Distilling knowledge through student-teacher model and BERT for sentiment analysis

by

Ximing Dong

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Master of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
August 2022

© Copyright 2022 by Ximing Dong

Thesis advisor
Dr. Parimala Thulasiraman

Author
Ximing Dong

Distilling knowledge through student-teacher model and BERT for sentiment analysis

Abstract

Bi-directional Encoder Representations from Transformers (BERT) is the state-of-the-art deep learning model for pre-training natural language processing (NLP) tasks such as sentiment analysis. The BERT model dynamically generates word representations according to the context and semantics using its bi-directional and attention mechanism features. The model, although, improves precision on NLP tasks, it is compute intensive and time consuming to deploy on mobile or smaller platforms. In this thesis, to address this issue, we use knowledge distillation (KD), "teacher-student" training technique, to compress the model.

We use the BERT model as the "teacher" model to transfer knowledge to student models, "first-generation" convolution neural networks and long-short term memory with attention mechanism (LSTM-atten). We conduct various experiments on sentiment analysis benchmark data-sets and show that the "student models" through knowledge distillation have better performance with 70% improvement in accuracy, precision, recall and F1-score compared to models without KD. We also investigate the convergence rate of student models, and compare the results to the existing models in the literature. Finally, we show that compared to the full-size BERT model, our RNN series models are 50 times smaller in size and retain approximately 96% performance on benchmark data-sets.

Contents

Abstract . . . ii
Table of Contents . . . iv
List of Figures . . . v
List of Tables . . . vii
Acknowledgments . . . viii
Dedication . . . ix

1 Introduction **1**
1.1 Sentiment Analysis (SA) . . . 4

2 Background **9**
2.1 Artificial Neural Network . . . 9
2.1.1 Recurrent neural network . . . 12
2.1.2 Long Short-Term memory . . . 13
2.2 Machine learning methods for text classification . . . 16
2.3 Word Embedding . . . 22
2.3.1 Static Word Embedding . . . 24
GloVe . . . 27
2.3.2 Dynamic Word Embedding . . . 28
ELMo . . . 29
BERT . . . 31
2.4 Transformer Model . . . 33
2.5 Student-Teacher Model . . . 42
2.5.1 Knowledge Distillation . . . 43

3 Literature Review on Sentiment Analysis **45**
3.1 Dictionary for "student" models . . . 52
3.2 Transfer learning in sentiment analysis . . . 53

4 Distilling knowledge through student-teacher model and BERT **58**
4.1 Overview . . . 58
4.2 Student Models . . . 59
4.2.1 Bi-LSTM . . . 60
4.2.2 Bi-Long Short-Term Memory with Attention . . . 63
4.2.3 Convolution Neural Network (CNN) . . . 65

4.3 Teacher Models . . . 67
4.3.1 Fine-Tuning Classification Tasks: . . . 74
BERT with linear model . . . 75
BERT with GRU . . . 75
4.4 Knowledge Distillation . . . 76

5 Experiments and Results **78**
5.1 Experimental Setup . . . 78
5.1.1 "Teacher model" . . . 79
5.1.2 "Student model" . . . 81
5.2 Evaluation: Results and Discussion . . . 83
5.2.1 Evaluation measures on dataset . . . 85
The process of knowledge distillation . . . 86
Evaluation . . . 88
5.2.2 Execution time . . . 93

6 Conclusion and Future Work **95**
6.1 Future Work . . . 96
6.1.1 Improvements . . . 97
6.1.2 Exploration of other models . . . 97

List of Figures

2.1 ANN model . . . 11
2.2 An overall of RNN series model . . . 12
2.3 An overall of multi-layer LSTM/GRU model . . . 13
2.4 An LSTM architecture . . . 16
2.5 BERT model . . . 31
2.6 An overall of Transformer model . . . 35
2.7 Position encoding in Transformer . . . 37
2.8 Attention score calculation in Transformer . . . 39
2.9 attention score calculation in transformer . . . 40
2.10 residual network . . . 40

4.1 Knowledge Distillation architecture . . . 59
4.2 Bi-LSTM architecture . . . 61
4.3 Bi-LSTM with attention architecture . . . 64
4.4 CNN architecture . . . 67
4.5 BERT embedding architecture . . . 68
4.6 BERT MLM architecture . . . 71
4.7 BERT NLP architecture . . . 72
4.8 BERT fine-tuning architecture . . . 73
4.9 BERT Bi-GRU architecture . . . 74
4.10 Knowledge Distillation architecture . . . 77

5.1 Model parameters for IMDB . . . 82
5.2 Model parameters for Twitter . . . 83
5.3 Best epoch in "teacher model" for validation set . . . 86
5.4 Best epoch in "student model" for validation set(IMDB) . . . 87
5.5 Best epoch in "student model" for validation set(Twitter) . . . 87
5.6 Evaluation matrix for IMDB . . . 88
5.7 Evaluation matrix for Twitter . . . 88
5.8 Comparison on IMDB dataset . . . 90
5.9 retain rate for IMDB dataset . . . 91
5.10 retain rate for Twitter dataset . . . 91
5.11 Different filters for IMDB dataset . . . 92
5.12 Different filters for Twitter dataset . . . 93

5.13 Execution time in IMDB 94
5.14 Execution time in Twitter 94

List of Tables

Acknowledgments

This thesis is dedicated to science and the thirst for knowledge.

Chapter 1

Introduction

Data processing using deep learning technology can be divided into three main categories: text, speech, and image. Among them, owing to huge data sources and wide applications, text processing is one of the important areas. Natural language processing (NLP) is an area of research that studies how computers can learn, and manipulate text similar to a human brain. There are few research tasks in NLP, such as Question and Answering (QA), Name Entity Recognition (NER), Machine Translation, Part-of-speech (POS) Tagging, Sentiment Analysis, etc.

In recent years, with the technological advancements in artificial intelligence and computer hardware, there has been significant development in all areas of natural language processing, in particular sentiment classification. In general, there are two main categories of classification

models used to solve problems in text classification. The first category consists of machine learning methods, such as Naive Bayes algorithm, k-nearest neighbor, support vector machine and decision tree algorithm.

The second category consists of deep learning methods based on neural networks such as convolution neural network (CNN) and recurrent neural network (RNN). However, the performance of text classification and other areas in NLP, using CNN or sequence neural network architectures (RNN or neural long-short term memory (LSTM)) models are computationally slower in comparison to deep learning algorithms used in computer vision. There are couple of reasons for this poor performance. First, the uniformity of the images. In general, image samples can be enlarged or reduced to a uniform size. However, in NLP, each text has a different sequence and data processing these images is a challenge. Second, deep learning algorithms require large labelled data-sets for training. The amount of labelled data-sets in NLP is not sufficient to train such networks. Using small data-sets to train deep learning networks would cause overfitting. Labeling data is time-consuming and very expensive. Third, transfer learning. In this technique, deep learning models are pre-trained using large data-sets so that the models can be used for other tasks. This is more efficient than deep learning mod-

els which learn from scratch [23] for each task. Unfortunately, transfer learning in NLP did not exist until the ELMO model introduced by AllenNLP [38] few years ago .

Attention is another mechanism applied in neural network architectures to gain increased performance in NLP tasks. The Transformer model with attention mechanism [48] has been successfully studied for applications such as machine translation and aspect based sentiment analysis. In 2018, Google [46] introduced the Bidirectional Encoder Representation from Transformer (BERT) based on Transformers. Since then there has been a variety of models based on BERT such as BART [28], ALBERT [26], RoBERTa [31] and so on. BERT is a very important technique on which lot of models are based on.

In general, NLP tasks are based on text sequences. Transformer or BERT models have some advantages in solving NLP tasks: (i) *Parallelization*: Unlike deep learning neural network architectures (such as RNN and LSTM models that take one token at a time as input, the Transformer or BERT models take the entire sequence as input, speeding up the computations through parallelization; (ii) *Unlabelled Data*: These models can be pre-trained using unsupervised learning which can in turn be trained by unlabelled data. The pre-trained models can be

used for training other NLP tasks. Large amounts of unlabelled data can be used to train better models to avoid overfitting. Also, it is easier to get unlabeled data for experimentation.

1.1 Sentiment Analysis (SA)

Sentiment analysis, also known as opinion mining refers to the computational study of how subjective information are expressed in textual units. SA aims at evaluating and classifying the intensity of the texts. Therefore, it can be categorized as a classification task. Sentiment analysis can be divided into three aspects [30]: document level, statement level and aspect level. The intensity of sentiments in the documents can be identified as either positive, negative or neutral polarities. For example, the sentence " The food was delicious but the service was awful." involves two contradicting sentiments (i.e. positive and negative) on two different aspects (i.e. food and service), respectively. Another application to sentiment analysis is in spam filtering systems that can automatically identify spam. The data available for these applications is extremely large. It is impossible to analyze a large number of text by reading manually. We need an intelligent system that can do the data analysis.

Sentiment analysis can be solved by supervised learning[53] techniques. Initially, techniques such as support vector machines [17], logistic regression [16], Naive Bayes, decision tree were used to solve sentiment analysis. This was followed by "first generation " models, such as CNN [21] and LSTM[15]. The last few years, the Transformer models have taken over the NLP research, such as [11] BERT and ALBERT[26].

Challenges in sentiment analysis: One of the challenges in sentiment analysis is determining how to express each word (or *token*) with numerals for computers to read the data. Traditional machine learning techniques use bag of word [50], term frequency, term frequency-inverse document frequency (TF-IDF) [40] as mutually selected information to determine the importance of a word or token as features. Then, an appropriate classifier is selected that uses selected features as input to perform the classification. This is not adequate as the context information such as relationship between words is ignored and mutually extracting information is not necessarily very accurate. Also, the sequence information of words is ignored. In deep learning models, the word2vec model [34] is a very useful word's representation. This model has two main ways to train word embedding: (1) using the context of current vocabulary to predict the current vocabulary; (2) using the cur-

rent vocabulary to predict the context of the current vocabulary. The initialization of word embedding has been studied by various researchers using word2vec model [10], [24] , [43] , [52]. However, word2vec is a static model. It can not solve issues such as polysemy. For instance, the word “apple” has several meanings. It can be referred to as “fruit” or “apple company”. If we say “ apple is a healthy fruit ”, that means “apple” is a fruit, if we say “ apple company will release a new IPhone ”, that means “apple” is a company. But, “apple” in both context represented in word2vec will be the same. This is an incorrect representation for the word “apple” in different contexts. Unfortunately, word2vec will be unable to provide a context word expression. Even if a word or token can have several representations, we cannot guarantee the emergence of new meanings.

Recently, with transfer learning [54], word representation problems can be solved using Transformer models such as BERT. BERT is the state-of-the-art model which has the best performance on NLP tasks such as question answering, sentiment analysis, reasoning, natural language inference and so on. BERT model can dynamically generate word’s representation according to the context and semantics. This is because the model is bi-directional and uses attention mechanism.

During the training phase, BERT learns from the right and left of a token's context. The BERT model has been pre-trained on a large unlabelled dataset. This allows the model to provide a good ability to solve downstream tasks. This pre-trained model can be fine-tuned to create models for other NLP tasks. That is, load the pre-trained model and then add a new layer to train new tasks.

Although the BERT model can improve a lot on precision, it can not be deployed on mobile or small platforms. This is because, BERT is a large neural network architecture with millions of parameters. Since BERT's base model has 12 layer attention mechanisms, the time complexity of each layer is approximately $\mathcal{O}(d \times n^2)$, where d is dimension of each word embedding and n is the length of the sequence of the sentence. Therefore, the pre-trained BERT model is too large and complicated compared to "first generation" networks such as LSTM and CNN [42].

In this research, to address the highly large scale BERT model, we use Knowledge distillation (KD) technique [14] to compress the model. KD is a training method based on "teacher-student" network. That is, knowledge is transferred from a large teacher model to a small student model. It is a technique that has been used in other NLP tasks [45]. The large teacher model is a fine-tuned model that produces good results

and will influence and teach the student model. The student model is a small model that learns from the teacher model and mimics the teacher model. In this research, we compress the network with knowledge distillation technique [1] [14]. Specifically we used the fine-tuned BERT model which has extremely high performance as the teacher model. Then, we chose long-short term memory with attention mechanism called *LSTM-atten* and convolution neural network as the student models, respectively.

Two kinds of loss functions can be used to transfer knowledge: mean-square-error and cross-entropy between "teacher"model and "student" models. In this thesis, we use both these loss functions. Experiments are conducted on Internet Movie Database (IMDB) and Twitter sentiment analysis data-sets.

The rest of the thesis is organized as follows. Chapter 3 discusses the literature review on sentiment analysis. Our proposed compression approach is discussed in Chapter 4. Experiments and results are discussed in Chapter 5 . Chapter 6 concludes this thesis with some future work.

Chapter 2

Background

This chapter provides some background material in artificial networks followed by sentiment analysis models.

2.1 Artificial Neural Network

Artificial Neural networks (ANN) are as series of algorithm that mimic the human brain and find the relationship between the sets of data without human designed features extraction. ANN is constructed by neurons layers, in which the number of neurons construct a layer. layers can connect each other directly by weights and(or) bias. In usual, there are three kinds of layers: input layer, hidden layer, and output layer. The input data will firstly go through input layer and transmit the data that the hidden layer. The hidden layer is core part of ANN

model. The number of hidden layers is not restricted and the depth of hidden layers can be set by user. In general, deep model(large number of layers) can have a better performance than shallow model, since deep model can capture more relationships between data. However, Excessively deep layer will cause overfitting problem, since deep model will learn some features from "noise" so that they can have a better performance in training data-sets but low performance in testing data-sets. The last layer is the output layer.

Figure 2.1 is an example of ANN with one hidden layer. There are 4 neurons in the input layer, 5 neurons in the hidden layer, and 4 neurons in the output layer. In general, hidden layers and output layer always have active function to introduce non-linearity into the network since the complexity of data is not limited to linear function. different activation functions have different functionality. For example, sigmoid function is one of the most widely used activation function. This function can take any input and convert the input into a non-linear distribution. Weight matrix assigns a weight to each input and add a bias to the final output. The final result computed by ANN and true result are compared by a loss function. the loss function such as mean square error (MSE) or cross-entropy will result in a value that

evaluate the accuracy of ANN. the weight matrix and bias vector are updated in every iteration by the back propagation algorithm based on loss function.

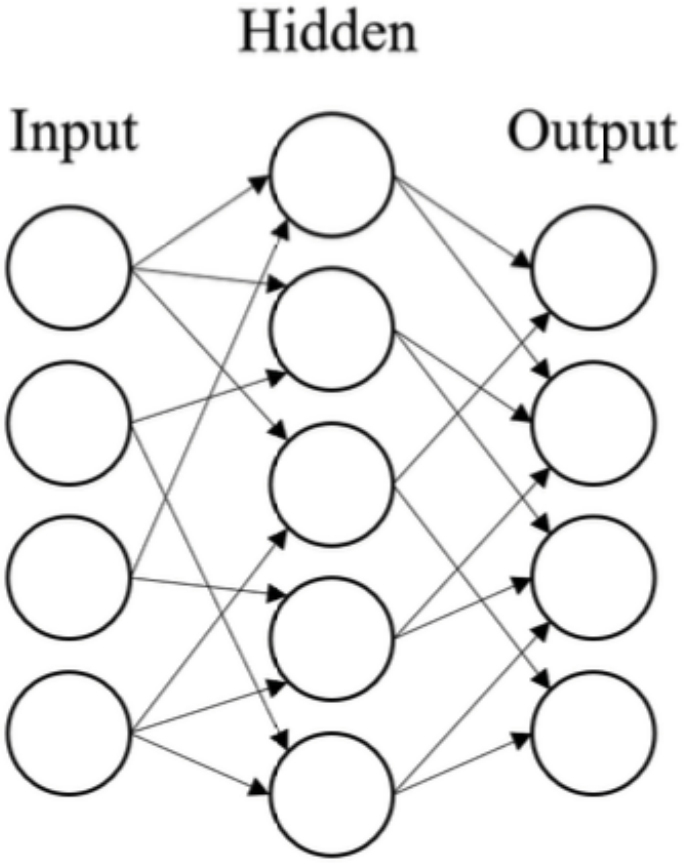


Figure 2.1: ANN model

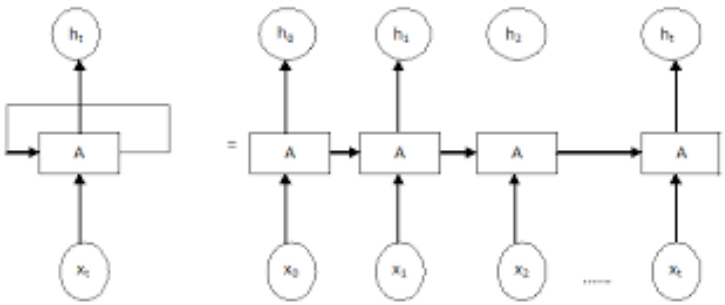


Figure 2.2: An overall of RNN series model

2.1.1 Recurrent neural network

Recurrent neural network (RNN) is one kind of ANN where hidden layers have feedback connections to themselves. That is, their current outputs are fed as next step inputs. The feedback connection of the neuron to itself acts as a kind of memory element. This element takes into account the present input, the history of decisions previously taken, and present state. RNN is therefore suitable for times-series data 2.2. Sentiment analysis is based on text which is in order. Therefore, we can use times-sequence model to “read” it.

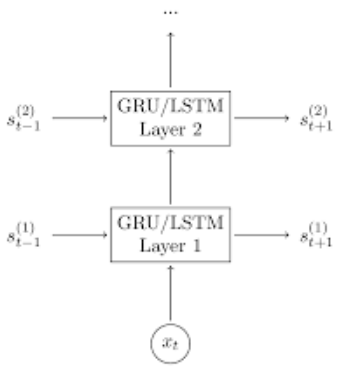


Figure 2.3: An overall of multi-layer LSTM/GRU model

2.1.2 Long Short-Term memory

Long Short-Term memory (LSTM) is one type of RNN model. RNN fails to remember long term dependencies. To overcome this drawback, LSTM was introduced [15]. LSTM can store long-term useful information and selectively forget some useless information. Figure 2.3shows a typical LSTM unit with feedback loop. It may consist of many layers. That is, the hidden output of first layer will be the second LSTM layer input. Figure 2.4 shows the structure of LSTM that mainly consists of four parts: cell, input gate layer, output gate layer and forget gate layer. The Input gate layer process the current input. The Forget gate layer forgets the unnecessary data. Output gate computes the final output. The memory cell layer stores the output from the previous iteration. Input gate, forget gate, and output gate apply sigmoid function which

transfers the input into sigmoid curve with range [0, 1]. It is defined in equation2.1.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.1}$$

We use notations as follow:

1) Input data: X = $(x_{1,2}, x_3, x_4 ... x_n)$

2) State of memory cell: C = $(c_1, c_2, c_3, c_4 ... c_n)$

3) Hidden Output: H = $(h_1, h_2, h_3, h_4 ... h_n)$

The forget gate decides what information will be forgotten. The current input x_t and the previous hidden output h_{t-1} are forget gate's input. The forget gate computes the sigmoid function with these two input which are concatenated firstly multiplies with a weight matrix W_f plus the bias vector b_f. The output of forget gate f_t (Equation 2.2) is a value in the range [0,1] in each dimension, where a date value of 0 represents withdraw this data and a data value of 1 represents completely keep this data.

$$f_t = \sigma(W_f \cdot [h_{t-1}; x_t] + b_f) \tag{2.2}$$

As show in equation 2.3 similar to forget gate, input gate also takes x_t and h_{t-1} as input. However, the purpose of this is opposite to For-

get gate. This gate decides what to store in memory for the current iteration.

Another copy of the input and hidden will be the input of modulation gate, which is just a modification of tanh 2.4 activation function. It changes in the output range from [-1, 1]. This is denoted as g. The output of the input gate and input modulation gate are added together to compute c_t, the data needed to be store in memory cell 2.5.

$$i_t = \sigma(W_i \cdot [h_{t-1}; x_t] + b_i) \tag{2.3}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.4}$$

$$c_t = f_t * c_{t-1} + i_t * g(W_c \cdot [h_{t-1}; x_t] + b_c) \tag{2.5}$$

The last step is to decide what to output. Similar to input gate and forget gate, the output gate o_t 2.6 will compute output's weight. g is tanh function. It takes the current memory c_t as input. hidden output h_t 2.7 will be as hidden output of current state.

$$o_t = \sigma(W_o \cdot [h_{t-1}; x_t] + b_o) \tag{2.6}$$

$$h_t = o_t * g(c_t) \tag{2.7}$$

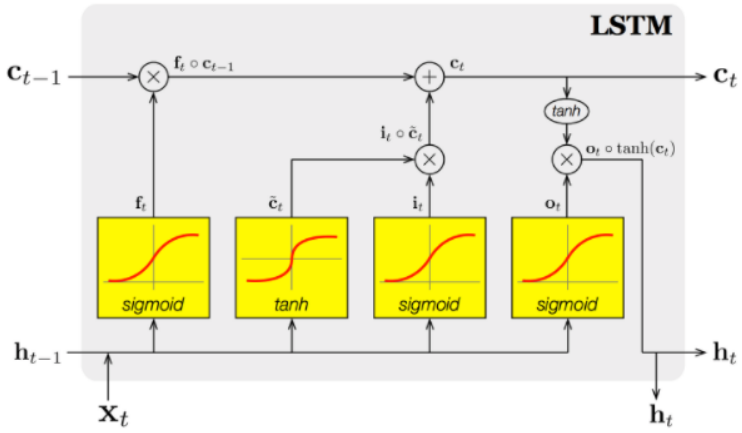


Figure 2.4: An LSTM architecture

We extend the LSTM architecture in our proposed research in Chapter 4.

2.2 Machine learning methods for text classification

Research in NLP has been ongoing for a very long time. Sentiment analysis (SA) also called opinion mining, was first put forward by Pang et al. [36]. The original method was proposed using one-hot encoding. In this encoding scheme, every word (or token) is represented as a vector, with a 0 or 1. The vector length is the number of words in

a sentence. Each word in a sentence is, therefore, made up of vectors, comprising of an array of vectors or a matrix. The weakness in this scheme is that the sparsity of the vectors for large data increases the error rate leading to the curse of dimensionality [47]. Latent Semantic Analysis (LSA) [9] is another technique that has been used to get potential semantic information based on Vector Space Model (VSM) [41]. However, LSA is time consuming and it has to be re-trained for every new input.

In general, text classifiers are represented as a bag of words. That is, a bag is like a set that allows repeating words. This is a simple representation that ignores the word order and keeps a record of which words are included in the document and how many times each word may occur.

There are well-known classifiers such as Naive Bayes and support vector machine to solve sentiment analysis.

Naive Bayes: Naive Bayes classifier is one of the most popular machine learning methods for solving sentiment analysis (or text classification). This method is based on Bayes' theorem and "Naive" is an assumption that the features (x) on a given class (A) dataset are mutually independent. That is, the classifier assumes that the effect of the value of a

feature x on a given class (A) is independent of the values of other features. The general notation of the posterior probability can be written as

$$P(\mathbf{A}|\mathbf{x}) = P(\mathbf{A})\frac{P(\mathbf{x}|\mathbf{A})}{P(\mathbf{x})} \tag{2.8}$$

where $P(\mathbf{A}|\mathbf{x})$ is the posterior probability of class (target) given feature (attribute). $P(\mathbf{A})$ is the prior probability of class; $P(\mathbf{x}|\mathbf{A})$ is the likelihood which is the probability of feature given class. $P(\mathbf{x})$ is the prior probability of feature.

Since, we use a naive assumption that each of the features are independent, the conditional probability, $P(\mathbf{X}|\mathbf{A})$ can be represented as:

$$P(\mathbf{X}|\mathbf{A}) = P(\mathbf{x_1}|\mathbf{A})P(\mathbf{x_2}|\mathbf{A})...P(\mathbf{x_n}|\mathbf{A}) \tag{2.9}$$

$P(\mathbf{X})$ which is called the evidence, means that the probability of encountering a particular pattern $\mathbf{X}$ is independent from the class label. Since it is constant for every class and it is a scaling factor, it can be ignored. Posterior probability can be estimated or calculated by conditional probability $\times$ prior probability, that is

$$P(\mathbf{A}|\mathbf{X}) = P(\mathbf{A})P(\mathbf{X}|\mathbf{A}) = P(\mathbf{A})P(\mathbf{x_1}|\mathbf{A})P(\mathbf{x_2}|\mathbf{A})...P(\mathbf{x_n}|\mathbf{A}) \tag{2.10}$$

One problem is that if some features are not in the dataset, then the class-conditional probability will be 0. That is, it will result in the posterior probability equal to zero. We can add "additive smoothing" to avoid the problem of zero probabilities.

There are two main Naive Bayes models: multinomial model and Bernoulli model. Multinomial model is word-level model and Bernoulli model is document-level model. The multinomial model classifies the entire document and counts how often words occur in the document. The Bernoulli model classifies the document with respect to whether the word occurs (1) or not (0). These models are useful for binary features.

We will explain these probabilistic models on a bag of words using Naive Bayes assumption.

For example, consider V the set of vocabularies.

$$V = \{dog, cat, homework, cookie, apple, red\} \tag{2.11}$$

Then $|V|$ contains 6 word types (feature x_i). The feature vector dimension is D $= |V| = 6$. Consider this sentence in a document: "the red cat and red dog ate my homework and cookie." Then, the Bernoulli feature vector for this document is $[1, 1, 1, 1, 0, 1]^T$. The features (word

type) that occur in the document is given 1, otherwise 0. The multinomial feature vector for this document is $[1, 1, 1, 1, 0, 2]^T$. We count the number of occurrences of the word. In this case, "red" appears twice in the sentence. The Bernoulli model ignores the number of occurrences of the features.

The Bernoulli model, therefore estimates $P(\mathbf{x_i}|\mathbf{A})$ as the fraction of documents of class A that contain x_i. Multinomial estimates $P(\mathbf{x_i}|\mathbf{A})$ as the fraction of words (tokens) or fraction of positions in documents of class A that contain the word x_i.

We also use term frequency–inverse document frequency (TF-IDF) to evaluate features' weight. Since some words are not that important such as stop words, we need to filter those words and assign them less weights. Term frequency indicates the number of times a word appears in a document

$$\mathbf{TF(t)} = \frac{\textit{term frequency in a document}}{\textit{total words in document}} \tag{2.12}$$

Inverse Document Frequency (IDF) indicates the total number of documents in corpus over number of documents with term (word).

$$\mathbf{IDF(t)} = \log_2 \frac{\textit{total documents in corpus}}{\textit{documents with term+1}} \tag{2.13}$$

The addition of 1 in the denominator is additive smoothing (or Laplace smoothing) to eliminate zero.

TF-IDF is

$$\textbf{TF-IDF(t)} = TF \times IDF \tag{2.14}$$

We can use TF-IDF as "count" in word representation. However, after feature extraction, TF-IDF only applies to multinomial Naive Bayes model as the bag of word and TF-IDF will be same in Bernoulli model.

The disadvantage of Naive Bayes model is that these models do not consider the relationship between words and the order of words, since Naive Bayes assumes that all predictors (or features) are independent, which rarely happening in real life scenarios. For example, "That is good news. " and "Is that good news?" will have same result if we don't consider punctuation. This is obviously not correct.

Support Vector Machine: Traditional machine learning methods have high accuracy and are widely used by industry. One of the best known methods is support vector machine (SVM) [22]. One of the advantages of SVM is that it can handle high dimensional input spaces. We could avoid high dimensional spaces by keeping only the relevant features, during feature selection. However, there are usually few irrelevant fea-

tures. SVM performs well in high dimension spaces. This is because, an SVM constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional input space. This is used to find support vectors. In general, a good separation is achieved by the line that maximizes the minimum margin. The maximum-margin separator is determined by a subset of the training data points or support vectors. The SVM decision function depends on the support vectors.

SVMs are well suited for dense and sparse document input vectors. SVMs do not directly provide probability estimates, but are calculated using an expensive five-fold cross-validation. In scikit-learn, there are several classes such as SVC, NuSVC and LinearSVC that can be used for text classification on a dataset.

Although SVM is a powerful tool for solving classification problems, they are compute and memory intensive. The memory requirements increases with increase in number of training vectors.

2.3 Word Embedding

One hot encoding and word embedding are two popular techniques in NLP. These techniques provide mechanisms on how to use a vector to represent a text.

One-hot encoding is a popular technique. It is very easy to implement. It creates a binary vector in the size of the total number of unique words. The disadvantages is when the number of words are too large. It becomes too complex and time-consuming to compute. Also, this technique can not capture the relationship between words. Since only unique words are listed in the vector the semantics of the different meanings of a word (eg: "apple" has two different meaning-fruit or computer) is not captured. That is polysemy is not captured.

Word and vector have a one-to-one relationship. Bag of words is one word embedding technique. Others are GloVe[37] and word2vec[34]. In general, the training data is used to build a dictionary.

There are two types of word embedding: *static* and *dynamic*. GloVe and word2vec are static word embedding techniques. BERT and ELMO are dynamic embedding techniques. The difference between BERT and ELMO is that BERT uses self-attention to generate word embedding, while, ELMO uses Long Short-term memory to generate word embedding. *In order to handle polysemy, our model will generate word embedding dynamically.*

2.3.1 Static Word Embedding

There are two kinds of algorithms to build word embedding: word2vec [34] and GloVe [37]. The general idea is to map each word to a word vector. The dimension of the word vector can be specified by the user according to the task requirements during training. Using the word vectors, we can use statistical methods to study the relationship between the words.

word2vec: There are two kinds of methods in word2vec: Continuous Bag-of-Words (CBOW) and skip-gram. The idea of CBOW is trying to predict the *target* word in the context of the surrounding word. The idea of skip-gram is to predict the surrounding context word from a target word (opposite of CBOW).

word2vec works as follows. Consider a sentence"Machine Learning is an important tool." We first create training data. The tool tries to find which words are close to far from certain words. That is, finds (target word, context word) pair. To do this, we set a context window. Assume the context window is 2. Then, we consider two words before and after the target word at each iteration by moving the window. Initially, the window is set at "Machine". Two words after this is "Learning is".

Then the pair (Machine, Learning) and (Machine, is) are part of the training data set. We move the window again. This time the target word is “Learning”. Two words before is “Machine” and two words after is “is an”. The pairs are then (Learning, Machine), (Learning is) and (Learning an) which are addded to the training data set.

After we collect the training data, we may use Skipgram or CBOW model. Skipgram model will try to predict each context word given a target word. We can use a simple neural network architecture for prediction. We can use one-hot encoding for example to represent the target word and context word. After training the neural network, we would obtain a prediction of the context word given a target word. The CBOW is the inversion of the Skipgram model and again we can use a neural network to do the prediction. Skipgram is better suited for small amounts of data while CBOW works well for large amounts of data and is faster. CBOW also has better representation for frequent words.

The word2vec model therefore, maximises the similarity between the vectors for words which appear close together in the text, and minimises the similarity of words that do not. Since there are a lot of words in the dictionary, computing context word with target word and all other words’ similarity can be very slow. To deal with this problem, they are

two optimization methods: *hierarchical softmax* and *negative sampling*.

The *negative sampling* method is to only modify a small percentage of the weights, rather than all of them for each training sample. It will randomly choose few other words and try to maximise the similarity between context word and target word and minimise the similarity between context word and random words.

The idea of *hierarchical softmax* method is based on Huffman tree. The algorithm traverses the tree, and calculates the probability of taking the left or right branch at every node. Therefore, we need to have a representation of each node. In contrast to the regular softmax and negative sampling methods, we no longer have output word embedding for every word. Instead, we have embedding for each node. We compute the dot product between hidden layer (for skip-gram it is context input word embedding and for CBOW it is average or sum of context input embedding) and node embedding and use *sigmoid* function to take the left or right branch at every node. Each leaf will have their own encoding. The rule is assuming a 0 bit for turning left and a 1 bit for turning right in the tree. In this manner, the compute complexity is $\Theta(log_2 V)$ instead of $\Theta(V)$, where V is the size of the vocabulary. The disadvantage of word2vec is that the semantic representation of a word

relies only on its neighbours, it is not an optimal solution.

GloVe

GloVe is another method for vector representation of words. This method does not only rely on local statistics, but also includes global statistics to obtain word embedding. It is unsupervised learning. It uses "word co-occurrence" matrix to get global information. That is, the model considers co-occurrence of words and assigns a probability. For example, assume the target words are "book" and "pen". Then, we can get the co-occurrence probability, $P(k|targetword)$ with various words (say, k) in the dictionary and store them in a matrix. Assume k = "bookmark" and k = "pencap". We can expect that "book" occurs more frequently with "bookmark" than "pencap" and therefore the co-occurrence probability $P(bookmark|book)$ for the former is higher. Similarly, "pen" occurs more frequently with "pencap" than with "bookmark". The ratios of the probabilities captures the closeness between words.

GloVe is a "count-based" model. Therefore, it does not use neural networks to train the data. Both word2vec and GloVe use loss functions to measure the performance of a classification model. There are three main differences between GloVe and word2vec: (i) loss func-

tion: word2vec uses cross-entropy and GloVe uses mean square error; (ii) Glove needs statistics of fixed corpus information to use global co-occurrence matrix. word2vec only considers local information, and can be trained by online learning; (iii) GloVe converges faster and the training period is shorter and more effective than word2vec. Therefore,*in our experiments, we used GloVe word representation as model input for the "student" models. That, the word embedding model used in the student models in our research is Glove.*

2.3.2 Dynamic Word Embedding

Dynamic word embedding is better to capture polysemy. A example of polysemy is,

1. I went to the bank to get money.

2. I went to the bank to catch a fish.

The general idea is that it generates word embedding according to context dynamically. In other words, each word may have different kinds of word representation according to semantic information. Word can have one-to-many embedding rather than one-to-one relationship. There are two main methods for dynamic word embedding: ELMo and BERT. ELMo is based on LSTM and char-level CNN models to generate dy-

namic word embedding. BERT is based on self-attention to get word representations.

ELMo

Deep contextualized word representation (ELMO) was first introduced at AllenNLP [38]. ELMo presented the idea of embedding from language models and used transfer learning for special tasks. Language modeling is a core idea in NLP tasks. A language model is a probability distribution over word sequences. In other words, a language model gives the probability of a certain word sequence being "valid". ELMo uses language model tasks to pre-train model and fine-tune the word representation in downstream tasks.

In particular, ELMO uses character level CNN model to generate character level word embedding (represent words of a text string into raw word vectors). ELMo word vectors are computed on top of a two-layer bidirectional language model. Each word then passes through two layers of Bi-LSTM model to predict the "next word" according to priority. The forward pass contains information about a certain word and the context (other words) before that word. The backward pass contains information about the word and the context after it. Using the words from the forward and backward pass, ELMo creates a pair of

intermediate words. The intermediate word vectors are inputted into the bi-directional model. In the end, the final representation is the weighted sum of the raw word vectors and the 2 intermediate word vectors.

Each LSTM has their own hidden state. Each word then has three kinds of word embedding: char-level word embedding, first layer hidden output and second layer hidden output. Using softmax function and cross-entropy the parameters of Bi-LSTM are tuned. The word embedding for each word will have different weights to generate ultimate word representation during fine-tuning. This training style is also called "feature-based pre-training". Compared to one directional LSTM, Bi-LSTM can capture global information. Bi-LSTM generates word embedding by combining forward and backward information rather than the whole sequence context information.

Note that the ELMo vector (token or word) is a function of the entire sentence containing that word. It takes the whole input sentence to calculate the word embedding. This means the same word can have different word vectors under different contexts and ELMo architecture can capture this. It can handle polysemy.

However, the disadvantage of ELMo is that that the Bi-LSTM to

generate word representation can not be computed in parallel.

BERT

To overcome information leaks, BERT uses self-attention. Bidirectional Encoder Representations from Transformers (BERT) was created by google team in 2018. This model is very famous and important to NLP to promote the development of NLP field. The core technique is attention. Transformer is based on attention mechanism for machine translation. In general, BERT is Transformer's encoder (Figure 2.5).

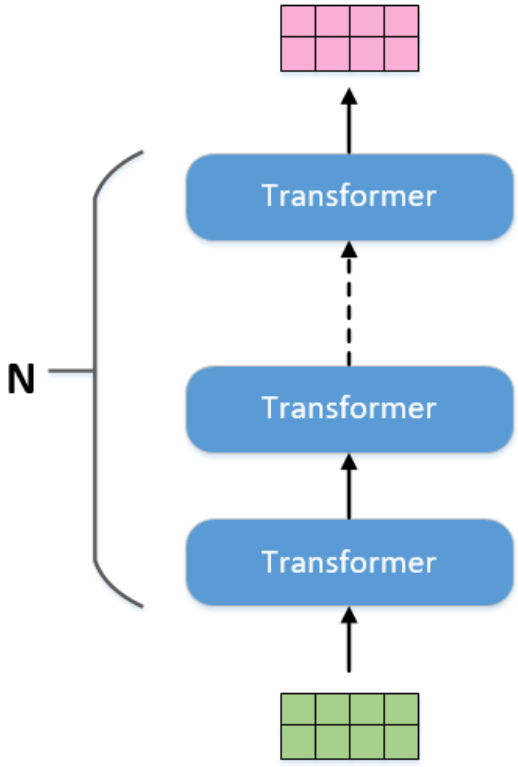


Figure 2.5: BERT model

The Transformer model is explained later in this chapter. Also, BERT can be computed in parallel, and accelerate model convergence and capture whole context information, simultaneously. Like ELMO, there are two ways to pre-train BERT: mask language model and next sentence prediction. During pre-training, the algorithm randomly masks out 15% of the words and lets the language model (BERT) predict these words.

BERT can learn some inter-sentence relationships from large-scale corpus. These tasks are unsupervised learning tasks. Since BERT (base) has about one hundred million parameters, it needs a large-scale corpus to train. Labeling data is expensive and time consuming. Therefore, mask language model and next sentence prediction tasks are used to pre-train models. These two tasks are not "standard task" in NLP. The purpose of tasks is to allow the model to have a better word representation and language understanding. The model outputs contextual word representations. In general, the meanings of words change while developing sentences in a document. As the words evolve and more words are added to the sentence(s) the context of the meaning changes. BERT captures this using the attention mechanism.

Consider for example, "The dog did not eat the biscuit because it was

stale." Here the word "it" is meant for "biscuit" and not "dog". BERT determines which prior word in the sentence the word "it" is referring to. The model then uses using its attention mechanism and calculates a score. The word with the highest calculated score is determined to be the correct similarity word to "it".

In this thesis, we use BERT with neural network (BERT with Gated Recurrent Unit and classification layer) as "teacher" model.

2.4 Transformer Model

In this thesis we use BERT with the Transformer encoder model.

The Transformer is a sequence-to-sequence architecture. Seq2seq is a neural network that transforms a given sequence of words into another sequence. It is widely used in machine translation. This is similar to the LSTM model. Before the Transformer model, the LSTM was the popular model.

Seq2Seq models consist of an encoder and a decoder. The encoder takes the input sentence and maps it to a higher dimensional space or an n-dimensional vector. This vector summarizes the whole contextual information for the input sentence. The decoder uses the vector as an initial hidden state. For LSTM, the performance is good for only

sentences of at most 30.

Bahdanau et al.[2] proposed the attention mechanism to improve the performance of Seq2Seq. The attention-mechanism looks at an input sequence and decides at each step which other parts of the sequence are important. The authors in [46] introduced a novel architecture called *Transformer*. The model adopted the attention mechanism. It is a deep learning model that uses the attention mechanism. The attention mechanism is similar to how a human brain behaves: focusing on one thing while ignoring the surrounding noise. The Transformer model weighs each of the words in the input and associates the significance of these words whole ignoring other words.

Similar to LSTM, The Transformer works with an encoder and decoder. However, it differs from the previously described models because it does not use any recurrent networks (RNN, GRU, LSTM) to generate an output.

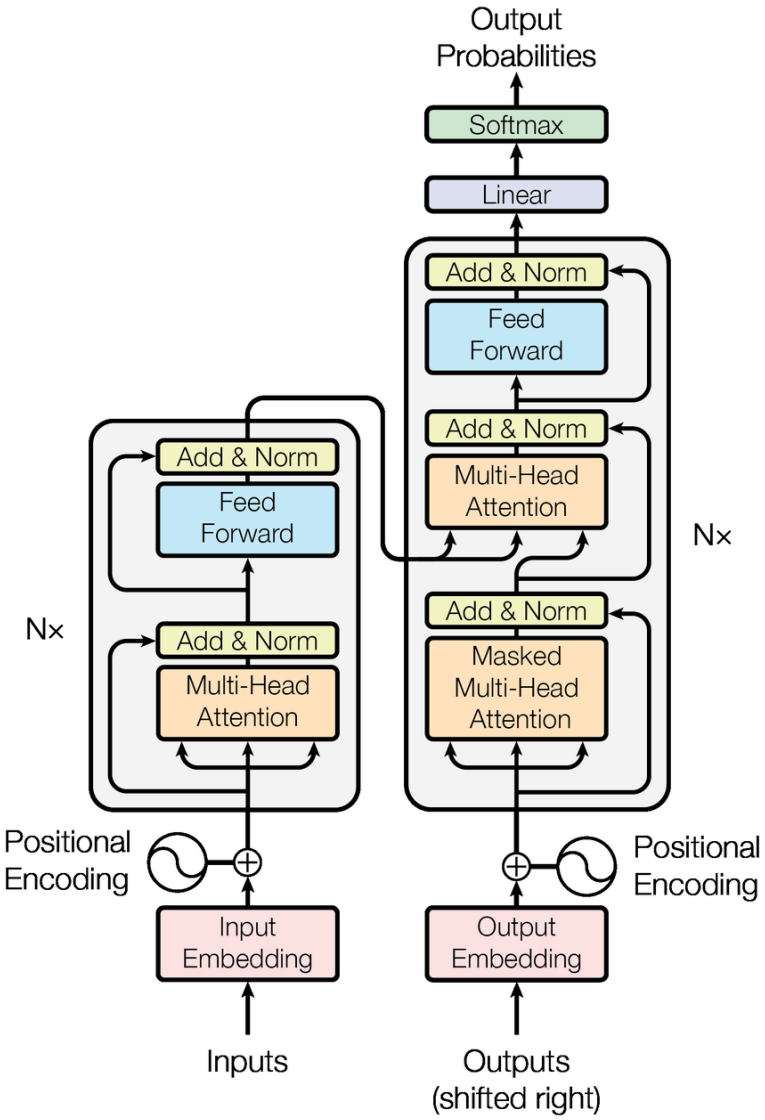


Figure 2.6: An overall of Transformer model

The architecture of the Transformer is shown in Figure 2.6. The encoder is on the left and the decoder is on the right. The encoder basically takes a input sequence and maps them to a sequence of continuous representations. This is fed into the decoder. The output sequence is generated by taking two inputs: the output from the encoder and the

output from previous time step from the decoder.

The **N** times in the figure, implies that layers or modules can be stacked on top of each other **N** times. There are two sub-layers within each layer in the encoder (the multi-head attention sub-layer and feed forward sub-layer). And there are sub-layers in the decoder. The additional sub-layer generates the output from the previous time step. The inputs and outputs of models are firstly embedded into an n-dimensional (512) space,as strings cannot be used as model input directly.

Embedding: In Figure 2.6, there are two kinds of embeddings as Transformers inputs: word embedding, and position embedding. The Transformer uses a random initialization of the weight matrix and tunes the embedding parameter during training-i.e. it does not use any pretrained word embedding such as GloVe or word2vec. Unlike recurrent models that processes sentences sequentially, the Transformer processes all the words in the input sequence at the same time, but without considering the word ordering and position information. To better allow and understand positions of words in the sentence, the positional encoding mechanism was added to the model.

Positional encoding enables the Transformer architecture to acquire

information related to word position. There are two ways to get positional encoding: *learned* positional embeddings and "*cos-sin*" positional embedding. The *cos-sin* method does not need to train parameters and this method can be used even if the length of the sentences have not appeared in the training set.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Figure 2.7: Position encoding in Transformer

Figure 2.7 shows the position encoding mechanism. pos refers to the position of the word in the sentence. i refers to i^{th} embedding dimension. d_{model} refers to the embedding dimension. For example, if we choose d_{model}=512, then i will count from 1 to 512. After getting word embedding and positional encoding for each word, it will add them together to generate word representation as model's input.

Encoder: Each encoder layer has 2 sub-layers: multi-head self-attention mechanism and position-wise fully connected feed-forward network as shown in Figure 2.6. Since the encoder is stacked by several encoder layers, there are two kinds of input: word representations are fist encoder layer's input and other layers inputs are their own previous layer's

output.

The attention mechanism consists of three parameters: *query (q, key (k), value (v).* These are represented in matrices: Q, K and V, for queries, keys and values, respectively. W_q, W_k and W_V are projection matrices. These are used in generating different subspace representations of the Q, K and V. The projection matrix for the multi-head output is represented as W_O.

The attention mechanism, therefore, generates query, key and value vectors ($vector_{q_i}$, $vector_{k_i}$, and $vector_{v_i}$) as input to generate a projection of the input sentence. The Transformer model implements a *scaled dot-product* attention mechanism. Given h attention mechanisms, a *multi-head attention* mechanism creates different projections of the same input (using queries, keys and values), simultaneously. Therefore, it creates multiple (h) outputs in parallel.

Each input's $vector_q$, $vector_k$, $vector_v$ is represented by matrix $\mathbf{Q}\in$ $\boldsymbol{d*l}$, key $\mathbf{K}\in$ $\boldsymbol{d*l}$ and value $\mathbf{V}\in$ $\boldsymbol{d*l}$ respectively, where d represents the embedding size and l represents the length of the sequence. Given a query, the Transformer's scaled-dot product attention function creates a dot product between the query will all the keys. The result of the dot product is divided by $\sqrt{d_k}$. It then applies a Softmax function

to obtain the weight which is used to scale the v values. As this dot product can be applied to the entire set of queries, simultaneously, Q, K and V matrices are fed into the SelfAttention formula 2.8 to calculate the attention score by Softmax function.

$$\begin{aligned} X_{\text{attention}} &= \text{SelfAttention}\,(Q, K, V) \\ &= \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \end{aligned}$$

Figure 2.8: Attention score calculation in Transformer

In general, the dot product of the QK^T will become large after using Softmax function. $\sqrt{d_k}$ is used to divide QK^T to allow the values to not exceed very large and to avoid gradient vanishing.

The transformer model uses 8 multi-head self-attention to capture contextual information. Each head generates their own contextual information and concatenates them together and converts it using a classification layer. Then each output adds it original input as next layer input (adding self-attention output and last layer output or initial word embedding). This architecture is called the residual network 2.10. The next step is layer normalization as the length of each input is different. This is fed to the feed forward network to generate new embeddings.

The feed forward network is made up of two fully-connected linear

network. The first layer activation function is ReLU activation function. There is no activation function in the second layer. The input dimension is the same as the output dimension. The newly generated embedding passes through the residual network (adding self-attention output and feed forward output) performs layer normalization again to generate final layer embedding. The final word embedding is outputted in repeating these several times.

$$\begin{aligned} X_{\text{attention}} &= \text{SelfAttention}\,(Q, K, V) \\ &= \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \end{aligned}$$

Figure 2.9: attention score calculation in transformer

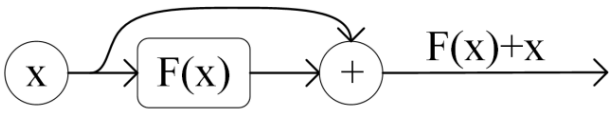


Figure 2.10: residual network

Decoder: The decoder is used for the translation task. Each encoder output is the probability of the current translation word. In general, token "**BOS**" is usually encoder's first input that can lead to decoder output words in sequence as the first correct word cannot be given during testing. During testing, we use previous output as next step input to predict current step words.

Decoder also uses word embedding and positional embedding to represent word embedding. As you can see in Figure 2.6, there are three layers for the decoder: masked multi-head attention, multi-head attention and feed forward network. In the encoder, the self-attention mechanism uses each word to generate word representation in whole sentence regardless of their positional information in the sentence. However, in the decoder, this is not possible. The decoder requires information about the preceding words in the sequence. To predict a word in position k, requires information about preceding words in the previous known output. The decoder, therefore, at the training stage cannot get any information about the current word without information from preceding words. This is done using a mask over the values (produced by the scaled multiplication of matrices Q and K) in the multi-head attention mechanism. This produces a matrix (mask(QK^T)). The masked matrix avoids information leakage before using the Softmax function.

The second layer implements a multi-head self-attention mechanism. This is similar to the first sublayer of the encoder. The new embedding passes through the residual network for layer normalization again to generate final masked layer embedding. This layer uses the cross attention connect encoder and decoder architecture. It receives the

encoder's $vector_{k_i}$ and $vector_{v_i}$, and queries from the previous masked layer embedding $vector_{q_i}$. The self attention formula is then used to calculate the attention score by using Softmax function. Finally, it uses encoder's V to summarize the contextual vector according different attention score. This enables the decoder to consider all the words in the input sequence.

The last layer of the decoder uses Softmax function to predict the next word. This is similar to the second sub-later in the encoder. During training, the decoder's output produces the "ground true' values to train the model.

2.5 Student-Teacher Model

Although BERT can significantly improve many NLP tasks. But, it is too large and complex. More importantly, this implies that it takes a long time to complete the task. This obviously does not conform to the characteristics of small, fast and flexible mobile devices or small platforms. Therefore, compressing the model is important. There are four main compression methods in deep learning algorithms:

1. Network Pruning: networks are typically over-parameterized, since some weight are closed to 0 or equal to 0 and some neuron output's

are 0. We can prune some unimportant weights or neurons;

2. Knowledge Distillation: Let the small model learn and imitate the output of the large model.

3. Parameter Quantization: Using less bits to store a neural network of weight , and clustering weights by k-means method.

4. Weight sharing: Some weights in the model share the same value as other parameters in the model. For example, ALBERT uses the same weight matrices for every single layer of self-attention in BERT.

2.5.1 Knowledge Distillation

The BERT model can be compressed using knowledge distillation, proposed by [3] and then generalized by [14]. Knowledge distillation is based on the student-teacher model. The teacher *transfers* its knowledge to the student. The student model then mimics the pre-trained larger model (e.g., BERT). The teacher model teaches the student model in two parts, (i) which type the data belongs to and (ii) which type the data looks like or is similar to. For example, suppose we have three tags (cat, dog, car). If the image belongs to a cat, the teacher teaches the student that the image is an animal and not a thing.

It also teaches the student that the image looks similar to a dog (as its an animal).

There are two main ways of knowledge distillation in BERT model. The first is pruning some weight and neurons from BERT by knowledge distillation such as TinyBERT [19]. The second is transferring knowledge from large-scale model which is based on BERT model to "first generation neural networks" such as CNN and LSTM [45]. The authors in [45], transferred knowledge from BERT into a single-layer BiLSTM and used word2vec for word embedding. Meanwhile, they used three different kinds of data augmentation methods to increase the number of training dataset. They show that the accuracy of this model is not as good as BERT. However, compared to the first generation network model, the accuracy is improved and saves time (small model's execution time is 490 times less than the larger BERT's model). The authors show that knowledge distillation has low time complexity with good accuracy.

Chapter 3

Literature Review on Sentiment Analysis

Sentences when expressed primarily as a vector, can be represented as inputs to deep learning models.

There has been some work in using SVM for text classification. In [18], the authors segmented the text and manually designed a set of target-dependent features. Then, the authors inputted selected features into a machine learning classifier, SVM. On the other hand, Go et al. [13] used distant supervision to label sentiment data. For example, if a sentence contains ":)" emotion, it will be marked as positive. If it contains ":(" emotion, it will be marked as negative. For feature extraction, the authors used Unigram to extract token information. They used machine learning classifiers (such as SVM and Naive Bayes) for

classification. Cui et al. [8] used high order N-gram language model and a passive-aggressive [8] algorithm based classifier for classification. They showed that their method produced better performance than low order N-gram model. Melody et al.[35] first pre-processed the data (data cleaning) to remove stop word or word stemming. Then, they used a multi-tier prediction model to do the classification. The model predicts if the the text acquires positive, negative or neutral polarity. This prediction is done in the first-tier. If the text is positive, then the model in the second tier predicts very positive or positive for the text. Negative prediction is similar to positive prediction. The authors showed that multi-tier models have better performance than single-tier model which uses only single-tier classifier. In these traditional machine learning methods, the features are manually extracted. Obviously, this is not good enough, as feature engineering is laborious and may lose some features during extraction.

In general, there are three kinds of fundamental models for solving sentiment analysis: convolutional neural networks (CNN), time series model(RNN) and BERT. Deep learning models, inspired by the human brain learn and predict without manual intervention. CNN technique is usually applied to compute vision field. However, there has been

some work using CNN for sentiment analysis. Kim [21] used CNN for sentence classification. The author used four kinds of word embedding (random initialization, freeze pre-trained word2vec (static), fine-tuned word2vec (non-static), freeze pre-trained word2vec (static) and fine-tuned word2vec (non-static) as two "channel"(only fine-tune non-static part during training)), a convolution layer followed by max pool layer to achieve good performance on text classification tasks. In this thesis, we used the CNN model as one of the "student" model.

Alexis et al. [7] inspired by deep convolution layers applied in CV field and the CNN model used to solve sentiment analysis by Kim [21], proposed a new model. Their approach used four models with different depths (9,17,29 and 49 CONV blocks) and character-level embedding. This implies each sentence will be represented by a sequence of characters and each character will have its own embedding. There are only 69 char-level tokens in the dictionary and the embedding size is 16 for each char. The step before last in the pooling layer, max pooling layer is used and in the last step, k-max pooling layer is used. k-max pooling can take the value of top–k among all feature values and keep the original order of these feature values. Because the relative order of these features is kept, k-max pooling retains some position information.

The authors showed that CNN can also solve document classification by increasing the depth of convolution layers and using character-level embedding. However, the previous level inputs do not contribute to the next level input in CNN model, even if it can be processed in parallel. On the other hand, convolution layers pay more attention to the information between words than the whole sentence information. This is a disadvantage. To capture the whole sentence information, RNN and LSTM [15] have proved to perform well in solving sequence problems, since they can imitate the way human beings read and understand context as they are time series models.

Xu et al. [24] used Bi-RNN followed by 1-D max pooling to handle classification. Specifically, they firstly used Bi-RNN to capture contextual information for word representation and combined it with pretrained word2vec embedding. Therefore each word has three kinds of word embedding and goes through a classification layer to compress dimension. Then their approach used a 1-D max pooling to extract text representation based on word representation. Although, max-polling is an efficient way to extract features, it will miss some context information due to the limitation of max-polling. On the other hand, disadvantages of RNN are gradient vanishing and exploding problems.

Therefore, most of the series models used in sentiment analysis tasks are LSTM, since they have forget gate and cell state to avoid gradient vanishing and exploding problems. To improve the performance in traditional RNN, an improved model is used instead of traditional RNN for text classification in Dan Li [29].

Ensemble model is based on the horizontal models. Cliche and Mathieu [6] used a series of LSTM and CNN models as ensemble models in parallel to solve sentiment analysis. They showed that their model had better performance than only using one of the models. More importantly, they used distance training to increase the polarity information. For example, "bad" and "good" are sentiment words that contradict each other. They used distant training to fine tune embedding to ensure that words with extremely different sentiment polarities are far away from each other. However, since the number of parameters is too huge in ensemble model, it is time consuming.

Gated Recurrent Unit (GRU) [5] is a variant of RNN. GRU can not only reduce the influence of gradient explosion or vanishing, but also reduce the computational complexity compared to LSTM. Because GRU only has two gates (rest and update gate), it will decrease execution time. In other words, GRU can quickly realize convergence. Faheem

et al. [55] indicated that GRU performs better than LSTM in Google Snippets and TREC data sets. Since BERT is a word encoder, each output represents contextual meaning. On the other hand, training BERT is time consuming even just fine-tuning. *We, therefore, used BERT with fixed parameters followed by GRU and classification layer as the "teacher" model in our experiments.* In other words, BERT's output will be GRU's input.

Although sequence model (LSTM, GRU) can reduce the influence of gradient explosion or disappearance, it cannot perform well on longer texts, especially in machine translation. In the last few years, attention mechanism has been followed and applied by many researchers. The first paper published by Bahdanau in 2014 [2] confirmed that attention mechanism has better performance in machine translation. However, with the increase in sentence length, the problem of gradient explosion or vanishing is inevitable even if LSTM or GRU is used. In their experiment, they have higher scores of Bilingual Evaluation Substitution (BLEU) than traditional encoder-decoder (two LSTM: one for encoder and one for decoder). This is because, attention mechanism can see the output of all encoders in the decoding process, thus avoiding gradient explosion or vanishing.

Lu and Shi [44] use stacked (double) bidirectional LSTM with attention mechanism to solve document-level semantic classification. They used two layers Bi-LSTM architecture to generate sentence-level and document-level representation. They used the first layer of Bi-LSTM to extract word information in each sentence since words have an nonequivalent effect on the meaning of sentence. Attention mechanisms extract words that are important to the meaning of sentence, based on first layer's output to generate a sentence embedding. The second layer of Bi-LSTM extracts sentence relationships to generate sentence representation by using sentence embedding as input. Similarly, attention mechanism gives each sentence representation with different weight to summarize a document-level representation. They used visualization of reviews to show which words are most important to determine the result. *In our experiments, we used LSTM with attention as a "student" model.*

Wang et al. [48] used Bi-GRU with single attention layer model to do classification. They indicated that concatenating word embedding and output hidden layer as word representation with attention layer have better performance than word representation without attention layer. Because compared to single direction GRU, bidirectional GRU

can see whole text information. However, their experiment only used testing data set and training data set. The authors did not consider validation data set. Validation data set can give a better evaluation of the model and avoid overfitting. *Therefore, in our work, we divided the whole dataset into three parts: training dataset, validation dataset and testing dataset.*

3.1 Dictionary for "student" models

In general, the training data is used to build a dictionary. Some words in the test dataset may not be in the training dictionary, even if pre-trained word embedding are used such as GloVe [37] or word2vec [34]. Such words are called out-of-vocabulary (OOV). The most common method is to use an "unknown word" (UNK) token to represent OOV words. Another method is to use sub-words to generate word embedding, called fastText [20]. Since the number of sub-words is too huge and depends on how many letters are used to generate a sub-word, it is extremely hard for computers to store these sub-words. *Therefore,in our experiment, we ignored the fastText method, instead using the whole words in GloVe as dictionary.*

Most of the related articles use either randomly initialized word em-

bedding or pre-trained word embedding such as (GloVe, word2vec, fastText) with "first generation" neural networks (CNN, LSTM, GRU) to solve sentiment analysis. However, these word embedding methods are static. In other words, each token or word only has consistent word embedding so that it cannot handle polysemy. For example, it is impossible for static word embedding to distinguish different meanings: "bank means riverbank" versus "bank means financial institution". Also, static embedding does not consider the order or position of words and can not capture context information. In tasks such as Named Entity Recognition (NER), the order of words is one of the important factors.

3.2 Transfer learning in sentiment analysis

BERT [11] and GPT-2 [39] are large and complex transfer learning models. There were several challenges in NLP that gave rise to transfer learning: large labelled dataset was difficult to obtain. Therefore, small dataset was used. But, this caused overfitting. Also, training the models from scratch is time consuming. Therefore, it is convenient to use pre-trained large-scale models and fine-tune them to each NLP task. Polysemy was another problem. It required a contextual lan-

guage model to solve the problem. Embedding from Language Models (ELMO) is one such language model. Before ELMO, there was Context Vectors (CoVe) [33]. In CoVe, the authors trained a high-performance translation model, and used translation encoder to encode every token initialized by GLOVE. During pre-trained part, it firstly used seq2seq model to train machine translation task. In specified NLP task, it concatenates trained encoder outputs with original GloVe embedding as ultimate word embedding. However, the model required a lot of translation pairs (such as English-German pairs), since it is a supervised learning. To overcome data collection, ELMO [38] is a language model which uses context word information to predict the word by using Bi-LSTM model.

LSTM Model has two layers, the output of first hidden layer is grammatical information, and the output of second hidden layer is semantic information. ELMO's embedding consists of three parts for each word (word embedding, grammatical information, semantic information). These parts have different weights to determine the results according to different tasks. In [38] the authors show that the first LSTM layer's output has higher weight in question-answering task and each LSTM layer's output weight is similar in sentiment analysis task. How-

ever, ELMO has three main shortcomings: (i) since this model was built using LSTM, extracting features was not good enough compared to attention mechanism; (ii) feature extraction can only be performed in one direction; (iii) it is impossible to parallelize,as it is time-consuming.

These shortcomings are overcome by BERT [11] based on Transformer[46] model.

The Transformer model introduced by Vaswani[46] is an Encoder-Decoder model based on self-attention and position embedding. Transformer model is core model for NLP task, since a lot of popular NLP models such as BERT [11], ALBERT[27], ERNIE[51] originate from Transformer model. The original BERT model had two main parts: (i) how to pre-train the model; (ii) how to fine-tune the model. For pre-training the model, BERT used two methods: Masked Language Model(MLM), to predict masked word and Next Sentence Prediction(NSP), to determine whether a pair of sentences are related. They added a special token("CLS") at the start of sentences which can generate a comprehensive vector to determine whether a pair of sentences are related. Pre-trained models were used to fine-tune the model to do popular tasks such as sentiment analysis, question-answering etc. BERT used the vector which is the output of "CLS" followed by a linear layer to

do sentiment analysis and each word contextual embedding to do such as question-answering and Part-of-speech (POS) tagging. Since BERT utilized self- attention model, it can see the whole contextual information (all sentences at once) and , therefore be parallelized. On the other hand, As attention mechanism does not consider order between tokens, BERT adds position embedding and segment embedding to obtain the position information of the tokens. Therefore, it is an advanced model, which improves performance in many NLP tasks. BERT can connect not only through the fully connected layer, but also through "first generation models", such as LSTM and CNN.

Cai et al. [4] used BERT-BiLSTM model to perform sentiment analysis between investors and consumers in energy market. They used BERT to generate contextual word representation and BiLSTM to generate comprehensive information based on BERT's output. Rao and Zhang[49] used BERT-DNN model to do aspect-level sentiment classification. Dong et al. [12] used BERT and CNN models to solve sentiment analysis in parallel. They used the CNN model, which utilized BERT's output as its word representations, to generate a sentence-level vector. The training is similar to the work in [21], BERT generates a sentence-level vector, the output of token "CLS". It combines two

vectors to accomplish the classification task. it is a ensemble model. However, these methods [4], [12], [49] take long training time since (based) BERT model has over one hundred million parameters, That is a potential problem.

In this thesis we compress the BERT model using knowledge distillation. We explore and implement two knowledge distillation methods which transfer knowledge from a "fine-tuned" BERT model to Bi-LSTM with-atten and CNN models, respectively.

The next chapter will explain the details of BERT, CNN and Bi-LSTM with-atten architectures and how to use "knowledge distillation" to transfer knowledge between teacher model and student models in our proposed approach.

Chapter 4

Distilling knowledge through student-teacher model and BERT

This chapter explains how we use BERT model to "teach" student model. We choose the Bi-LSTM and Bi-LSTM-atten, CNN as "student" models. For the "teacher" model, we can either use BERT model with linear function or BERT model with GRU and linear function. We also explain the loss objective function.

4.1 Overview

We will use the figure below to explain our proposed methodology.

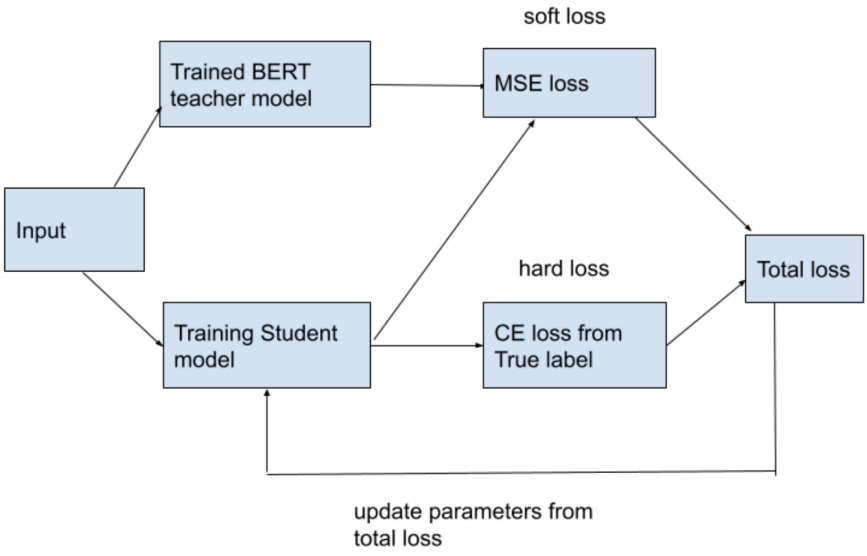


Figure 4.1: Knowledge Distillation architecture

4.2 Student Models

We will first explain the student models studied in this thesis.

In the student models considered the first two steps mentioned below are the same for word embedding.

1. First, we build the dictionary. The model in general uses training dataset to build up the dictionary. However, we choose pre-trained word embedding GloVe (Section 2.3.1) to do this. The size of GloVe's dictionary is 40000 which can incorporate common English words.

2. The second step is data cleaning. To speed up training, we remove

stop-words which are not important in sentences. This can reduce the length of the sentences. GloVE obtains vector representations for words. Each word has its own original GloVE embedding. We add two special tokens [PAD] and [UNK] in the dictionary. Since the length of each sentence is different, we use token [PAD] to make sure each sentence is of the same batch and has same size. Some words may not be in GloVe dictionary. These words are replaced by a special symbol [UNK].

4.2.1 Bi-LSTM

Figure 4.2 shows the Bi-LSTM architecture which will be used in this thesis.

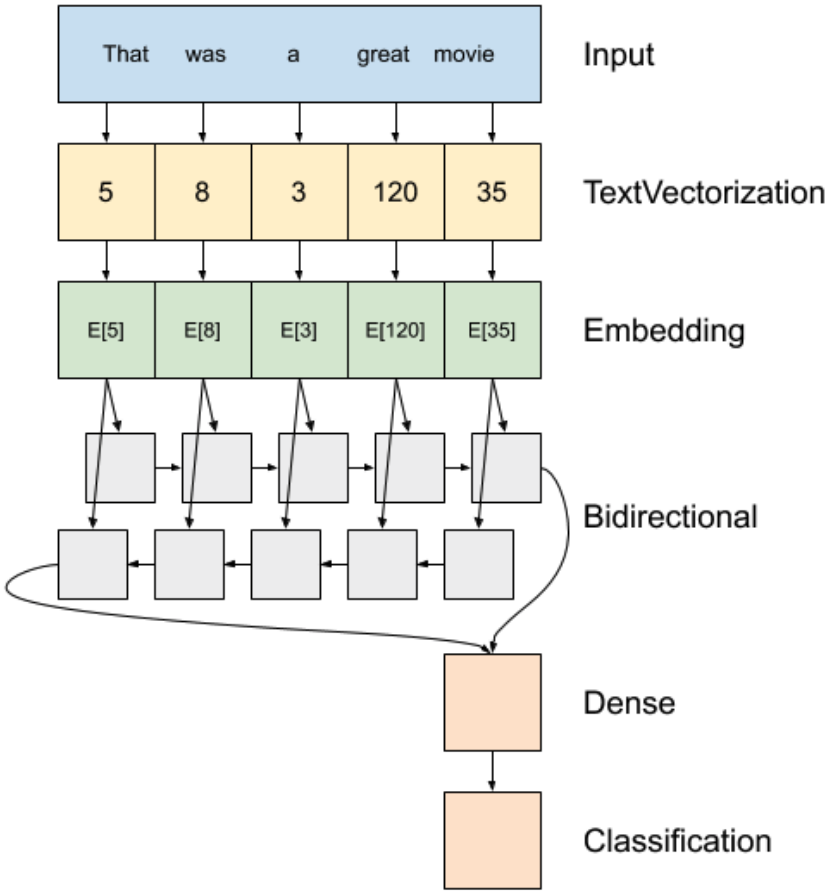


Figure 4.2: Bi-LSTM architecture

1. After embedding, in the next step these sentences pass through Bi-LSTM to get last hidden outputs H_t ($H_t \in R^{b \cdot 2H}$) (Equation 4.1), where **H** represents the size of hidden dimension and **b** represents batch size. H_t uses classification layer ($W \in R^{K \cdot 2H}$) to generate final output H_f (Equation 4.2). **K** is the number of labels.

$$[H]H_t = [\overrightarrow{h}_t, \overleftarrow{h}_t] \tag{4.1}$$

$$H_f = [H_t \cdot W^T] \tag{4.2}$$

2. After we obtain the final output, we apply a loss function for comparison and to determine the accuracy of the result. We use cross-entropy loss function.

3. Once we get the loss, we apply back propagation algorithm to update model parameters. Back propagation through time is a technique designed for LSTM/GRU back propagation.

   In this thesis, we choose Adam optimizer. Adam optimizer is the combination of adaptive gradient algorithm and root mean square propagation. It takes advantages of both these techniques and works well with sparse gradients and non-stationary settings. Adam optimizer requires some hyper-parameters: learning rate, γ, β_1 and β_2 represents betas, θ_0 represents initial parameters of model. Adam optimizer algorithm is given below:

Algorithm 1 Adam Optimizer

Require γ,β_1,β_2,λ,θ_0

for *t=1 to ...* **do**

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

if $\lambda \neq 0$ **then**

$g_t \leftarrow g_t + \lambda\ \theta_{t-1}$

$m_t \leftarrow \beta_1\ m_{t-1} + (1\text{-}\beta_1) g_t$

$v_t \leftarrow \beta_2\ v_{t-1} + (1\text{-}\beta_2) {g_t}^2$

$\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$

$\widehat{v_t} \leftarrow \frac{v_t}{1\beta_2^t}$

$\theta_t \leftarrow \theta_{t-1} \frac{\gamma \widehat{m_t}}{\sqrt{\widehat{v_t}}+\varepsilon}$

return θ_t

4.2.2 Bi-Long Short-Term Memory with Attention

We also consider the Bi-LSTM with attention model for better performance. This is another student model considered in this thesis.

Figure 4.3 shows the Bi-LSTM with attention mechanism architecture. Each step in Bi-LSTM's output (H) passes through *tanh* activation function. It has a new linear function as attention vector $W_a (W_a \in R^{2H \cdot 1})$, which finds relationship between the output of each step LSTM's output and normalizes the weight by Softmax function, Equation 4.4. This vector can be updated during training. Finally, it generates the in-

formation, Equation 4.5 by *tanh* activation function, Equation 4.6. Loss function and back propagation algorithm are same as update Bi-LSTM model discussed previously.

Experiments indicate that Bi-LSTM-attention method is better than the Bi-LSTM model without attention mechanism.

$$M = tanh(H) \tag{4.3}$$

$$\alpha = softmax(W^T \cdot M) \tag{4.4}$$

$$r = H \cdot \alpha^T \tag{4.5}$$

$$H_f = tanh(r) \tag{4.6}$$

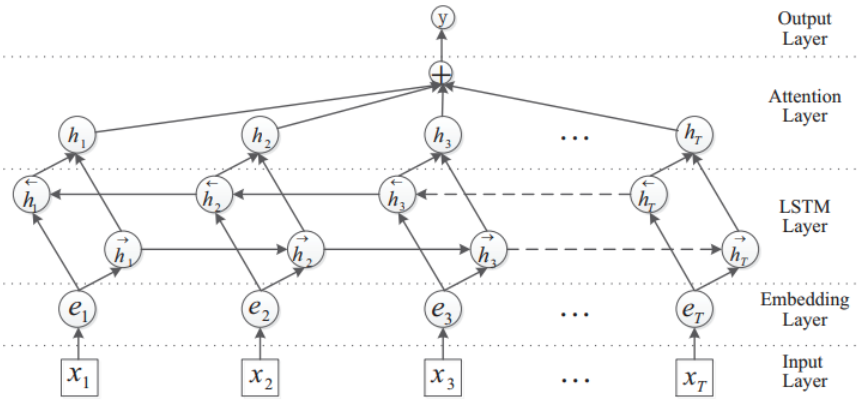


Figure 4.3: Bi-LSTM with attention architecture

Example

4.2.3 Convolution Neural Network (CNN)

The third student model considered is CNN, Figure 4.4. Using one kind of filter is not comprehensive to capture information and one size of filter is not good enough to find relationship between words. The model, therefore, extracts information by using different filters to extract different features.

In general, the dimension of each filter is the same as the dimension of word embedding. A convolution operation involves a filter $w \in R^{h \cdot k}$, which is applied to a window of h words to produce a new feature. For example, an extracted feature c_i is generated from a window of words $x_{i:i+h-1}$ by Equation 4.7.

$$[h]cf_i = f(w \cdot x_{i:i+h-1} + bias) \tag{4.7}$$

f is a non-linear function such as the hyperbolic tangent. This filter is applied to each possible window of words in the sentence $x_{1:h}, x_{2:h+1}, ..., x_{n-h+1:n}$ to produce a feature list, Equation 4.8

$$f = [f_1, f_2, ..., f_{n-h+1}] \tag{4.8}$$

with $f \in R^{n-h+1}$.

We then apply a max-pooling function. That is, take the maximum value $\hat{f}$= max$\{f\}$ as the feature corresponding to the filter. We have described the process by which one feature is extracted from one filter. The model uses multiple filters (by varying the size of filter) to obtain multiple features. That is, the filters have different size and the same size filters have different parameters. Each filter will output one value after max-pooling operation. These values are concatenated together to obtain general information which will go though classification layers.

For example in Figure 4.4, there are 3 different sizes of filters (2,3,4) and there are 3 kinds of filters of the same size. Each filter will output one value, a total of 9 values. Those values passes through a classification layer and uses loss by cross entropy.

Back propagation algorithm is the same as update Bi-LSTM model as in previous sections. Experiments indicate that CNN model has the least computation time.

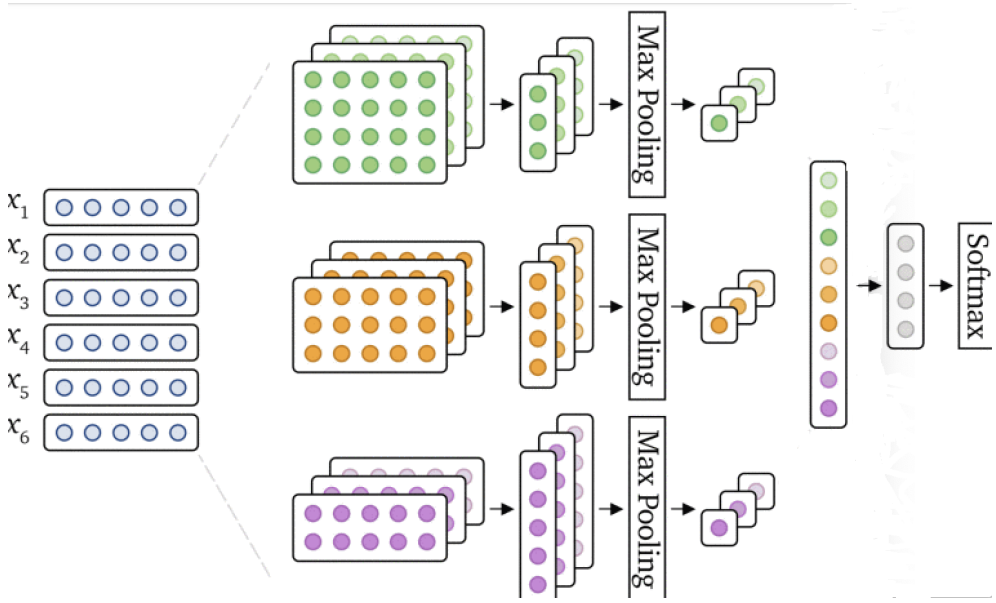


Figure 4.4: CNN architecture

4.3 Teacher Models

There are three kind of embedding words in BERT: token embedding, segment embedding and position embedding. BERT expects a certain format for input. It also has special tokens for task training. We will explain this using the example in Figure 4.5.

CPS : This classification token is placed first in any input sequence. This is mainly used for classification tasks. For other tasks it is ignored.

SEP : This token separates sentence pairs. If there is only one sentence sequence, it is placed at the end of the sentence.

MASK : Used for masked words in pre-training. In the example, one of the masked word is "likes".

BERT's input sequence must follow the format in Figure 4.5.

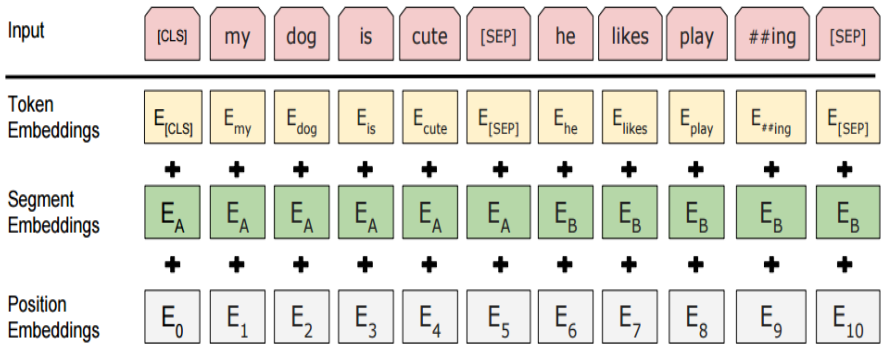


Figure 4.5: BERT embedding architecture

Token embedding: This is similar to that used in the Transformer model. Each word is represented by a vector. The BERT Tokenizer follows a WordPiece algorithm. The original word is split into smaller sub-words and character. For example, the token "playing" can be split into "play" and "##ing". This is to allow out of vocabulary (OOV) words.

BERT Tokenizer's vocabulary limit size is 30522 which includes whole words, subwords occurring at the front of a word or in isolation such as "im" of "importantly". Subwords not at the front of a word, are represented by '##" such as "##ly" of "importantly"and individual characters.

The Tokenizer first checks if the whole word is in the vocabulary. If not, it divides the word into the largest possible subwords contained in the vocabulary. In the worst case, it will divide the word into individual characters. Therefore, words that are not in the vocabulary are divided into subwords and character tokens.

Segment embedding: After token embedding, segment embedding is done (Figure 4.5. This is to separate sentence or determine whether two texts are semantically similar. The sentences are simply spliced together and then sent into the model. The first and second sentences will encode in 0 and 1, respectively, if there are two sentence pairs. In general, it provides a numeric class to distinguish between sentences. For classification task, since the input is a whole, each token will have the same segment embedding 0.

Next is **positional embedding**. Unlike the Transformer model, the position embeddings in BERT will be learned during the training phase. Each token is provided its position in the input sequence (Figure 4.5.

Thus, each token in BERT will be associated with three kind of embedding. This will will be added together as initial input in BERT 4.5. Since the the dimension of the three kinds of embedding is the same and the length of input is at most 512.

Pre-training task: There are two kinds of pre-training tasks: Masked Language Mask (MLM) and Next Sentence Prediction (NSP).

In the training process, the user randomly masks 15% of the tokens and then predicts these masked tokens 4.6 unlike continuous bag of words (CBOW) model that predicts every word. Since token [Mask] is fine-tuned, after the word to be masked is determined, 80% of the time this will be directly replaced with token [Mask], 10% of the time will be replaced with any other word, and 10% of the time will keep the original token.

The finial loss function only calculates the tokens that were dropped by mask. The core of MLM architecture (Figure 4.6) is in masking words. In the BERT model, as it is based on the Transformer model and attention mechanism, it can see contextual information in two directions simultaneously. The MLM architecture, therefore, is a good fit to the BERT architecture due to its two-direction properties.

Understanding the relationship between two sentences can improve performance in many downstream tasks such as Question Answering (QA) and Natural Language Inference (NLI). These can not be done by traditional language model. Therefore, in order to understand sentence relationships, BERT uses a binary task called next sentence prediction

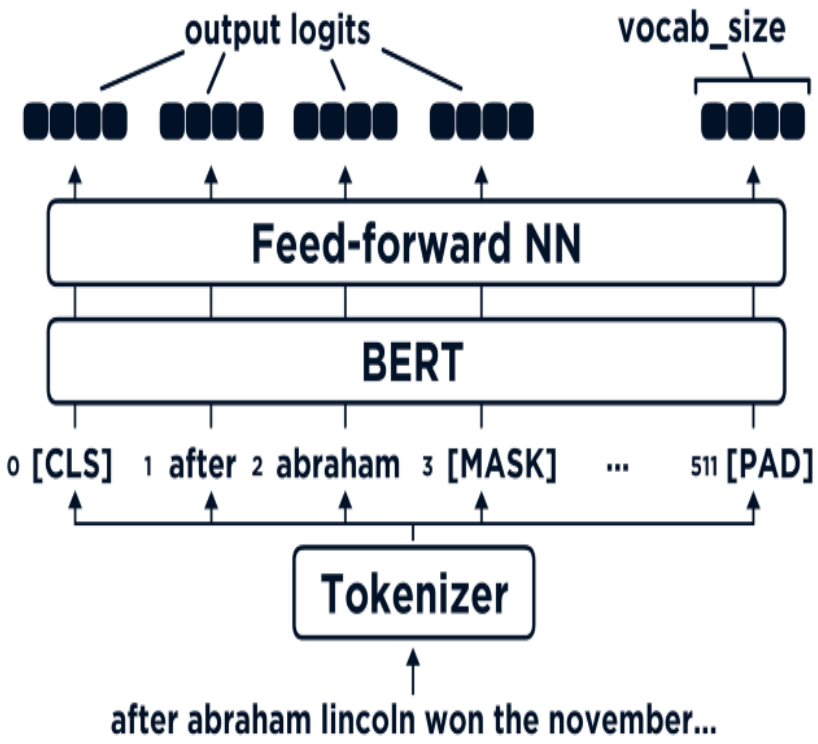


Figure 4.6: BERT MLM architecture

(NSP) to train the model.

In NSP, when choosing sentence A and sentence B for each sentence pair during pre-training, 50% of the time B is the actual next sentence that follows A,and 50% of the time it is a random sentence from the corpus. As mentioned earlier in this section, the model adds a special token [CLS] which represents the overall information of the input in front of the first sentence and token [SEP] to distinguish sentence pair in front of the second sentence. As can be seen in figure 4.7, **C**, the output [CLS] token is used for next sentence prediction by a classification layer.

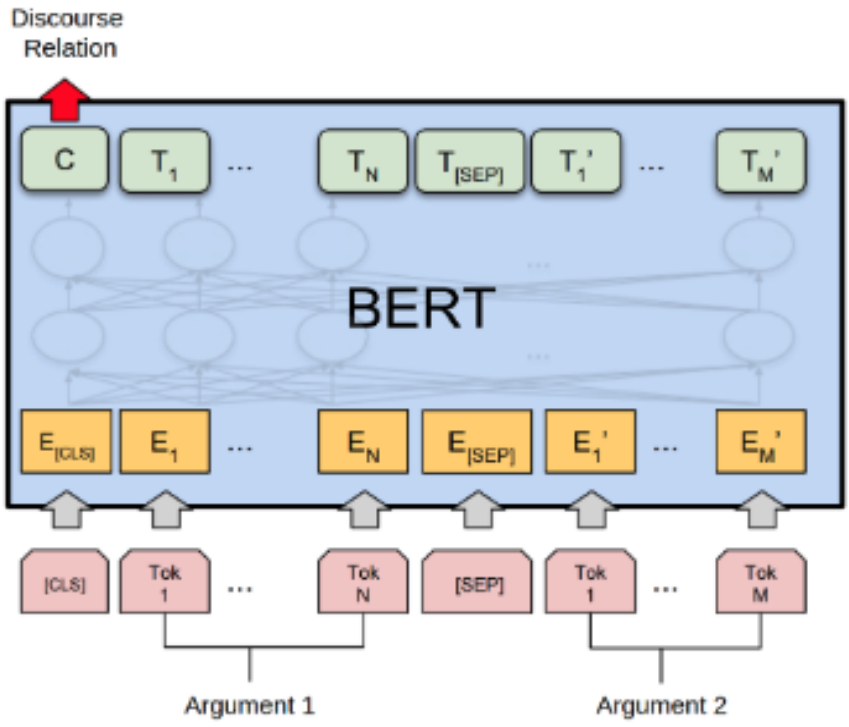
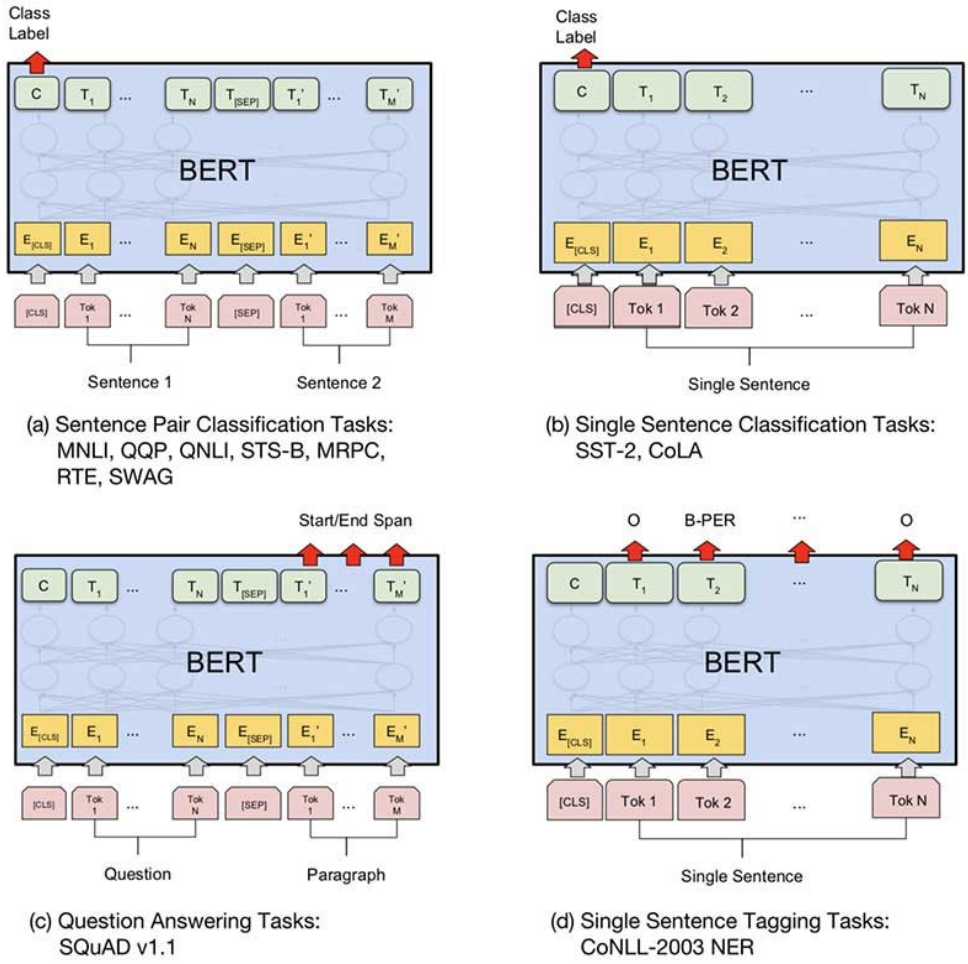


Figure 4.7: BERT NLP architecture

After training BERT on a large number of corpus, it can be applied to downstream tasks of NLP. For other tasks, we can also make corresponding predictions according to BERT's output information.

Figure 4.8 shows BERT's models in 11 different tasks. **In all these models, the only additional requirement to fine-tune specific tasks is adding an output layer to BERT.** Note that as has already made the model learn information, it will have a good performance quickly in fine-tuning. *For the thesis, we focus on fine-tuning the sentiment analysis classification task.*



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Figure 4.8: BERT fine-tuning architecture

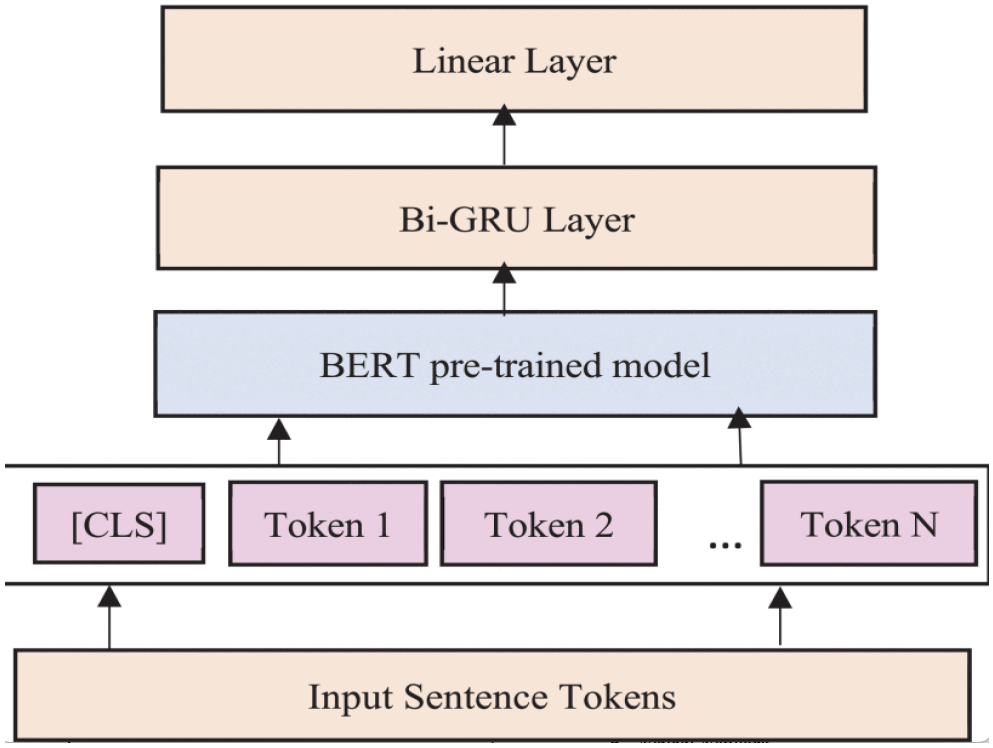


Figure 4.9: BERT Bi-GRU architecture

4.3.1 Fine-Tuning Classification Tasks:

In classification tasks, there are two kinds of fine-tuning methods: (i) using the output of token [CLS] followed by a classification layer (Section 4.3.1). In Figure 4.8 b, **C** represents the overall information of the sentence. $T_1 ... T_N$ represents the BERT representation of each word; (ii) using the BERT output of each word followed by "first generation model" such as LSTM (Section 4.3.1).

In this thesis we study both methods. Experiments show that one method is more suitable for some dataset than the other. For example,

we found that the second method has good performance than the first in IMDB dataset, while the first method has better performance than the second in Twitter dataset.

Below we discuss the "teacher model" architectures.

BERT with linear model

Specifically, we used token [CLS] $C \in R^{b \cdot H}$ as classification layer input, where b represents batch size and H represents the dimension of the hidden output (the dimension of BERT output is 768). The new parameters introduced during fine-tuning are classification layer weights. The dimension of the classification layer is $W \in R^{K \cdot H}$, where K, the number of labels can be varied.] depending on the dataset. A standard classification loss is computed with C and W by Softmax function, i.e., $\log(softmax(CW^T))$. Figure 4.8 b illustrates the technique.

BERT with GRU

The proposed architecture, BERT with Bi-LSTM, is illustrated in Figure 4.9. First, each input sequence passes through the BERT model to get their own contextualized word embedding. Then, it passes through through Bi-LSTM model (dynamic word embedding) step by step. The forward and backward outputs will be concatenated. The last step of

the hidden output ($Y_n \in R^{b \cdot 2H}$) will be overall the sentence information as classification layer $W \in R^{K \cdot 2H}$ input. The new parameters introduced during fine-tuning are classification layer weights and Bi-LSTM weights. the loss function as the is the same as above.

4.4 Knowledge Distillation

Now that we have introduced teacher and student models, we will discuss knowledge distillation (Figure 4.1). Knowledge Distillation (Section 2.5) is one of the model compression methods. The core idea is to allow the "teacher" model to teach the "student" model. In this thesis, use "BERT model with first generation network" as "teacher" model.

There are two kinds of loss functions: *soft loss* and *hard loss*. Hard loss function is cross-entropy (CE) and the soft loss function is mean square error (MSE). We computing soft loss between BERT's logit output and student model's logit output through MSE. The argument of the Softmax function is known as logits. Logit is a linear function output. We compute hard loss between student's logit output by using cross-entropy loss.

BERT's final logits will be used as the soft learning objective of the "student" model by MSE. True label will be used as the hard learning

objective of the 'student" model by CE.

Below is the proposed knowledge distillation architecture, Figure 4.10. The original input will go through "teacher" model and "student" model, respectively. The loss function is shown in Equation 4.9, where $L_{distill}$ 4.10 is the mean-squared-error (MSE) loss between the student network's logits against the teacher's logits. $z^{(B)}$ and $z^{(S)}$ are the teacher's and student's logits, respectively. Our "student" model has to learn to mimic the behavior of the BERT model.

$$L = \alpha \cdot L_{CE} + (1 - \alpha) \cdot L_{distill} \tag{4.9}$$

$$L_{distill} = ||z^{(B)} - z^{(s)}||_2^2 \tag{4.10}$$

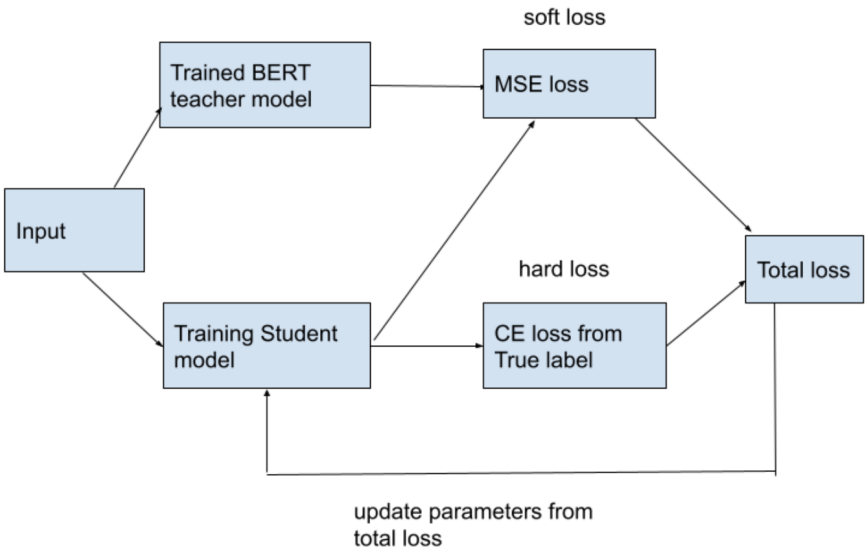


Figure 4.10: Knowledge Distillation architecture

Chapter 5

Experiments and Results

5.1 Experimental Setup

Dataset: The first dataset is the IMDB movie review is collected from Stanford University [32]. This is recorded in a CSV file. The data gives 25000 highly polar movie reviews for training and 25,000 reviews for testing. It can be publicly accessed from the Stanford University website: https://ai.stanford.edu/~amaas/data/sentiment/. There are two kinds of labels: negative and positive.

In addition, data from the Sentiment140 data set with 1.6 million Twitter [13] data is also collected. This is publicly available on the Kaggle website: https://www.kaggle.com/kazanova/sentiment140. There are two kinds of labels: positive, negative. We divided the dataset into training data, validation data, and testing data as a ratio of 8:1:1

in Twitter dataset and as a ratio of 4:1:5 in IMDB dataset (there are two sub-dataset in IMDB,that are training dataset and testing dataset. We divided the training dataset into training data, validation data at a ratio of 4:1).

Data cleaning: First, the data is cleaned to ensure each text has the same standard. Specifically, we remove useless symbols, stop-words, and URL to reduce the size of the text. In addition, we also do zero padding according to the longest text, so that each text has same length, because the input to the model in each text should be the same.

Languages and Systems In the experiments, all programs are written in the *pytorch* programming language and executed on a NVIDIA Tesla p100 with 128 GB memory. The operating system is Linux 7 (core).

Hyperparameter setting

5.1.1 "Teacher model"

The teacher model structure follows the general architecture and hyper-parameters of BERT and transformer. The hidden size is 768 and maximum sequence length is 256 in IMDB dataset and 32 in Twitter dataset. Firstly we train the teacher models. We use training dataset

to fine-tune the teacher model. we have 2 kinds of teacher model: BERT with linear model and BERT with GRU. For BERT with linear model, we choose hugging face pre-trained BERT model with linear function. We used the output of token [CLS] as linear function input (after applying dropout). The learning rate is initialized as 3e-5, since we just fine-tune the model.

For BERT with GRU and linear model, we freeze the pre-trained model and use each time step output as "contextual word embedding". These output will be the GRU model input. The GRU model has two hidden layer and it is a bidirectional model. The final hidden state, hidden, is the same shape as LSTM. We obtain the top two hidden layers from the first dimension, hidden[-2,:,:] and hidden[-1,:,:], and concatenate them together before passing them to the classification layer (after applying dropout).The output of logits will be evaluated by cross entropy loss Equation 5.1.The learning rate is initialized as 0.01. For the back propagation algorithm, Adam optimizer is used in all models during training. Figures 5.1 and 5.2 show the model parameters.

5.1.2 "Student model"

All the word embedding is pre-trained GlOVe embedding (GLOVE.6B.100d). The dimension of word embedding is 100. The proposed time-series (LSTM, LSTM-attention) models for student and baseline have 2 LSTM layers and they are bi-directional. Therefore, the input dimension is 100 and the output dimension for each step is 512. The dropout layer rate is 0.25 and the last output layer of LSTM will not apply dropout function. The purpose of this layer is to avoid over fitting: this means that the LSTM is so well trained with the current data set that it does not work well for other new data sets. The final hidden state, hidden, has a shape of [num-layers * num-directions, batch-size, hid-dim]. The hidden dimension is 256, these are ordered: $forward_layer_0$, $backward_layer_0$, $forward-layer_1$, $backward_layer_1$, ..., $forward_layer_n$, $backward_layer_n$.

As we want the final (top) layer (forward and backward hidden states), we obtain the top two hidden layers from the first dimension, hidden[-2,:,:] and hidden[-1,:,:], and concatenate them together before passing them to the classification layer (after applying dropout). The output of logits is evaluated by cross entropy loss Equation 5.1.

For CNN and KD_CNN, we use three kinds of window size 3,4 5 with 64 filters for each of them. The intuition is that we will be looking for

the occurrence of different tri-grams, 4-grams and 5-grams that are relevant for analysing sentiment of movie reviews. Each filter will output one value by max-pooling operation and concatenate them together to generate contextual information of dimension 256. This is passed to the classification layer (after applying dropout). The input word embedding is same as LSTM model.The output of logits is evaluated by cross entropy loss Equation 5.1. For the back propagation algorithm, Adam optimizer is used. The learning rate is initialized as 0.01 in all the student models (CNN,LSTM and LSTM attention).

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{5.1}$$

IMDB	Batch size	Learning rate	Input dimension	output dimension	hidden dimension
CNN	32	0.01	100	2	256
KD_CNN	32	0.01	100	2	256
LSTM	32	0.01	100	2	512
LSTM_atten	32	0.01	100	2	512
KD_LSTM	32	0.01	100	2	512
KD_LSTM_atten	32	0.01	100	2	512
BERT_linear	8	0.001	768	2	768
BERT_GRU	32	0.01	768	2	512

Figure 5.1: Model parameters for IMDB

Twitter	Batch size	Learning rate	Input dimension	output dimension	hidden dimension
CNN	128	0.01	100	2	256
KD_CNN	128	0.01	100	2	256
LSTM	128	0.01	100	2	512
LSTM_atten	128	0.01	100	2	512
KD_LSTM	128	0.01	100	2	512
KD_LSTM_atten	128	0.01	100	2	512
BERT_linear	64	0.001	768	2	768
BERT_GRU	128	0.01	768	2	512

Figure 5.2: Model parameters for Twitter

5.2 Evaluation: Results and Discussion

In this thesis, we evaluate the model performance using the following metrics: execution time, accuracy (Equation 5.2), precision (Equation 5.3), recall (Equation 5.4) and F1-score (Equation 5.5).

Accuracy is a metric that describes how the model performs across all classes. It is useful when all classes are of equal importance. In order to reflect the fairness of the experiment, the number of positive and negative samples is the same. It is calculated as the ratio between the number of correct predictions to the total number of predictions as shown in Equation 5.2.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.2}$$

where, TP and TN are True Positive and True Negative, respectively. FP and FN are False Positive and False negative, respectively.

The precision metric is calculated as the ratio between the number of positive samples correctly classified to the total number of samples classified as positive (either correctly (true positive) or incorrectly (false positive)). The precision measures the model's accuracy in classifying a sample as positive.

$$Precision = \frac{TP}{TP + FP} \tag{5.3}$$

The recall metric is calculated as the ratio between the number of positive samples correctly classified as positive to the total number of true positive and false negative samples. The recall measures the model's ability to detect positive samples. The higher the recall, the more positive samples detected.

$$Recall = \frac{TP}{TP + FN} \tag{5.4}$$

F1-Score is the harmonic mean of precision and recall and gives a better measure of the incorrectly classified cases than the accuracy met-

ric.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{5.5}$$

5.2.1 Evaluation measures on dataset

As usual,the period with the minimum loss value in validation dataset will be selected as the parameter of the model. In our experiments, we find that BERT with linear models have better performance than BERT with GRU model in all benchmark datasets. This is because,

- BERT with linear model is based on self-attention, it can see whole sentence simultaneously. GRU does not allow text to be read simultaneously.

- BERT with linear is a fine-tuned model. It converges quickly. GRU is trained from scratch and converges slowly. The lowest rate appears in epoch 1 and 2 for BERT with linear model and 6 and 7 for BERT with GRU (Figures 5.3).

Therefore, in our experiments, we used BERT with linear model as the final "teacher" model.

teacher model	Best epoch	Best accuracy
BERT+linear IMDB	1	92.75
BERT+GRU IMDB	6	92.24
BERT+linear Twitter	2	87.66
BERT+GRU Twitter	7	86.01

Figure 5.3: Best epoch in "teacher model" for validation set

The process of knowledge distillation

The teacher model is overfitting for knowledge distillation if the number of epochs is greater than the best epoch. Therefore, the lowest loss rate model parameters are stored when fine-tuning the "teacher model". During training the "student models", we load "teacher model" parameters to compute the soft loss. In the experiments, we find that "student models" through knowledge distillation have better performance than those models without knowledge distillation. However, compared to the "teacher models", "student models" converge slowly. There are two main reasons: (i) all "student models" were trained from scratch and takes time to converge; (ii) compared to the teacher model, we used a larger batch size, that takes more epochs to update parameters. Interestingly, "Student models" with knowledge distillation (KD) need more epochs to converge. The models with the proposed knowledge distillation is specified as KD_** in the table. For example, KD_LSTM,

KD_CNN and KD_LSTM_Atten require 13, 19 and 15 epochs, repectively compared to their counterparts without KD. since at first, soft loss was too high, which caused the update to be slow. As can be seen from the figure 5.4 and 5.5, the update time of models with KD are almost twice that of the model without KD.

Models	Best Epoch	accuarcy	loss rate
LSTM	7	87.76%	0.295
KD_LSTM	13	89.13%	0.286
CNN	9	85.24%	0.329
KD_CNN	19	86.08%	0.319
LSTM_Atten	8	88.21%	0.284
KD_LSTM_Atten	15	89.68%	0.279

Figure 5.4: Best epoch in "student model" for validation set(IMDB)

Models	Best Epoch	accuarcy	loss rate
LSTM	3	84.37%	0.353
KD_LSTM	7	85.67	0.349
CNN	3	82.63%	0.393
KD_CNN	6	83.30%	0.389
LSTM_Atten	5	84.39%	0.354
KD_LSTM_Atten	10	85.85%	0.345

Figure 5.5: Best epoch in "student model" for validation set(Twitter)

Evaluation

IMDB	Accuracy	Precision	Recall	F1-score	hidden dimension
CNN	0.8524	0.8526	0.8526	0.8526	256
KD_CNN	0.8610	0.8610	0.8610	0.8610	256
LSTM	0.8776	0.8779	0.8776	0.8776	512
LSTM_atten	0.8822	0.8834	0,8822	0.8822	512
KD_LSTM	0.8914	0.8916	0.8914	0.8914	512
KD_LSTM_atten	0.8969	0.8969	0.8969	0.8969	512
BERT_linear	0.9275	0.9289	0.9275	0.9274	768
BERT_GRU	0.9223	0.9223	0.9223	0.9223	512

Figure 5.6: Evaluation matrix for IMDB

Tiwtter	Accuracy	Precision	Recall	F1-score	hidden dimension
CNN	0.8263	0.8263	0.8263	0.8263	256
KD_CNN	0.8330	0.8331	0.8330	0.8330	256
LSTM	0.8437	0.8438	0.8437	0.8437	512
LSTM_atten	0.8439	0.8439	0.8439	0.8439	512
KD_LSTM	0.8568	0.8567	0.8567	0.8567	512
KD_LSTM_atten	0.8585	0.8586	0.8585	0.8585	512
BERT_linear	0.8766	0.8767	0.8766	0.8766	768
BERT_GRU	0,8601	0.8601	0.8601	0.8601	512

Figure 5.7: Evaluation matrix for Twitter

Figures 5.6 and 5.7 show the evaluation metrics for the dataset, IMDB and Twitter. The models with the proposed knowledge distillation is specified as KD_** in the table. All student models have improved the performance by using knowledge distillation. Specifically, the student models with knowledge distillation has improve approxi-

mate 0.7% on accuracy, precision, recall and F1-score for both dataset in CNN model.

The performance of Bi_LSTM_atten with knowledge distillation models are not as good as BERT model. There are three main reasons: (i) Bi_LSTM_atten models do not have "dynamic word embedding", therefore, they do not consider contextual information; (ii) the number of model parameters are smaller than BERT model in Figure 5.1 and 5.2; (iii) BERT model was pre-trained on large-scale corpus, so that it can capture more comprehensive corpus information to represent the teacher model.

We also compared our model to the results in the article [48]. In this article, the authors applied the Skip-gram algorithm to train word embedding. using the default parameters in word2vec. The authors used a bi-directional recurrent neural network to capture the features, in particular, Bi-GRU. They tested their model on IMDB dataset. Figure 5.8 shows the comparison between our Bi_LSTM_atten model with knowledge distillation and the results in the article. There is a 3.2% and 5.2% on F1-score in IMDB dataset.

The Bi_LSTM_atten model with knowledge distillation have improved approximate 1.5% on accuracy, precision, recall and F1-score in all

datasets. The Bi_LSTM models with knowledge distillation have improved approximate 1.3% on accuracy, precision, recall and F1-score in all datasets.

Type	Model	F1-score
Full size(Our)	BERT	0.9275
Full size	Bi-GRU +Attention+Embedding[49]	0.8667
Full size	Bi–LSTM + Attention[49]	0.8449
Compressed(Our)	Bi–LSTM with KD	0.8914
Compressed(Our)	Bi–LSTM + Attention with KD	0.8969

Figure 5.8: Comparison on IMDB dataset

We also evaluate the retain rate and size rate. retain rate defines how much percentage accuracy was retained in knowledge distillation 5.6. The size rate defines how many times the "teacher model" compares with the "student model" 5.7

$$Precision = \frac{studentmodelaccuracy}{teachermodelaccuracy} \tag{5.6}$$

$$Precision = \frac{thenumberof\,\text{“}teachermodel\text{”}parameters}{thenumberof\,\text{“}studentmodel\text{”}parameters} \tag{5.7}$$

Model	accuracy	retain rate	Paramaters	size rate
KD_CNN	0.8610	92.8%	77378	1414
KD_LSTM	0.8914	96.1%	2311170	47.37
KD_LSTM_atten	0.8969	96.7%	2311426	49.5
BERT_linear	0.9275	100%	109483778	1

Figure 5.9: retain rate for IMDB dataset

Model	accuracy	retain rate	Paramaters	size rate
KD_CNN	0.8330	95%	77378	1414
KD_LSTM	0.8568	97.7%	2311170	47.37
KD_LSTM_atten	0.8585	98%	2311426	49.5
BERT_linear	0.8766	100%	109483778	1

Figure 5.10: retain rate for Twitter dataset

As we can see from Figure 5.9 and 5.10, compared to the full-size 12-layer BERT model, our RNN series models, LSTM and LSTM_attention, are nearly 50 times smaller (from 110 to 2.3 million of parameters), and retain 96.1% and 96.7% of performance in IMDB dataset. On the other hand, our RNN series models retain 97.7% and 98% of performance in Twitter dataset. This shows that our model has learned the knowledge of the teacher's model.

Using attention mechanism we can improve performance. A gain of 0.6% is observed between LSTM and LSTM_attention in IMDB dataset and 0.3% in Twitter dataset. The mechanism of attention enhancement

in Twitter data set is not obvious, mainly because the average length is very short. Attention mechanism can reduce the problem of long-sequence forgetting.

In CNN model, we also explored the relationship between the number of filters and accuracy. We find that appropriately increasing the number of filters can improve accuracy. As we can see in Figure 5.11 and 5.12, the best filter (denoted a xxx(y), where y is the filter size) for each window size is 64 in our experiments. In both datasets, with filter size 128, KD_CNN is almost the same as with 64 filters. Different filters have different functionality. Increasing the number of filters will lead to functional saturation. On the other hand, increasing the number of filters is equivalent to increasing the model parameters. It is time-consuming and has the possibility of overfitting.

Model	accuracy	Number of parameters
CNN(32)	83.19%	67778
CNN(64)	85.24%	77378
CNN(128)	85.28%	96578
KD_CNN(32)	83.79%	67778
KD_CNN(64)	86.10%	77378
KD_CNN(128)	86.12%	96578

Figure 5.11: Different filters for IMDB dataset

Model	accuracy	Number of parameters
CNN(32)	80.19%	67778
CNN(64)	82.63%	77378
CNN(128)	82.64%	96578
KD_CNN(32)	80.79%	67778
KD_CNN(64)	83.30%	77378
KD_CNN(128)	83.29%	96578

Figure 5.12: Different filters for Twitter dataset

5.2.2 Execution time

As Figure 5.13 and 5.14 show, the execution time of the model with knowledge distillation is almost the same as that without knowledge distillation, since they have the same number of parameters. The longest execution time model is BERT_GRU model, since it need BERT output "dynamic word embeddings" which are GRU input. The smallest execution time model are CNN and KD_CNN model, since the number of it's parameters is the smallest.

Overall, the execution time on IMDB dataset is longer than that on Twitter dataset, since the average length of IMDB dataset (each samples has approximate 250 tokens) is longer than Twitter dataset (each sample has approximate 17 tokens). The execution time of Bi_LSTM_atten model is longer than Bi_LSTM model, since Bi_LSTM_atten model has

more parameters than Bi_LSTM model.

IMDB	Trainable parameters	Execution Time	Number of samples
BERT_linear	109483778	675s	25000
BERT_GRU	6437890	733s	25000
LSTM	2311170	51s	25000
LSTM_atten	2311426	57s	25000
KD_LSTM	2311170	49s	25000
KD_LSTM_atten	2311426	53s	25000
CNN	77378	21s	25000
KD_CNN	77378	21s	25000

Figure 5.13: Execution time in IMDB

Tiwtter	Trainable parameters	Execution Time	Number of sample
BERT_linear	109483778	345s	160000
BERT_GRU	6437890	356s	160000
LSTM	2311170	14s	160000
LSTM_atten	2311426	20s	160000
KD_LSTM	2311170	14s	160000
KD_LSTM_atten	2311426	19s	160000
CNN	77378	13s	160000
KD_CNN	77378	13s	160000

Figure 5.14: Execution time in Twitter

Chapter 6

Conclusion and Future Work

In conclusion, we proposed to compress the large scale deep learning neural network BERT model using knowledge distillation (KD) technique for deployment on mobile or smaller platforms. The KD technique follows a teacher-student model. In this thesis we use Bi-directional Encoder Representations from Transformers (BERT) as the teacher model for pre-training sentiment analysis tasks. First generation convolution neural networks and long-short term memory with attention mechanism were used as student models.

Our experiments on benchmark datasets indicate that BERT with linear models have better performance than BERT with GRU model. We, therefore, used BERT with linear model as the "teacher" model. The teacher model was overfitting with knowledge distillation if the number of epochs was greater than the best epoch. We also found

that the "student models" through KD had better performance than those models without KD. However, we also observed compared to the "teacher model", "student models" converge slowly.

The student model with KD had approximately 70% improvement in accuracy, precision, recall and F1-score. Compared to the full-size 12- layer BERT model, our RNN series models, LSTM and LSTM attention, was nearly 50 times smaller and had improved performance on datasets. We also showed that the attention mechanism improved performance. In CNN model, we also explored the relationship between the number of filters and accuracy. We found that appropriately increasing the number of filters can improve accuracy. With respect to execution, there was not much difference for the model with or without KD.

6.1 Future Work

This thesis opens an interesting research topic in using "first generation" neural networks to extract knowledge from BERT model for sentiment analysis. There are still a lot of explorations and improvements that can be further investigated.

6.1.1 Improvements

Data Augmentation refers to increase in the diversity of training examples without collecting new data. Data augmentation technologies include, Easy Data Augmentation (EDA), Back-translation, and MIXUP. We could use any of these technologies for futher improvement in our model. EDA is a technique used for synonym replacement. Back-translation is used in machine translation, the main idea is that we can translate target language to source language and mixing both original source sentence and back-translated sentence to train a model. MIXUP, also called Mixed Sample Data Augmentation (MSDA) it uses deep learning model to generate sentence embedding. Each embedding and it's label have their own weight to mix up a new embedding and new label, respectively.

There is still room for improvement in word representations. We could use character-level n-gram (replacing Out of vocabulary representation) by dividing words into sub-word forms.

6.1.2 Exploration of other models

We could also explore KD on the following two models. FastText is a classification model that can speed up training and testing while

maintaining high precision. FastText uses average word embedding of sentences to predict sentence label instead of word itself. RCNN is also a popular model for sentiment analysis [25]. RNN combined with CNN has higher performance than only RNN or CNN based models.

Bibliography

[1] Lei Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2654–2662, Cambridge, MA, USA, 2014. MIT Press.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, pages 1–15, January 2015. 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015.

[3] Christian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, page 535–541, Philadelphia, Pennsylvania, August 2006.

[4] R. Cai, B. Qin, Y. Chen, L. Zhang, R. Yang, S. Chen, and

W. Wang. Sentiment analysis about investors and consumers in energy market based on bert-bilstm. *IEEE Access*, 8:171408–171415, 2020.

[5] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555:1–9, 2014.

[6] Mathieu Cliche. BB_twtr at SemEval-2017 task 4: Twitter sentiment analysis with CNNs and LSTMs. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 573–580, Vancouver, Canada, August 2017.

[7] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1107–1116, Valencia, Spain, April 2017. Association for Computational Linguistics.

[8] Hang Cui, Vibhu Mittal, and Mayur Datar. Comparative experiments on sentiment classification for online product reviews. In

Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2, AAAI'06, page 1265–1270. AAAI Press, 2006.

[9] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[10] Jayati Deshmukh, KM Annervaz, Sanjay Podder, Shubhashis Sengupta, and Neville Dubash. Towards accurate duplicate bug retrieval using deep learning techniques. In *2017 IEEE International conference on software maintenance and evolution (ICSME)*, pages 115–124, Shanghai, China, 2017. IEEE.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[12] J. Dong, F. He, Y. Guo, and H. Zhang. A commodity review senti-

ment analysis based on bert-cnn model. In *2020 5th International Conference on Computer and Communication Systems (ICCCS)*, pages 143–147, Shanghai, China, 2020.

[13] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 1(12):2009, 2009.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, pages 1–9, 2015.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

[17] Long Jiang, Mo Yu, Ming Zhou, Xiaohua Liu, and Tiejun Zhao. Target-dependent twitter sentiment classification. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, pages 151–160, Portland, Oregon, USA, 2011.

[18] Long Jiang, Mo Yu, Ming Zhou, Xiaohua Liu, and Tiejun Zhao.

Target-dependent Twitter sentiment classification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 151–160, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[19] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online, November 2020. Association for Computational Linguistics.

[20] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, Valencia, Spain, April 2017. Association for Computational Linguistics.

[21] Y. Kim and Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, page

1746–1751, Doha, Qatar, 2014. Association for Computational Linguistics.

[22] Svetlana Kiritchenko, Xiaodan Zhu, Colin Cherry, and Saif Mohammad. Nrc-canada-2014: Detecting aspects and sentiment in customer reviews. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 437–442, Dublin, Ireland, 2014.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[24] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 2267–2273. AAAI Press, 2015.

[25] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Proceedings*

of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI'15, page 2267–2273. AAAI Press, 2015.

[26] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020.

[27] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020.

[28] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.

[29] Dan Li and Jiang Qian. Text sentiment analysis based on long short-term memory. In *2016 First IEEE International Conference*

on Computer Communication and the Internet (ICCCI), pages 471–475, 2016.

[30] R. Liu, Y. Shi, C. Ji, and M. Jia. A survey of sentiment analysis based on transfer learning. *IEEE Access*, 7:85401–85412, 2019.

[31] 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.

[32] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011.

[33] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. *arXiv preprint arXiv:1708.00107*, 2017.

[34] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781:1–12, 2013.

[35] M. Moh, A. Gajjala, S. C. R. Gangireddy, and T. Moh. On multi-tier sentiment analysis using supervised machine learning. In *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 1, pages 341–344, Singapore, Singapore, 2015.

[36] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. *arXiv preprint cs/0205070*, pages 79–86, July 2002.

[37] 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.

[38] Matthew 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.

[39] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[40] Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation - J DOC*, 60:503–520, 10 2004.

[41] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[42] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108:1–5, 2019.

[43] M. Shi, K. Wang, and C. Li. A c-lstm with word embedding model for news text classification. In *2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS)*, pages 253–257, Beijing, China, 2019.

[44] X. Shi and R. Lu. Attention-based bidirectional hierarchical lstm networks for text semantic classification. In *2019 10th International*

Conference on Information Technology in Medicine and Education (ITME), pages 618–622, Qingdao, China, 2019.

[45] Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling task-specific knowledge from BERT into simple neural networks. *CoRR*, abs/1903.12136:1–8, 2019.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[47] Michel Verleysen and Damien François. The curse of dimensionality in data mining and time series prediction. In *ACL*, IWANN'05, page 758–770, Berlin, Heidelberg, 2005. Springer-Verlag.

[48] B. Wang, F. Miao, X. Wang, and L. Jin. Text classification using a bidirectional recurrent neural network with an attention mechanism. In *2020 International Conference on Culture-oriented Science Technology (ICCST)*, pages 265–268, Beijing, China, 2020.

[49] Y. Zhang and Z. Rao. Deep neural networks with pre-train model

bert for aspect-level sentiments classification. In *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, pages 923–927, Chongqing, China, 2020.

[50] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1-4):43–52, 2010.

[51] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. Ernie: Enhanced language representation with informative entities, 2019.

[52] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. A C-LSTM neural network for text classification. *CoRR*, abs/1511.08630:1–10, 2015.

[53] Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National science review*, 5(1):44–53, 2018.

[54] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

[55] Muhammad Zulqarnain, Rozaida Ghazali, Muhammad Ghulam

Ghouse, and Muhammad Faheem Mushtaq. Efficient processing of gru based on word embedding for text classification. *JOIV: International Journal on Informatics Visualization*, 3(4):377–383, 2019.