Multi-label Classification and Sentiment Analysis on Textual Records

MULTI-LABEL CLASSIFICATION AND SENTIMENT ANALYSIS ON TEXTUAL RECORDS

BY

XINTONG GUO, B.Eng.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2019) McMaster University

(Electrical & Computer Engineering) Hamilton, Ontario, Canada

TITLE:	Multi-label Classification and Sentiment Analysis on Textual Records
AUTHOR:	Xintong Guo B.Eng., (Automation Science and Electrical Engineering) Beihang University, Beijing, China
SUPERVISOR:	Dr. Jun Chen
NUMBER OF PAGES:	xi, 59

To my dear parents and beloved wife

Abstract

In this thesis we have present effective approaches for two classic Nature Language Processing tasks: Multi-label Text Classification(MLTC) and Sentiment Analysis(SA) based on two datasets.

For MLTC, a robust deep learning approach based on convolution neural network(CNN) has been introduced. We have done this on almost one million records with a related label list consists of 20 labels. We have divided our data set into three parts, training set, validation set and test set. Our CNN based model achieved great result measured in F1 score.

For SA, data set was more informative and well-structured compared with MLTC. A traditional word embedding method, Word2Vec was used for generating word vector of each text records. Following that, we employed several classic deep learning models such as Bi-LSTM, RCNN, Attention mechanism and CNN to extract sentiment features. In the next step, a classification frame was designed to graded. At last, the start-of-art language model, BERT which use transfer learning method was employed.

In conclusion, we compared performance of RNN-based model, CNN-based model and pre-trained language model on classification task and discuss their applicability.

Acknowledgements

I wish to express my gratitude to all people who have helped me to complete this thesis. First and foremost, I would like to express my sincere appreciation to my supervisor Dr. Jun Chen, not only for the initial and ongoing support for this thesis, but also for the enthusiastic support, patience, encouragement and guidance during my Master program.

Furthermore, I would like to thank Dr. Sorina Dumitrescu and Dr. Jiankang Zhang for being members of my defence committee. I feel grateful for their time reviewing my paper and providing valuable feedback.

Last but not least, I am also grateful for the advice and assistance of Mr. Zichao Zhao and Mr. Yexin Cui, which in keeping my progress on schedule.

Abbreviations and Notation

Bi-LSTM	Bidirectional Long Short-Term Memory
BERT	Bidirectional Encoder Representations from Transformers
CBOW	Continnuous Bag of Words
CNN	Convolutional Neural Network
GPU	Graphics Processing Unit
GRU	Gated Recurrent Units
KNN	K-Nearest Neighbor
LSTM	Long Short-Term Memory
MCC	Multi-class Classification
MLC	Multi-label Classification
MLTC	Multi-label Text Classification
MSE	Mean Square Error
NLP	Natural Language Processing

RCNN	Recurrent Convolutional Neural Network
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SA	Sentiment Analysis
SVM	Support Vector Machine

Contents

List of Figures

Chapter 1

Introduction and Motivation

1.1 Introduction

Given one or several sentences, Multi-Label Text Classification(MLTC) is a complex natural language processing task which requires to predict multiple labels related to one instance. First of all, To describe the MLTC task we would like to define some notations, assuming that we have a label list with **a** labels in total, $\mathcal{Y} = \{Y_1, Y_2, Y_3, \cdots, Y_a\}$, MLTC is asked to assign a subset *y* containing *n* labels in the label space $\mathcal{Y}$ to instance *x*. Unlike traditional Single-Label Classification that each sample has only one label, each instance in MLTC task may have several related labels. The MLTC task can be modelled as finding a label list subset y^* that maximizes the conditional probability $p(y|x)$:

$$p(y|x) = \prod_{i=1}^{n} p(y_i|y_1, y_2, y_3, \cdots, y_{i\text{-}1}, x) \tag{1.1}$$

where y_i is the next predict label to a sequence x, and $y_1, y_2, \cdots, y_{i-1}$ are predicted labels.

There is a baseline model called fastText(Joulin *et al.* (2016)). This model's advantages are fast and simple, it can train large amount of data in a few minutes. Because this method uses n-gram features as its input, it is able to capture enough features of text sentence. As for the performance, this model is good in some cases, however, according to many experiments, it turns out that this model is not robust enough for modelling complex problem. To build a stronger model, we explore convolutional neural network based model(TextCNN), this model takes advantage of the feature extraction ability of CNN and focus on emphasizing key features of an instance. Results shows that it exceeded the baseline model in a large margin, and achieved great result on our MLTC task .

Based on previous MLTC work, we further construct a new semantic complex data set for Sentiment Analysis. SA is aiming at identifying and categorizing the sentiment expressed by an author in text, normally it can be transfer to a Single-label Classification task. SA has a wide range of applications in industry, such as forecasting market trend based on sentiment comment in social media.

Now most of the SA works are focusing on speech processing, our target is to develop a sentiment degree classification model analyzing textual data. Our model are consists of three main parts, word embedding, feature extraction and classification. We firstly use a traditional word embedding method, Word2Vec to generate word vector. Second, to get features from an instance, our baseline method is the Bi-LSTM, it is a recurrent neural networks model that connects two hidden layers of opposite directions to the same output, so that the last output layer can get information from past (backwards) and future (forward) states in the same time. However, this method is extremely slow to train and not suitable for our data set.

To optimize baseline model, we implement several methods. First, we deploy Recurrent Convolutional Neural Network(RCNN) to take advantage of both CNN and RNN; Then we use Attention mechanism to optimize Bi-LSTM, Attention enables the recurrent neural network to focus on relevant parts of the input more than those irrelevant parts. We also implement convolutional neural network based model(TextCNN), compared with recurrent neural networks, it based on max-pooling to get the most important information from a text. Finally we implement a language model: Bidirectional Encoder Representations from Transformers(BERT). BERTs key innovation is applying the bidirectional training of Transformer, which is a popular Attention-based model, to language modelling. This method is in contrast to previous works which looked at a text sequence either from left to right or reverse.

1.2 Motivation

The motivation of this paper is to effectively solve limited-information text classification problem. Our data set is an auto repair records data set which was originally generated by ourselves. The data set contains almost 1 million records, where many of them are just repair orders, short descriptions and a few key words. This data set has two main characteristics: 1.Most instance are short text data which including less than 30 words. 2.Each label in the label list is highly correlative to some other labels for both MLTC and SA tasks.

Both Multi-Label Text Classification and Sentiment Analysis for short texts records are challenging because of the limited contextual information and semantic diversity which can lead to interference building language model. To build a robust and effective model, an intuition concept is to combine limited text content with prior technical knowledge by

using more informative word embedding methods and implementing more powerful feature extraction method to represent information-limited data.

In this paper, we compared result from each model and it shows that convolutional neural network is the most suitable feature extraction model for key words contributing data set, further we discussed the contribution between key words and contextual information in a practical text classification problem.

1.3 Thesis Structure

The rest of this paper is arranged as follows:

In Chapter 2, the background of Multi-label text classification problem and Sentiment Analysis problem are presented.

Next, in Chapter 3, word embedding method and several deep learning models of neural network that related to proposed method are introduced.

Then, in Chapter 4, the proposed multi label classification and sentiment analysis classification structures have been explained in detail, experiments results in numeric comparisons have presented in table.

Finally, in Chapter 5, main contribution of our work have been summarized, after that we make a conclusion and take a forecast in future works.

Chapter 2

Problem Background

2.1 Multi-label Text Classification

Multi-label Text Classification (MLTC) is an important task in the field of natural language processing (NLP), which can be applied in many real-world scenarios, such as information retrieval(Gopal and Yang (2010)), tag recommendation (Katakis *et al.* (2008)), and so on. The target of the MLTC task is to assign multiple labels to each sample in the data set. The difference between Multi-class Classification and Multi-label Classification is the number of labels that can be assigned to one instance.

Consider an example of three classes $C = [Sun, Moon, Cloud]$. In Multi-class case, each sample belongs to only one of C classes, while in Multi-label case, more than one class in C can associate with a sentence, Fig2.1 shows a scenario to compare MCC and MLC. Normally, for a Multi-label Classification problem, the data set is more complicated to solve. Currently, there are two groups of methods to solve this type of problem: Problem Transformation and Algorithm Adaptation.(Zhang and Zhou (2014))

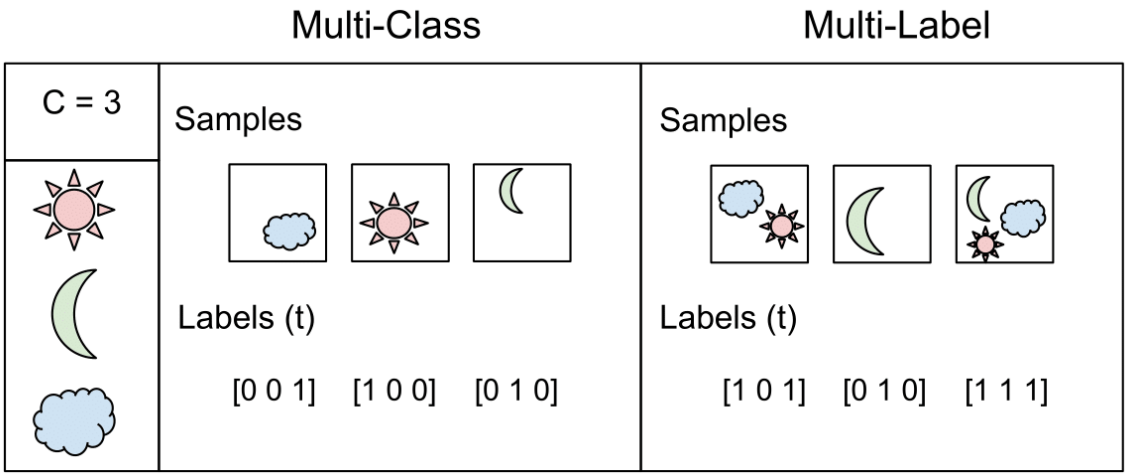

Figure 2.1: MCC and MLC(source:$gombru.github.io$)

The first attempt of Problem Transformation techniques is transforming the Multi-label Classification problem into one or more Single-label Classification problems. One way to accomplish this task is called Binary relevance (BR)(Boutell *et al.* (2004)) which will create an independent binary classifier for each label of the problem, this is one of the earliest attempts to solve the MLTC task by transforming concept. The main problem of the BR method is that it neglects the correlations between labels, thus ignoring some characteristics of the problem.

One way to solve this is through the Classifier chains (CC) method(Read *et al.* (2011)), The CC method uses the output of a binary classifier as an input attribute to the next classifier, hence adding relationships among classes in a problem. However, the main flaw of this method is the difficulty of getting the best order of the classifier, and experiments show that it is computationally expensive processing large data sets.

Another solution to MLTC task is algorithm adaptation method. The concept is trying to work directly with a Multi-label problem by using data mining algorithms. One optimization is a Multi-label lazy learning approach named ML-KNN (Zhang and Zhou (2007)), in this work, the authors perform an adaptation of the KNN algorithm to allow the use of Multi-label data. In the first step, all the k nearest neighbours of each instance are identified, after identification, statistical information obtained from the neighbours label sets is used to determine labels.

However, just as other similar methods such as ML-DT (Clare and King (2001)), Rank-SVM(Elisseeff and Weston (2002)), they are only able to capture label correlations within the first or second order, and computationally expensive as well.

In recent years, neural networks have achieved great success in many fields including NLP. Some neural network models have also been applied in the MLTC task and achieved important progress. For instance, (Zhang and Zhou (2006)) used fully connected neural network with pairwise ranking loss function to tackle the problem. (Kim (2014)) is the first to implement convolutional neural network into Sentence-Level Classification tasks, (Kurata *et al.* (2016)) introduced CNN to Multi-label Classification. (Chen *et al.* (2017)) use CNN and RNN to capture the semantic information of texts. (Yang *et al.* (2016)) purposed Hierarchical Attention Networks(HAN) to solve Document-level Classification task, theis hierarchical structure has two levels of Attention mechanisms used at the word and sentence-level, it purposed an idea of capturing important information in different levels, which is useful especially in scenario that each instance has many sentences.

2.2 Sentiment Analysis

The growth of using social networks, such as Twitter, Amazon, and Instagram, has proved that nowadays the way people expressing their opinions and feelings about services and products are influencing our life. Sentiment Analysis is an important growing task(Liu (2015)), whose goal is to classify sentiments degree expressed in text which will help industry and companies optimize their products.

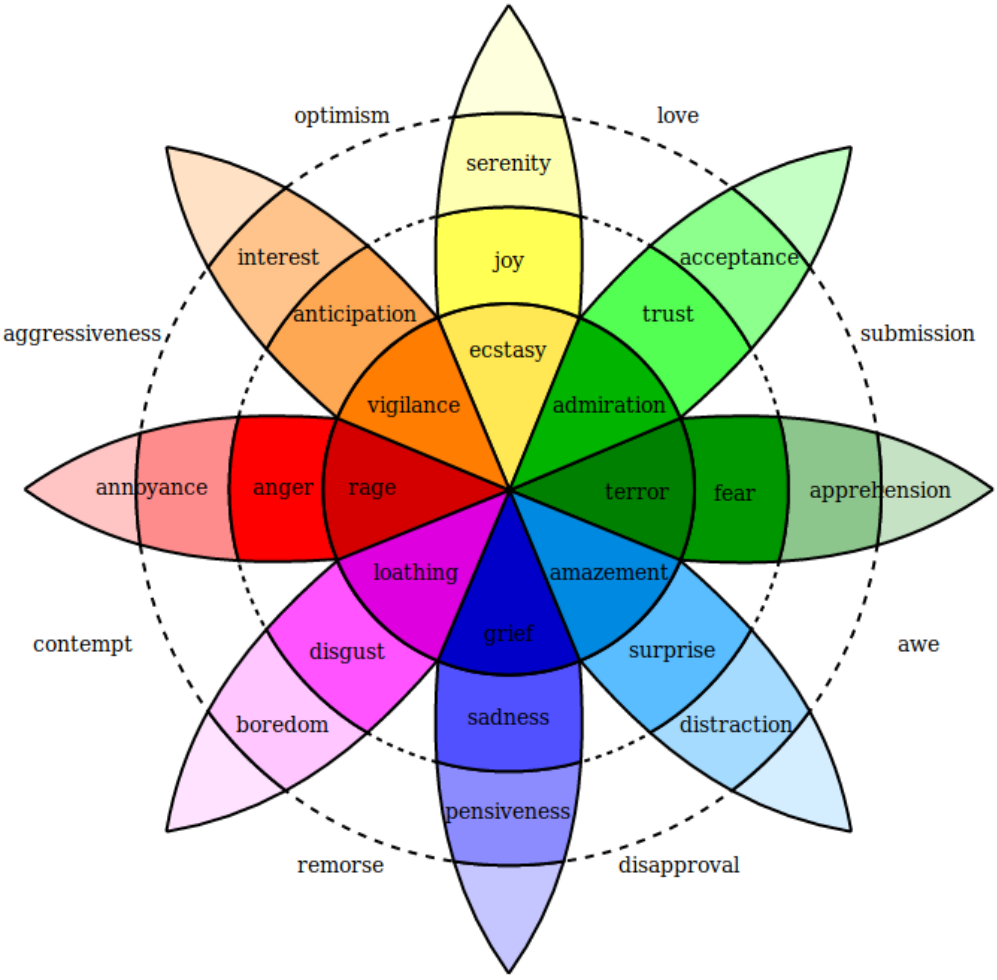

Figure 2.2: Sentiment Class Wheel(source:$Plutchikswheelofemotions$)

The dominant approaches in sentiment analysis are based on machine learning techniques(Wang and Manning (2012)). The overall structure of machine learning solutions are consists of three steps: extracting features from text, estimating relevant features and selecting an appropriate classification algorithm.

Traditional methods use the Bag of Words (BOW) model which is a simple and efficient way to map a whole text instance to a feature vector, then classify it by machine learning algorithms. However, some language information will lost such as word order and grammar structures. Therefore, higher order features have been exploited to enrich the information extraction of a sentence, such as n-grams(Pak and Paroubek (2010)). Prior information about sentiment can be added as addition information in the analysis. For instance, Sentiment lexicons(Cambria (2016)) is added to the features as a supplement of subjective sentiment knowledge. Nevertheless, manual feature engineering is the dominant part of traditional approaches, which is very time consuming.

Recently, Deep learning techniques for Sentiment Analysis have become very popular. These methods utilize automatic feature extraction with better representation and can provide better performance than traditional methods(Pang *et al.* (2008), Collobert *et al.* (2011)). The main idea of deep learning techniques is to learn complex features extracted from data using deep neural networks(Bengio *et al.* (2009)), without pass manually crafted features, deep neural networks automatically learn new complex features through training. While, deep learning approaches have an important character that large amounts of data is required to perform well.

It has been proved that representations of words as vectors is an effective technique in many NLP tasks, including Sentiment Analysis(Tang *et al.* (2014b)). Word2Vec is one of the most popular approaches that allows modelling words as vectors(Mikolov *et al.* (2013b)). The backbone model are Skip-gram and Continnuous Bag of Words(CBOW) to perform the computation of the distributed representations. CBOW aims to predict a word given its context while Skip-gram predicts the context given a word. A very large data set is needed to build Word2Vec model, which computes continuous vector representations of words form rich corpus. These word-level embeddings are encoded by column vectors in an word embedding matrix W:

$$W \in \mathbf{R}^{d \times V} \tag{2.1}$$

where V is the size of vocabulary, d is the embedding size.

Each column in $W_i \in \mathbf{R}^d$ corresponds to a word embedding vector for the i-th word in the vocabulary. The transformation of a word w into its word embedding vector r_w is made by using the matrix-vector product:

$$r_w = W v_w \tag{2.2}$$

where v_w is the one-hot vector of size $|V|$ which just has value index at word w. The matrix parameters of W will be learned in training, and we need to manually select the dimension of the word vectors. In sentiment classification, experiments(Zhang *et al.* (2015)) shows that Word2Vec is a better way to generate vector representations than traditional classifier such as logistic regression(Hosmer Jr *et al.* (2013)). An additional method in word embedding learning is the auto-encoder(Chen *et al.* (2014)), which has been used for many kind

of machine learning tasks, especially distorted data.

Another import approach in Sentiment Analysis is to enrich the embedding vectors with other sources of information, especially the sentiment specific embedding information can be useful(Tang *et al.* (2014a)). (Severyn and Moschitti (2015)) use supervised data to refine the parameters of the unsupervised neural network model. (Kim *et al.* (2013))use a collaborative filtering algorithm to add sentiment information from a small subset of the data. (Li *et al.* (2015)) introduce recursive neural network (RNN) in parallel to another neural network architecture into sentiment analysis. In general, adding additional information to the word embedding generated by deep learning networks has been more and more popular.

As shown by(Socher *et al.* (2013)), compositionality in the sentiment classification task has proven to be relevant, and become another trend in SA. This work proposes the Recursive Neural Tensor Network (RNTN) model which represents a phrase using word vectors and a parse tree, it use tensor-based composition function to generate vectors for higher level nodes in the parse tree.

Another successful attempt in ensemble schemes, is presented by (Mesnil *et al.* (2014)) they used a mean computation rule to combine three sentiment models: a weighted BOW, a language model and continuous representations of sentences model. That ensemble exhibits shows better result in sentiment classification of movie reviews. More works by (Araque *et al.* (2017)) try to combine existing sentiment classifiers, including traditional model, sentiment-based word embedding and manually selected features, and improve the result in social application Sentiment Analysis.

Chapter 3

Deep learning Related Work

3.1 Word Embedding Method in NLP

3.1.1 Word2Vec

Word embedding is the most popular representation of words, it can capture the semantic and syntactic meaning from word vocabulary. Before that, if we want to embed a sentence x which have 6 words. First we need to construct a vocabulary *V* that have each words in the sentence, $V = \{w1, w2, w3, w4, w5, w6\}$. Then we need to create a one-hot encoded vector for each of these words in V. Length of these vector would be equal to $|V|$ which is the size of vocabulary, then each word have their encoding: $w1 = [1, 0, 0, 0, 0, 0]$, $w2 = [0, 1, 0, 0, 0, 0]$,...etc. In this one-hot vocabulary embedding method, We can think of a six dimensional space, where each word occupies one of the dimensions and does not have any relation to other words, which is not true. We want to let those words which have similar context place at closer positions in the vector space. Mathematically, according to cosine similarity, the cosine value of the angle between similar context word vectors should be

close to 1, and then comes the idea of generating distributed representations.

Word2Vec(Mikolov *et al.* (2013a)) is a two-layer neural networks that are trained to reconstruct representations of words called "word embedding". Word2Vec takes a large corpus of text as its input and produces a vector space, which can be set as hundred of dimensions. Each word in the corpus have a unique vector in the space.

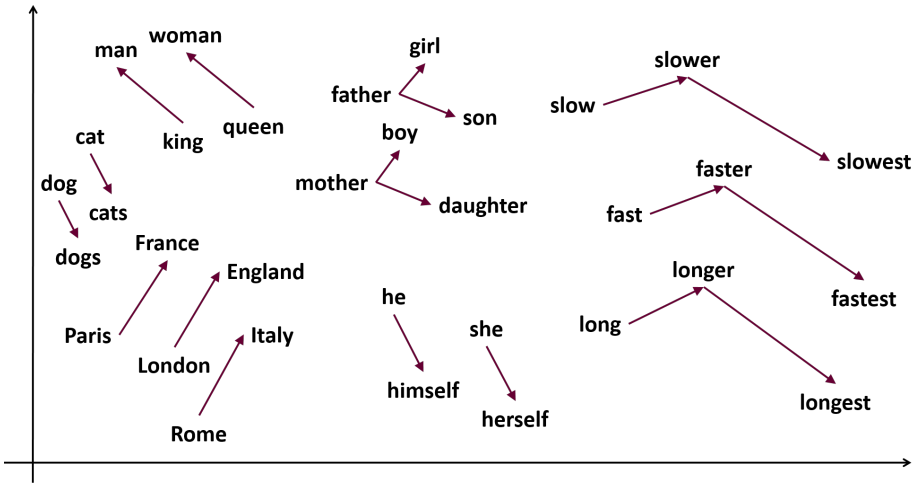

Figure 3.1: Word Vectors in Vector Space(source:$/samyzaf.com/ML/nlp$)

As shown in Fig3.1, due to the training process, words which have similar context meaning are positioned closely in the vector space, and because of the vector representation, a word can be formed by addition of other words. Here is a common example to represent word by linear translations, the result of a vector calculation: : Assuming that W_s is the word space, R^n is an n-dimensional vector space. Now Let ψ be the word embedding mapping from $W_s \rightarrow R^n$, we will have: $\psi('king') - \psi('man') + \psi('woman')$ is closer to $\psi('queen')$ than to any other word vector.(Mikolov *et al.* (2013b))

To construct Word2Vec model there are two methods: Skip-Gram and Common Bag of Words(Mikolov *et al.* (2013c)). The goal of CBOW is predicting word from a window of its surrounding context, CBOW assumes that the order of would in a sentence not influence prediction. In the contrast, the Skip-Gram uses a given word to predict the surrounding window of other words. Both of these two methods have their own advantages and disadvantages. According to the author, Skip-Gram works better with small amount of data and represent rare words well, but CBOW is faster in most of time especially in large data set which would be more industrial friendly.

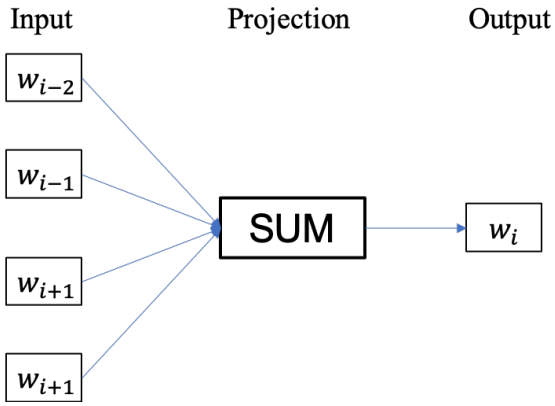

Figure 3.2: Architecture of Continuous Bag of Words

As shown in Fig3.2, In the CBOW, the neural network has multiple context words w as input, and we have multiple input layers sharing the same weights W, which are connecting to the hidden layers. The output w_i from the hidden layers is the average of the weighted inputs for all the surrounding context words.

As shown in Fig3.3, the Skip-gram model predict the surrounding words from the target word w_i, which is inverse of CBOW. In a training instance, there is only one embedding representation of input word been computed by the input weight matrix, but there are multiple output context words for each input. In this case we assume that we have multiple

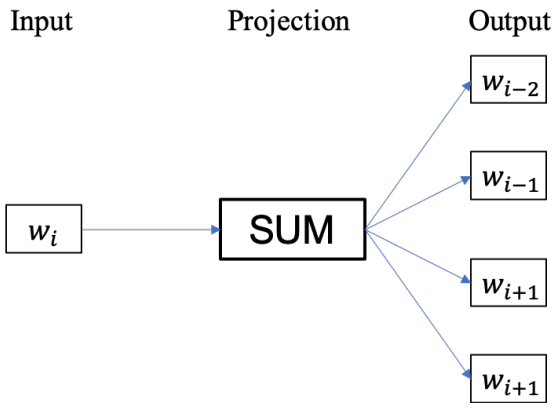

Figure 3.3: Architecture of Skip-Gram Model

output layers and all of them sharing the same set of weights W, the result is a score for each word in the vocabulary, which is the probability of the word being a surrounding word.

Take Skip-gram model as an example, The training goal is to find word representations that are useful for predicting the surrounding words in a sentence. Mathematically, given a sequence of training words $w_1, w_2, w_3, w_4, ...w_L$, the objective of the Skip-gram model is to maximize the average log probability:

$$\frac{1}{L}\sum_{l=1}^{L}\sum_{-k\leq j\leq k, j\neq 0}\log p(w_{l+j}|w_l) \tag{3.1}$$

where k is the window size of training corpus, here can be seen as a function of centre word w_i. Skip-gram formulation defines $p(w_{l+j}|w_l)$ by using the softmax function.

To make the algorithm computationally efficient, Negative Sampling(Goldberg and Levy (2014)) is used to optimize training process, we will not get into details in this paper.

3.1.2 Transformer

As we have discussed above, Word2Vec helps us to build a model that can translate words into numeric form and then perform some algorithms on it. However, it is not enough to just capture the static word level information of natural language, we also want extract contextualized meaning from text in certain scenario and avoid confusion, consider a simple example of polysemous:

I like *play* basketball.

What an amazing *play*!

Note that the word *play* has a different semantic meaning in each sentence. However, before with Word2Vec as embedding method, the word *play* has a fixed vector representation no matter in what sentence. Now with a contextualized language model, the embedding of the word *play* would have a different vector representation which makes it more powerful for NLP tasks.

More advanced model has been presented by Google, called Transformer(Vaswani *et al.* (2017)), which has more advantages than the conventional sequential models (LSTM and GRU). Transformer has a Sequence-to-Sequence architecture, which consist of an Encoder and a Decoder, it can transform a sequence of words into another. The output sequence can be in another language, or just a symbol. The author proved that an architecture with only attention-mechanism, no any RNN units, is even better. Below is the overall architecture of Transformer.

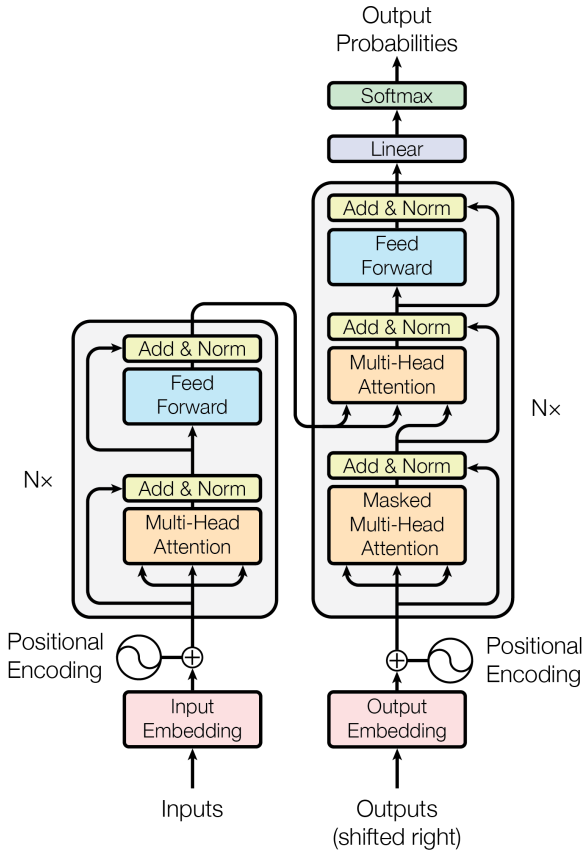

Figure 3.4: Architecture of Transformer(source:Vaswani *et al.* (2017))

The Encoder is on the left and the Decoder is on the right. Both of them are composed of modules that can be stacked on top of each other multiple times, which is described by Nx in the figure. Multi-Head Attention and Feed Forward layers are the two main part of this module. Get into detail of the special attention based model, there are two novel attention bricks, In the Scaled Dot-Product Attention model, output can be described by the following equation:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \tag{3.2}$$

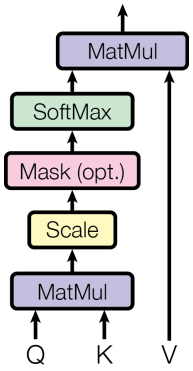

Figure 3.6: (a).Scaled Dot-Product Attention

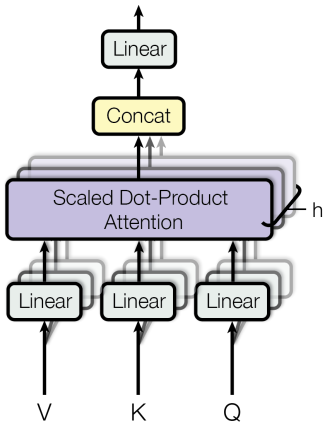

Figure 3.7: (b).Multi-Head Attention(source:Vaswani *et al.* (2017))

where Q is a matrix that contains the vector representation of one word in the sequence, K are the keys for representations of all the words in the sequence, V are the values related to K. Additionally, the softmax function is applied in order to make the output a distribution between 0 and 1.

The Multi-Head Attention model describes how this attention-mechanism can be parallelized into multiple mechanisms that can be used side by side. The attention mechanism is repeated multiple times with linear projections of Q, K, V, This allows the system to learn different representations of Q, K, V. After that Q, K, V and weight matrices W are multiplied to get the output vector. Weight matrices W are learned during training process.

The Multi-Head Attention module that connects the encoder and decoder will make sure that the input information from both encoder and decoder are taken into account. After that, the feed-forward network has identical parameters for each position according to a given sequence.

3.2 TextCNN

Convolutional neural network(Krizhevsky *et al.* (2012)) is the main building box to solve computer vision problems. We would like to explore convolutional neural network based model to solve the MLTC problem. For a traditional CNN model, assuming that we have N training data, denoted as $\{(x_i, y_i)\}^N$, where each x_i are related to a y_i, it's goal is to minimize:

$$\arg\min_{\theta} \mathcal{L}(\theta) + \Phi(\theta) \tag{3.3}$$

where

$$\mathcal{L}(\theta) = \sum_{k=1}^{N} \boldsymbol{L}(y_i, f(x_i, \theta)). \tag{3.4}$$

Here $f(\cdot)$ is the activation function, $\boldsymbol{L}(\cdot)$ is a loss function, which is used to measure the difference between predicted value $f(x_i, \theta)$ and ground truth y_i. In our MLTC task we use cross entropy loss, the $\Phi(\theta)$ is the regularization term. The training procedure can be seen as to find a best value θ that can map the input data to output label.

Now we will show how CNN can be used for text classification. The author (Kim (2014)) simply trained a CNN with only one layer of convolution based on word vectors obtained from Word2Vec, the language data sets are around 100 billion words of Google News. In our work, we did not just keep the initial word vectors static, we have fine-tuning the word embedding model in our specific task. It is proved(Sharif Razavian *et al.* (2014)) that CNN can perform well when obtain feature extractors from a pre-trained deep learning model and then apply on a variety of tasks that are very different from the original task. Following are the TextCNN processing steps.

The convolutional neural network based classification model are consists of four parts:

The first step is word embedding; each sentence length is different at beginning, in order to operate them together, we use padding mechanism to get fixed length l for each sentence. As for each word/token in the sentence, then we use word embedding method to get vectors with fixed embedding size d. Now, our input is a matrix with two dimension(l, d), the input format for our text classification problem is similar with image classification.

Secondly, we will do convolution to our embedded input. It is an element-wise multiply between a filter and input. For filters, the first dimension is their size f, the second dimension is the word embedding size d. We use k number of filters, each filter is a 2-dimension matrix (f, d). Now the output will be k lists, each list has a length of $l - f + 1$, each element of the list is a scalar. To acquire rich features from input, several different size of filters are used.

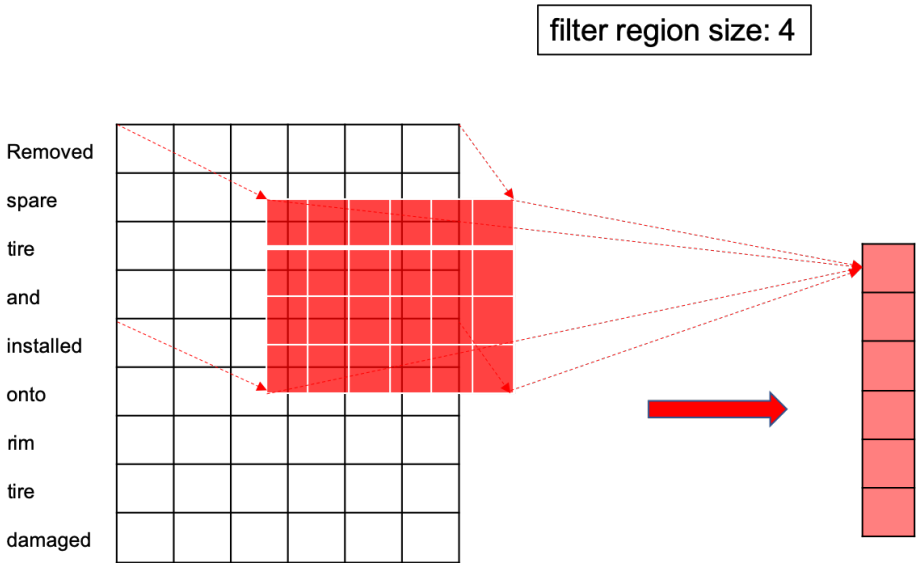

Figure 3.7: A Convolution Instance

After that, we will do max pooling for the output of convolutional operation. For k lists, we will get k number of scalars, they are in different colours in the whole diagram as shown in 3.8, implicating that they are computed from different filters. Concatenating all scalars, we can form the final features of the input instance, it is a fixed-size vector, which is not depend on the size of filters we use.

Finally, we use fully connected layer to project these features to per-defined labels.

The whole processes can be shown in diagram below:

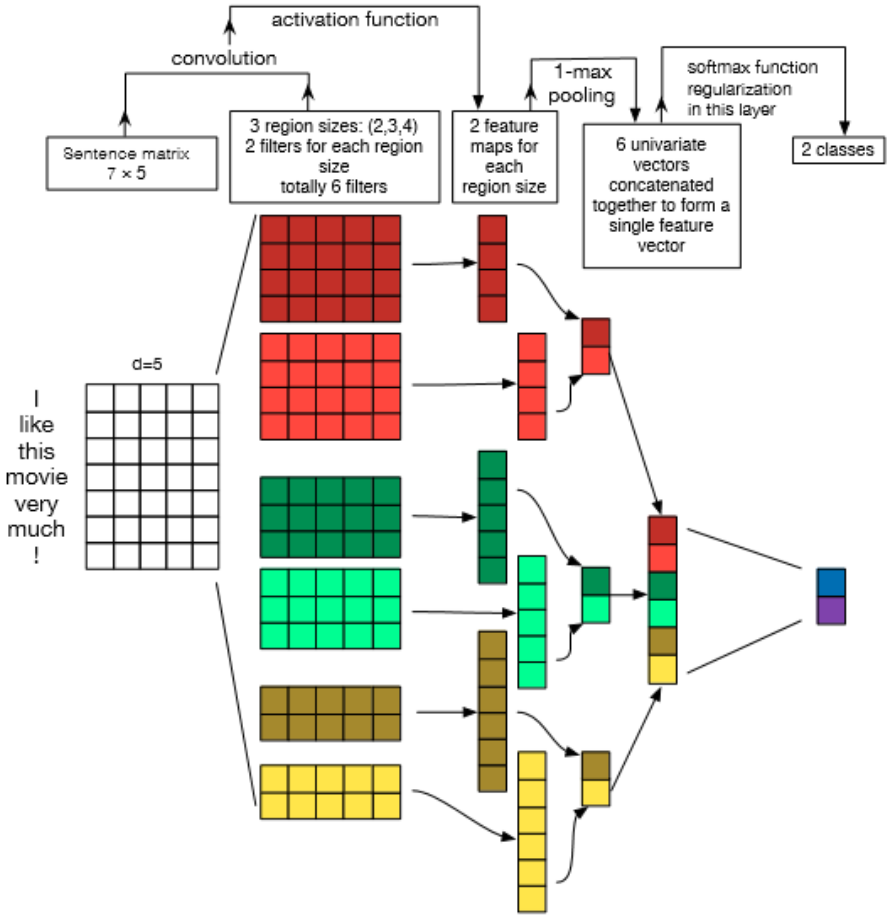

Figure 3.8: Diagram of TextCNN(source:Zhang and Wallace (2015))

As for the instance shown in this diagram, we have an input:

I like this movie very much !

whose length is 7, embedding dimension is 5. We use 3 different sizes:(2,3,4), two filters for each region size, so in total, we have 6 filters. After convolution between input and filters, it result in 6 lists, length range from 4 to 6. Then the result of max-pooling are 6 scalars. We concatenate them to get a fixed size vector which consists of all important features of the input. Finally we use linear layer to project this vector to the label set, and get the possibility distribution of labels.

3.3 Bi-LSTM

RNN has the ability of preserve sequence information over time, that's why it can be applied in many NLP tasks. First proposed by(Hochreiter and Schmidhuber (1997)), LSTM has the ability to overcome gradient vanishing problem of recurrent neural network, it has an adaptive gating mechanism, which can take a balance between the previous state and features extracted from the current input. Given an instance x whose length is l, $x = \{w_1, w_2, w_3, ..., w_l\}$, At time step t, hidden state h_t and current state c_t will be updated by following rules:

$$\begin{bmatrix} f_t \\ o_t \\ i_t \\ \hat{c}_t \end{bmatrix} = \begin{bmatrix} \alpha \\ \alpha \\ \alpha \\ \tanh \end{bmatrix} W \odot [h_{t-1}, x_t] \tag{3.5}$$

and

$$c_t = f_t \odot c_{t-1} + i_t \odot c_t \tag{3.6}$$

$$h_t = o_t \odot \tanh(c_t) \tag{3.7}$$

where f, o, i are the forget gate activation, output gate activation and input gate activation respectively, $\hat{c}$ denotes the current state, α denotes sigmoid activation function, $\odot$ denotes computing Hadamard product, *W* term denote weight matrices and x_t is the input at the current time-step.

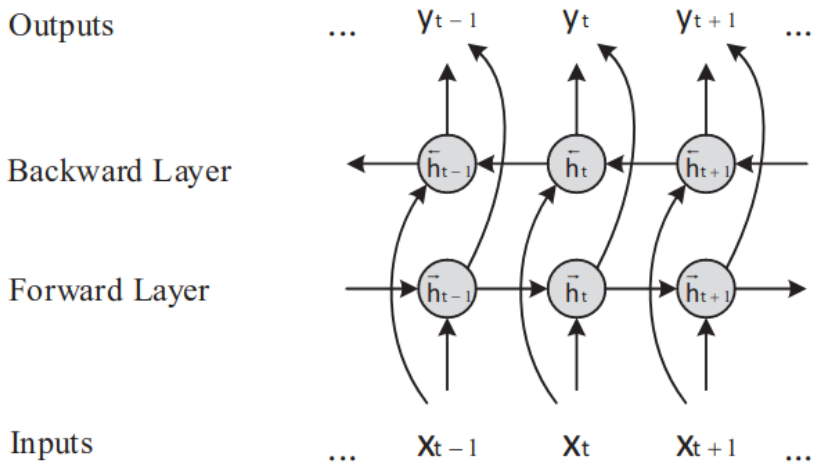

Figure 3.9: Bidirectional LSTM Model(source:Zhang and Ma (2017))

For sequence modelling tasks, to obtain future as well as past context(Zhou *et al.* (2016b)) is really helpful. Apparently, as we illustrated above, LSTM networks process sequences in temporal order, which will ignore future context. Bi-LSTM successfully solved

this problem, as shown in Fig3.9, the networks extend normal LSTM networks by adding a second hidden layer, where the $hidden - hidden$ connections can flow in opposite temporal order. Therefore, this model is able to exploit information from both the past and the future.

For our sentiment analysis task, in order to capture the long-range context information of a sentence to analysis sentiment degree, we use Bi-LSTM to computes the forward hidden sequence $\overrightarrow{h_i}$ and the backward hidden sequence $\overleftarrow{h_i}$. The network have two sub-layers for the forward and backward sequence context respectively. Finally, The output of the i^{th} word is shown in the following equation:

$$h_i = \left[\overrightarrow{h_i} \oplus \overleftarrow{h_i}\right] \tag{3.8}$$

where $\oplus$ denotes the element-wise sum.

3.4 RCNN

Purposed by (Lai *et al.* (2015)), Recurrent Convolutional Neural Network(RCNN) combines both the advantage of recurrent structure that can capture contextual information as far as possible and the outstanding features of max-pooling layer that can automatically take those parts of input that play key roles in a classification task. The reason why we do not use other types of pooling layer such as average pooling layers(Collobert *et al.* (2011)) is that for our data set, we estimate that only some key words have important contribution to classify different instance into different label, which is agree with the concept of max pooling.

The problem with RNN are the influence by bias and greater influence by recent inputs which will result in error. As for CNN, it can extract important features from max pooling layer while neglecting long range contextual information. The core of RCNN is creating a special word representation $y^{(1)}$ that consists of the left side context, word embedding, and right side context in the word embedding part, then use max pooling layer before softmax layer. The Left context is constructed from a forward RNN, and the right context is structure from a reverse RNN. The structure of RCNN can be shown in a graph:

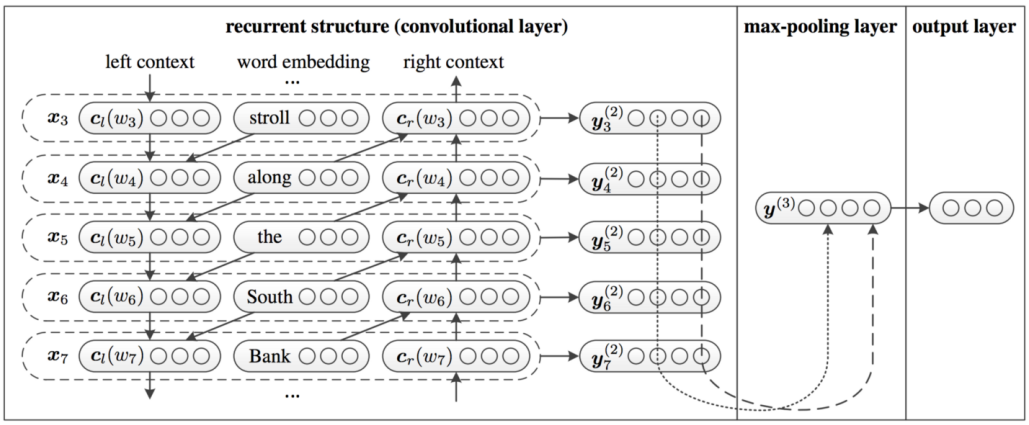

Figure 3.10: The Structure of RCNN(source:Lai *et al.* (2015))

$c_l(w_i)$ is the left context of word w_i, $c_r(w_i)$ is the right context of word w_i, they are both computed from weight matrix and word embedding. Take $c_l(w_i)$ as an instance, it is computed from following equation:

$$c_l(w_i) = f(W_{(l)} \cdot c_l(w_{i-1}) + W_{(sl)} \cdot e(w_{i-1})) \tag{3.9}$$

where f is a non-linear activation function, $W_{(l)}$ is a matrix that transforms the former hidden layer into the latter. $W_{(sl)}$ is a matrix used to combine the semantic of the current

word with the next words left context. $c_l(w_{i-1})$ is the left-side context of the previous word w_{i-1}, $e(w_{i-1})$ is the word embedding of word. The right-side context $c_r(w_i)$ is computed in the same way. After getting the left-side and right-side context vector $c_l(w_i)$ and $c_r(w_i)$, we concatenate them with the word embedding $e(w_i)$ as the word representation $y_i^{(1)}$:

$$y_i^{(1)} = [c_l(w_i); e(w_i); c_r(w_i)]. \tag{3.10}$$

Then we apply a linear transformation together with the tanh activation function on the representation of word w_i:

$$y_i^{(2)} = \tanh(W \cdot y_i^{(1)} + b). \tag{3.11}$$

The convolutional neural network in this model is designed to represent the text, the recurrent structure we previously mentioned is the convolutional layer.

When all the representation of words are calculated, the max pooling layer will apply element-wise max function on $y_i^{(2)}$, so that the k-th element of $y^{(3)}$ is the maximum value in the k-th elements of $y_i^{(2)}$:

$$y^{(3)} = \max_{i=1}^{n} y_i^{(2)}. \tag{3.12}$$

The final layer output $y^{(4)}$ is calculated through a linear layer with $W \cdot y^{(3)} + b$. Finally, a softmax activation function is applied to convert the output numbers into probabilities:

$$p_i = \frac{\exp(y_i^{(4)})}{\sum_{k=1}^{n} \exp(y_k^{(4)})}. \tag{3.13}$$

3.5 Attention Mechanism

Attention mechanism has been proved useful in not only computer vision tasks but also in natural language processing tasks. In this section, we introduce this method to solve sentiment degree classification.

We did not use Multi-level Hierarchical Attention networks as (Yang *et al.* (2016)) because each sentence in our data set is not very long so that we do not need to capture intrinsic structure from a document level. Based on the baseline model, Bi-LSTM, we put Attention layer after the Bi-LSTM embedding layer to get a weighted feature vector, which can capture the most important semantic information in a sentence. The Attention mechanism with Bi-LSTM network structure can be shown in following diagram.

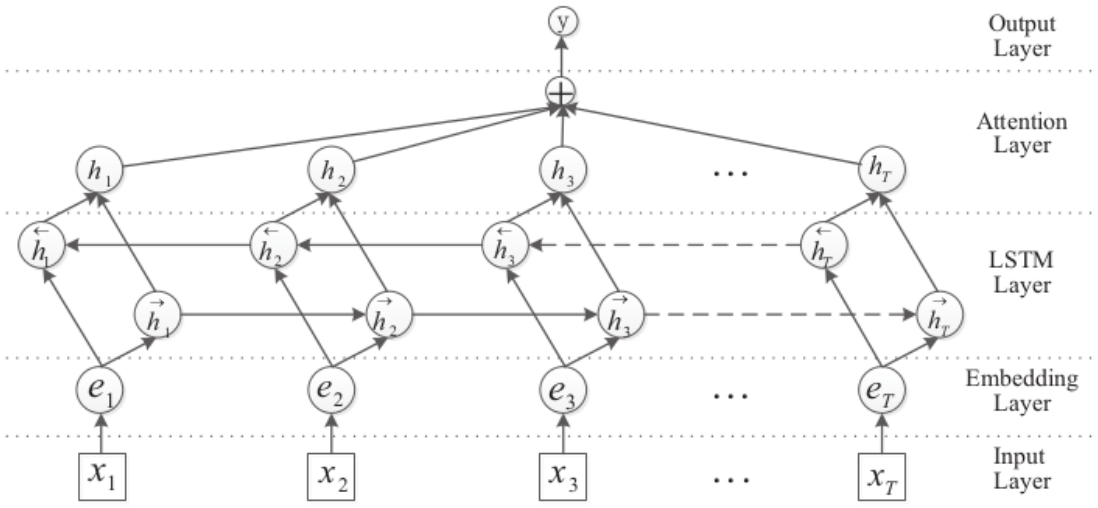

Figure 3.11: Bi-LSTM with Attention Mechanism (source:Zhou *et al.* (2016a))

As we can see from the diagram, the Input Layer, Embedding Layer and LSTM layer are the same as Bi-LSTM network, after LSTM layer get the high-level features from the embedding layer, we use Attention Layer which have produced a weight vector in the

training step. It has ability to construct a higher level feature vector representing a whole sentence through combining those word-level feature vectors in each time step. Assuming that H is the LSTM output vectors $[h_1, h_2, h_3, ..., h_l]$, l is the sentence length, let s be the representation of the input sentence x, it can be computed by:

$$P = \tanh(H) \tag{3.14}$$

$$\alpha = softmax(\omega^T P) \tag{3.15}$$

$$s = H\alpha^T \tag{3.16}$$

where $H \in \mathbf{R}^{d^\omega \times l}$, d^ω is the dimension of the word vectors, ω^T is the transpose of a trained parameter vector, the dimension of α, s are T and d^ω respectively. So the final representation used for classification is:

$$y = \tanh(s). \tag{3.17}$$

This model does not rely on any pre-defined NLP tools or vocabular resources to build, it just uses our raw text with position markers as input. This model can be seen as an optimization of Bi-LSTM baseline model.According to our experiment result, using Attention layer on top of Bi-LSTM can accelerate traditional Bi-LSTM by a large margin.

3.6 Language Model in Classification

More recently, Generative Pre-Training(Radford *et al.* (2018)) is a method of pre-training language models on a large network with a large amount of un-labeled data and fine-tuning in lower level tasks has made a breakthrough in several natural language understanding tasks such as commonsense reasoning, question answering and textual entailment.

ULMFiT(Howard and Ruder (2018)) is an effective transfer learning method that also achieve very good results in the text classification task, it brings up focusing on better way to fine-tuning a language model. The problem with ULMFiT is that it use special fine-tuning that only adapt to classification task, for other more complicated task it need other ways to pre-train and fine-tune. Embedding is a key tool in transfer learning in NLP, (Peters *et al.* (2018)) introduced ELMo, it can embedding words into vector space using bidirectional LSTMs which was trained on a language model objective. But these language model have a common problem, only significant information from individual words can be captured but not relationships between them based on their relative positions in a sentence.

Googles Bidirectional Encoder Representations from Transformer(BERT, (Devlin *et al.* (2018))) is different from previous model intuitively because it use a modified method to produce representation called masked language modeling. Due to the network structure, previous language model could not take advantage of both left and right contexts simultaneously. BERT randomly erased some words in a sentence and replaced with a unique token ”masked”, then it use multi-layer bidirectional Transformer to train on plain text for masked word prediction and next sentence prediction tasks. BERT can learn how to predict a word from surrounding words from these specific task therefore it automatically learns better word representation. Similarly, only last layer for downside task need to be change because most parameters of this language model is pre-trained.

The backbone model is Transformer(Vaswani *et al.* (2017)) which we already look in details before. A big difference is that previous method looked at a text sequence either from left to right or combined left-to-right and right-to-left training, but BERT use the Attention-based encoder which reads the entire sequence of words at once, this characteristic allows the model to have a deeper understanding of language context.

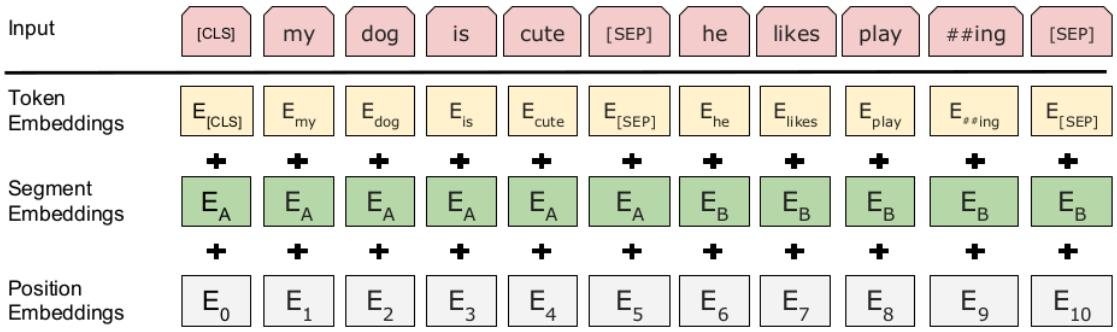

Figure 3.12: The Input Representation of BERT(source:(Devlin *et al.* (2018)))

Fig3.12 is the input format that BERT expects, there are three special tokens that BERT authors used for fine-tuning and specific task training:

[CLS]: The first token of every sequence. A classification token which is normally used in conjunction with a softmax layer for classification tasks.

[SEP]: A sequence delimiter token which was used at pre-training for sequence-pair tasks (for instance, Next sentence prediction). Must be used when sequence pair tasks are required. When a single sequence is used it is just appended at the end.

[MASK]: A token used for masked words, only used in pre-training.

The input layer is the vector representations of the sequence together with the special tokens. The 'Token Embeddings' layer are the vocabulary IDs for each of the tokens including special tokens. The 'Sentence Embeddings' layer is a numeric class to distinguish

between sentence A and B. The last layer 'Positional Embeddings' layer indicate the position of each word in the sequence.

As we have discussed above, BERT learns text representations by pre-training on two tasks: Masked Language Modelling and Next Sentence Prediction.

1. **[MASK Language Modelling]**: In a sentence with two words removed, BERT is trained to predict what those two words are, such as below:

Input: The car had a heavy $[MASK]_1$. We need to do a complete $[MASK]_2$

Labels: $[MASK]_1 = accident$; $[MASK]_2 = inspection$

2. **[Next Sentence Prediction]**: Given two sentences, BERT is trained to determine whether one of these sentences comes after the other in a piece of text, or whether they are just two unrelated sentences.

Exp1: **Labels**: In Next

Sentence A: The car had a heavy accident.

Sentence B: We need to do a complete inspection.

Exp2: **Labels**: Not In Next

Sentence A: The car had a heavy accident.

Sentence B: Your dog is so cute.

By doing these two tasks in pre-training, the training sets can be obtained without too much supervised efforts. As a result, BERT can be pre-trained on a massive corpus of text data(such as Wikipedia) and can learn rich representations of language that are impossible to learn with small data sets. The final layer can be fine-tuned on a task of our choice later.

Chapter 4

Method Implementation and Experimental Result

For MLTC task, we implement TextCNN with multiple filters, performance is reported by positioned F1 score. For Sentiment Analysis task, we firstly fine-tuned our own Word2Vec embedding model, then we implement several neural networks. Our baseline model is Bi-LSTM, which is an RNN model without any optimization. Then we implement Attention with Bi-LSTM to optimize feature vector. RCNN is introduced to try to mix up CNN and RNN. Furthermore, we implement TextCNN which emphasizes keywords contribution compares to RNN based model that focused on contextual meaning. Finally we replace traditional neural network model with state-of-art language model, BERT, to solve this task with transfer learning method. As for it is a Multi-class Text Classification problem, we use classification accuracy as performance score. Experiments results in numeric comparison have been presented in table.

The training is carried out on a PC with one NVIDIA GeForce GTX 1080Ti.

4.1 Multi Label Text Classification Method

4.1.1 Dataset for Training and Testing

Our MLTC dataset is extracted from a SQL database, which consists of 901945 instances after pre-processing, each instance have at least 1 label, up to 14 labels. Our label set has 20 labels in total, we encoded each word in each instance by words vocabulary, the vocabulary size is 18009, and we encoded each instances labels by multi-hot encoding method as well. We divided our dateset to three parts, training set has 871944 instances, validation set has 10000 instances, test set has 20000 instances. To train the network, we padded each instance to 200 long with 0. Here is an example of training instance:

$$TrainX[2] = [788, 66, 8, 13, 1018, 4, 251, 9982, 1687, ..., 0]$$

$$TrainY[2] = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0]$$

for TrainX[2], $[788, 66, 8]$ are the index of words in the vocabulary. Each pair of instance is similar to this example, validation set and test set has the same structure. For this instance, it has 5 related labels which are $label1, label13, label14, label18,$ and $label19$. The statistics of our dataset compared with two classic dataset, Reuters Corpus Volume I (RCV1-V2) and Arxiv Academic Paper Dataset (AAPD) is shown in Table4.1, it is apparently that our dataset is a short text classification dataset:

Table 4.1: Statistics information of our dataset.

Dataset	total samples	label sets	words/sample	labels/sample
RCV1-V2	804144	103	123.94	3.24
AAPD	55840	54	163.42	2.41
Ours	**901945**	**20**	**9.83**	**1.81**

4.1.2 Network Architecture

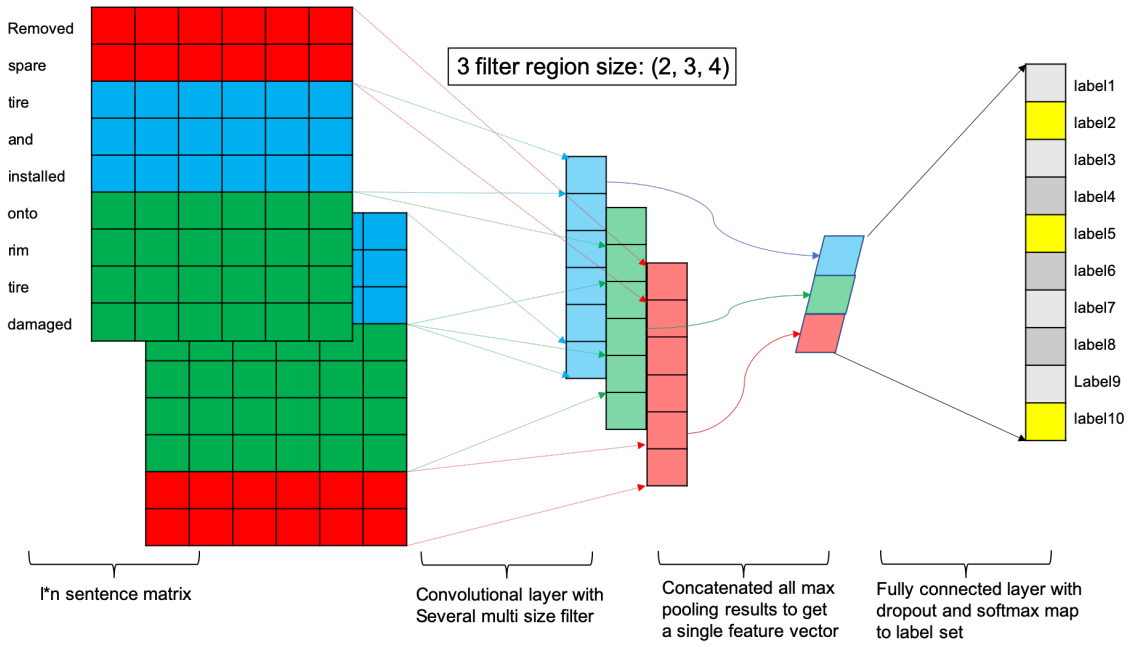

Figure 4.1: MLTC Model Architecture for an Example Sentence.

The model architecture, shown in Figure 4.1, is a CNN based Multi-label classification architecture. Assuming an instance x_i is a n-dimensional word vector corresponding to the i-th word in an instance. The instance length is l which has been padded to a fixed length 200. The instance $x1$ can be represented as:

$$x1 = [v_1, v_2, v_3, ...v_l] \tag{4.1}$$

where [ ] here means concatenation operator, $v_1, v_2, v_3, ...v_l$ are the vector representation of words in instance $x1$. The convolution layer will involves a filter with size **f1**, applied to a

window of m words to produce an intermediate feature:

$$e_1 = f(\mathbf{f1} \cdot x_{i+m-1} + b) \tag{4.2}$$

where b is the bias item, f is a non-linear function. Filter **f1** will traversal the whole instance to produce a feature map.

$$E1 = [e_1, e_2, e_3, ...e_{l-m+1}]. \tag{4.3}$$

Then we apply ReLU activation function to add non-linearity, after that, we use max pooling operation to take the maximum value as the key feature related to this size of filter.

For regularization we employ dropout with a constraint on l_2-norms of the weight vectors, this operation can reducing overfitting in neural networks by preventing complex co-adaptations on training data(Hinton *et al.* (2012)). Finally, we use fully connected layer to project the final features to per-defined labels.

During the training procedure, we use $logits$, which is the linear layer output of the network and ground truth $trainY$ to compute loss, because each instance have at least 1 label, here we use cross entropy loss(De Boer *et al.* (2005)). Cross entropy takes our two distributions, estimated distribution $\hat{y} = logits$ and true distribution $y = trainY$, is given by

$$L = -y \cdot log(\hat{y}) \tag{4.4}$$

where $\cdot$ is the vector dot product.

4.1.3 Validation Metric

Below is the confusion matrix for classification:

Table 4.2: Confusion matrix

Test Result	Truth: "A"	Truth: "Not A"
"A"	TP	FP
"Not A"	FN	TN

Where TP means True Positive, FP means False Positive, FN means False Negative, TN means True Negative. Precision is the fraction of TP among all the positives recalled:

$$Precision = \frac{TP}{TP + FP}. \tag{4.5}$$

Recall is fraction of TP among all the correct events:

$$Recall = \frac{TP}{TP + FN}. \tag{4.6}$$

F1 score is the harmonic mean of the Precision and Recall:

$$F1 = 2\frac{Precision * Recall}{Precision + Recall}. \tag{4.7}$$

Our dataset is Multi-label classification dataset, so we use $F1 - macro$ and $F1 - micro$ score. Assuming we have n labels, for $F1 - macro$, we compute the Precision and Recall for each label, and then we have:

$$Precision_{macro} = \frac{Precision_1 + Precision_2 + ... + Precision_n}{n} \tag{4.8}$$

$$Recall_{macro} = \frac{Recall_1 + Recall_2 + ... + Recall_n}{n} \tag{4.9}$$

$$F1 - macro = 2\frac{Precision_{macro} * Recall_{macro}}{Precision_{macro} + Recall_{macro}}. \tag{4.10}$$

For $F1 - micro$:

$$Precision_{micro} = \frac{TP_1 + TP_2 + ... + TP_n}{TP_1 + TP_2 + ... + TP_n + FP_1 + FP_2 + ... + FP_n} \tag{4.11}$$

$$Precision_{micro} = \frac{TP_1 + TP_2 + ... + TP_n}{TP_1 + TP_2 + ... + TP_n + FN_1 + FN_2 + ... + FN_n} \tag{4.12}$$

$$F1 - micro = 2\frac{Precision_{micro} * Recall_{micro}}{Precision_{micro} + Recall_{micro}}. \tag{4.13}$$

4.1.4 Experimental Results

While training the neural network, we normalize each input of size 200×128, the first dimension is the fix length of each instance, which was padded to 200, the second dimension is embedding size, which was used to get an initial embedding matrix. We use tensorflow API tf.random_normal_initializer to generates tensors with a normal distribution. We train the network for 10 epochs in total, learning rate is set as 0.0002, batch size is 64 and we have reduce the learning rate by half every 1000 steps.

Every 3000 steps we use validation set to do a evaluation, in an evaluation, if $logits_i \geq 0$, $label_i$ would be considered as one of the predict label list, we use predict label list $\{label_i\}$ and target label list $\{evalY\}$ to compute confuse matrix, then using confuse matrix

to compute $F1-macro$ and $F1-micro$ score. At every end of epoch we use same strategy to evaluate on test set data, and report $F1$ score.

After 10 epoch training, our training loss goes down to 0.047, test loss is 0.042, $F1-macro$ score is 0.998, $F1-micro$ score is 0.998. The result shows that our CNN based model can perfectly predict the label list related to test data. Here are some randomly selected prediction examples from test set after last epoch training:

Table 4.3: Several examples of our textCNN model prediction result.

Example	targetY	predictY	target-label-logit
exp1	[3, 5, 12]	[3, 5, 12]	[8.0786, 10.5817, 10.3979]
exp2	[18]	[18]	[10.7559]
exp3	[10, 12]	[10, 12]	[11.0153, 10.0365]
exp4	[9]	[9]	[10.6712]
exp5	[1, 7, 8, 13]	[1, 7, 8, 13]	[8,2970, 5.7032, 10.0377, 11.2335]

As we can see from the test result, our model has a great performance on test set data. For an instance, the predicted label list are those index of network output logits which equal or bigger than 0, we can see not only the predictY is same with targetY, but also the logit value of all predict label are relatively high, much bigger than 0. Which means our model can classify multi label related to an instance very well.

4.2 Sentiment Analysis Classification Method

4.2.1 Dataset for Training and Testing

Our SA dataset is based on MLTC dataset, we despite those instances which have less than 5 words, because we assume sentiment classification need more information. After pre-processing, there are 259451 instances in total, each instance have only one label indicating the sentiment degree for the instance. For our auto repair dataset, the sentiment degree level vary from 0 to 4, which can be seen as normal to severe of a car damage condition. So the size of label list is 5. We divided our dateset into three parts, training set has 241290 instances, validation set has 12972 instances, test set has 5189 instances. To train the network, we padded each instance to 200 long with 0. Here are 2 examples of training instance:

$$TrainX[1] = [97, 80, 118, 114, 58, 115, 116, ..., 0] | TrainY[1] = 1$$
$$TrainX[2] = [10, 111, 1, 120, 76, 121, 0, 1, ..., 0] | TrainY[2] = 0.$$

For the word embedding part, in Bi-LSTM, Bi-LSTM with Attention, RCNN and TextCNN model, we use Word2Vec embedding method to generate network input and then load it in the training step. We append 2 tokens 'PAD' and 'UNK' to the vocabulary, which means padding 0 and unknown words respectively. We set 0 and random distribution as the related vector value. The embedding matrix has size 7860×200, the first dimension is vocabulary size, the second dimension is embedding size that we use to build embedding matrix at the beginning. In language model, BERT, we use pre-trained language model from (BERT-BaseUncased) released by google($github.com/google-research/bert$)

4.2.2 Network Architecture

Here we would like to show the difference among these model, compare the results and get further discussion.

Bi-LSTM

Bi-LSTM model is an RNN-based model. Firstly, we use word embedding method to get the embedding vector of input. Secondly, with Bi-LSTM block, we can get the contextual information from two direction of an instance, then concatenate two direction of hidden layer outputs $\overrightarrow{e}$ and $\overleftarrow{e}$ together to get the final feature vector

$$h = [\overrightarrow{e} \oplus \overleftarrow{e}] \tag{4.14}$$

Finally, we use fully connected layer to map this feature vector to our label set, which contains 5 class. During the training step, we train the network model with cross entropy loss, and we combine dropout with L2 regularization loss to alleviate overfitting.

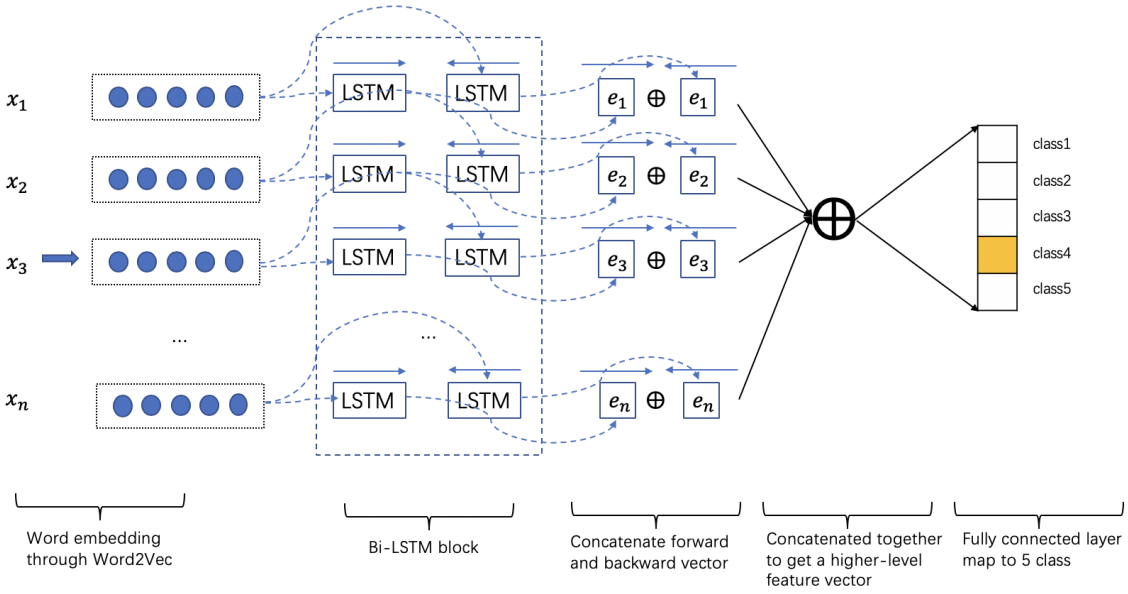

Figure 4.2: Bi-LSTM Structure for Sentiment Classification.

Bi-LSTM+Attention

In Bi-LSTM with Attention model, Attention layer can helps to capture the most important semantic information in a sentence.

The former part of structure is similar to Bi-LSTM model, after getting the high level features from Bi-LSTM layer, we add Attention layer which can multiply this high level features vector by a weight vector to merge word-level features from each time step into a sentence-level feature vector. Finally we use this sentence-level feature vector h to apply softmax classifier, and get the predict label y from classes set:

$$l = softmax(Wh + b) \tag{4.15}$$

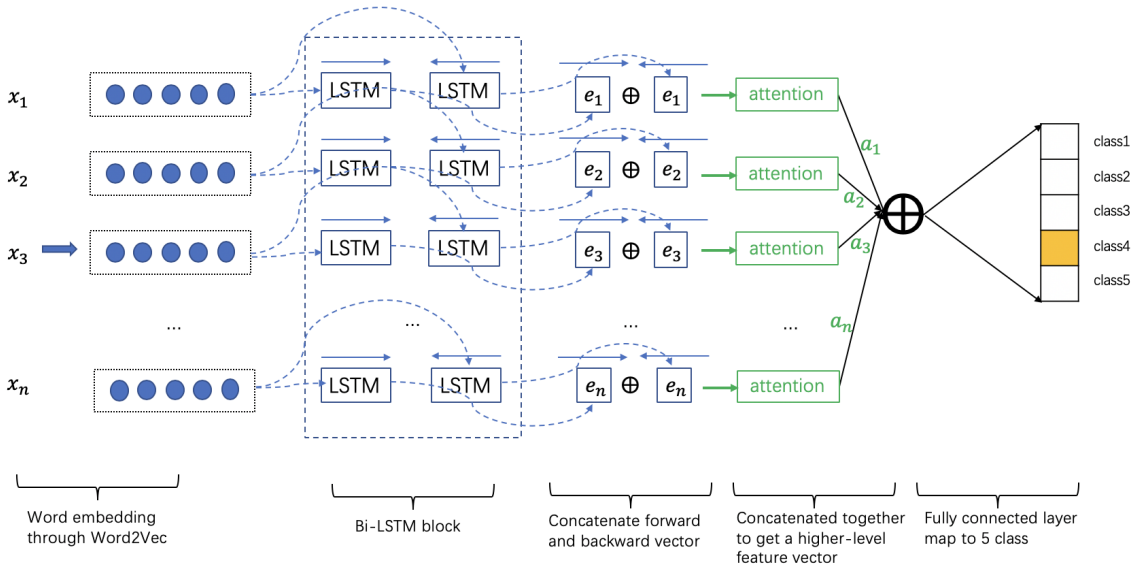

Figure 4.3: Bi-LSTM with Attention Structure for Sentiment Classification.

$$y = argmax(l) \tag{4.16}$$

where W and b is weight matrix and bias in the network, the loss function is same to Bi-LSTM model.

TextCNN

TextCNN model is a CNN-based model, the neural network structure is similar to TextCNN in MLTC, except in sentiment classification we use Word2Vec to do word embedding, so that we can take advantage of pre-trained embedding method, and because it is a single class classification, we use sparse softmax cross entropy loss to compute loss in the training step.

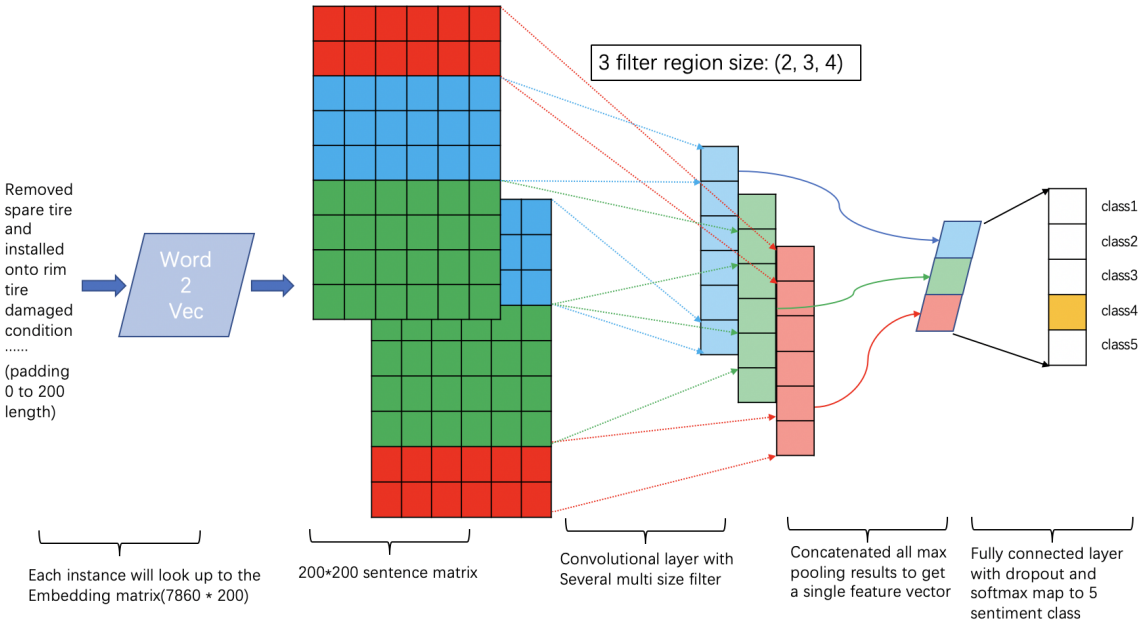

Figure 4.4: TextCNN with Word2Vec for Sentiment Classification

RCNN

In RCNN model, we also use Bi-LSTM model to get contextual information, but here we concatenate two direction of hidden layer outputs $\overrightarrow{e}$ and $\overleftarrow{e}$ together with word embedding vector v as the representation of word.

$$h = [\overrightarrow{e} \oplus v \oplus \overleftarrow{e}] \tag{4.17}$$

Then we apply a linear transformation together with the tanh activation function to get a latent semantic vector y, after that, we apply a max-pooling layer which will do an element-wise max function, then each element of $\hat{y}$ is the maximum value among y_i:

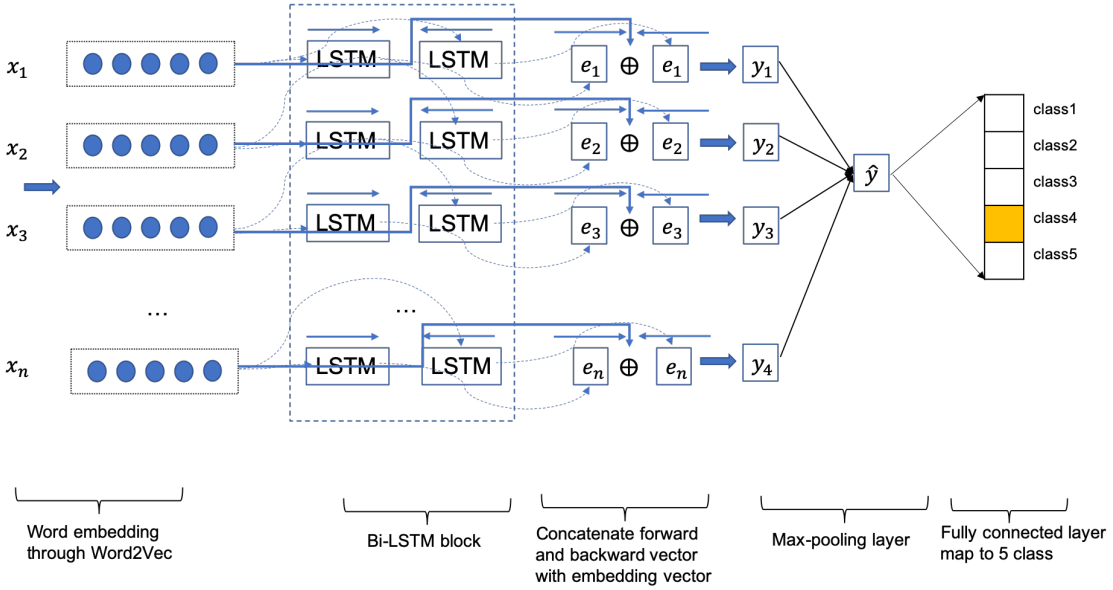

Figure 4.5: RCNN for Sentiment Classification

$$y_i = tanh(Wh_i + b) \tag{4.18}$$

$$\hat{y} = max(y_i). \tag{4.19}$$

With max-pooling layer, we can capture the information throughout the entire text. All in all, this model can use recurrent network to capture contextual information and then constructs the word representation by using convolutional neural network.

BERT

BERT model is totally different with previous model, because it is a language model which most of parameters has been pre-trained, only last layer for classifying need to be modified.

In our classification task, due to the size of our data set, we have trained a middle size model, so our base model is BERT-$base$. In order to make the training process fast, we slightly modified the model parameters. The vocabulary size of input is the same with previous model, the size of encoder layers and the pool layer are both 128. For the Transformer encoder, we use 4 hidden layers and 8 attention heads for each Attention layer, the intermediate(feed-forward) layer size is 1024. The overall structure is shown below.

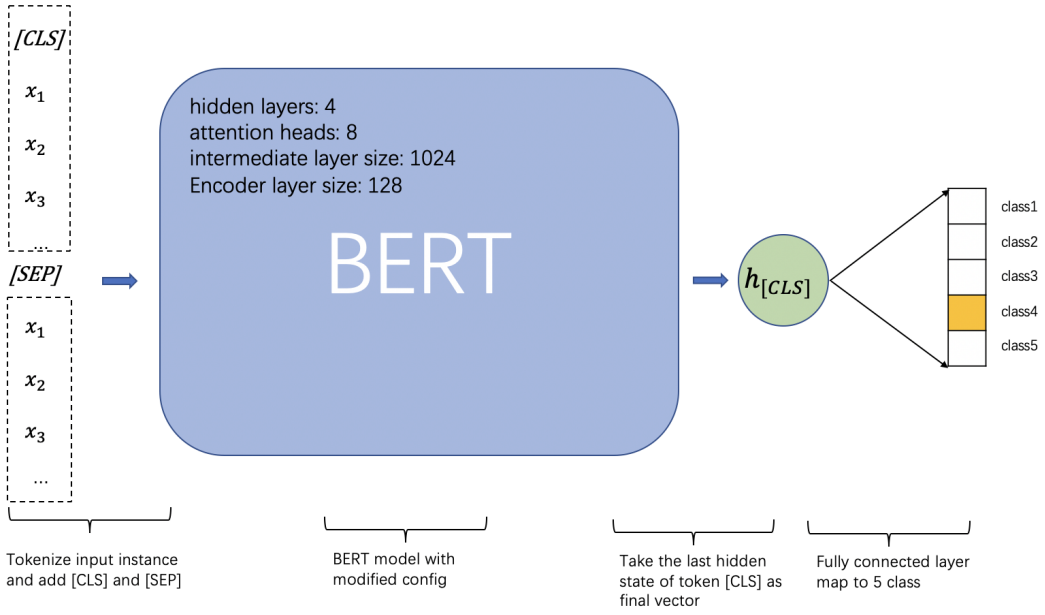

Figure 4.6: Text Classification Task with BERT

Fig4.7 is the input form of BERT in SA, the original **Input** is under a preprocessing called Word2Index, where we produce the vocabulary of the dataset and use index to express each word in an instance. In this vocabulary, we insert key **2** for token **[CLS]**, in order for later use. We change the format of each instance to get the **Input id**, it has one or two segments that the first token of the sequence is always set to **[CLS]** which contains the special classification embedding and another token **[SEP]** is used for separating several parts if we have more than one sentence for an instance.

Following the concept of BERT, we will produce **Input mask** and **Segment ids** of an instance before feed it into model, the shape of both **Input mask** and **Segment ids** are $[64, 200]$ which are same with input size, the first dimension is batch size and the second dimension is fixed max sequence length. For those padding tokens, **Input mask** will be 0, value of other place is 1, **Segment ids** are set to be 1 at all place of the matrix. The input format processing for training BERT model can be shown as:

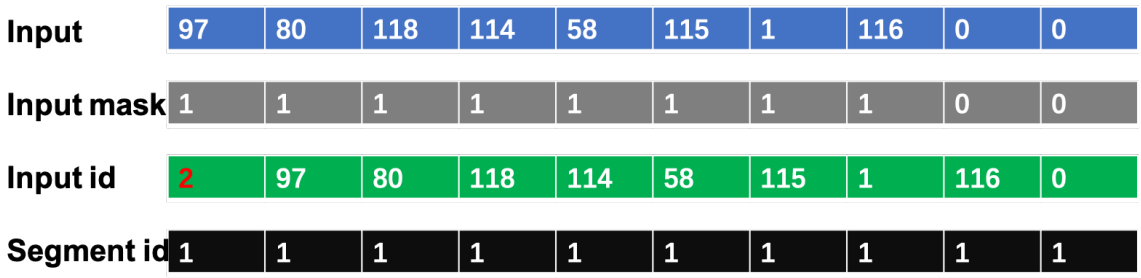

Figure 4.7: BERT Input Format for Sentiment Classification

We take the hidden state of the first token, which is **[CLS]**, as the representation of the input instance, feed to the final dense layer, A simple softmax classifier is added after Transformer operation to predict the probability of label y using the final hidden state h:

$$P(y|h) = softmax(Wh) \tag{4.20}$$

where W is the task-specific parameter matrix, we minimize the sparse cross entropy loss of the model to fine-tune all the parameters.

4.2.3 Validation Metric

Sentiment Analysis on our dataset can be seen as a Single-label Classification problem, so here we use $accuracy$ to evaluate model, as we have already introduced confusion matrix and TP, TN, FP, FN in MLTC, $accuracy$ can be simply calculated in this way:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \tag{4.21}$$

4.2.4 Experimental Results

We normalize each input of size 200×200 except BERT model, here the first dimension is embedding size, the second dimension is fixed max sequence length. The original input of BERT model is just use index of each word in vocabulary to express an instance, the length of each instance is set to 200 as well.

The Bi-LSTM network is trained for 5 epochs and the learning rate(0.0001) is reduced

by 0.65 every 1000 steps, batch size is 64, same as Bi-LSTM+Attention network and RCNN network, but their learning rate are 0.001 and 0.005 respectively. The TextCNN network is trained for 10 epochs, batch size is 64 and learning rate is 0.0001. The language model BERT is trained for 5 epochs, batch size is 64 and learning rate is 0.00007.

In the evaluation part, we simply use argmax function on model output **logits** to get the index as prediction, and then return the truth value of $predictionY == groundtruthY$, after that we transfer this result data format to float32, and then compute the mean of elements across all dimensions. Finally, all dimensions are reduced and we get a single element as accuracy. We add up accuracy for every instance in a batch and get the final accuracy in an evaluation batch. For each model, we did 5 repetitive experiments and use the mean value as the final result. The numerical results of all model for our sentiment classification data set are shown as below:

Table 4.4: All model prediction results reported in ACC(accuracy).

Model	Train_ACC	Test_ACC	training_time(s)
Bi-LSTM	0.755	0.751	7278.7
Bi-LSTM+Attention	0.830	0.806	7303.1
RCNN	0.810	0.790	7296.9
TextCNN	**0.839**	**0.811**	**711.8**
BERT	0.832	0.794	1841.2

As we can see from the table, TextCNN achieves best result in training(0.839) and classifying test data(0.811), and also use the shortest time(711.8 seconds), which is around $1/10$ of three RNN models. Our baseline model, Bi-LSTM, gets the lowest accuracy in training and testing. Although we use two optimization method for basic Bi-LSTM, which are adding attention layer and RCNN model, both of them achieve better result in training and testing but still worse than TextCNN model.

On the other hand, the predict accuracy with Attention layer is just slightly lower than TextCNN, however the training time is ten times slower. Because in order to memorize states of different time and get the contextual information of an instance, recurrent neural network spend much more time to memorize than convolutional neural network just do convolution and select the max value.

Another interesting result is that our three RNN based model are consuming similar time to train 241290 instances, the difference among them are less than 30 seconds, which shows that training additional Attention weighted layer and adding max pooling layer cost little time, but can improve the result to a remarkable extent.

For BERT, its performance is better than RCNN for both training and testing, even though its prediction accuracy gets 0.832 in training set which is better than Bi-LSTM + Attention model, on test data it perform worse than Attention model a lot. And the difference between its score on training set and test set are significant, which shows obvious overfitting. Although we tried different parameters sets of the model, our present result is the best. Last but not least, BERT training speed is much faster than RNN based model, but still spend over twice training time compared to TextCNN.

Chapter 5

Conclusion and Future Works

We have proposed solutions to both Multi-label Text Classification task and Sentiment Analysis task. For MLTC task, based on convolutional neural network, we embed our input and use filter convolution operation to replace traditional n-gram information extraction, after that we apply max pooling operation to get the most valuable information from lots of features. This CNN based model accomplish great result in both training set and test set.

For Sentiment Analysis task, the dataset is much smaller than MLTC task, but more semantically complex. We implement three RNN based models, CNN based model and pre-training language model, BERT to compare results. From experiment results we can see that in our data set, key features of an instance are more valuable than contextual information of the whole instance as for CNN get better result than RNN and BERT. Pre-trained language model, BERT is much faster than Bi-LSTM based model, and its classification result is better than Bi-LSTM based model and RCNN model, but BERT is not the best classification model at all time especially in key-words dominant data set.

The main contribution of our work is to effectively implement several mainstream neural network models and fine-tuned language model, BERT for our specific task. We are

the first to successfully implement CNN on short text Multi-label classification task to get great result. We also firstly compare CNN based model, RNN based model and language model on short text classification task. According to our experiment result, using convolution layer with max-pooling layer is the best way to do feature extraction for short text classification task, therefore in that case, CNN model would be more powerful and much faster than RNN model and language model.

In summary, for those long text classification task, each instance has several sentences and a lot of words, which make it semantically rich, therefore it is important to extract contextual feature and give it priority, then consider key word feature contribution. In that case, BERT should be considered as the most suitable model, and those RNN based models should be competitive as well. However, no technique is universal, for short text classification task such as our data set, there are only a few words in an instance which make it very semantic restricted, and this time those useless features will influence contribution of key features, hence it is redundant to consider contextual information in an instance, on the contrary, the proportion of key features should be magnified, as a result, the model would not be distracted by irrelevances.

On the next step, a further extension of the Sentiment Analysis framework would be focusing on take advantage of correlation among labels. Since sentiment degree are with laddering nature, we can extract this characteristic as an additional feature to the context feature we extracted from neural networks which can definitely enrich the feature vector. Exploring the ability of neural network to learn relation among different sentiment degree in training is challenging. Another possible optimization is to investigate different fine-tuning methods of BERT, according to our experiment result, the over-fitting happened in training can be improved. Main directions include taking domain data into pre-training,

selecting useful layer in its backbone model to simplify model, and designing better forgetting algorithm. The power of pre-trained language model can be exploit in further study.

Bibliography

Araque, O., Corcuera-Platas, I., Sanchez-Rada, J. F., and Iglesias, C. A. (2017). Enhancing deep learning sentiment analysis with ensemble techniques in social applications. *Expert Systems with Applications*, **77**, 236–246.

Bengio, Y. *et al.* (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, **2**(1), 1–127.

Boutell, M. R., Luo, J., Shen, X., and Brown, C. M. (2004). Learning multi-label scene classification. *Pattern recognition*, **37**(9), 1757–1771.

Cambria, E. (2016). Affective computing and sentiment analysis. *IEEE Intelligent Systems*, **31**(2), 102–107.

Chen, G., Ye, D., Xing, Z., Chen, J., and Cambria, E. (2017). Ensemble application of convolutional and recurrent neural networks for multi-label text categorization. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2377–2383. IEEE.

Chen, M., Weinberger, K., Sha, F., and Bengio, Y. (2014). Marginalized denoising auto-encoders for nonlinear representations. In *International Conference on Machine Learning*, pages 1476–1484.

Clare, A. and King, R. D. (2001). Knowledge discovery in multi-label phenotype data. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 42–53. Springer.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, **12**(Aug), 2493–2537.

De Boer, P.-T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. (2005). A tutorial on the cross-entropy method. *Annals of operations research*, **134**(1), 19–67.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Elisseeff, A. and Weston, J. (2002). A kernel method for multi-labelled classification. In *Advances in neural information processing systems*, pages 681–687.

Goldberg, Y. and Levy, O. (2014). word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*.

Gopal, S. and Yang, Y. (2010). Multilabel classification with meta-level features. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 315–322. ACM.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, **9**(8), 1735–1780.

Hosmer Jr, D. W., Lemeshow, S., and Sturdivant, R. X. (2013). *Applied logistic regression*, volume 398. John Wiley & Sons.

Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. (2016). Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Katakis, I., Tsoumakas, G., and Vlahavas, I. (2008). Multilabel text classification for automated tag suggestion. In *Proceedings of the ECML/PKDD*, volume 18.

Kim, J., Yoo, J., Lim, H., Qiu, H., Kozareva, Z., and Galstyan, A. (2013). Sentiment prediction using collaborative filtering. In *Seventh international AAAI conference on weblogs and social media*.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Kurata, G., Xiang, B., and Zhou, B. (2016). Improved neural network-based multi-label classification with better initialization leveraging label co-occurrence. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 521–526.

Lai, S., Xu, L., Liu, K., and Zhao, J. (2015). Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*.

Li, C., Xu, B., Wu, G., He, S., Tian, G., and Zhou, Y. (2015). Parallel recursive deep model for sentiment analysis. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 15–26. Springer.

Liu, B. (2015). *Sentiment analysis: Mining opinions, sentiments, and emotions*. Cambridge University Press.

Mesnil, G., Mikolov, T., Ranzato, M., and Bengio, Y. (2014). Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews. *arXiv preprint arXiv:1412.5335*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013a). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013b). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Le, Q. V., and Sutskever, I. (2013c). Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.

Pak, A. and Paroubek, P. (2010). Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, volume 10, pages 1320–1326.

Pang, B., Lee, L., *et al.* (2008). Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, **2**(1–2), 1–135.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*.

Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2011). Classifier chains for multi-label classification. *Machine learning*, **85**(3), 333.

Sasaki, Y. *et al.* (2007). The truth of the f-measure. *Teach Tutor mater*, **1**(5), 1–5.

Severyn, A. and Moschitti, A. (2015). Twitter sentiment analysis with deep convolutional neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 959–962. ACM.

Sharif Razavian, A., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Tang, D., Wei, F., Qin, B., Liu, T., and Zhou, M. (2014a). Coooolll: A deep learning system for twitter sentiment classification. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 208–212.

Tang, D., Wei, F., Yang, N., Zhou, M., Liu, T., and Qin, B. (2014b). Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1555–1565.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Wang, S. and Manning, C. D. (2012). Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th annual meeting of the association for computational linguistics: Short papers-volume 2*, pages 90–94. Association for Computational Linguistics.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489.

Zhang, D., Xu, H., Su, Z., and Xu, Y. (2015). Chinese comments sentiment classification based on word2vec and svmperf. *Expert Systems with Applications*, **42**(4), 1857–1863.

Zhang, L. and Ma, L. (2017). Coattention based bilstm for answer selection. In *2017 IEEE International Conference on Information and Automation (ICIA)*, pages 1005–1011. IEEE.

Zhang, M.-L. and Zhou, Z.-H. (2006). Multilabel neural networks with applications to

functional genomics and text categorization. *IEEE transactions on Knowledge and Data Engineering*, **18**(10), 1338–1351.

Zhang, M.-L. and Zhou, Z.-H. (2007). Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition*, **40**(7), 2038–2048.

Zhang, M.-L. and Zhou, Z.-H. (2014). A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering*, **26**(8), 1819–1837.

Zhang, Y. and Wallace, B. (2015). A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*.

Zhou, P., Shi, W., Tian, J., Qi, Z., Li, B., Hao, H., and Xu, B. (2016a). Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 207–212.

Zhou, P., Qi, Z., Zheng, S., Xu, J., Bao, H., and Xu, B. (2016b). Text classification improved by integrating bidirectional lstm with two-dimensional max pooling. *arXiv preprint arXiv:1611.06639*.