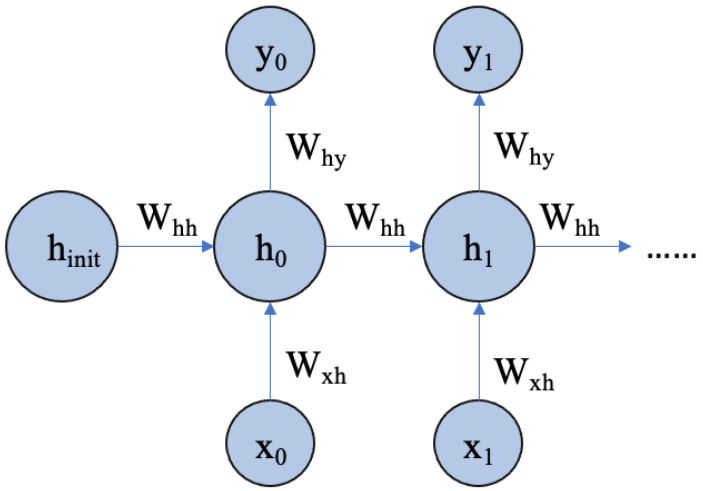


Figure 2.4: Architecture of a recurrent neural network

learnable parameter for the transformation between a to b. The bias term is omitted in this figure. Following is the formular of forward propagation:

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \tag{2.8}$$

$$y_t = f(W_{hy}h_t + b_y) \tag{2.9}$$

RNN not only uses the information from the current time stamp, it also takes account of the hidden state from previous time stamp. Since it also considers historical information, it is more sufficient to capture the patterns from the data that has a sequential order such as videos or text. Another advantage of RNN is the parameters can be shared over the time. Hence, the model size does not increase as the input length increase.

However, this model structure also has some disadvantages. Unlike FC networks can process all input at the same time, it has to process each input data after going through all the previous input. Consequently, the training and inference speed is slower than other models. In addition, for some relatively long sequences such as paragraphs, traditional RNN is not good at remembering the information from a long time ago. If we roll out RNN with a very long input sequence, it can be seen as a neural network with many layers. As the network goes deep, the multiplicative gradient could increase or decrease exponentially and the model cannot learn anything about the long-term dependency [26, 49, 57]. This issue is also named as gradient explosion and vanishing.

There are different methods to solve some of the potential problems. Pascanu et al. proposed gradient clipping, which clips the magnitude of gradient and it makes the gradient will not be dramatically large [**?**]. It is also proved by Zhang et al. that this method can accelerate the training of network [92]. More advanced recurrent neural network structures, such as gated recurrent unit (GRU) [11] and long short-term memory unit (LSTM) [27], are developed to solve the issue of long-term dependency. They use memory cell and gate unit to control what to memorize or forget in each time step. Besides, bidirectional RNN is proposed by Schuster et al., and it is able to process information from future time frame additionally [65].

2.1.3 Word Embeddings

Transforming the sentences or texts into the type that can be understood by machine is the foundation of NLP tasks. Word embeddings are the most common representations for words. They encode words into fixed-length, dense, and distributed vectors that can be easily taken as input for machine learning algorithms and strategies. These vectors can also be used to calculate similarities or the distance between words.

The idea of word embeddings starts from learning a distributed representation for words which allows each training sentence to inform the model about semantically similar sentences [47]. Instead of generating the embeddings directly, a neural network language model (NNLM) is trained to predict the following word based on the input sequence. It contains a projection layer and a hidden layer besides input and output layer. The projection layer projects the raw words vectors into word embeddings.

To minimize computational complexity and improve data efficiency, Mikolov et al. proposed two new model architectures, the continuous bag-of-words model (CBOW) and the continuous skip-gram model [43]. CBOW predicts the current words given the context, whereas Skip-gram predicts the surrounding words according to the current word. More importantly, these two models attempt to explore simpler architectures, so they remove the hidden layer, and the projection layer is shared for all words. The result showed that simple models have higher semantic and syntactic accuracies.

Word2vec and Glove are two well-known word embedding models that has been frequently used [44, 51]. Based on skip-gram model, Word2vec used negative-sampling method to improve the quality of the vectors. In addition, by subsampling of the frequent words, the training speed is boosted and the accuracy of the learned vectors of the rare words is improved. In comparison, Glove is a global log-bilinear regression model which combines the advantages from global matrix factorization method and local context

window method. The training strategy efficiently using the statistical information from the word-word concurrence matrix. This model also outperforms other related models on word analogy, word similarity, and named entity recognition tasks.

2.2 Encoder-decoder Architecture

In the field of NLP, most tasks have a variable length of input sequence and output sequence. For example, in the question answering task, the length of the answer depends on the type of question. For the case when the input length and output length are unclear, the encoder-decoder architecture is developed [12, 75]. It is also known as the sequence-to-sequence model because it takes a sequence as input and produces a sequence as output. The general framework contains a pair of encoder and decoder which is shown as follows:

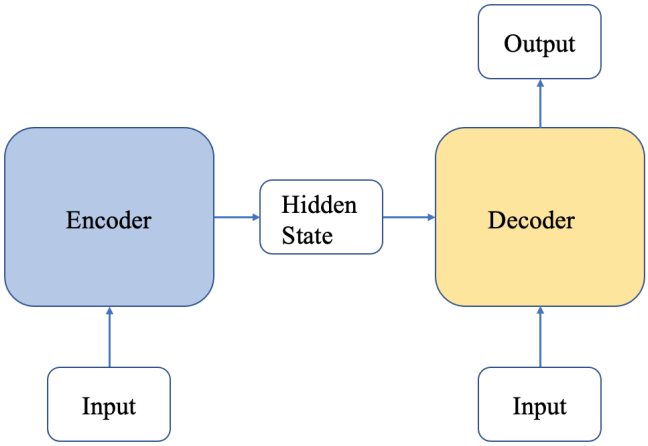


Figure 2.5: Encoder-decoder Architecture

The encoder takes input and transforms it into a fixed length hidden state, which is called a context vector. The decoder takes the hidden state and uses it to make prediction. The standalone RNN introduced in Section2.1.2 also has some attributes of an encoder-decoder model. The weights W_{xh} can be seen as the encoder and W_{hy} is similar to the encoder part. However, a better approach is to use two RNNs, one of them is only used to extract features from the input sequence, and the other focuses on decoding the extracted features into the label or probability. The encoder reads all the words till the last one, and

passes the last hidden state as the context vector to the decoder. On the decoder side, it takes the context vector as the initial state and reads one input token at a time. For machine translation task, it translates each token in the source sentence until reach the end.

2.2.1 Attention Mechanism and Transformer

When the input sentence is very long, the fixed size of the context vector limits the representation power of the RNN-based encoder-decoder model. It is also unreasonable to process different target words in a sentence with the same context information. To address this problem, Bahdanau et al. proposed the attention mechanism, which achieved much better performance on long sentences[3]. The combination of attention and RNN also had great success in a variety of tasks, including modeling word embedding, answering questions, and machine reading comprehension [52, 69, 67].

For attention mechanism, when decoder receives an input token, it identifies the relevant part from the source sentence and generates the context vector for this input. In simpler terms, the decoder does not always use the same context vector any more. The calculation of context vector c_i for i_{th} word is showed as follows:

$$c_i = \sum_{j=1}^{n} \alpha_{ij} h_j \tag{2.10}$$

It is the weighted sum of all the hidden states in the source sentence with length n. the weights α_{ij} are computed by:

$$\alpha_{ij} = \frac{exp(f(d_{i-1}, h_j))}{\sum_{k=1}^{n} exp(f(d_{i-1}, h_k))} \tag{2.11}$$

The formular takes the softmax of attention score between previous decoder hidden state d_{i-1} and the encoder hidden state h_j. It evaluates the relevance between all the words in source sentence and current word at position i. The scoring function f is parameterized by the FC network, and it can be trained within the back propagation.

Several studies have reported that the attention mechanism significantly improves RNN's performance, but they did not show what the performance of the standalone attention is [3, 69, 67, 52]. To answer this question, Vaswani et al. introduced the transformer model which used the self-attention layer and the fully connected layer, an advanced attention technique, without any recurrent structure in both the encoder and decoder [79].

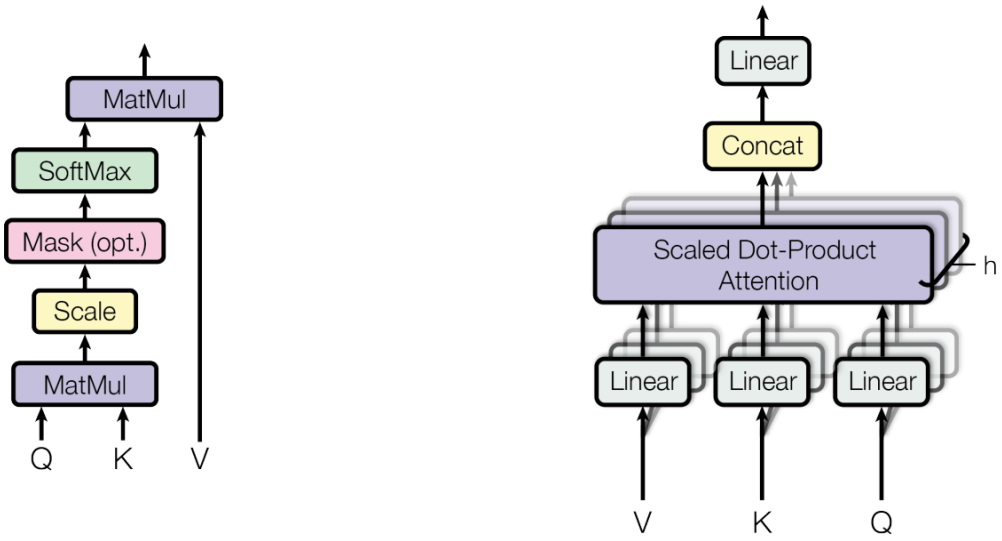


Figure 2.6: Left: Self-Attention. Right: Multi-head Attention [79]

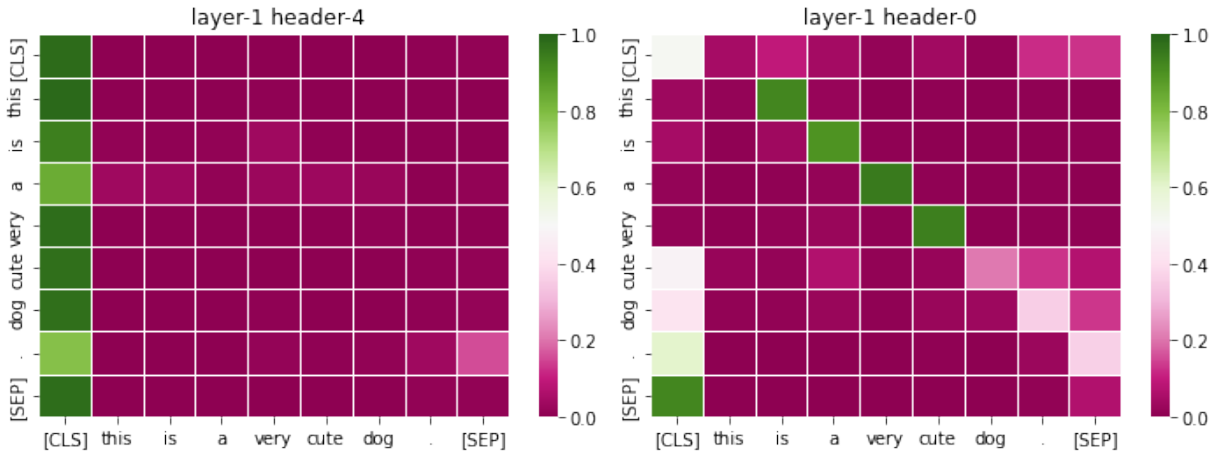
The multi-head self-attention technique is showed in Figure2.6. To recall, the classic attention considers the dependency relationship for the words in source sentence against the words in target sentences. Self-attention did one more step, it first computes the relationship between words in encoder and decoder side independently, then it aggregates them to get the dependency between source and target.

The equation in detail is

$$Self\text{-}attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}})V, \tag{2.12}$$

where d is the dimension of query.For example, if we want to derive the output of self-attention for one word in the source sentence. First, this word is mapped into three vectors which are query vector, key vector, and value vector. Second, its query vector is multiplied with the key vector of each word in the sentence to get their relationship. The scaling is to avoid the magnitude getting too large and influence training. The masking is used to make the prediction does not seen the data in the future time step. Finally, similar to the original attention layer, the softmax function is applied to get the attention scores, and they are multiplied by value vector to get the final output.

The operation in the left part of Figure2.6 can be considered as one attention head, and multiple attention heads are used in the transformer as the following:



(a) (b)

Figure 2.7: Attention Head Patterns from Multi-head Attention

$$Multi\text{-}head(Q, K, V) = concat(head_1, head_2, ..head_n) \tag{2.13}$$

where

$$head_i = self\text{-}attention(Q_i, K_i, V_i) \tag{2.14}$$

They used concatenation to connect the output from each attention head as a whole, and each head can learn different sets of parameters to obtain query, key, and value vectors.

Figure2.7 shows the heatmap of attention head pattern. In this map, each row represents the attention score of each token according to all the other tokens in this sentence. The attention scores in each row should add up to 1. Both graph is from the attention head in the same layer but they captures different relationships between the words. The first one focuses on the first special token [CLS] while the second one pays more attention to the surrounding words.

Compared to single-head attention, multi-head attention can achieve the same amount of representation ability with much less layer, enhancing training stability [39]. Another benefit is that it can identify more linguistic knowledge in different attention patterns [14]. An et al. claimed that multiple attention heads act as more sample data points to estimate the underlying posterior distribution [1].

Position embedding is used to give more information about the position of each word in the sentence. This is important because BERT does not know the order of the words

without it, and different orders can change the meaning of the sentence. Transformer used a relative position embedding showed as follows :

$$PE(pos, 2i) = sin(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.15}$$

$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.16}$$

pos is the position of the model, i is the index of the embedding dimension and d_{model} is the model dimension. Using sinusoidal function can map the position encoding in the range from -1 to 1 and tt provides the relative position between the words because it is periodic. Moreover, it also works well if the input sentence has a longer length than the training data.

2.2.2 BERT

Motivated by the success of transformer and the efficiency of transfer learning in computer vision domain, more researchers have started investigating how to carry out transfer learning in NLP domain. Devlin et al. introduced a large pre-trained model BERT which stands for Bidirectional Encoder Representations from Transformers [18]. At the time it was published, it obtained new state-of-the-art results on eleven NLP tasks and achieved notable improvement.

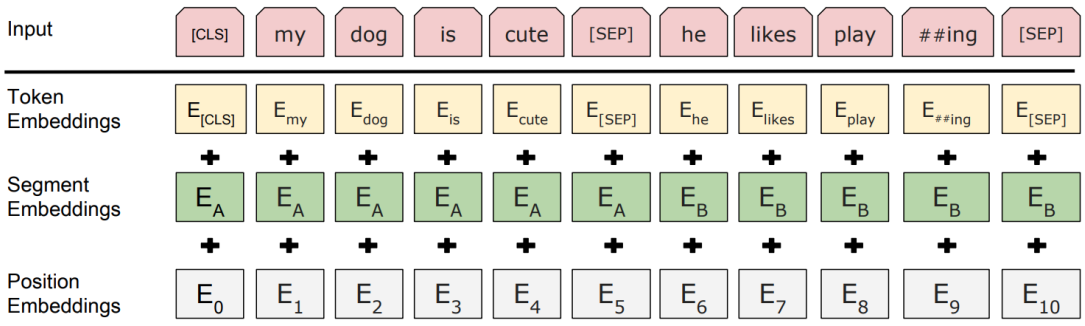


Figure 2.8: BERT Input Layer [18]

The architecture of BERT is similar to the transformer model, but it has some changes in the input layer so it can be easily fine-tuned on different kinds of down stream tasks. Figure2.8 shows the BERT input representation, and some special tokens are add to the

original sentence. Each input sequence starts with a token [CLS] and ends with the token [SEP]. When pretraining on next sentence prediction task, the output vector at [CLS] token is used to make the prediction. Therefore, it is also used to perform the sentence classification in downstream task. If the input is a sentence pair, then two sentences are also split by the token [SEP].

The input sequence is mapped into the sum of three embeddings, namely token embedding, segment embedding, and position embedding. The authors use WordPiece embeddings, which contain more than 30000 tokens in the vocabulary [87]. Segment embedding is used to identify whether the sentence is the first or the second one. Unlike the transformer, BERT uses a learnable absolute positional embedding. The experiments showed that it generally results in a better classification performance [83].

BERT's remarkable success comes from the pretraining-then-fine-tuning framework. It is pre-trained on BooksCorpus and English Wikipedia with more than 3000M words and learns a general-purpose language model by two unsupervised tasks, masked language modeling (MLM) and next sentence prediction (NSP).

For the MLM task, the tokens are randomly masked with a special token [MASK] under a certain percentage, and the model is trained to predict the masked tokens. Since the token [MASK] will not appear in the downstream tasks, the original token is maintained in 10% of the time. BRET inherits the bidirectional design from the transformer encoder, and the purpose of MLM is to learn a bidirectional model. A language understanding model that can process the word from both ends of the sentence can extract more context information than a model that only considers one direction [64].

The other task next-sentence prediction takes two sentences as input, and it is required to classify whether the second is the actual next sentence of the first one. Some NLP tasks, such as answering questions and identifying phrases, require knowledge of sentence relationship, and MLM does not cover this domain. NSP is conducted to make BERT learn a sentence-level representation. An ablation study is performed to validate the importance of NSP. Removing this task results in a critical drop in performance for all tasks that take sentences pairs as input [18].

BERT has high flexibility in the fine-tuning stage because it can be conveniently used for almost any downstream task that takes a single sequence or sequence pair. For example, for a question answering task, the first sequence can be the document, and the second is the question. BERT simply concatenates two sequences with [SEP] and takes the entire sequence as input. Similarly, in the text similarity task, one sentence is the reference sentence and the other the source sentence. The self-attention mechanism allows BERT to extract features from different types of sequences.

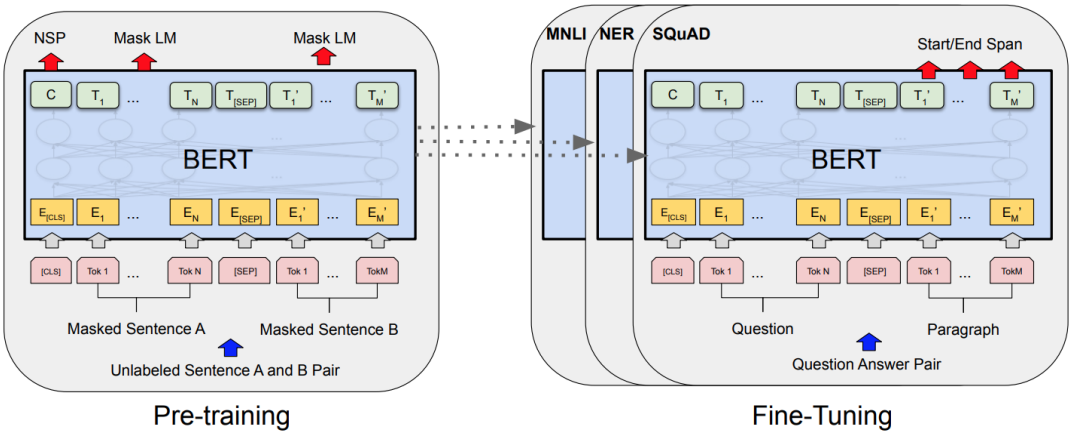


Figure 2.9: Pre-training and Fine-tuning [18]

It is also very straightforward to adapt to the expected output type. For classification tasks, we can simply stack another linear layer that follows the last hidden state of [CLS] token. As we mentioned before, the hidden state of this token is used for NSP in the pre-training stage, so it contains the information of the whole sequence. For question answering, this task provides a passage and a question, and the model needs to label the answer in the passage with the start and end positions. In this case, a start vector V and an end vector E are introduced. We first take the dot products between the final hidden state of each word and V, and the probability of a word to be at the start position is the softmax of its dot product result over all the words. Then, we repeat the same thing for the end position to localize the answer. The illustration of the pre-training and fine-tuning is shown in Figure2.9.

In the fine-tuning stage, BERT is initialized with exactly the same pre-trained parameter. Depending on the target downstream task, the corresponding classifier layer is randomly initialized and attached to the pre-trained model. Once the whole model is set up properly, its parameters can be fine-tuned with parallel data end-to-end. Since the parameters have been trained, the model can be fine-tuned with 3 epochs and a relatively low learning rate for each task [71].

To conclude, BERT provides an easy and powerful solution to use transfer learning in NLP. It shows the ability to extract useful features from massive amount of data. More importantly, this model can fit into different contexts and problems through fine-tuning while achieving impressive accuracy.

2.3 Model Compression Methods

2.3.1 Quantization

Quantization means reducing the number of bits in the weights or converting the floating point number into integers. Thus, the size of the model decreases and takes less space on the disk. Also, by reducing the precision level, the numerical operations on integers are more efficient than floating-point number. The latency will be shorter and the energy consumption can be reduced. The most common data type is the 32-bit floating point and the 64-bit floating point, and can be converted to 16-bit, 8-bit, 4-bit, 2-bit or 1-bit using different approaches [17, 4, 13, 72].

Quantization can be divided into Post Training Quantization (PTQ) and Quantization Aware Training (QAT) depending on when it is performed. PTQ first computes the cut-off point of the bit width and then truncates the parameters into the desired bit width. Subsequently, the model does not need to be fine-tuned [78]. On the other hand, PTQ computes the target bit width and updates the bit width of the quantized weights while training the model [6].

2.3.2 Pruning

Network pruning refers to the identification of irrelevant parameters and components and their removal from the model. When the number of operations is reduced, the inference speed and run-time memory are improved. Regarding the parts that are pruned, pruning involves unstructured pruning and structured pruning. Structured pruning focuses on removing components as a whole, such as layers or channels. Figure2.10 shows how the model becomes compressed with structured pruning. In the resultant model, the second layer and all the neurons it contains are removed. Then, layer 1 is connected to layer 3 directly. Its benefit is that the pruned model is easy to reconstrue by updating the hyper-parameters. The drawback is that it is not as flexible as unstructured pruning, because the pruned components can still contain some important parameters. These important ones have to be pruned along with the redundant ones.

Unstructured pruning refers to the removal of individual weights or parameters, which is shown in Figure1.1. The original model structure is still maintained, but some connections between neurons are not linked. It is more difficult to decide which parameter to prune because we have to evaluate the importance of every one of them. Despite this, it avoids great damage to the original performance because it only drops the unnecessary ones.

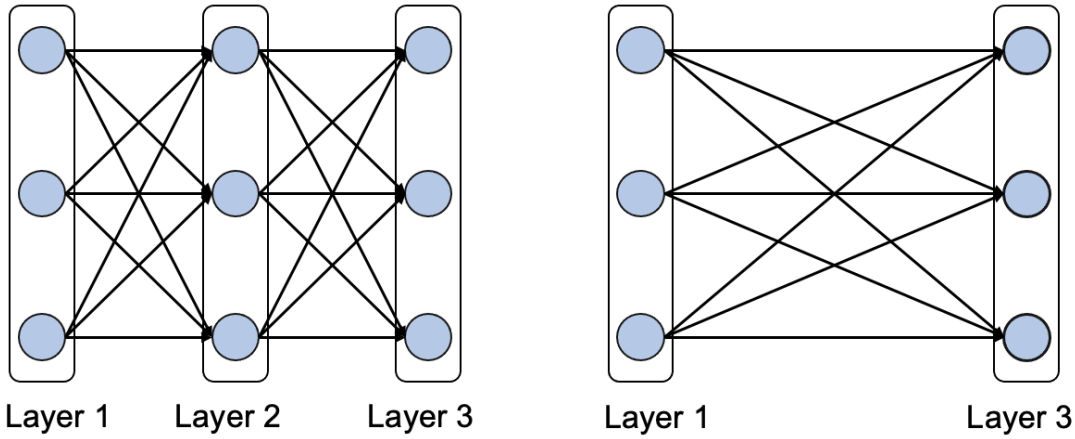


Figure 2.10: Structured Pruning

There are different ways to evaluate the importance of the parameters. The most straightforward is pruned on the basis of the magnitude of the values [23, 33]. If the parameter of the neuron is close to zero, then its input is ignored by the mode. Thus, it is safe to remove this neuron. Another method is iterative pruning, which is used mainly in structured pruning. In each iteration, it removes one component from the model and computes the negative impact on accuracy. In each iteration, the component with the least impact is pruned until the accuracy reaches certain thresholds [5, 48]. There is also a method that uses the reweighted $l1$ minimization algorithm to find a sparse subnetwork [24] or prune parameters based on redundancy and similarities [16].

2.3.3 Knowledge Distillation

The process of knowledge distillation uses the large pre-trained model, which is the teacher, to train a smaller model, namely the student [25]. The student model does not have to use the same structure as the teacher. It can use a completely different one or replace some blocks from the teach model. For example, Mukherjee et al. use BiLSTM as the student for the pre-trained transformer [46]. The transformer needs to take all input tokens in one shot, but LSTM can take them sequentially. Hence, LSTM requires less memory and is more suitable for resource restricted devices.

Different types of knowledge from the teacher model can be used for training. Knowledge can be grouped mainly into three categories on the basis of its properties. The first is response-based knowledge, which generally is the output state of the teacher. The target is

to make the output logits of the last hidden layer in the student model match the teacher's [91]. It is the most direct method, and the pattern is easy to learn, but it fails to learn the features extracted in the hidden layer. Another kind of knowledge is Feature-Based Knowledge. That is, not only the output state, the hidden states in the hidden layers are also used to guide the student network [58]. In this way, the student model learns the feature map of the input data and the relationship between hidden layers.

2.4 Related Work

Previous studies have proposed different kinds of method to compress the BERT-based model. The idea of removing attention heads in Multi-head self-attention layer was first proposed by Voita et al. [81]. They evaluated the contribution of each head using layer-wise relevance propagation. They showed that a large portion of the heads can be removed without significant degradation in performance. Michel et al. also made a similar observation that some layers only need one head [42]. Kovaleva et al. revealed the fact that there is repeated information encoded in attention heads that BERT can benefit from disabled some of them [34]. Although all these works focused on removing entire attention head and did not consider the redundancy in feed-forward network, but our approach prunes the redundant part in both self-attention layers and fully connected layers.

There are also structured pruning methods that consider not only the attention heads. For example, Peer et al. developed a greedy algorithm for layer-wise pruning. It iteratively pruned the layer with minimum loss and made the remaining layers optimal [50]. Lin et al. identified redundant nonlinear mapping in the residual module and removed the corresponding heads and feed-forward layers [38]. Our method uses a more fine-grained strategy and prunes at the neuron levels. Khetan and Karnin reduced the depth and width of the original BERT model and obtained multiple variants of it, which are called schuBERT [30]. Afterwards, they pre-trained and fine-tuned them from scratch. In comparison, we leverage the knowledge from pre-trained weights and compress the model during the tuning stage. We don't have the time and resource cost on pretraining the model again. Li et al. proposed a hardware-friendly pruning method. They divided the weight matrix into blocks and pruned them by Reweighted Group Lasso Optimization on the magnitude of the weights [36]. It is similar to ours since both methods perform structured pruning at the finest level. However, our method has a different pruning strategy, so we learn the importance of each individual neuron first and then drop the parameters of the unimportant ones. Network Slimming and Vision Transformer Slimming used a similar way to identify the unimportant neurons, but they focused on the computer vision tasks [41, 9].

Another popular technique to reduce the size of the model is knowledge distillation, and it is first utilized in pre-trained language models by Sanh et al. [62]. They created a 6-layer BERT model, namely DistillBERT. It is trained with the same corpus and settings as BERT. To further improve the training objectives, Sun et al. proposed additional loss functions for the student models to learn the hidden states of the teacher model [73]. Truc et al. proposed pre-trained distillation that first pre-trained the student model on the objective of the masked language model and then tuned it for downstream tasks [73].

Chapter 3

Approach

3.1 Motivation and Overview

Several attempts have been made to use a customized method to estimate the importance of parameters, such as Center Kernel Alignment similarities [16], magnitude of weights and bias [23], or gradient-based optimization [37]. Although they managed to identify the redundant neurons from the network, they are very inefficient because all of them need more post hoc computation after fine-tuning is done. To reduce the additional work, we attempt to model the importance of neurons and approximate these values during the fine-tuning stage.

Furthermore, BERT-based models are usually not as deep as large computer vision models that have more than 100 layers [56]. Instead, they usually have a larger width, as demonstrated by a hidden size of 768 for the base version. Compared to CV models, it is more risky to prune BERT by layers because they store more relationship between their input and output. Thus, we also wonder if there is a way to estimate the contribution of each neuron individually, so that we can easily rank and prune the unimportant ones. Motivated by these ideas, we propose neural slimming, which prunes redundant neurons and preserves essential ones based on their contribution to the output.

This chapter is orgainzed as follow, we will (1) provide the detailed explanation of the problem definition and the proposed neural slimming approach, (2) introduce the architecture of SlimBERT, where we perform neural slimming on the BERT model, (3) discuss the design of object function, and (4) explain the method to analyze the redundancy in static word embeddings.

3.2 Problem Definition

We first introduce the notation to describe the variables in the problem and our approach as follows.

- x: Input into the model
- y: Output from the model
- L_i: i_{th} layer
- W_{ij}^{l}: Weight that transforms neuron i at layer L_l to neuron j at layer L_{l+1}
- n_i^j: i_{th} neuron at j_{th} layer
- θ: All the learnable parameter in the model
- n_{dim}: size of dimension in hidden layer

In this work, we focus on pruning the pre-trained language model BERT [18]. It uses the architecture of the encoder part of the transformer [79]. There are different sizes of BERT with adjustable encoder number and hidden dimension size. We use the most common one, BERT-Base, in BERT compression problems. It has 12 encoders and the hidden dimension n_{dim} is 768. There is no recurrent structure, so all data are first passed into the embedding layer to retrieve the token embeddings and then flow through the encoders sequentially to obtain the context vector of the input sequence.

Each encoder block is stacked by two subnetworks. The features first enter the multi-head self-attention layer, and there are 12 attention heads for BERT-Base. The self-attention layer is connected to a linear layer and these two layers form the first subnetwork. The following subnetwork is a position-wise feed-forward network with two layers. The first network captures the relationship between tokens and sentences, while the second network extracts syntactic and semantic features.

The BERT sublayers and their parameters are shown below.

- Embedding Layer: $768 * 30522 \approx 23.4M$
- Self-attention Layer: $768 * 768/12 * 3 * 12 \approx 1.7M$
- Linear Layer: $768 * 768 \approx 0.6M$

- Feed-forward Layer 1: $768 * 768 * 4 \approx 2.36M$ (the size of hidden layer is 4 times the hidden size)
- Feed-Forward Layer 2: $768 * 4 * 768 \approx 2.36M$
- 1 encoder: $1.7M + 0.6M + 2.3M + 2.3M \approx 6.9M$
- Total: 84M without embedding layer, 109M with embedding layer

Most layers use the hidden size following the same setting of n_{dim} in BERT-Base, except for the first feed-forward layer. It uses a much wider hidden size to extract more complex features. Within the BERT model, the majority of parameters are located in the embedding layer, the self-attention layer, and the first feed-forward layer. We will perform pruning on all these layers to fully compress the network.

In the model pruning problem, we are given the base model BERTBase, and we would like to obtain a pruned model M_{pruned} with the same structure but fewer parameters. We need to consider two objectives. The first is to minimize the difference between the prediction and the ground truth, which is the same as the original model. Consider the pruned model as the a function f that maps the input features to the output label, the definition of the first object is

$$\min_{\theta} L(f_{pruned}(x, \theta), y) \tag{3.1}$$

The second is to minimize the number of neurons in the pruned model, which can be written as

$$\min \sum_{j=1}^{12} (n^j_{self-attention} + n^j_{linear} + n^j_{feed-forward\,1} + n^j_{feed-forward\,2}) + n_{embedding} \tag{3.2}$$

In summary, the overall objective is to remove the parameters that have a minimal impact on $M_{original}$'s prediction. When removing the same number of neurons, we always want to obtain the optimal substructure from the base model.

3.3 Neural Slimming

According to the type of components that are pruned, pruning is classified as structured pruning and unstructured pruning. With regard to structured pruning, it prunes the

components as a whole, such as network layers or encoder blocks. It is easier to measure the importance of these components, so this method is more straightforward and easier to implement. Unfortunately, it usually has to scarify either the compress rate or the accuracy. This is because it is very unlikely that all the parameters of the component will be useless. Especially in the last few iterations of pruning, the remaining components contain a larger portion of important parameters, and the performance drops sharply if we remove any one of them.

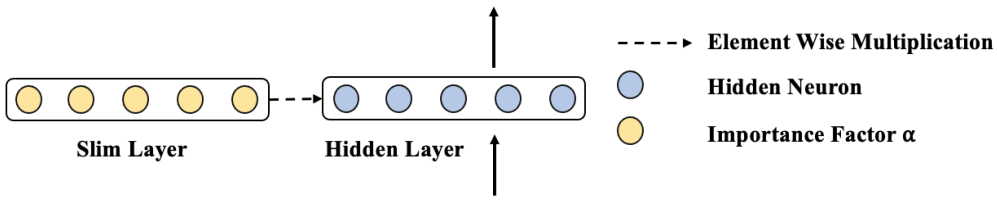


Figure 3.1: Slim Layer

Unstructured pruning aims to remove individual weights and biases that are less important. It can obtain a higher compression rate than the structured one, which leads to a smaller model size. However, it does not accelerate the inference speed because its weight matrix cannot be reshaped into a smaller size, regardless of many of them being zeros.

In the case of BERT, it contains a limited number of layers and attention blocks, so removing them as a whole results in missing a great number of features. Performing structured pruning by neurons provides a nice balance between compression rate and inference speed. The reason is that removing neurons can save as many important parameters as possible. Meanwhile, removing one neurons can help us completely eliminate one column of weights from the weight matrix. Thus, the connection between layers can be presented by a smaller matrix, and the time cost on mathematical operation can also be shortened.

To estimate the contribution of each neuron, we introduce a importance factor α which is a learnable parameter. Figure3.1 shows the usage of a slim layer. A slim layer is a group of independent importance factors that are individually optimized. Each time we want to perform pruning on a certain layer, we connect the slim layer to this layer. α ranges from 0 to 1, and we perform element wise multiplication between α and each neuron to scale their values. Thus, let n' denotes the original neuron values and their scaled values in layer i will be:

$$(n_i^1, n_i^2, ..., n_i^j) = (n_i^1 * \alpha_i^1, n_i^2 * \alpha_i^2, ..., n_i^j * \alpha_i^j) \tag{3.3}$$

We initialize all the importance factors to 1 before fine-tuning starts. When they are all 1, the values in the hidden layers do not change, and the model behaves normally. At this time, all neurons contribute equally. Once fine-tuning starts, all the parameters including α are updated using the loss function.

$$L = L_{data} + L_{compression} \tag{3.4}$$

where:

$$L_{data} = CELoss(y, y'), \tag{3.5}$$

$$L_{compression} = \lambda \sum_{i=1}^{n} \beta_{L_i} \sum_{j=1}^{m} \log(1 + (\alpha_i^j)^2) \tag{3.6}$$

The proposed loss function contains two terms, namely, model loss and compression loss. The prior one is the loss of model on ground truth, which optimizes the accuracy. BERT model is used mainly to predict the probability of the target token in different scenarios, so the cross entropy loss is used here [28].

For the latter, we use it to constrain the importance of redundant neurons. It takes the sum of factors from all layers and can be optimized by pushing each α towards zero. Since removing unnecessary neurons does not damage model loss, it can be safely scaled down, and its importance factor can be decreased. The importance neurons are still reserved because the first loss term guides the pruning procedures, and dropping them causes the first term increase.

Since it is not applicable for α to be exactly zero, we set a very small number as the threshold. Once it is smaller than the threshold, the output of this neuron can be ignored. At the same time, we can also remove the corresponding parameters related to this neuron, and the resulting model required fewer mathematical operations.

There are two hyperparameters about the compression term worth pointing out. First, the compression weighting term λ defines the ratio between model loss and compression loss and is used to balance the pruning rate and model performance. If we set it to zero, then the model is fine-tuned as what is used to be, and no neuron is deleted. To perform pruning and fine-tuning at the same time, this hyperparameter needs to be tuned. This is because the compression term is normally larger than the model loss, and we do not want the model to get over-compressed. The other adjustable hyperparameter is the layer weighting factor β. It is multiplied by the compression loss of each layer. To guide the compression rate of different layers, we can assign different values. We can also increase this value to accelerate the compression of some specific layers. Normally, we always set this value to 1 so that all layers are pruned fairly.

3.4 Architecture of SlimBERT

Due to the flexibility of the slim layer, we can easily apply it to the parts that we want to prune in the model. For BERT, we use it on all the layers, including the embedding layer, the multi-head self-attention layers, and the fully connected layer. We can consider the linear layer and the feed-forward layer as FC layers because they all have the same structure. Figure3.2 shows how the slim layer is applied to one of the encoder blocks in BERT. The size of each slim layer matches the number of neurons in each connected layer. The use of neural slimming in other encoder block and embedding layer follows the same pattern.

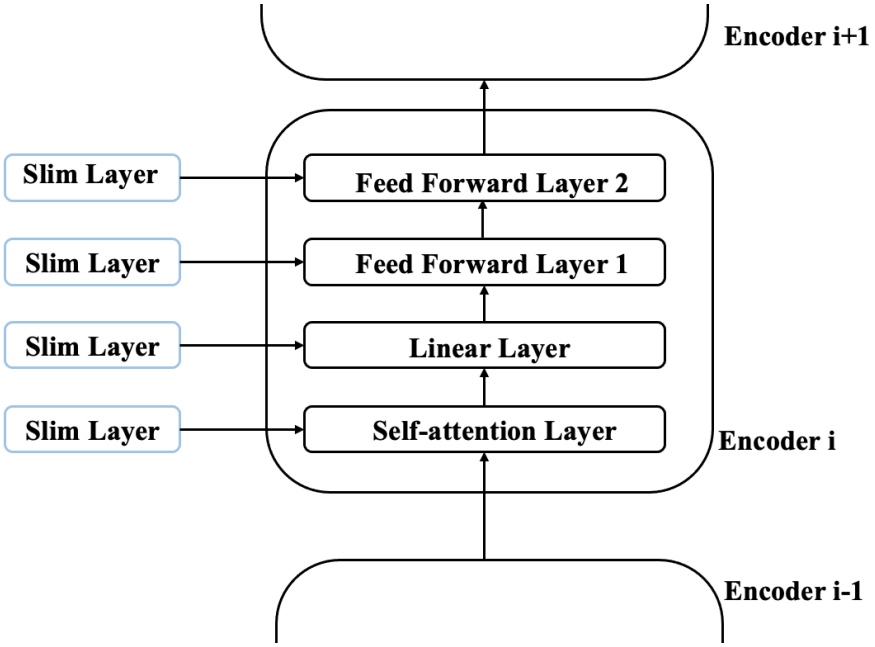


Figure 3.2: SlimBERT

After neural slimming is performed, the output of the FC layer will be

$$n^{i+1} = A \odot W^i n^i \tag{3.7}$$

where A denotes the vector of the slim layer. If any of the neurons can be pruned, we can correspondingly remove the weights in the weight matrix. For example, if neuron n_m^l gets pruned, then we can also safely remove the weights $W_{1m}^{l-1}, W_{2m}^{l-1}, ..., W_{nm}^{l-1}$ in layer $L^l - 1$ and weights $W_{m1}^l, W_{m2}^l, ..., W_{mk}^l$ in layer L^l.

The embedding layer works similarly to a dictionary. It maps each token to its corresponding embeddings, so its parameter is a weight matrix $W_{emb} \in \mathbb{R}^{v*dim}$ where v is the vocabulary size. The BERT vocabulary contains 30522 tokens. Let I be the index vector of the input token, and the pruned embedding layer is

$$embedding = A \odot W_{emb} I. \tag{3.8}$$

With respect to the number of parameters in each layer, we can observe that the embedding layer holds a large portion of parameters, which is more than the size of three encoder blocks. Each time we prune one neuron v_i^{emb} in embedding layer, we can remove 30522 parameters because the model does not need to know the value in dimension i for all words.

For multi-head self-attention layer, we perform pruning for each attention head and then concatenate them together.

$$Scaled Multi\text{-}head(Q, K, V) = concat(A_1 \odot head_1, A_2 \odot head_2, ..A_n \odot head_n) \tag{3.9}$$

where

$$A_i \odot head_i = A \odot softmax(\frac{QK^T}{\sqrt{d}})V. \tag{3.10}$$

Different attention heads capture different kinds of relationship patterns. In this case, we train the scaling factors separately. Once the prunable neurons are identified, we can remove the parameters at the same location in the query vector, key vector, and value vector together.

3.5 Tuning and pruning strategies

Concerning the neural slimming method, we propose two tuning methods to make the pruned model fit the downstream tasks. The first one is called prune-after-tune, which is similar to the standard BERT fine-tuning procedures. First, we load the pre-trained BERT model. We also initialize the scaling factor α for each neuron with 1. At this time, all neurons are equally important and the model acts as a normal BERT classifier. Then, we set up the hyperparameters the same as the original BERT mode, and then fine-tune it on the target dataset with the loss function we proposed for neural slimming. After the tuning is complete, the importance of each neuron is also learned for this task.

To perform pruning, we can set a pruning threshold and discard any neurons with a factor smaller than this threshold. For example, figure 3.3 demonstrates the neuron

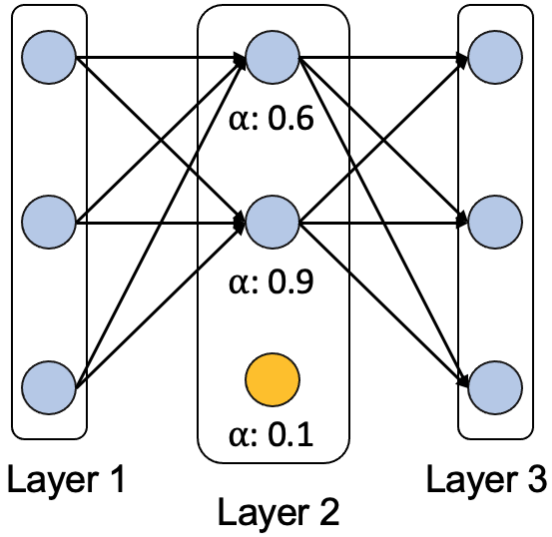


Figure 3.3: Neuron slimming when threshold is 0.2

slimming when the threshold is 0.2. The third neuron in the second layer has an importance of only 0.1. Therefore, we cut off its connection with all neurons in the adjacent layer, and the dimension of the second layer becomes 2. Finally, the pruned model is obtained.

Although the weights of the unimportant neurons in the pruned models from the one-shot pruning are very small, they still impact the prediction result. We introduce the prune-then-tune method to tune a clean version of the pruned model. Before doing so, we first rank the learned scaling factor from the lowest to the highest. After that, we can set the pruning threshold that yields the desired density ratio and mask all corresponding neurons. Following that, we load the weights from the original pre-trained BERT, and then tune this pruned model with standard loss function according to the type of task.

Chapter 4

Experiments and Results

4.1 Datasets

The General Language Understanding Evaluation (GLUE) benchmark involves 9 tasks and their corresponding datasets to evaluate and compare the performance of NLP models [82]. We use the following 7 of them.

The Microsoft Research Paraphrase Corpus (MRPC) contains pairs of sentences in English extracted from news sources on websites [20]. They are annotated by humans on whether they are equivalent on paraphrase and semantics. The label is not balanced and 68% of them are positive samples.

Multi-Genre Natural Language Inference Corpus (MNLI) is a collection of crowd-sourced sentence pairs labeled with textual entailment information [86]. The first sentence is the premise, and the other is the hypothesis. The task is to classify whether their relationship is entailment, contradiction, or neutral. The premises are collected from government reports, audio, and novel. There are two validation sets because the data set is from different genres and domains. The matched version means the source of the training set and the development set matches. Mismatch means that they do not match.

The Quora Question Pairs (QQP) is a collection of questions pairs from the social question-and-answer website Quora. They are classified by whether a pair of questions is semantically equivalent. It is also an unbalanced dataset with 63% negative samples.

Question-answering NLI (QNLI) consists of question-passage pairs, where the passages are from Wikipedia. It is transformed from the Stanford Question Answering Dataset,

Dataset	Training Set Size	Dev Set Size
MRPC	3668	408
MNLI	392702	9815(matched)/9832(mismatched)
QQP	363870	40431
QNLI	104743	5463
SST-2	67350	873
RTE	2491	277
CoLA	8551	1043

Table 4.1: Size of 7 Datasets

which collects the question and answer pairs [54]. The task is to identify whether the passage entails the expected answer.

The sentences in SST-2, The Stanford Sentiment Treebank, are extracted from movie reviews on rottentomatoes.com [68]. Each sentence is labeled as 1 for positive sentiment or 0 for negative sentiment by Amazon Mechanical Turk, and the label reflects the writer's preference for the movie.

Recognizing Textual Entailment Dataset (RTE) aggregates data from a few text entailment challenge benchmarks, including RTE1, RTE2, RTE3, etc [15, 22]. Unlike other entailment datasets, it converts neutral and contradiction into one class, non-entailment. Therefore, it becomes a binary classification at the end.

CoLA, The Corpus of Linguistic Acceptability, is a collection of sentences from linguistic publications [85]. Each of them is labeled according to its grammatical acceptability by their original authors. It is also a single sentence classification task.

The size of each dataset is shown in Table 4.1. Our experiments contain both small and large datasets. For example, RTE and MRPC have fewer than 4,000 training sentences. On the other hand, MNLI and QQP have more than 0.36 million records. Therefore, we can reveal the performance of our method with different amounts of training data.

4.2 Evaluation Metric

In order to perform a fair comparison of neural slim and other pruning methods, we use the following criteria to quantify their compression power and performance on downstream tasks.

Regarding the evaluation of how much the proposed method compresses the BERT model, we used the compression rate, which is calculated by

$$compression\ rate = \frac{remaining\ parameters}{total\ number\ of\ parameters} \tag{4.1}$$

Its value falls in the range from 0 to 1. 1 means that no parameter is pruned at all. As it approaches 0, it means that the pruned model uses fewer parameters compared to the original structure. Not all compression methods were applied to the word embedding layer. To compare with these methods, we also compute the compression rate without embeddings, which only considers the remaining parameter number and the total parameter number in the encoder blocks. Furthermore, we compute the density of the BERT model and its components with

$$density = \frac{remaining\ neurons}{total\ number\ of\ neurons}. \tag{4.2}$$

Since removing one neuron leads to the removal of different numbers of parameters in different layers, we also want to measure the density ratio at the neuron level. It helps us to analyze the redundancy in different components, such as linear layers and attention layers.

Concerning the performance of the pruned models, we followed the metrics required for each task from the GLUE benchmark. Accuracy is the most common and straightforward metric. It is calculated by

$$accuracy = \frac{correct\ predictions}{all\ predictions} \tag{4.3}$$

Higher accuracy shows that the model makes more accurate predictions given all the input data. However, it is not a reliable metric if the dataset is imbalanced. For example, when the dataset has 90% positive samples and the model naively predicts a positive label for all input data, it still achieves a 90% accuracy, but it does not learn anything about the negative samples.

A better metric for an imbalanced dataset is the f1 score. It is computed as shown in the following definition.

$$f1\ score = 2 * \frac{precision * recall}{precision + recall} \tag{4.4}$$

where

$$precision = \frac{True\ Positives}{True\ Positives + False\ Positives}, \tag{4.5}$$

$$recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}, \tag{4.6}$$

Precision computes the portion of correct predictions in all records predicted as positive, while recall measures the one in the actual positive labels. F-score combines both metrics into one with a harmonic mean of them, with 1 being the best score and 0 being the worst value. It evaluates the overall performance on precision and recall. On the other hand, it is an asymmetric metric if the classes are imbalanced. If we swap the positive and negative labels, then the f1 score also changes. The class that is considered positive has a higher contribution than the negative one, so this metric is also not normalized.

Matthews correlation coefficient, MCC, does not have the same problem as the f-score [10]. It is calculated as

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TN + FP)(TP + FN)(TN + FN)}}. \tag{4.7}$$

The value falls within the range of -1 to 1 where -1 means that the model makes the opposite prediction all the time, 0 means a random prediction, and 1 means a perfect prediction. No matter the dataset is balanced or not, all labels are considered equally important in MCC. It also remains the same if we exchange positive and negative classes.

4.3 Tasks

We used seven natural language inference tasks to evaluate the performance of neural slimming in different scenarios. They cover two kinds of input, single sentence and sentence pairs. For single sentences, the task focused on identifying some properties or characteristics of the given sentences. In contrast, the sentence pair tasks required the model to understand the relationship between them.

Tasks fall mainly into four categories depending on what they want the model to predict. The first is similarity classification, which is performed on MRPC and QQP. In these tasks, we want to determine whether two sentences are paraphrases of each other. Both datasets have an unbalanced class distribution, so we evaluate the accuracy and f1 score for them. MNLI, QNLI, and RTE are used for natural language inference tasks. The model is asked to interpret the textual entailment information within the sentence pairs. Each corpus has different data sources. MNLI collects sentences in written English and speech, QNLI involves questions and answers, and the data in RTE is from Wikipedia and news.

Hyperparameter	Selections
compression weighting term λ	1e-4, 1.5e-4, 2e-4
epoch	3, 5, 10
learning rate for BERT parameters	3e-5, 2e-5, 5e-5
learning rate for importance factor α	1e-3,1e-4,5e-4
batch size	16, 32

Table 4.2: Selections of Hyperparameters

Task	Epoch	lr_bert	lr_compression	λ	Batch size
MRPC	10	2e-5	0.0015	1e-4	32
QQP	3	2e-5	0.0003	1e-4	32
QNLI	3	2e-5	0.00075	1e-4	32
SST-2	3	2e-5	0.001	1e-4	32
RTE	10	2e-5	0.002	1e-4	32
CoLA	10	2e-5	0.002	1e-4	32
MNLI	3	2e-5	0.0002	1e-4	32

Table 4.3: Optimal Hyperparameter Combination

The remaining two categories are binary classifications of a single sentence. SST-2 is for sentiment classification, which predicts whether the sentiment of a sentence is positive or negative. The input sentences are movie reviews. For CoLA, the task is to tell if each sentence contains grammatical errors. The sentences in CoLA are from formal documents, such as journal articles, and SST-2 contains movie reviews.

4.4 Experiments setup

In this work, we focus on the pruning of BERTBase, so we follow the original setting of it and construct the model with 12 encoders, 12 attention heads for multi-head attention, and 768 for hidden size [18]. We use Pytorch for the implementation of our model. We also loaded its open-source pre-trained parameters for all the downstream tasks.

For each task, we first performed prune-after-tune to determine the importance of each neuron. Then, we pruned the model until we got the desired parameter number. Finally, we loaded the pre-trained weights in it and tuned the compressed model on the training set. Its performance on the validation set is also used to compare with other compression

methods because all other methods used the validation set as a benchmark. The selection of the hyperparameters is given in Table 4.2 and we choose the combination with the best performance. Table 4.3 shows the selected combination that gives the optimal results. We tuned the number of epochs and the learning rate of α. The selection of them depends on the size of the dataset. For a small dataset such as MRPC that has about 3000 records, we need 10 epochs and a larger learning rate for compression to fully identify the unimportant neurons. But for larger datasets, we can just tune it for 3 epochs and set the $lr\_compression$ smaller.

We compared our pruning method with other approaches, including

- BERTBase: The original BERT model without any pruning [18].
- MvP: It used first-order information of the model and only kept the weights that were moving away from zero when training [63].
- SNIP: A structured pruning approach based on Spectral-Normalized Identity Prior [38].
- schuBERT: Pruning components that give the minimum increase in mode loss [30].
- Block structured pruning was a hardware friendly method that uses reweighted group Lasso optimization [36].
- PD-BERT: It performed knowledge distillation during the pre-training stage to obtain a smaller general-purpose language representation model [77].
- DistillBERT used additional distillation signals in the loss function when performing knowledge distillation [62].
- BERT-PKD introduced a "patient" teacher-student mechanism to leverage information from the teacher's intermediate layers as well [73].
- BERT-of-Theseus trained smaller student modules using teacher modules and then used them to replace teacher modules [88].
- Greedy-Layer Pruning used a locally optimal strategy on layer-wise pruning to obtain a global optimal solution [50].
- LEAP used a learnable pruning threshold and an adaptive regularization coefficient to perform unstructured pruning [90].

Method	C.R.	MRPC	SST-2	RTE	MNLI	QNLI	QQP	CoLA
BERTBase	100	87.75	93.1	69.1	84.8	91.7	89.35	60.5
Knowledge Distillation								
DistilBERT	60	87.5	90.7	59.9	79.0	85.3	84.9	43.6
PD-BERT	60	87.2	91.1	66.7	83.0	89.0	89.1	-
BERT-PKD	60	85.7	91.3	66.5	81.3	88.4	88.4	45.5
BERT-of-Theseus	60	**89.0**	91.5	**68.2**	82.3	89.5	**89.6**	51.1
SlimBERT (ours)	60	85.7	**92.6**	66.4	**84.4**	**90.6**	89.0	**58**
Structured Pruning								
Block Structured Pruning	70	**88.3/-**	89.3	63.9	82.9	88.2	**90.7/-**	52.6
SchuBERT	60	86.7/-	91.7	**68.5**	83.8	-	-	-
Greedy-Layer Pruning	50	77.9/85.4	90.7	60.6	81.2	87.7	89.7/86.3	45.0
SchuBERT	50	**86.3**/-	90.9	**67.3**	83.5	-	-	-
SNIP	50	-/88.1	91.8	-	82.8	89.5	88.9/-	-
SlimBERT (ours)	50	84.8/**88.8**	**92.1**	65.2	**83.9**	**90.7**	90/**86.5**	**55.6**

Table 4.4: Performance of the BERTBase model and pruned models on GLUE development set. C.R. stands for compression rate.

4.5 Results

4.5.1 GLUE tasks

The experimental results on the GLUE development set are shown in Table 4.4. Different methods use different metrics in their work. For knowledge distillation methods, we report the average accuracy and f1 for MRPC and QQP. For structured pruning, we report these scores separately. We take the average accuracy of MNLI-matched and MNLI-mismatched and MCC score for CoLA. For the rest of the tasks, the accuracy is evaluated. We also fine-tune the original BERTBase model on each task and report the performance as a baseline. The results of all other methods are from their original papers.

SlimBERT achieves remarkable accuracy across these seven tasks, and it gets around 90% accuracy for SST-2, QNLI and QQP. Although it has lower scores on RTE, MNLI, and Cola, they are very close to the fine-tuned base model we got. That is, the cost on the original performance is minor after pruning 40% parameters. It shows that BERT has a large number of neurons that are not helpful in downstream tasks, and our method successfully identifies these unimportant ones.

We compared our method with knowledge distillation (KD) methods and structured pruning (SP) methods. We observed that KD usually has a larger model size than SP

Method	C.R.	MRPC	SST-2	RTE	MNLI	QNLI	QQP	CoLA	Avg.
BERT-of-Theseus	60	**-0.5**	**0**	-2.9	-1.2	-1.7	**-0.2**	-3.2	-1.39
SlimBERT (ours)	60	-2	-0.5	**-2.7**	**-0.4**	**-1.1**	-0.4	**-2.5**	**-1.37**
Greedy-Layer Pruning	50	-5.7	-2.1	-4	-3.5	-3.8	-1.3	**-3.4**	-3.4
SlimBERT (ours)	50	**-0.95**	**-1**	**-3.9**	**-0.8**	**-1**	**-1.1**	-4.9	**-1.94**

Table 4.5: Drop on accuracy scores

because it aims to train a new model instead of extracting a subnetwork. To make a fair comparison, we first mask the same amount of parameters and then fine-tune the pruned model on each dataset.

When the compression rate is 60%, SlimBERT significantly outperforms four KD-based methods, including DistilBERT, PD-BERT, BERT-PKD, and BERT-of-Theseus on SST-2, MNLI, QNLI and CoLA. It means that BERT can be compressed efficiently without additional training strategies and objective functions. Our method also only has a 0.35% decrease on the QQP dataset.

Compared to other structured pruning methods, SlimBERT also obtains the best accuracy with a similar or even smaller model size in most of the tasks. Under the same concept of removing structured units, our method can reserve more important parameters and information for the downstream task in the BERT model. On QQP, it has a lower score than Block Structured Pruning by 0.7%, but it uses 20% less weights at the same time.

We also notice that SlimBERT does not perform as well as some of the competitors on MRPC and RTE. One potential reason is that each pruning method is performed on BERTBase models with slightly different performance due to the hyperparameters or random seed selection. Consequently, a terrible base model can limit the performance of the pruned model. Based on this discrepancy, we will analyze the performance damage on original models from the pruned ones.

In Tables 4.5 and 4.6, we calculate the differences in accuracy scores between the fine-tuned BERT and its compressed model. Our method is compared to other competitive methods that achieve the highest score in some of the tasks. After removing the noise from the inconsistency on base models, we can quantify the loss on the original performance. The result shows the ability of each method to recover the prediction made by the base model. It can be observed that a model with a higher accuracy can have a greater decrease on its original one.

After removing 40% or 50% of the parameters, SlimBERT shows impressive perfor-

Method	C.R.	MRPC	SST-2	RTE	MNLI	QNLI	QQP	Avg.
SchuBERT	50	-1.5	-1.2	-4.7	**-0.6**	-	-	-2
SNIP	50	-2.8	**-0.9**	-	-1.9	-2.1	-1.7	-1.88
SlimBERT (ours)	50	**-0.95**	-1	**-3.9**	-0.8	**-1**	**-1.1**	**-1.45**

Table 4.6: Accuracy Degradation of Structured Pruning at 50% Compression Rate

mance in maintaining original performance on the QNLI/MNLI natural language inference tasks, sentiment classification SST-2, and the QQP question pair similarity task. The drop is about or less than 1% on these tasks. RTE and CoLA are more challenging among all tasks; all compression methods have obvious shrinkage. SlimBERT has an average of around 2.5% for these tasks. We also found that the baseline accuracy reported on MRPC has a greater variance and its value ranges from 84% to 89%. Although SlimBERT has a lower accuracy on MRPC, it nearly reproduces the output from the base model.

For models with a 60% compression rate, SlimBERT is compared with BERT-of-Theseus, which is the best among four knowledge distillation methods. The proposed method has smaller decreases in all three language inference tasks and the grammatical error classification task. It shows that our method has a better ability to maintain the information on the sentence relationship. Our method also performs better on average, indicating that it has a greater representation power of the BERT model.

We also apply the same metric to other structured pruning methods after removing half of the weights. SlimBERT dramatically outperforms layer-level pruning on all tasks except CoLA, and the average drop is also 2.5% lower. Instead of pruning the entire layer, we only remove unnecessary neurons in each layer. Therefore, we can reserve more essential neurons within each layer, as demonstrated by the empirical results. Compared to the other two SP methods, our approach also has a higher score on 4 tasks. Despite SNIP and SchuBERT having a better performance on SST-2 and MNLI respectively, our model has a very similar performance on these tasks. Among the four structured pruning methods, SlimBERT has the lowest loss on average, which means that our method performs better in recovering the results from the base model. It is also very flexible because it can easily compress the model to the desired size while always minimizing the loss in accuracy.

Finally, we list the results of the state-of-the-art unstructured pruning, soft MvP and LEAP, and perform a direct comparison in Table 4.7. We use the same way from these works to compute the compression ratio by excluding the number of parameters in embeddings layer. Unstructured methods generally have a lower compression rate because they focus on the smallest unit that can be pruned in a network. For MNLI and QQP,

Method	MNLI		
	C.R.	Matched Acc./Mismatched Acc.	Inference Speed Ratio
soft MvP	17˜21	**82.3/82.7**	1
LEAP-1	17˜21	81.0/81.0	1
SlimBERT	23	81.9/81.8	**1.28**
	QQP		
	C.R.	Accuracy/F1	Inference Speed Ratio
soft MvP	11˜15	-/86.8	1
LEAP-1	11˜15	**90.5/87.3**	1
SlimBERT	13	88.88/85.7	**1.38**

Table 4.7: Accuracy of SlimBERT and Unstructured Pruning Methods

unstructured methods can reduce the compression rate to 0.17˜0.21 and 0.11˜0.15 respectively. We observed that SlimBERT can also maintain model quality with fewer than 20% parameters, as demonstrated by compressing encoders into 23% and 13% of the original size. Regarding accuracy, MvP and LEAP achieved the best score on MNLI and QQP. Although SlimBERT has a slightly lower score, the gap ranges from 0.4% to 1.6%, which is still acceptable. Although unstructured methods truncate a large portion of the parameters, they cannot reduce the time consumption to make predictions. Unlike these two methods, SlimBERT is more competitive in inference speed. In particular, it is 28% faster on MNLI and 38% faster on QQP. It shows that a pruned model can achieve competitive accuracy and improve running efficiency at the same time.

4.5.2 Ablation Studies

In the last section, we used prune-after-tune only to determine which neuron to drop, but did not analyze the models produced by this method. We list their performance in Table 4.8.

We found that neurons in the embedding layer have much lower priority than other neurons and almost none of them is pruned. This is reasonable because embeddings are the input to the model and should be more important than hidden neurons. As a reference, we calculated the compression rate with and without the embedding layer. There is an gap of 20% between them due to the size of the token embeddings.

We observed that prune-after-tune can generate SlimBERT models with different levels

Methods	MRPC		QQP		QNLI	
	C.R.	acc./f1	C.R.	acc./f1	C.R.	acc.
BERTBase	100/100	85.8/89.7	100/100	90.9/87.8	100/100	91.7
Prune-after-tune	38.9/22.3	82.8/87.7	34.5/16.2	87.8/83.8	29.6/10.3	85.1
Prune MHSA	84.2/80.3	83.8/88.2	77.1/71.1	90.2/87.3	78.4/72.8	90.3
Prune FF	57.1/45.5	82.8/87.5	49.5/35.8	88.9/85.3	45.0/30.1	86.4
Methods	SST-2		RTE		CoLA	
	C.R.	acc.	C.R.	acc.	C.R.	acc./mcc.
BERTBase	100/100	93.1	100/100	69.1	100/100	83.9/60.5
Prune-after-tune	29.5/10.2	87.4	32.9/14.5	64.1	34.7/16.9	79.1/48.8
Prune MHSA	70.8/63.1	89.5	86.1/82.6	67.9	75.7/69.4	81.5/54.3
Prune FF	50.7/37.4	89.2	49.2/35.4	66.8	51.3/38.2	80.1/49.9

Table 4.8: Ablation studies of SlimBERT. MHSA stands for multi-head self-attention layers, and FF stands for feed-forward layers.

of compression ratios depending on the task. The model used the most parameters of 22.3% on MRPC and recovers around 97% of the original accuracy and f1. For QNLI and SST-2, the compression rate for the encoder units is only 10% and the accuracy drop is 6%. Shows that prune-after-tune removes more parameters if the task is relatively easier. The density ratio and performance loss for the other three tasks are approximately 16% and 5%. In all these tasks, SlimBERT suggests preserving at least 10% parameters to avoid significant degradation.

We also performed pruning on multi-head self-attention layers (MHSA) and feed-forward layers (FF) independently and investigated the contribution for each of them. Due to the size of each subnetwork, there is a clear pattern that pruning MHSA reserves more parameters than FF. It also illustrates that the accuracy of MHSA pruning is slightly lower than the original performance. On the contrary, the accuracy of FF pruning is similar to that of fully pruned models. It indicates that SlimBERT prunes the parameters in both the attention layer and the feed-forward layer rather than only focusing on one of them.

We also observed that a fully pruned model with the same sparsity can have a different distribution of pruned neurons in its subnetworks. More specifically, SlimBERT has a compression rate of around 10.2% on both QNLI and SST2, but it reserves 7% more parameters from FF and 9% less parameters from MHSA in SST-22. It shows that neural slimming can dynamically extract the optimal structure from the original model. It also indicates that the BERT model relies on different components to perform its downstream tasks.

Methods	MRPC	QQP	QNLI	SST-2	RTE	CoLA	MNLI
Prune-after-tune	**82.78**	87.7	89.2	90.5	**67.0**	51.8	81.2
Prune-then-tune	82.0	**88.6**	**90.2**	**92.1**	66.3	**54.0**	**82.5**

Table 4.9: Ablation study of Prune-after-tune and Prune-then-tune

Components	MRPC	QQP	QNLI	SST-2	RTE	CoLA	MNLI
Self-attention Layer	51%	61%	72%	79%	61%	69%	45%
Linear Layer	25%	39%	59%	43%	27%	41%	16%
Feed-forward Layer 1	89%	93%	95%	93%	92%	88%	82%
Feed-forward Layer 2	4%	7%	10%	26%	9%	27%	3%

Table 4.10: Percentage of Pruned Neurons in Each Component

Also, we were interested in how much the pruned model benefits from prune-then-tune. We evaluated them at the same sparsity and listed the results in Table 4.9. According to the result, prune-then-tune improves accuracy by more than 1% on 5 tasks, and there is a reduction of less than 0.8% for the rest. It shows that prune-then-tune has better performance in general and helps to increase the accuracy of pruned models. For the prune-after-tune approach, neurons with minimal weight are still involved in forward propagation and back propagation, but they are cut off during the testing. They introduce some additional noise into the model and damage the overall performance. The same issue does not occur in prune-then-tune because it eliminates the pruned neurons since the model is initialized. Consequently, fine-tuning a pruned model starting from the pre-trained weights gives a better performance.

4.5.3 Analysis of SlimBERT Architecture

In this section, we analyze the structures of the pruned models for each task. Figure 4.1 illustrates the heat map for density in each layer. For BERT, each layer is an encoder block that contains the attention layer and feed-forward layers. We observe that the shallow layers normally contain more important neurons than the deep layers. There a special pattern for RTE, the number of neurons is evenly distributed in each layer.

Furthermore, the last layer usually contains the least number of neurons, which is approximately 15%. For sentence pair tasks including natural language inference and sentence similarity, their last two layers are pruned heavily. There is also a clear pattern for RTE and CoLA. They reserve less than 10% neurons from the last three layers. It

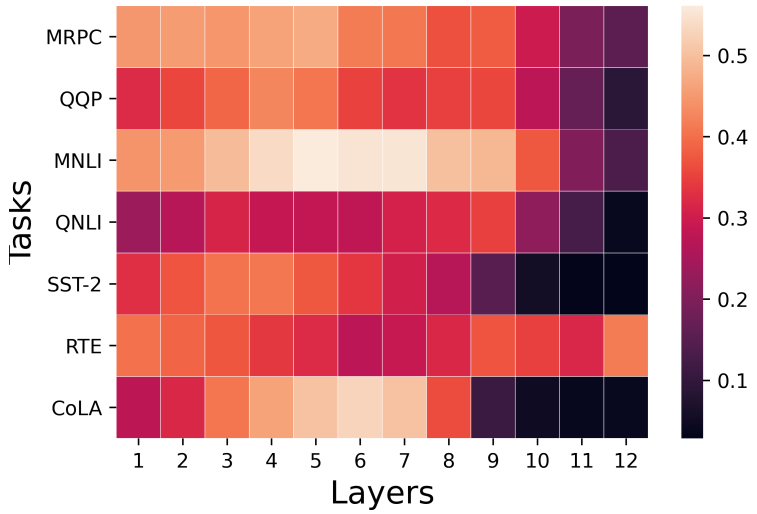


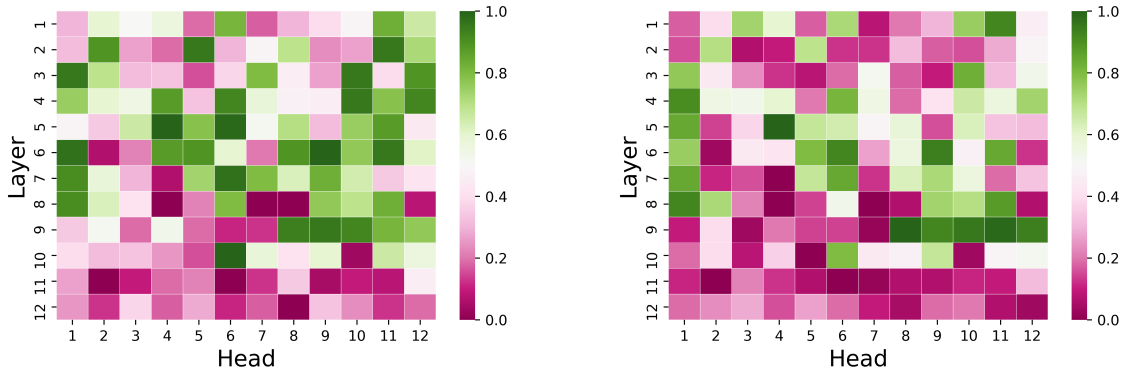
Figure 4.1: Percentage of Remaining Neurons in Layers

shows that not all layers are necessary and that the last layer can be safely removed for some of the tasks.

More specifically, we count density in each self-attention head and compute the percentages of them. The statistical result is shown in Figure 4.2. We observe a similar pattern of the usage on attention heads again for sentence pair classification of paraphrases or textual entailment. We define important heads of self-attention as those having more than 50 % neurons, and unimportant ones having less than 50% neurons. The distribution of them is almost identical in MRPC and QQP. The important ones are mainly located in the intermediate layers from layer 4 to layer 9.

In addition, MNLI has more important heads between layers 5 and 7. This is probably because MNLI is a more challenging task, and the model required more features from the heads. Similarly, QNLI is relatively easier, so it does not need as many attention layers features as the previous two tasks. For these four tasks, another observation is that the unimportant heads are centered in the last two layers, which follows the pattern we identified in the encoder units.

The pattern in RTE is different from other inference tasks. Like the distribution of its remaining neurons in each layer, the important heads are also evenly distributed over 12 attention layers. One significant difference is that the last attention layer has the most important heads. We surprisingly find that the last five attention heads in layer

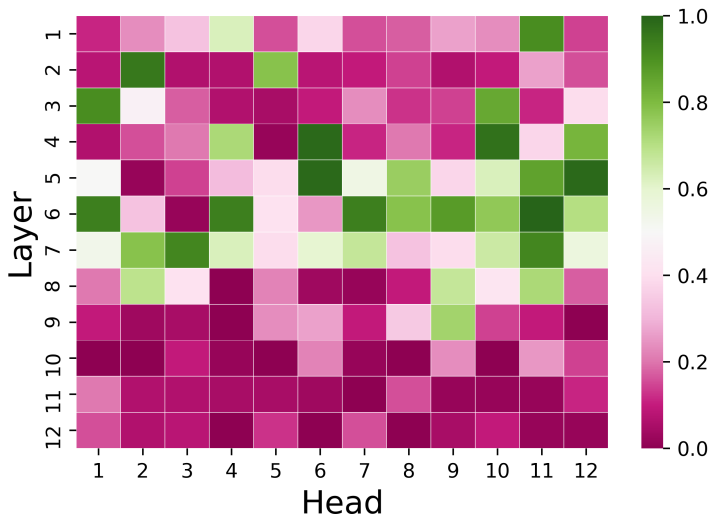


(a) MRPC (b) QQP

(c) MNLI (d) QNLI

(e) SST-2 (f) RTE

Figure 4.2: Percentage of Remaining Neurons in Attention Heads



(g) CoLA

Figure 4.2: Percentage of pruned Neurons in Attention Heads

9 consistently have a high importance on all tasks taking a pair of sentences. However, the same pattern does not appear in the single sentence classification. It indicates that these components can be used to extract information about the relationship between two sentences.

Unlike sentence pair tasks, single sentence classification tasks have different shapes of attention patterns. In particular, SST-2 contains only about 10 attention heads within the first seven layers. Similarly, the first seven layers for CoLA are more important, and only a small portion of the heads are pruned in layer 6 and 7.

Table 4.10 shows the percentage of pruned neurons in each component. Their names follow the description in figure 3.2. The sizes of the pruned attention layers range from 51% to 79%. It also depends on the difficulty and objective of the tasks. Feed forward layer 1 is the widest layer with 3072 neurons. It has very high redundancy and at least 82% of neurons are pruned in all scenarios. It shows that a large number of neurons are not really used to extract the features from the attention layer, and removing them can save a lot of space. In the second feed-forward layer, the number of pruned neurons falls into two groups, less than 10% for sentence pair classification or around 26% for single sentence classification. It shows that as the input sequence gets longer, BERT is representing the information of input with a higher-dimensional vector in the encoder output.

Chapter 5

Conclusion and Future Work

In this work, we propose a fine-grained compression method for the pre-trained language model BERT, namely neural slimming. It efficiently evaluates the importance of each individual neuron while tuning the model on the downstream task. We make this method adapt to all the subnetworks including token embedding layer, multi-head self-attention layers, and feed-forward layer. After pruning the redundant neurons, we can obtain a more compact BERT model with no or minor performance loss.

We evaluated our method on seven GLUE tasks involving sentence similarity, natural language inference, sentiment classification, and grammatical error identification. The empirical results show that the proposed approach outperforms other knowledge distillation methods and structured pruning methods after removing 40% to 50% of the parameters. The proposed method also minimizes the accuracy gap and the compression rate gap between structured pruning and unstructured pruning. At the same time, it reduces the size of the model and the run-time memory and improves the inference speed by more than 28%.

On each dataset, we analyze the architecture of the pruned model based on the learned importance scores. From the statistical data, we discover the following insights of the BERT model.

- According to our method, all neurons in the word embedding layer are important. Less than 5 of them are pruned each time. As the input to the BERT, pruning any of the them can cause a notable drop in accuracy.
- There are a huge number of unimportant neurons in the last encoder block and

feed-forward layer 1. More than 80% of them can be removed without significant degradation.

- For similar tasks, the model relies on the same group of self-attention heads. The importance heads for sentence pair tasks are located in the first 9 attention layers. The number of essential heads increases as the difficulty of the task increases. For single sentence classification, SST-2's importance heads are in the shallow layers, and the neurons in the middle layers for CoLA have higher importance.

With regard to future directions, more work can be done on how to compress the size of the embedding layer. In the original BERT, more than 22% of the parameters are from this subnetwork. Although our method has a good density ratio on the encoder blocks, it rarely prunes any neuron from the token embeddings. If there is a more compact way to represent the input tokens, the size of the model can decrease dramatically.

Since our method localizes the important neurons in BERT, another future research topic may be analyzing what kind of knowledge is encoded in these neurons. It improves the explainability of this model and helps others understand what information is encoded in the network.

Our work focused on applying the neural slimming technique to BERTBase. It can also be applied to the encoder part of the transformer, such as GPT. The major difference between them is GPT uses the masked attention heads to behave as an autoregressive model. Slim layer can be easily attached to this type of structure and perform compression as well. Our tuning strategy can also adapt to the generative tasks easily. A future research topic can utilize this method on the decoder part and compare its performance with the encoders.

References

[1] Bang An, Jie Lyu, Zhenyi Wang, Chunyuan Li, Changwei Hu, Fei Tan, Ruiyi Zhang, Yifan Hu, and Changyou Chen. Repulsive attention: Rethinking multi-head attention as Bayesian inference. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 236–255, Online, November 2020. Association for Computational Linguistics.

[2] Matthias Aßenmacher and Christian Heumann. On the comparability of pre-trained language models. *arXiv preprint arXiv:2001.00781*, 2020.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2015.

[4] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 5151–5159, Red Hook, NY, USA, 2018. Curran Associates Inc.

[5] Maximiliana Behnke and Kenneth Heafield. Losing heads in the lottery: Pruning transformer attention in neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2664–2674, Online, November 2020. Association for Computational Linguistics.

[6] Yoonho Boo and Wonyong Sung. Fixed-point optimization of transformer neural network. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1753–1757, 2020.

[7] Rich Caruana, Steve Lawrence, and C. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

[8] Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. LEGAL-BERT: The muppets straight out of law school. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2898–2904, Online, November 2020. Association for Computational Linguistics.

[9] Arnav Chavan, Zhiqiang Shen, Zhuang Liu, Zechun Liu, Kwang-Ting Cheng, and Eric P Xing. Vision transformer slimming: Multi-dimension searching in continuous optimization space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4931–4941, 2022.

[10] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21, 01 2020.

[11] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics.

[12] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[13] Jungwook Choi, Swagath Venkataramani, Vijayalakshmi Srinivasan, K. Gopalakrishnan, Zhuo Wang, and Pierce I-Jen Chuang. Accurate and efficient 2-bit quantized neural networks. In *MLSys*, 2019.

[14] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does BERT look at? an analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Florence, Italy, August 2019. Association for Computational Linguistics.

[15] Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In Joaquin Quiñonero-Candela, Ido Dagan, Bernardo Magnini, and Florence d'Alché Buc, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[16] Fahim Dalvi, Hassan Sajjad, Nadir Durrani, and Yonatan Belinkov. Analyzing redundancy in pretrained transformer models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4908–4926, Online, November 2020. Association for Computational Linguistics.

[17] Barry de Bruin, Zoran Zivkovic, and Henk Corporaal. Quantization of deep neural networks for accumulator-constrained processors. *Microprocess. Microsystems*, 72, 2020.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. cite arxiv:1810.04805Comment: 13 pages.

[19] Bin Ding, Huimin Qian, and Jun Zhou. Activation functions and their characteristics in deep neural networks. In *2018 Chinese Control And Decision Conference (CCDC)*, pages 1836–1841, 2018.

[20] William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[21] Jianli Feng and Shengnan Lu. Performance analysis of various activation functions in artificial neural networks. *Journal of Physics: Conference Series*, 1237:022030, 06 2019.

[22] Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, RTE '07, page 1–9, USA, 2007. Association for Computational Linguistics.

[23] Mitchell Gordon, Kevin Duh, and Nicholas Andrews. Compressing BERT: Studying the effects of weight pruning on transfer learning. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 143–155, Online, July 2020. Association for Computational Linguistics.

[24] Fu-Ming Guo, Sijia Liu, F. Mungall, Xue Lin, and Yanzhi Wang. Reweighted proximal pruning for large-scale language representation. *ArXiv*, abs/1909.12486, 2019.

[25] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. cite arxiv:1503.02531Comment: NIPS 2014 Deep Learning Workshop.

[26] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 04 1998.

[27] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[28] Katarzyna Janocha and Wojciech Czarnecki. On loss functions for deep neural networks in classification. *Schedae Informaticae*, 25, 02 2017.

[29] M I Jordan. Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986.

[30] Ashish Khetan and Zohar Karnin. schuBERT: Optimizing elements of BERT. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2807–2818, Online, July 2020. Association for Computational Linguistics.

[31] Sarit Khirirat, Hamid Reza Feyzmahdavian, and Mikael Johansson. Mini-batch gradient descent: Faster convergence under data sparsity. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2880–2887, 2017.

[32] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[33] Vinnie Ko, Stefan Oehmcke, and Fabian Gieseke. Magnitude and uncertainty pruning criterion for neural networks. pages 2317–2326, 12 2019.

[34] Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. Revealing the dark secrets of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4365–4374, Hong Kong, China, November 2019. Association for Computational Linguistics.

[35] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.

[36] Bingbing Li, Zhenglun Kong, Tianyun Zhang, Ji Li, Zhengang Li, Hang Liu, and Caiwen Ding. Efficient transformer-based large scale language representations using hardware-friendly block structured pruning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, Online, November 2020. Association for Computational Linguistics.

[37] Zhenge Li, Soroush Ghodrati, Amir Yazdanbakhsh, Hadi Esmaeilzadeh, and Mingu Kang. Accelerating attention through gradient-based learned runtime pruning, 04 2022.

[38] Zi Lin, Jeremiah Liu, Zi Yang, Nan Hua, and Dan Roth. Pruning redundant mappings in transformer models via spectral-normalized identity prior. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 719–730, Online, November 2020. Association for Computational Linguistics.

[39] Liyuan Liu, Jialu Liu, and Jiawei Han. Multi-head or single-head? an empirical comparison for transformer training. *arXiv preprint arXiv:2106.09650*, 2021.

[40] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[41] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.

[42] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[43] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. cite arxiv:1301.3781.

[44] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[45] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[46] Subhabrata Mukherjee and Ahmed Hassan Awadallah. XtremeDistil: Multi-stage distillation for massive multilingual models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2221–2234, Online, July 2020. Association for Computational Linguistics.

[47] Departement Operationnelle, Y. Bengio, R Ducharme, Pascal Vincent, and Centre Mathematiques. A neural probabilistic language model. 10 2001.

[48] Michela Paganini and Jessica Zosa Forde. On iterative neural network pruning, reinitialization, and the similarity of masks. *ArXiv*, abs/2001.05050, 2020.

[49] Razvan Pascanu, Tomas Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *30th International Conference on Machine Learning, ICML 2013*, 11 2012.

[50] David Peer, Sebastian Stabinger, Stefan Engl, and Antonio Rodríguez-Sánchez. Greedy-layer pruning: Speeding up transformer models for natural language processing. *Pattern Recognition Letters*, 2022.

[51] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[52] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[53] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[54] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.

[55] Ajna Ram and Constantino Reyes-Aldasoro. The relationship between fully connected layers and number of classes for the analysis of retinal images, 04 2020.

[56] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018. cite arxiv:1804.02767Comment: Tech Report.

[57] Antônio H. Ribeiro, Koen Tiels, Luis A. Aguirre, and Thomas B. Schön. Beyond exploding and vanishing gradients: analysingrnn training using attractors and smoothness. In *Proceedings of the 23rd International Conferenceon Artificial Intelligence and Statistics (AISTATS).*, volume 108. PMLR, 2020.

[58] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv*, 12 2014.

[59] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[60] Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, abs/1609.04747, 2016.

[61] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

[62] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

[63] Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20378–20389. Curran Associates, Inc., 2020.

[64] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[65] Mike Schuster and Kuldip Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45:2673 – 2681, 12 1997.

[66] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *CoRR*, abs/1907.10597, 2019.

[67] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *ArXiv*, abs/1611.01603, 2017.

[68] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical*

Methods in Natural Language Processing, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[69] Alessandro Sordoni, Philip Bachman, and Yoshua Bengio. Iterative alternating neural attention for machine reading. *ArXiv*, abs/1606.02245, 2016.

[70] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for modern deep learning research. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(09):13693–13696, Apr. 2020.

[71] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In Maosong Sun, Xuanjing Huang, Heng Ji, Zhiyuan Liu, and Yang Liu, editors, *Chinese Computational Linguistics*, pages 194–206, Cham, 2019. Springer International Publishing.

[72] Qigong Sun, Xiufang Li, Yan Ren, Zhongjian Huang, Xu Liu, Licheng Jiao, and Fang Liu. One model for all quantization: A quantized network supporting hot-swap bit-width adjustment. *arXiv preprint arXiv:2105.01353*, 2021.

[73] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*, 2019.

[74] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170, Online, July 2020. Association for Computational Linguistics.

[75] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[76] Tomasz Szandała. *Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks*, pages 203–224. Springer Singapore, Singapore, 2021.

[77] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*, 2019.

[78] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[79] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[80] Francesco Ventura, Salvatore Greco, Daniele Apiletti, and Tania Cerquitelli. Explaining the deep natural language processing by mining textual interpretable features. *CoRR*, abs/2106.06697, 2021.

[81] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy, July 2019. Association for Computational Linguistics.

[82] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[83] Benyou Wang, Lifeng Shang, Christina Lioma, Xin Jiang, Hao Yang, Qun Liu, and Jakob Grue Simonsen. On position embeddings in {bert}. In *International Conference on Learning Representations*, 2021.

[84] Yingying Wang, Yibin Li, Yong Song, and Xuewen Rong. The influence of the activation function in a convolution neural network model of facial expression recognition. *Applied Sciences*, 10(5), 2020.

[85] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2018.

[86] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[87] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan

Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[88] Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. BERT-of-theseus: Compressing BERT by progressive module replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7859–7869, Online, November 2020. Association for Computational Linguistics.

[89] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[90] Zhewei Yao, Linjian Ma, Sheng Shen, Kurt Keutzer, and Michael W Mahoney. Ml-pruning: A multilevel structured pruning framework for transformer-based models. *arXiv preprint arXiv:2105.14636*, 2021.

[91] Feng Zhang, Xiatian Zhu, and Mao Ye. Fast human pose estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[92] J. Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arXiv: Optimization and Control*, 2020.