Master's Research Project Report

A Survey on Deep Learning Models for Aspect-based Sentiment Analysis

1810199 YU Yang

Supervisor:	Professor HUYNH, Nam-Van
Main Examiner:	HUYNH, Nam-Van
Examiners:	Yukio Hayashi
	DAM Hieu Chi
	Hideomi Gokon

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
[Knowledge Science]

February, 2020

Abstract

Due to the development of Internet technology, nowadays, our society has entered the Internet era. People around the world can share their lives on the Internet easily. For both buyers and sellers in marketing, these contents (such as Weibo, comments, ...) are the worthiest resources for not only satisfying costumers' requests but also exploring a new market or exploiting a new product.

However, it is challenging to monitor the enormous numbers of comments on the Websites manually. Additionally, different experts have different opinions, and that can easily cause biases in practical work. In recent years, there is a new research direction in Natural Language Processing (NLP) called Sentiment Analysis (or Opinion Mining), which can systematically and automatically analyze text with opinions.

This thesis first survey some related knowledge about sentiment analysis and fundamental concepts of deep learning models. Secondly, we investigate the word embedding and some deep learning models, such as Recurrent Neural Network and its variants, Convolutional Neural Network, Attention model, and Transformer. Finally, we analyze the performance of sentiment analysis using those above deep learning models and make our conclusion.

As the results showed, among these deep learning models, the Transformer-based sentiment analysis models have achieved the new state-of-the-art accuracy. It also shows the trends of development of NLP, which could contribute further research in this domain.

Keywords: sentiment analysis - word representation - deep leaning models.

Acknowledgments

With boundless love and appreciation, I would like to extend my heartfelt gratitude and appreciation to the people who have supported me, not only finishing this Master dissertation but throughout my Master's study.

My deepest gratitude is first to my supervisor, Professor HUYNH Van Nam from Japan Advanced Institute of Science and Technology (JAIST) for his constant encouragement and guidance.Without his usual and enlightening guidance, this paper would not have been in its present form.

Secondly, I would like to express my heartfelt thanks to Professor Takaya Yuizono, who gave me a lot of suggestions to improve my thesis. In addition, I am also very grateful to Mr. NGUYEN Ba Hung, a 3rd-year doctoral student at Huynh-lab, JAIST, for his guidance, paper revision, and encouragement through my Master study.

I would like to thank my dear family for their love and great faith in me for many years. I also want to thank my friends and lab members, especially TRAN Xuan Thang for giving me precious help and time to listen to my opinions and help me solve my problems in the difficult process of my paper.

Finally, I would like to thank my fiancée, YU Fangyu, for comforting me when I felt frustrated and inspiring me to overcome difficulties. I want to dedicate this paper to the celebration of the forthcoming wedding of us.

Thank you,

YU, Yang

February 2020

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

With the explosive growth of social media on the Internet, people can share and review their opinions all over the world. However, it is difficult to find and monitor all opinion websites and extract valuable information manually to help improving services or making decisions. Each site normally contains a huge volume of opinionated text. Moreover, it is also known that different experts have different opinions. There is a saying which goes: "People only see what they want to see". As a result, without professional training, it is hard to ensure the quality of information that refined by human readers. Due to these reasons, the research of Sentiment Analysis was established.

About **"What is Sentiment Analysis ?"**, In *Sentiment Analysis and Opinion Mining*, Authors gave a definition as:

"Sentiment analysis or opinion mining is the computational study of people's opinions, appraisals, attitudes, and emotions toward entities, individuals, issues, events, topics and their attributes [1]*."*

In short, sentiment analysis can help computers to analyze the positive or negative opinions expressed in text. The task has important practical significance. For example, Potential customers want to know the feelings of using service or purchasing products from other users before, and businesses also aim to receive customers' feedback about their products and services.

According to the investigation of export[1], China has become the world's largest e-commerce market, and more than 50% of global e-commerce transactions was happening in China. The online retail transactions of China reached 1.33 trillion in USD, and China's digital buyers exceeded 560 million in 2018. The total Gross Merchandise Volume (GMV) for the Alibaba is CNY 268.4 billion, or 38.4 billion in USD on the past 2019 11.11 Global Shopping Festival. Under this situation, it is impossible to monitor all customers' reviews and comments manually. In finding a way to review such kind of comments and react on time, we apply sentiment analysis to extract customers' attitude from feedback. Typically, From a large amount of customer review data, we apply some deep learning models to identify customer sentiment opinions expressed in text reviews.

In March 2016, at the Google DeepMind Go Challenge, AlphaGo has beaten the Korean chess player Lee Sedol 9-dan. After the competition, the discussion of artificial intelligence and machine learning rapidly heated up, attracting attention from all over the world. Nowadays, deep learning replaces machine learning as the hottest trend in artificial intelligence due to the development of deep learning frameworks such as PyTorch and TensorFlow, and the increasing computational performance. Today, deep learning has already transformed traditional internet businesses like online advertising. Also, deep learning is helping society reducing the reliance of human labors. The famous researcher Andrew Ng said:" AI is the new Electricity." With AI, we can overcome the physical limitations of human beings to get better results and avoid duplication work.

1.2 Thesis Outline

From Chapter 2 to Chapter 5 is organized as follows:

- In Chapter 2, we survey sentiment analysis and fundamental concepts of deep learning models.

- In Chapter 3, we first mainly introduce word embedding, which is related to converting words into vectors. After that, we introduce some deep learning models in NLP and analyze their structures.

[1]https://www.export.gov//

- In Chapter 4, we first introduce Task 4 of SemEval-2014 datasets, which have aspect categories with polarity (positive, neutral or negative). Then we compare the performance of sentiment analysis in different deep learning models and analyze their advantages and disadvantages. Finally we implement an experiment to testify the performance of those deep learning models and get our conclusion,
- In Chapter 5, we summarize the development trend of deep learning models in NLP and future work for the sentiment analysis.

Chapter 2

Related Work

2.1 Different levels of Sentiment Analysis

From the previous section, we have known the background of “Sentiment Analysis“. The most common use of Sentiment Analysis is to classify a text into a class. In this section, we will introduce the composition of sentiment classification tasks. there are mainly three different levels of sentiment classification tasks: document level, sentence level and aspect level.

2.1.1 Document & Sentence Level Sentiment Analysis

Document level sentiment classification classifies an opinionated document, such as a review of product. It considers the whole document as the entire input and output an overall opinion of positive or negative. Compared to document level,the range of sentence level sentiment classification is much “smaller“. It only focuses on individual sentences.

2.1.2 Aspect Level Sentiment Analysis

Different from document level and sentence level, aspect level sentiment classification or **aspect-based sentiment analysis** considers both the sentiment and the target information. A target is usually an entity or an entity aspect. Both **entities** and **aspects** are usually just called **aspects**. Aspect-based sentiment classification aims to infer the sentiment polarity of different target aspects. For example, in the sentence “the screen is very clear but the battery life is too short.” if the target aspect is “screen” then the sentiment is positive, however if the target aspect is “battery life” then the sentiment is negative [2].

2.2 Artificial Neural Networks

Deep Learning is a subfield of machine learning that uses multiple layers to extract higher level features from input, which is inspired by neural circuit of human brain. In this section, we will briefly introduce Neural Network as shown in Figure 2.1.

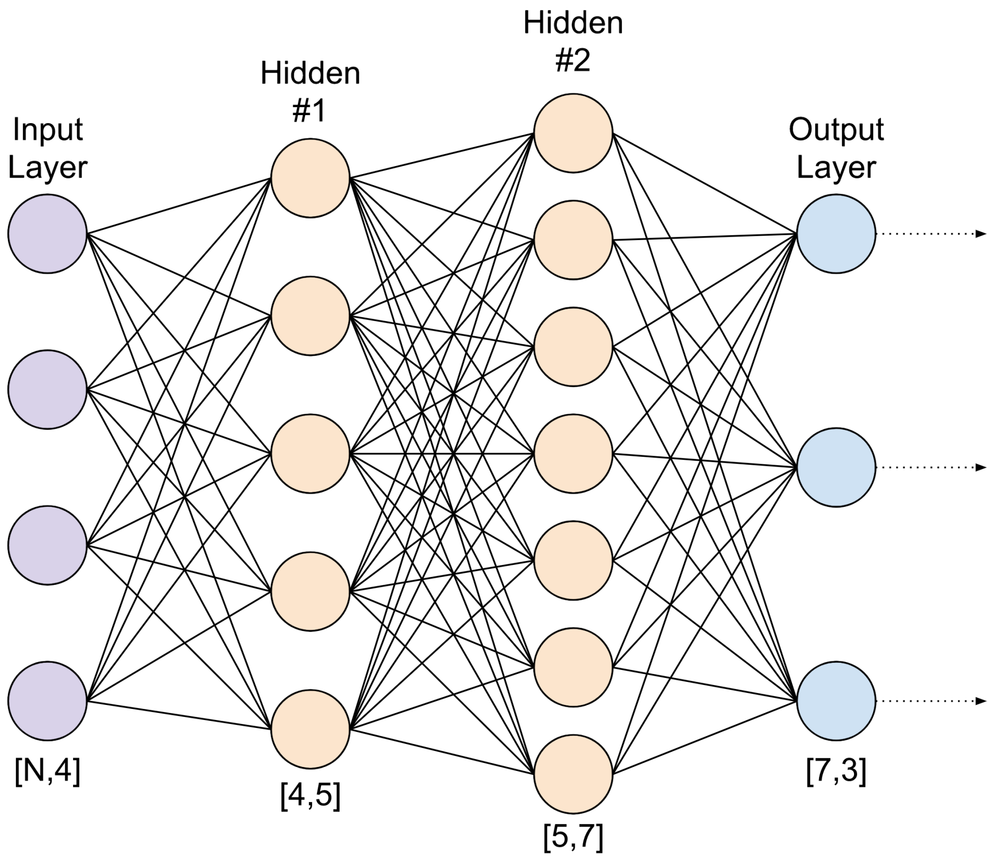


Figure 2.1: An example of deep learning neural network.

2.2.1 Neuron

We can consider neural networks as composite functions. We input some data into this function, and it will output the result. A neural network is made up of many neurons connected. The basic architecture of a single neuron contains three parts as shown in Figure 2.2:

1. Weight
2. Bias
3. Activation function

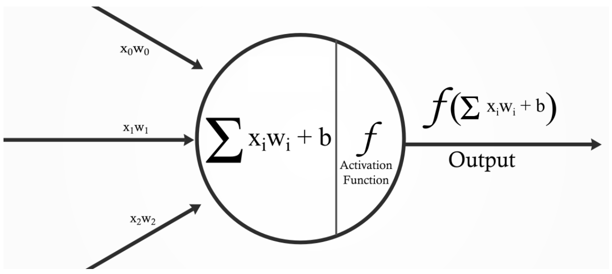
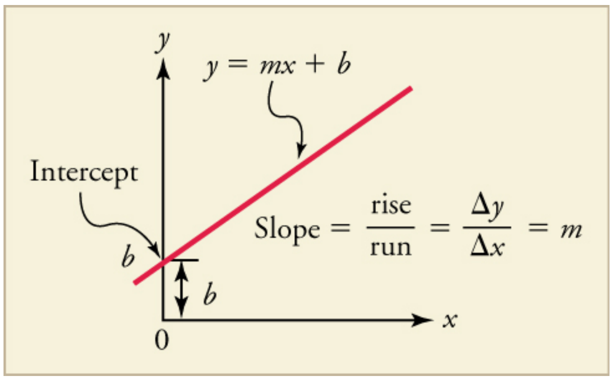


Figure 2.2: The architecture of one neuron.

Here is an example of how a neuron works:

1. A neuron has two ends - the input and the output. The data first enters the neuron, multiplies the weight by the data, and adds a bias to the multiplied data. It is similar to a liner function as shown in Figure 2.3:



y
$y = mx + b$
Intercept
b
Slope $= \frac{\text{rise}}{\text{run}} = \frac{\Delta y}{\Delta x} = m$
b
0
x

Figure 2.3: Liner function.

2. The output of these functions will be sent to another part in the neuron - activation function. The activation function is also called a mapping function. It receives data from previous output and output values over a limited range (in most cases). Mostly, they

are used to convert a large output of previous output to a smaller value. The activation function we choose can significantly affect the result of the neural network output. We can choose different activation functions for different units. Figure 2.4 shows some common activation functions:

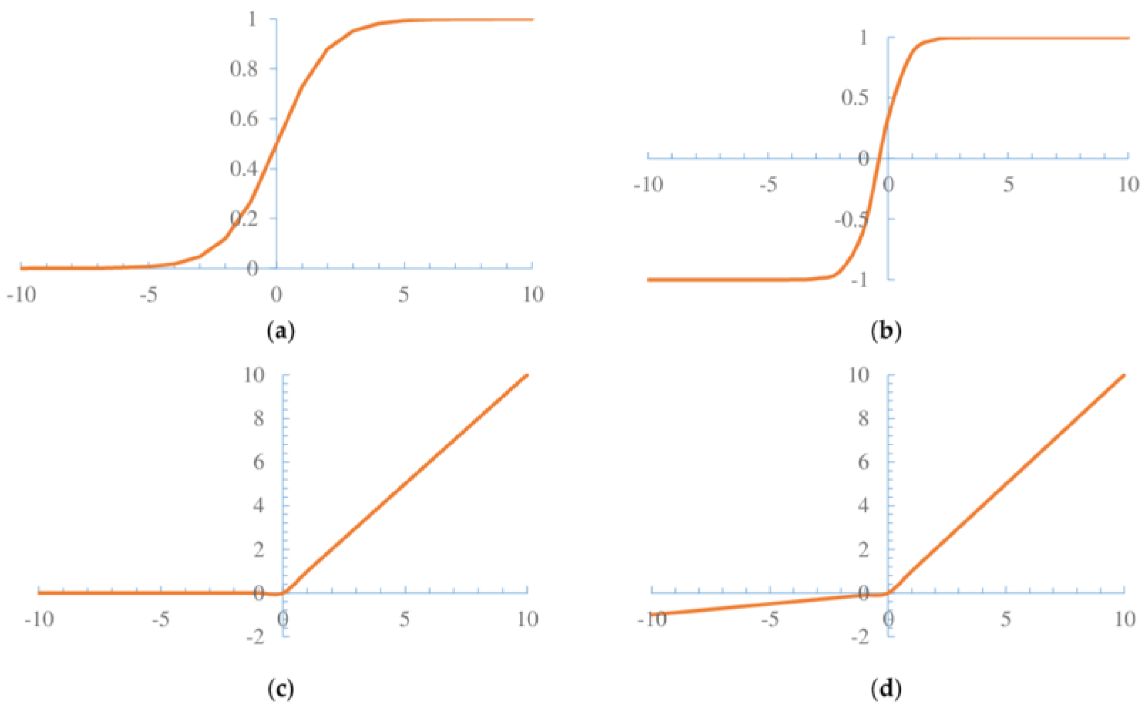


Figure 2.4: Activation function.

2.2.2 Hyperparameter

A hyperparameter is a parameter whose value is set by humans instead of learned from input data, such as learning rates, layers of neural networks, Regularization & Regularization rates. These values must be set manually, while parameters we mentioned above can be learned from training.

When we're starting constructing a new neural network, it's almost impossible to correctly guess all of these hyperparameters at the first time. In fact, deep learning is a highly iterative process, and we need to try different values such as learning rates, number of hidden units to achieve better result. After running an experiment and get a feedback that reflects how well this neural network, then we refine the ideas and change the configuration and try again in order to compare and get a better neural network.

Figure 2.5 shows an example of hyperparameters in Neural Network:

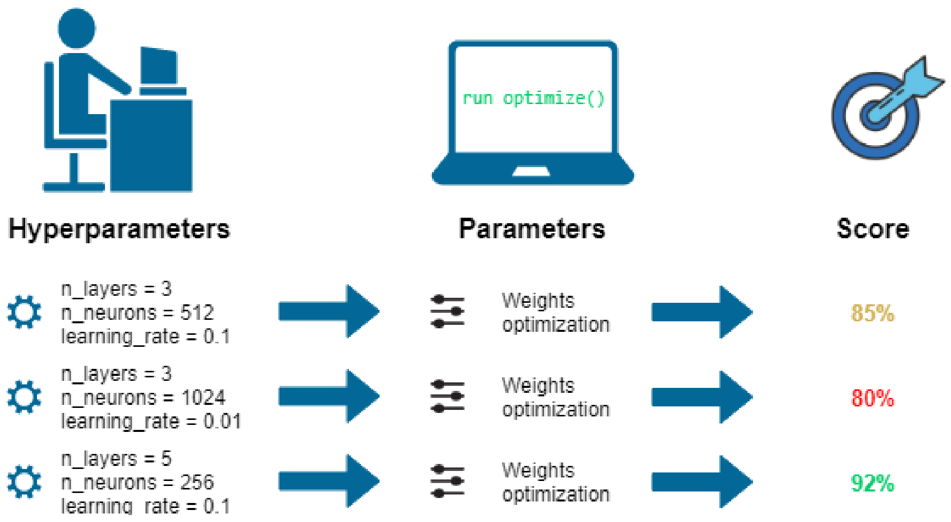


Figure 2.5: Hyperparameters in Neural Networks.

In the Figure above, the number of layers, number of neurons, and learning rates are hyperparameters. It is obvious that the score changes when we set different hyperparameters. The correct settings will accelerate the learning speed of neural networks.

Chapter 3

Methodology

3.1 Word Representation

To deal with NLP tasks, we must first solve the problem of word representation. In computer vision, because the image itself is a numerical matrix, the computer can handle it directly. There is no problem with the representation method in the image. On the contrary, language, as a tool for the expression of high-level information generated by the evolution of human society for millions of years. Language is abstract. It is difficult to portray the connection between them if the two words are literally different. Although we can clearly understand the meaning of the word, the computer can only do mathematical calculations, and the word needs to be expressed in a form that the computer can handle. That is the reason why we have to solve this problem in Natural Language Processing : **How to represent words in a digital way so that the computer can "understand" and "handle".**

3.1.1 TF - IDF

As one of the statistic models for information retrieval and data mining, TF-IDF is used to reflect the importance of a word to a document in a corpus or collection [3,4]. Because this algorithm is efficient and easy to understand, it is often used by the industry for the initial text data cleaning. TF-IDF has two meanings, one is "Term Frequency" (TF), and the other is "Inverse Document Frequency" (IDF). TF-IDF algorithm is made of these two algorithms multiplied together:

1. **Term Frequency (TF)**:

TF is how frequent a word appears in a document.

$$TF_{w,D_i} = \frac{\text{count}(w)}{|D_i|} \tag{3.1}$$

The parameters in this equation are explained as follows:

- count(w) is the number of occurrences of the keyword w.
- $|D_i|$ is the number of all words in the document D_i.

2. **Inverse document frequency (IDF)**:

IDF is how unique a word in a docment.

$$IDF_w = \log \frac{N}{1 + \sum_{i=1}^{N} I(w, D_i)} \tag{3.2}$$

The explanation of parameters in this equation is as follows:

- N is the total number of all documents.
- $I(w, D_i)$ indicates whether the document D_i contains keywords. If it is included, $I(w, D_i)$ is 1, and if not, it is 0. If the word w does not appear in all documents, the denominator of IDF formula becomes 0. Therefore, we add 1 to the denominator, which is called *smooth*.

Once we get TF (word frequency) and IDF (inverse document frequency), multiply these two words to get the TF-IDF value of one word.

$$TF - IDF_{w,D_i} = TF_{w,D_i} * IDF_w \tag{3.3}$$

The result of TF-IDF suggest that the larger the TF-IDF of a word in the article, the higher the importance of the word in this article in general. By computing the TF-IDF of each word in one article, it is easy to find out the keywords.

The advantages of TF-IDF algorithm are **simple**, **fast**, and **easy to understand**. However, the disadvantages of TF-IDF are also obvious. The disadvantages are shown as follows:

1. The frequency of a word is not enough to measure the importance of a word in an article (e.g., sometimes a important word may not appear enough).
2. TF-IDF cannot reflect the position information.
3. TF-IDF cannot reflect the importance of the word in context.

To overcome these difficulties, we will introduce word2vec algorithm in the following sections.

3.1.2 Language Model

The language model considers the language (the sequence of words) as a random event and assigns a corresponding probability to describe how possibly it belongs to a collection of languages. By given a vocabulary set V, for a sequence of words $S = \langle w1, \cdots, wT \rangle \in V_n$, the statistical language model assigns this sequence a probability P(S) to measure the confidence of S in conforming to the grammatical and semantic rules of natural language. In short, the language model is the model that computes the probability of a sentence. The higher the probability of scoring a sentence, the more the sentence conforms to the natural sentence of human expression.

3.1.3 One-hot Representation

One-hot Representation is the easiest and the most intuitive word representation mothod in NLP. Elements of each vector is consist of only one "1", and the remaining elements are "0". This dimension represents the current word. Figure 3.1 shows an example:

Each word is consists of a single 1 and all the others 0. This One-hot Representation is succinct if stored in a sparse way - assign a numeric ID to each word. This simple representation method with the Maximum entropy, SVM, CRF and other algorithms has successfully completed various mainstream tasks in the NLP field.

However, one-hot is not perfect. There are two **disadvantages** of one-hot:

1. The dimension of the vector increases as the number of words in the sentence increases.

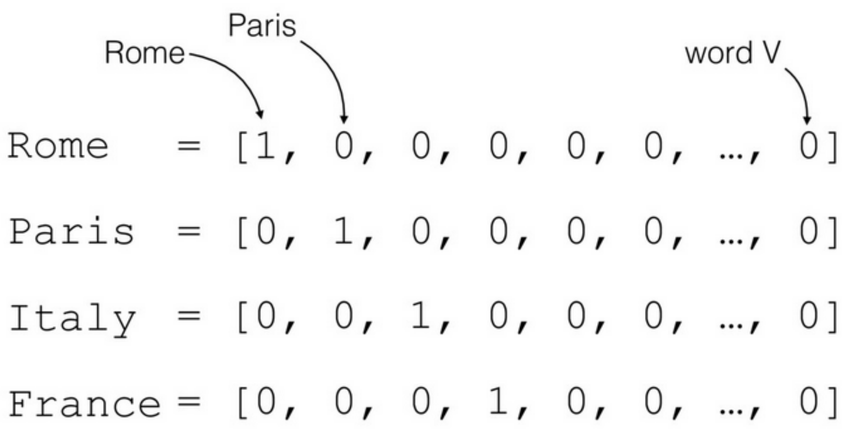

Paris
Rome
word V
Rome = [1, 0, 0, 0, 0, 0, …, 0]
Paris = [0, 1, 0, 0, 0, 0, …, 0]
Italy = [0, 0, 1, 0, 0, 0, …, 0]
France = [0, 0, 0, 1, 0, 0, …, 0]

Figure 3.1: One-hot vector.

2. Any two words are isolated and cannot express relevant information between words at the semantic level.

3.1.4 Word Embedding

A featurized representation with words based on neural networks is generally called as word embedding or distributed representation. For the two weaknesses of one-hot we mentioned above, the distributed representation can solve the one-hot representation problem. The idea is to map each one -hot word vector to a shorter vector through training, and all of these word vectors constitute the vector space so that we can study the relationship between words and words in a common statistical way. We can specify the dimension by ourselves during training [5]. This process is sometimes referred to as ”feature extraction” in NLP.

For the one-hot vector we make some improvements like this:

1. Change each element of the vector from integer to floating-point to become the representation of the entire real range.

2. Embed the original sparsely large dimension into a smaller dimension.

By implementing these, the original one-hot vector will be mapped into low dimension. The advantage of representing words as vectors is that they are suitable for mathematical operators such as add and subtract. Figure 3.2 is a typical example shows the mapped word vectors.

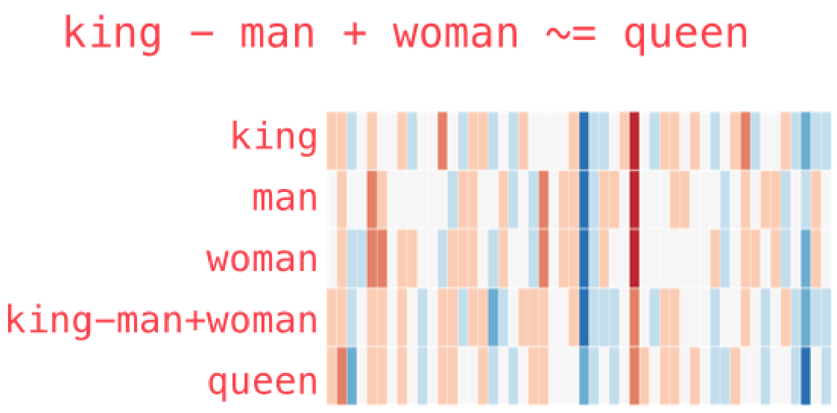


Figure 3.2: Word embedding.

From Figure above, we can first subtract one meaning (i.e. ***male***) from the ***king***'s word vector, then add another meaning (***female***), and finally the result indicate that the vector of (***king*** − ***man*** + ***woman***) is closest to the ***queen***'s. The number in the word vector represents the distribution weight of the word in each dimension. In a simplified sense, each dimension represents a meaning, and the numerical weight of a word in that dimension reflects its degree of association with that meaning. Therefore, after using word embedding, we can use a smaller dimension vector than one-hot to represent words, and we can also easier understand the relationship between words and words.

Word2Vec [6, 7] is one of the popular methods to train word embedding in Natural Language Processing. It includes two training approaches as in Figure 3.3 :

- The Continuous Bag of Words (CBOW) Model: Given the source context words (surrounding words), and predict the current target word (the center word).
  E.g. The cat ate ___ . Fill in the blank, in this case, it's "food".

- The Skip-gram Model: Given the target word (the center word), and predict the source context words (surrounding words).
  E.g. ___ ___ ___ food. Complete the word's context. In this case, it's "The cat ate"

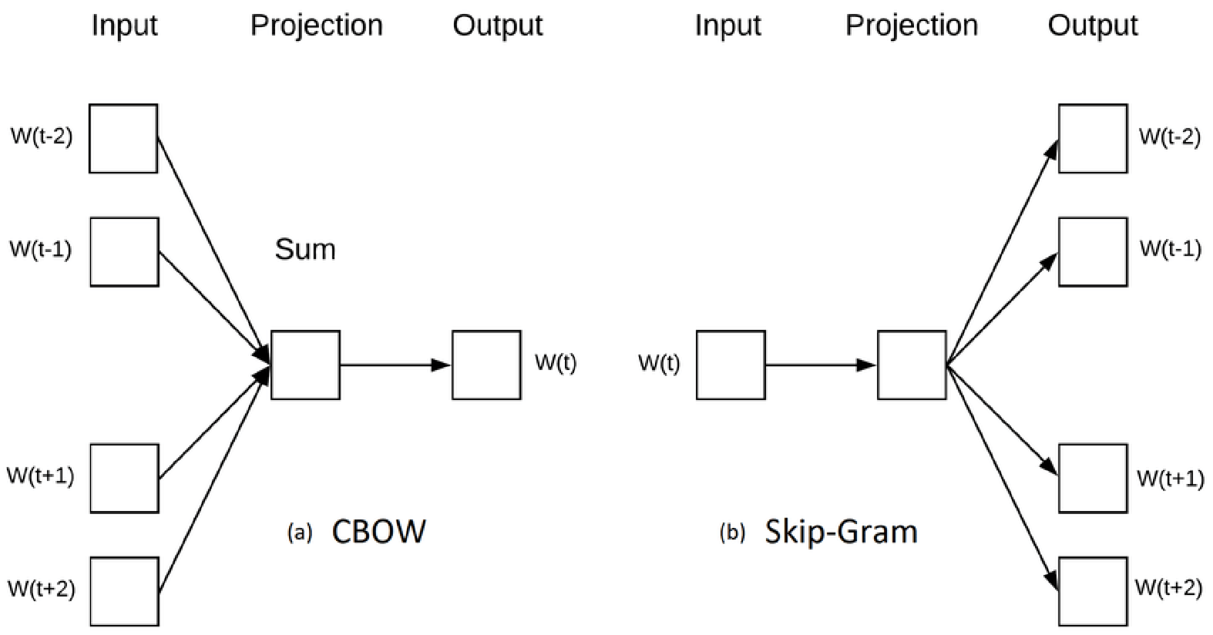


Figure 3.3: Illustration of the Skip-gram and CBOW models.

3.2 Deep Learning for NLP

3.2.1 Recurrent Neural Network & Variants

There are many such situations where the sequence information determines the event itself. For examples:

- The language we are using - the order of the words defines their meaning.
- Time series data - time defines the occurrence of an event.
- Genomic sequence data - each sequence has a different meaning.

If we try to use this type of data to get useful output, we need a network that can access to some prior knowledge about the data in order to fully understand the data. Because the RNN has some memory of what happened before in the data sequence. This helps the system achieve the context. Therefore, for a sequence model, using Recurrent Neural Network (RNN) is necessary.

Figure 3.4 is the architecture of the Recurrent Neural Network. It contains an input layer, a hidden layer, and an output layer:

The parameters are explained as follows:

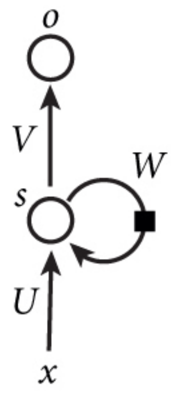


Figure 3.4: Recurrent Neural Network.

- x is a vector that represents the value of the input layer;
- o is a vector that represents the value of the output layer;
- s is a vector that represents the value of the hidden layer;
- U is the weight matrix from the input layer to the hidden layer;
- V is the weight matrix from the hidden layer to the output layer;
- W is the weight matrix from the previous hidden layer.

If we remove the circle with the arrow near W, it becomes the most common fully connected neural network. Figure 3.4 suggests that the value s not only depends on the input x but also on the value s from the last hidden layer. If we unfold the RNN above, it can also be described as in Figure 3.5:

We can use the following formula as in Equation 3.4 and 3.5 to represent the calculation method:

$$\circ_t = g\left(V_{\mathrm{S}_t}\right) \tag{3.4}$$

$$\mathrm{s}_t = f\left(U_{\mathrm{X}_t} + W_{\mathrm{S}_{t-1}}\right) \tag{3.5}$$

Equation 3.4 is the calculation formula of the output layer. Here g is the activation function. Equation 3.5 is a calculation formula for the hidden layer. Here f is another activation function. From the formula above we can see that the recurrent layer has an extra weight matrix W compared with the fully connected layer.

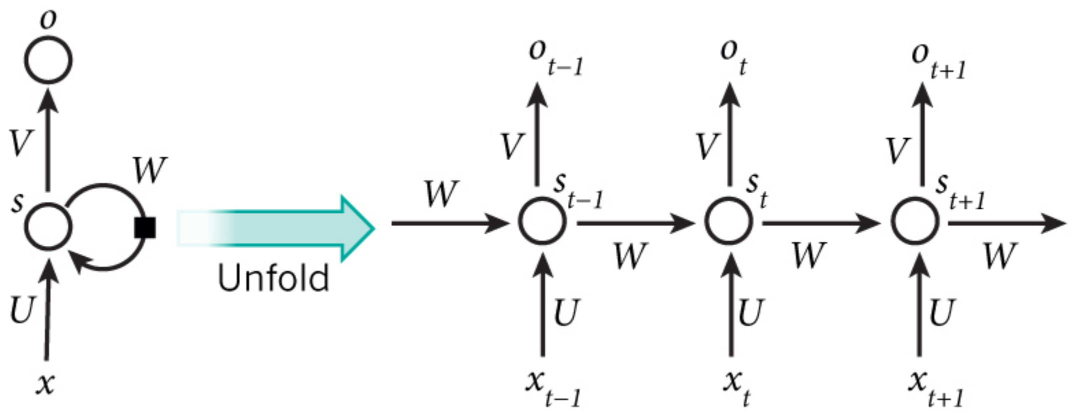


Figure 3.5: Unfolded Recurrent Neural Network.

If we repeatedly substitute Equation 3.5 into Equation 3.4, we will get Equation 3.6:

$$\begin{aligned} \mathrm{o}_t &= g\left(V\mathrm{s}_t\right) \\ &= Vf\left(U\mathrm{x}_t + W\mathrm{s}_{t-1}\right) \\ &= Vf\left(U\mathrm{x}_t + Wf\left(U\mathrm{x}_{t-1} + W\mathrm{s}_{t-2}\right)\right) \\ &= Vf\left(U\mathrm{x}_t + Wf\left(U\mathrm{x}_{t-1} + Wf\left(U\mathrm{x}_{t-2} + W\mathrm{s}_{t-3}\right)\right)\right) \\ &= Vf\left(U_{\mathrm{x}t} + Wf\left(U_{\mathrm{x}t-1} + Wf\left(U_{\mathrm{x}t-2} + Wf\left(U_{\mathrm{x}_{t-3}} + \ldots\right)\right)\right)\right) \end{aligned} \tag{3.6}$$

From the equations above, We can see that the output value o_t of the recurrent neural network is affected by the previous input values X_t, X_{t-1}, X_{t-2}, X_{t-3} ... That is why the recurrent neural network can “memory” previous layers’ information.

Bidirectional Recurrent Neural Network

For the language model, it is not enough to only look at the earlier sequence, such as the following sentence:

My phone is broken, I am planning to ___ a new phone.

If we only look at the words in front of the blank, it is unpredictable to fill in the blank with repair, buy, etc. But if we also see that the word behind the blank is *a new mobile phone*, then the probability of filling the word ***buy*** on the blank is much higher.

Figure 3.6 is the architecture of the Bidirectional Recurrent Neural Network(Bi-RNN) [8]. The hidden layer of the Bi-RNN saves two values: s_t for forward recurrent layer and

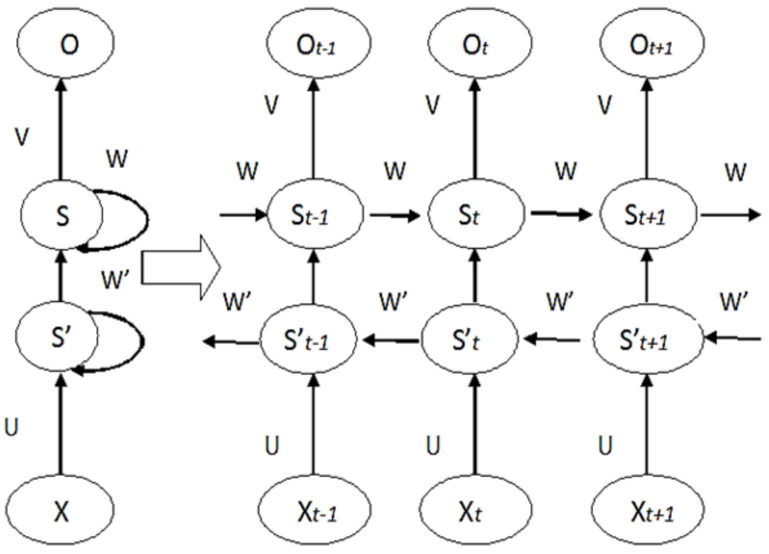


Figure 3.6: Bidirectional Recurrent Neural Network.

the other s'_t for backward recurrent layer. In the forward recurrent layer, the value s_t of the hidden layer is related to s_{t-1}; in the backward recurrent layer, the value s'_t is related to s'_{t-1}; the final output depends on the sum of the forward and backward recurrent layer calculations. The calculation methods of Bi-RNN are as follows:

$$\circ_t = g\left(Vs_t + V's'_t\right) \tag{3.7}$$

$$s_t = f\left(Ux_t + Ws_{t-1}\right) \tag{3.8}$$

$$s'_t = f\left(U'x_t + W's'_{t+1}\right) \tag{3.9}$$

From the three formulas above, we can see that U and U', W and W', V and V' are different weight matrices. That is to say, the forward and backward recurrent neural networks do not share the weight.

Vanishing/Exploding gradients with RNNs

Figure 3.7 shows that the output was primarily influenced by the values in the sequence close to it. On the other hand, there are situations when some sentences have long dependencies, meaning some words within the sentence are related to other ones much earlier in the sequence. Here is an example:

– The **cat**, which already ate ... , **was** full.

– The **cats**, which already ate ... , **were** full.

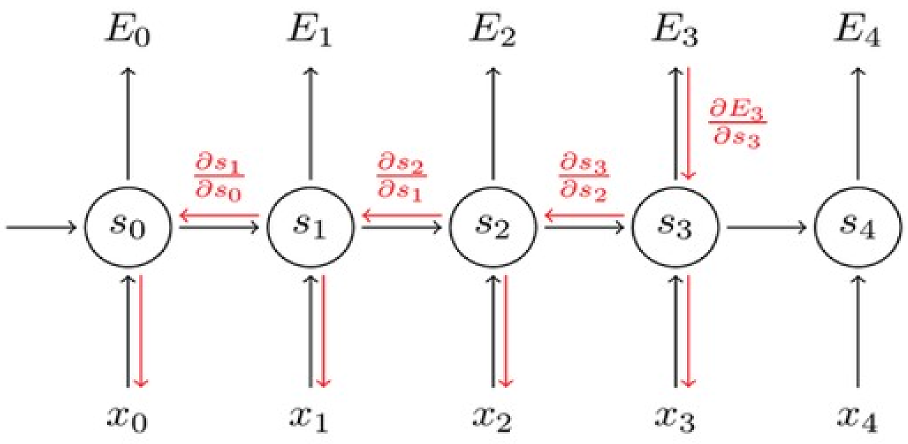

E_0
E_1
E_2
E_3
E_4
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4

Figure 3.7: Backpropagation through time.

Vanishing/exploding gradients with RNNs will cause the problem that it might be difficult to get a neural network to realize that it needs to memorize the just see a singular noun or a plural noun. Basic RNNs are not good enough to capture these long-term information [9].

Here is the mathematical analysis:

1. Now we suppose that our time sequence has only three segments, t_1, t_2, and t_3. Now we only compute the partial derivate of U, W and V with respect to $t = 3$. Based on chain rule we can get:

$$\frac{\partial L_3}{\partial V} = \frac{\partial L_3}{\partial o_3}\frac{\partial o_3}{\partial V} \tag{3.10}$$

$$\frac{\partial L_3}{\partial W} = \frac{\partial L_3}{\partial o_3}\frac{\partial o_3}{\partial s_3}\frac{\partial s_3}{\partial W} + \frac{\partial L_3}{\partial o_3}\frac{\partial o_3}{\partial s_2}\frac{\partial s_2}{\partial W} + \frac{\partial L_3}{\partial o_3}\frac{\partial o_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W} \tag{3.11}$$

$$\frac{\partial L_3}{\partial U} = \frac{\partial L_3}{\partial o_3}\frac{\partial o_3}{\partial s_3}\frac{\partial s_3}{\partial U} + \frac{\partial L_3}{\partial o_3}\frac{\partial o_3}{\partial s_2}\frac{\partial s_2}{\partial U} + \frac{\partial L_3}{\partial o_3}\frac{\partial o_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial U} \tag{3.12}$$

$$\frac{\partial L_3}{\partial U} = \sum_{k=0}^{3}\frac{\partial L_3}{\partial o_3}\frac{\partial o_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial U} = \sum_{k=0}^{3}\frac{\partial L_3}{\partial o_3}\frac{\partial o_3}{\partial s_3}\left(\prod_{j=k+1}^{3}\frac{\partial s_j}{\partial s_{j-1}}\right)\frac{\partial s_k}{\partial U} \tag{3.13}$$

2. Replace 3 with t, it turns to the partial derivation of U with respect to time t. The partial derivation result of W with respect to time t is similar:

$$\frac{\partial L_3}{\partial U} = \sum_{k=0}^{3} \frac{\partial L_3}{\partial o_3}\frac{\partial o_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial U} = \sum_{k=0}^{3} \frac{\partial L_3}{\partial o_3}\frac{\partial o_3}{\partial s_3}\left(\prod_{j=k+1}^{3}\frac{\partial s_j}{\partial s_{j-1}}\right)\frac{\partial s_k}{\partial U} \tag{3.14}$$

3. Now we have the partial derivate of W. Focus on the part of products ($\prod$). The activation function here usually is Tanh. Therefore:

$$s_j = \tanh\left(W_x X_j + W_s S_{j-1} + b_1\right) \tag{3.15}$$

$$\prod_{j=k+1}^{t}\frac{\partial s_j}{\partial s_{j-1}} = \prod_{j=k+1}^{t} \tanh' W \tag{3.16}$$

Vanishing gradient:

The functional graph of tanh is shown in Figure 3.8. When the activation function is a tanh function, it can be seen that the derivative of the tanh function has a maximum value of 1, and the derivative of tanh is less than 1 for most of the training process. That is to say, most of the numbers that are less than 1 are doing the multiplication. If t is large, the above formula will tend to 0. This is why the vanishing gradient happens in the RNN.

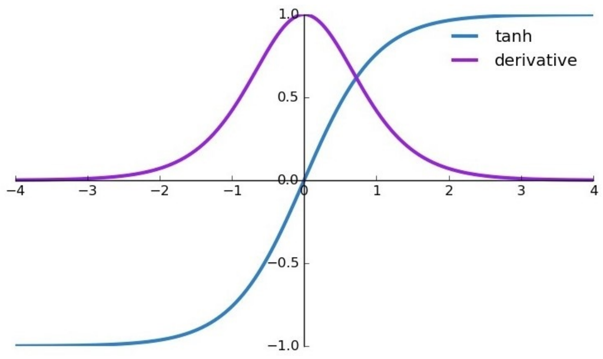


Figure 3.8: Tanh function.

Exploding gradient:

Apart from $\tanh'$, the parameter W is also required. If the value in parameter W is too large. As the length of the sequence getting longer, it also has a long-term dependency, which will result in an exploding gradient.To solve the vanishing/exploding gradient of RNN, there are usually two waysto do this:

$$\prod_{j=k+1}^{t} \frac{\partial s_j}{\partial s_{j-1}} = \prod_{j=k+1}^{t} \tanh' W \tag{3.13}$$

1. Using other activation functions such as *ReLU*. The functional graph of *ReLU* is as in Figure 3.9:

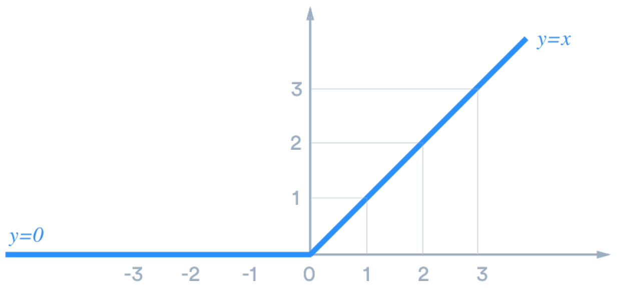


Figure 3.9: ReLU function.

The derivative of the *ReLU* function in the domain greater than 0 is always equal to 1, which solves the problem of vanishing gradient. In addition, the computation speed is fast, and network training can be accelerated. On the other hand, the negative part of the domain is always equal to 0, which will cause the neurons cannot be activated.

2. Change the internal structure to solve the vanishing/exploding gradient problems, which is the LSTM and GRU to be discussed next.

Long Short-Term Memory

Due to the vanishing/exploding gradient of RNN, it is difficulties to deal with long-term dependence. In order to solve this problem, researchers proposed many solutions, such as ESN (Echo State Network), or add leaky units (Leaky Units). The most popular solution is the Gated RNN, and LSTM is one of the Gated RNN. Sepp Hochreiter and Jürgen Schmidhuber first proposed Long-short term memory (LSTM) as a variant of RNN in 1997. The Gated RNN allows the weights to be changed at different times and allows the network to forget that it has accumulated Information [10].

From Figure 3.10 we can observe, each black line represents the flow of vectors; the pink circle represents a pointwise operation such as the sum of vectors; the yellow matrices

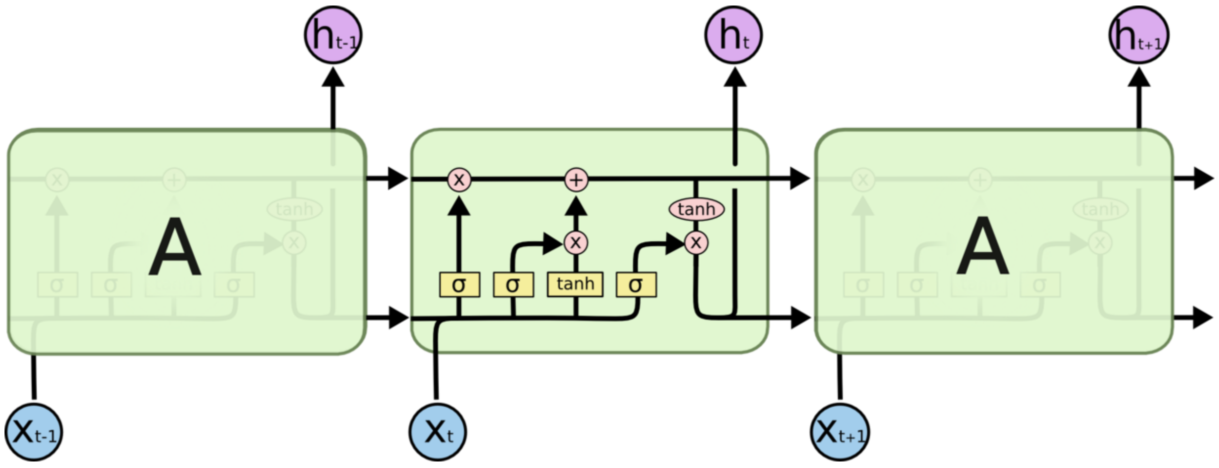


Figure 3.10: Long short-term memory.

are learned from the neural network. The standard LSTM model is a special form of RNN.

The core idea of the LSTM model is the "Cell state" as in Figure 3.11. "Cell state" is similar to a conveyor belt. It runs directly across the chain with only a few linear interactions. This architecture can easily keep the information flowing on it.

LSTM is able to modify cell state through a structure called "Gate" as in Figure 3.12. "Gate" is a way to pass selected information. Each "Gate" consist of a sigmoid function and a pointwise multiplication operation.

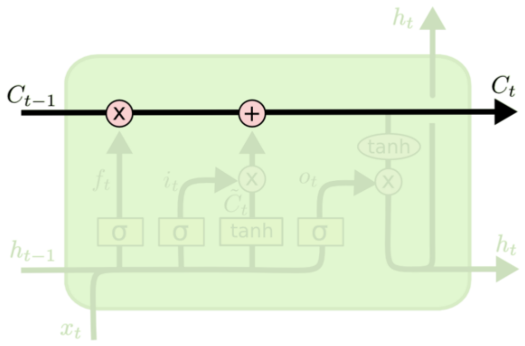


Figure 3.11: Cell state in LSTM.

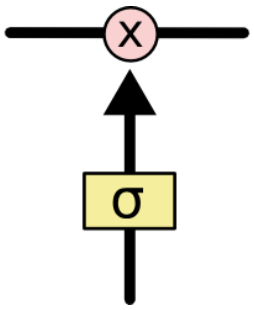


Figure 3.12: Gate in LSTM.

The progress of LSTM forward propagation is as follows:

1. Decide what information we will discard from the "Cell". This operation is implemented by a forget gate layer. This layer loads the input x and the information h from previous hidden layer , and f_t determines the discarded information. Output **1** means

"completely reserved" and **0** means "completely discarded".

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{3.17}$$

2. Determine the new information stored into the "Cell". This step contains two layers: The sigmoid layer as the "input gate" layer to determines the value that we will update; the *tanh* layer creates a new candidate value vector $\tilde{C}_t$ that we will add into the "Cell".

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{3.18}$$

$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \tag{3.19}$$

3. Update the states of the old "Cell": update C_{t-1} to C_t. We multiply the old state C_{t-1} by f_t. Then multiply i_t by $\tilde{C}_t$. Lastly we get the sum. This is the new candidate value.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.20}$$

4. This is the last step, which is to determine the output. First, we determine which part of "Cell" will be output by implementing a sigmoid layer. After this step, we input "Cell" into tanh function and multiply it by the sigmoid from output gate. In the end we will only output the value and copy the output to next layer.

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \tag{3.21}$$

$$h_t = o_t * \tanh\left(C_t\right) \tag{3.22}$$

Gated Recurrent Unit

In 2014, Cho, et al. proposed Gated Recurrent Unit (GRU) as a variant of LSTM. It combines the $forgetgate$ and the *inputgate* into a single *updategate*. Also, the cell state and the hidden state are mixed into one. As a result, GRU is simpler than the standard LSTM model and also is a very popular variant [11,12]. Figure 3.13 shows the architecture of GRU.

The progress of GRU forward propagation is as follows:

1. Here we introduce the two gates of the GRU: "reset gate" r_t and "update gate" z_t. The computation method is similar to LSTM:

$$r_t = \sigma\left(W^r x_t + U^r h_{t-1}\right) \tag{3.23}$$

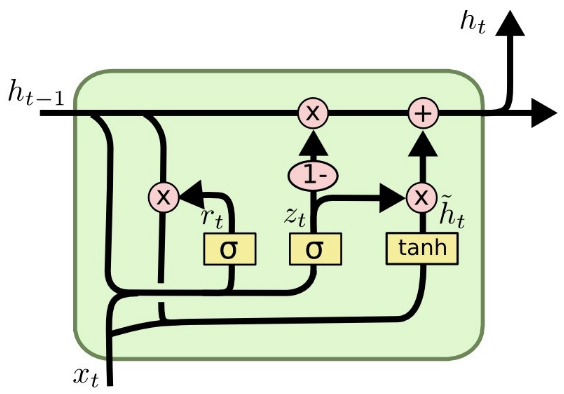


Figure 3.13: Gated Recurrent Unit.

$$z_t = \sigma\left(W^z x_t + U^z h_{t-1}\right) \tag{3.24}$$

2. Compute the candidate hidden layer $\tilde{h}_t$. This candidate hidden layer is similar to $\tilde{C}_t$ in LSTM. It can be considered as new information at the current moment, where r_t is used to control how much previous information is needed. For example, if r_t is 0, then $\tilde{h}_t$ only contains information about the current word:

$$\tilde{h}_t = \tanh\left(W x_t + r_t U h_{t-1}\right) \tag{3.25}$$

3. Use z_t to control how much information from the hidden layer h_{t-1} at the previous moment needed to be forgotten, and how much information from hidden layer $\tilde{h}_t$ at the current moment needs to be added. Finally, we get new h_t, which is both the last output and the new hidden layer information. Compared with LSTM, there is no output gate in the GRU:

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{3.26}$$

3.2.2 Convolutional Neural Network in NLP

When we hear the convolutional neural network (CNN), mostly it reminds us of its application in computer vision, especially CNN makes a huge breakthrough in image classification. From the automatic images annotation of Facebook to Self-driving cars, CNN has been used widely. However, researchers also applied CNN to NLP and found some interesting results. In this section, we will simply review the case where CNN is used for NLP.

Convolutional layer

Let's suppose there is a black and white image shown as in Figure 3.14, each small square represents a pixel, 0 for black, and 1 for white (usually grayscale images have 0-255 for pixels). The sliding windows are often called as kernels, filters, or feature detectors. The size of the above convolution kernel we used above is 3x3, each element is multiplied by a specific value (e.g. elements in diagonal are 1 and the rest are 0), and then sum all the values together to get an element value input to a new matrix. In order for all elements to be convolved, the "window" alsp has to be slid.

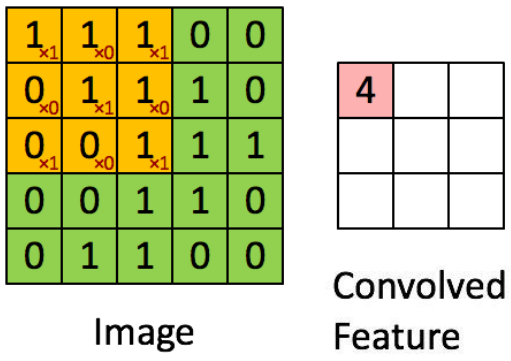


Figure 3.14: Convolutional Filter.

Pooling layer

Another prominent feature of convolutional neural networks is the pooling layer, which is often connected after the convolutional layer. The pooling layer is a subsampling of the convolutional layer. For example, For each region represented by the filter, we will take the maximum of that region and create a new output matrix where each element is the maximum of the region in the original input. This process is called max-pooling as shown in Figure 3.15.

Convolutional Neural Network (CNN) for NLP

A classic Convolutional Neural Network (CNN) usually includes convolutional layers, pooling layers and fully connected layers. Normally, RELU will be adopted as activation function. CNN for NLP as shown in Figure 3.16 uses a sentence or a document as an input matrix typically. Each row of the input matrix represents a token. Token usually

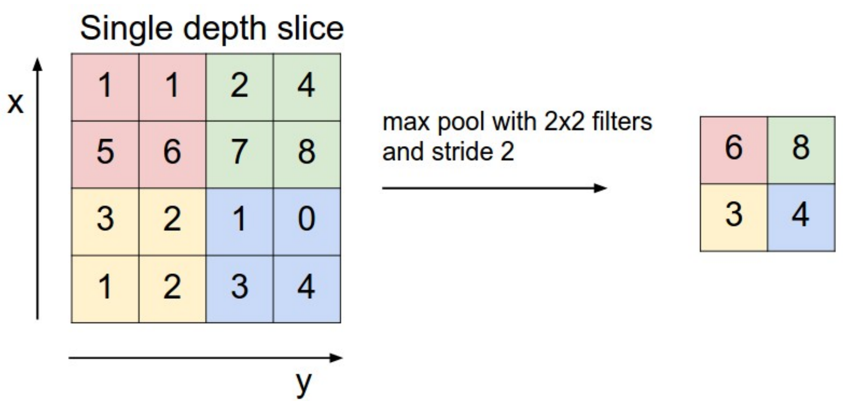
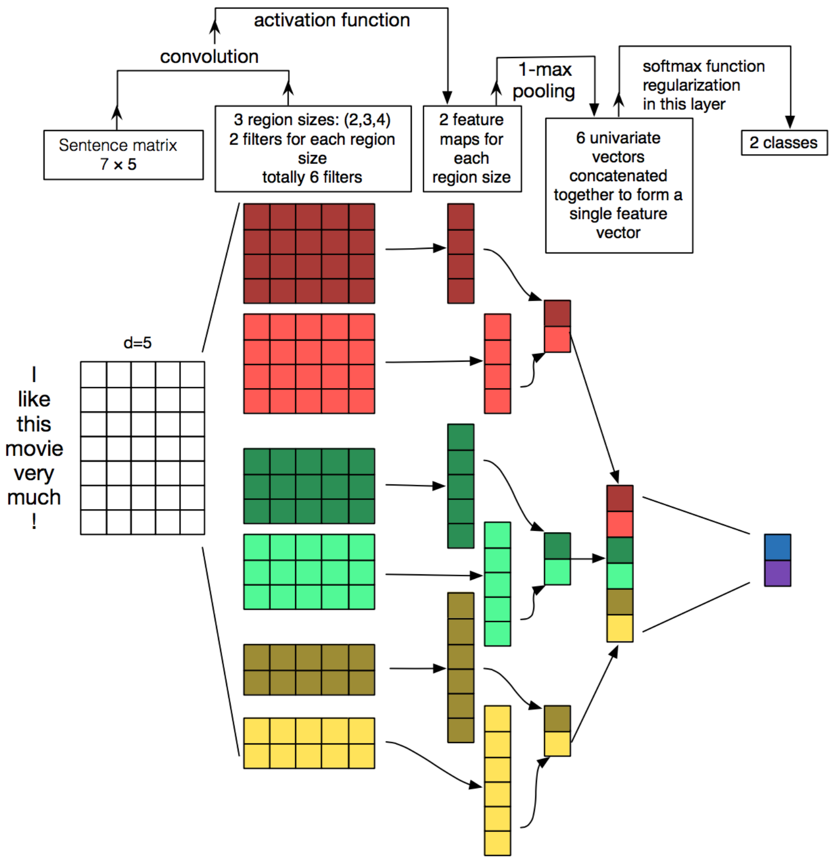


Figure 3.15: Max-pooling.

is a word, or just a letter. Unlike in Computer Vision, in NLP, we generally use filters to slide across the entire row of the matrix. The width of the filters is usually equal to the input matrix, while the height always changes.



activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Figure 3.16: Convolutional Neural Network in NLP.

There are several researches that imported CNN in NLP.

1. Kim, Y. (2014) evaluated the CNN architecture on various datasets, mainly about sentiment analysis and topic classification tasks. CNN has achieved good performance on datasets and has reached new levels in some areas [13].
2. Johnson, R., & Zhang, T. (2015) trained CNN from the beginning without using any pre-trained word vectors such as word2vec or GloVe. It directly applies convolution to one-hot vectors [14].
3. Zhang, Y.et al. (2015) evaluated the influence of hyperparameters in the CNN architecture. They examined the performance and variance by selecting different hyperparameters [15].

3.2.3 Attention Model

The Attention mechanism is a milestone in the history of NLP. Attention model not only domains NLP, but the idea of Attention-based model has profoundly affected the development of NLP. In recent years, more and more research fields of NLP have adopted the attention mechanism [16], such as image processing or speech recognition. Therefore, it is essential for NLP researchers to understand the operating principle of attention mechanisms.

Attention in Nature

Figure 3.17 shows an example of a heat map. It visualizes how humans efficiently allocate limited attention resources when seeing an image. The red area indicates the target of the visual system. It was evident that for the scene shown in this figure, people will pay more attention to it, especially the human face, the title of the text, and the first sentence of the article. This heat map demonstrate the attention mechanism in nature.

Attention in Deep Learning

In essence, the attention mechanism of deep learning is similar to the visual attention mechanism of human beings. The ultimate goal is also to filter out more valuable information from the information that is more important to the current targets.

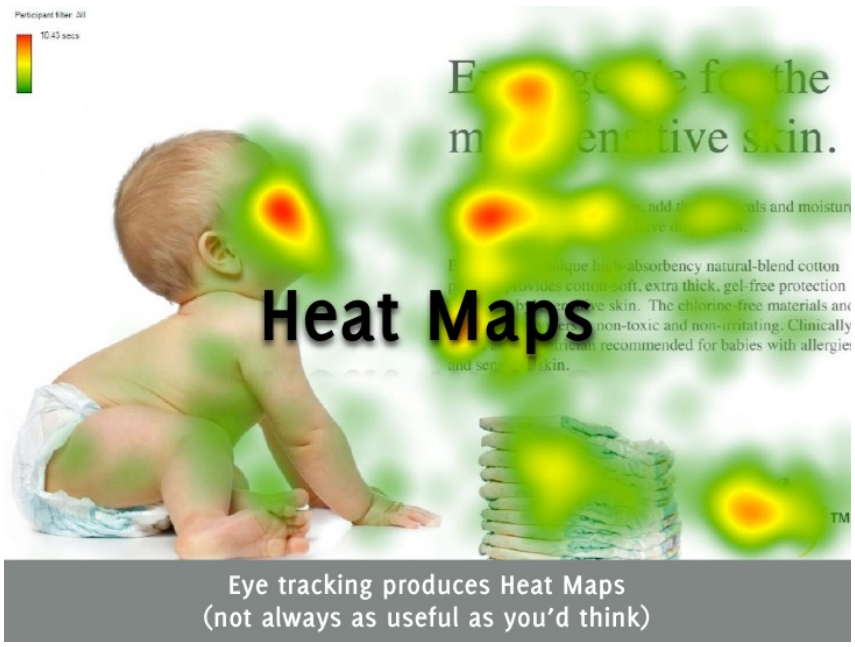

Participant filter: All
10.43 secs
Heat Maps
Eye tracking produces Heat Maps
(not always as useful as you'd think)

Figure 3.17: Heat map.

In *Attention is all you need*, Authors gave a definition of attention as well:
"An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key [17]*."*

Overview of Attention model

In the previous section, we mentioned that RNN is applied to NLP. For the problem of the vanishing/exploding gradient of RNN, we imported "Gate" into the standard RNN to solve long-term dependences (LSTM/GRU) to some extent.

However, we can see the sequence path from the previous LSTM unit to the current unit still exists. In fact, because this path is constantly added and will forget the information, it will become more and more complicated. LSTM, GRU, and its variants can learn a lot of long-term information, but they still have limitations. For longer sentences, it is difficult to convert the input sequence into a fixed-length vector and all valid information can be saved. In short, to LSTM/GRU, the longer the source sentences, the worse the performance of translation [9, 11].

In order to solve the problem above, the attention mechanism was introduced. In the attention model, when we translate the current words, we will look for the corresponding words in the source sentence, and combine the previous translated parts to make the corresponding translation. Figure 3.18 shows a Chinese-English translation by using the Attention mechanism.

As shown in the following figure, when we translate *"knowledge"*, we just focus on the *"knowledge"* part of the original sentence, when we translate *"power"*, just focus on *"power"*. In this way, when the Decoder predicts the target translation, it can see all the information from Encoder, not like what RNN does.

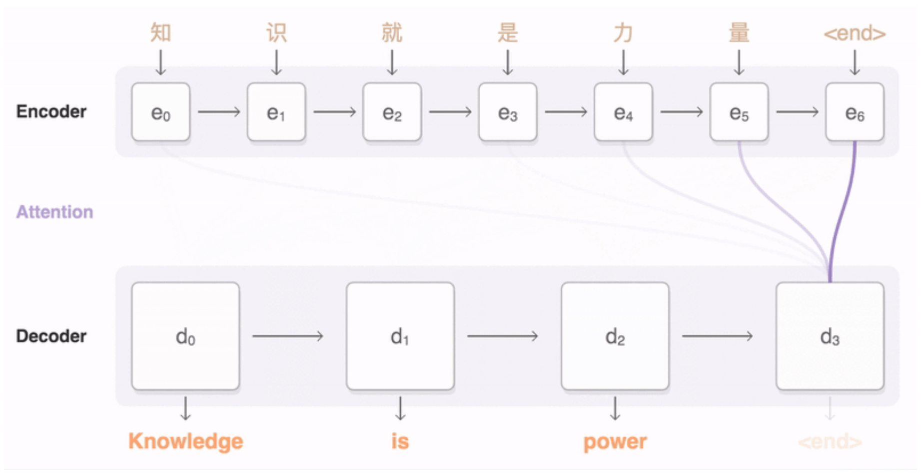


Figure 3.18: Attention mechanism.

Architecture of attention model is shown as Figure 3.19:

There are two separate LSTMs in this model (see diagram on Figure 3.19). At the bottom of the picture, there is a "pre-attention" Bi-LSTM, which is a bidirectional LSTM before the attention mechanism. At the top of the figure, there is a "post-attention" LSTM, which is the LSTM after the attention mechanism. The pre-attention Bi-LSTM goes through T_x time steps; the post-attention LSTM goes through T_y time steps.

Similarly, Attention is a mechanism for improving the effects of the RNN (LSTM or GRU) based on the Encoder-Decoder model. That is why Encoder and Decoder also exist in the Attention model. Here is the analysis for architecture of attention model.

(1) Encoder

1. The Encoder accepts word embedding of every word and the hidden state at the

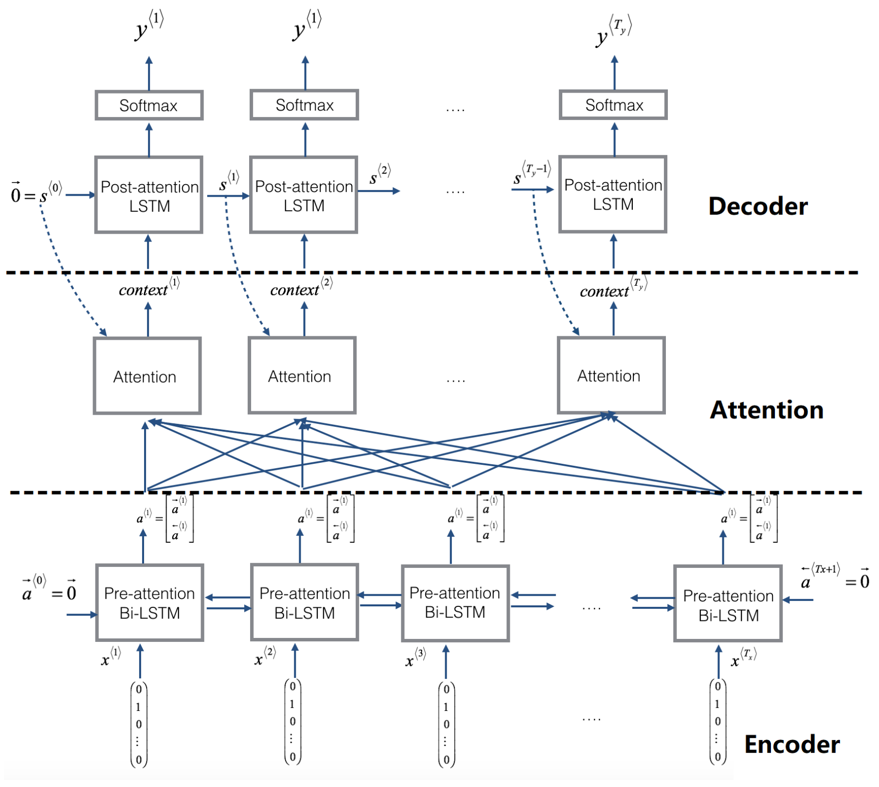


Figure 3.19: Architecture of attention model.

previous moment. The output is the hidden state at the current moment.

$$a_t = Bi - LSTM_{enc}\left(x_t, a_{t-1}\right) \tag{3.27}$$

2. The Encoder accepts word embedding of every word and the hidden state at the previous moment. The output is the hidden state at the current moment.

$$s_t = LSTM_{dec}\left(y_{t-1}^{\wedge}, s_{t-1}\right) \tag{3.28}$$

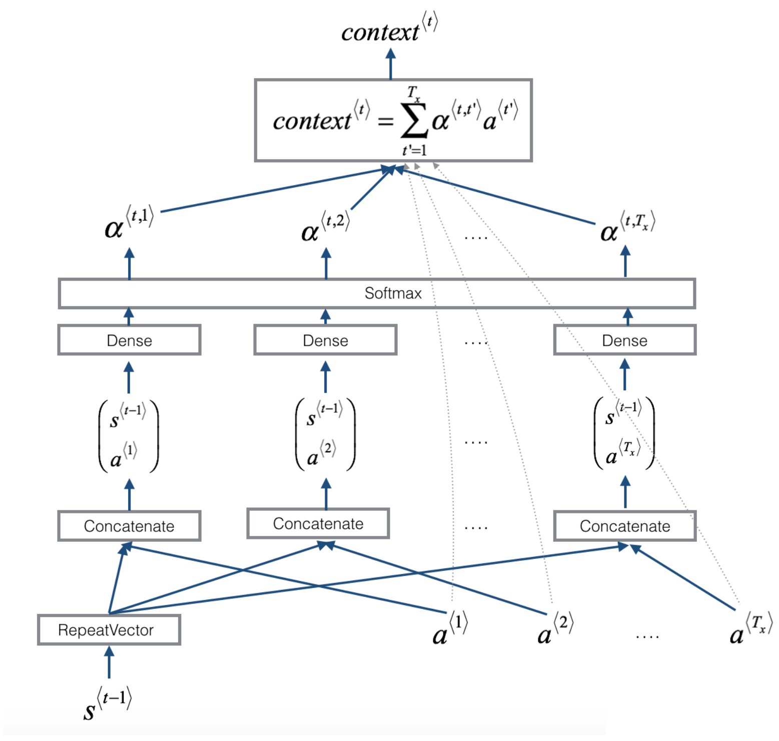
(2) Attention,as shown in Figure 3.20.

3. The Encoder accepts word embedding of every word and the hidden state at the previous moment. The output is the hidden state at the current moment.

$$a^{\langle t' \rangle} = \left[\vec{a}^{\langle t \rangle}; \overleftarrow{a}^{\langle t \rangle}\right] \tag{3.29}$$

4. Concatenate $s^{\langle t-1 \rangle}$ and $a^{\langle t' \rangle}$ and train a small neural to compute $e^{\langle t,t' \rangle}$.

$$e^{\langle t,t' \rangle} = \text{score}\left(s^{\langle t-1 \rangle}, a^{\langle t' \rangle}\right) \tag{3.30}$$



$context^{\langle t \rangle}$
$context^{\langle t \rangle} = \sum_{t'=1}^{T_x} \alpha^{\langle t,t' \rangle} a^{\langle t' \rangle}$
$\alpha^{\langle t,1 \rangle}$
$\alpha^{\langle t,2 \rangle}$
$\alpha^{\langle t,T_x \rangle}$
Softmax
Dense
Dense
Dense
$s^{\langle t-1 \rangle}$, $a^{\langle 1 \rangle}$
$s^{\langle t-1 \rangle}$, $a^{\langle 2 \rangle}$
$s^{\langle t-1 \rangle}$, $a^{\langle T_x \rangle}$
Concatenate
Concatenate
Concatenate
RepeatVector
$a^{\langle 1 \rangle}$
$a^{\langle 2 \rangle}$
$a^{\langle T_x \rangle}$
$s^{\langle t-1 \rangle}$

Figure 3.20: Architecture of Attention.

5. Pass $e^{\langle t,t' \rangle}$ through a softmax to compute $\alpha^{\langle t,t' \rangle}$

$$\alpha^{<t,t'>} = \frac{\exp\left(e^{<t,t'>}\right)}{\sum_{t'=1}^{Tx} \exp\left(e^{<t,t'>}\right)} \tag{3.31}$$

6. Output "context" as the weighted sum of the attention weights.

$$context^{<t>} = \sum_{t'=1}^{T_x} \alpha^{<t,t'>} a^{<t'>} \tag{3.32}$$

(3) Decoder

7. Concatenate context vector and hidden states of Decoder.

$$\hat{s}_t = \tanh\left(W_c\left[context^{\langle t \rangle}; s^{\langle t \rangle}\right]\right) \tag{3.33}$$

8. Compute and output the finial probability.

$$p\left(y_t | y_{<t}, x\right) = \operatorname{softmax}\left(W_s \hat{s}_t\right) \tag{3.34}$$

In summary, the Attention mechanism assigns a weight to each element in the sequence, and the model can be focus on all input information that is important to the next target word so that the model effect is significantly improved. Another advantage of the attention model is that by observing the change of the attention weight matrix, we can better know which part of the source text corresponds to which part of the translation. It helps us to better understand the operating mechanism of the model, as shown in Figure 3.21:

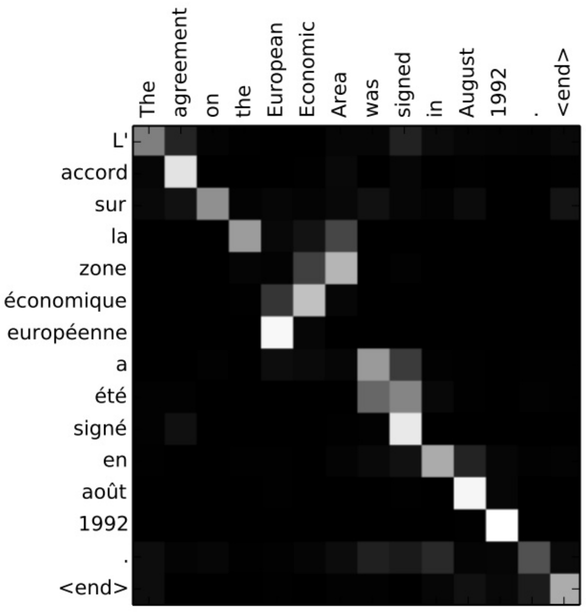


Figure 3.21: Attention distribution between input and output sentences.

Attention gives the model the ability to distinguish the importance between different parts of sentences at different times, making the learning of neural network models softer. Attention itself can be used as an alignment relationship to explain what the model has learned and to provide us with a window to explore the black box of deep learning. At the same time, it will not bring more burden to the computation and storage of the model. That is the reason why the attention mechanism is so widely used.

Of course, Attention also has some defects. The most obvious shortcoming is that the attention mechanism is not a ”distance-aware”. It cannot capture the order of the words (here is the word order, the order of the elements). However, the order of sequence is essential in NLP, and the natural language word order contains too much information. If this information is confirmed, the results will often be discounted. In the end, the attention mechanism is a bag-of-words (BoW) model. Transformer and BERT added a new concept called position-embedding (location vector), we will explain in the next section.

3.2.4 Transformer

Attention is all you need is a paper published by Google that extremely developed the potential of Attention mechanism. This paper proposed a brand new model called Transformer, which completely abandons the CNN and RNN used in deep learning tasks. In this article, Google has proposed many innovative architectures, such as Self-Attention, Multi-Head Attention, which brings new ideas to the NLP industry [17]. Figure 3.22 is the full architecture of Transformer:

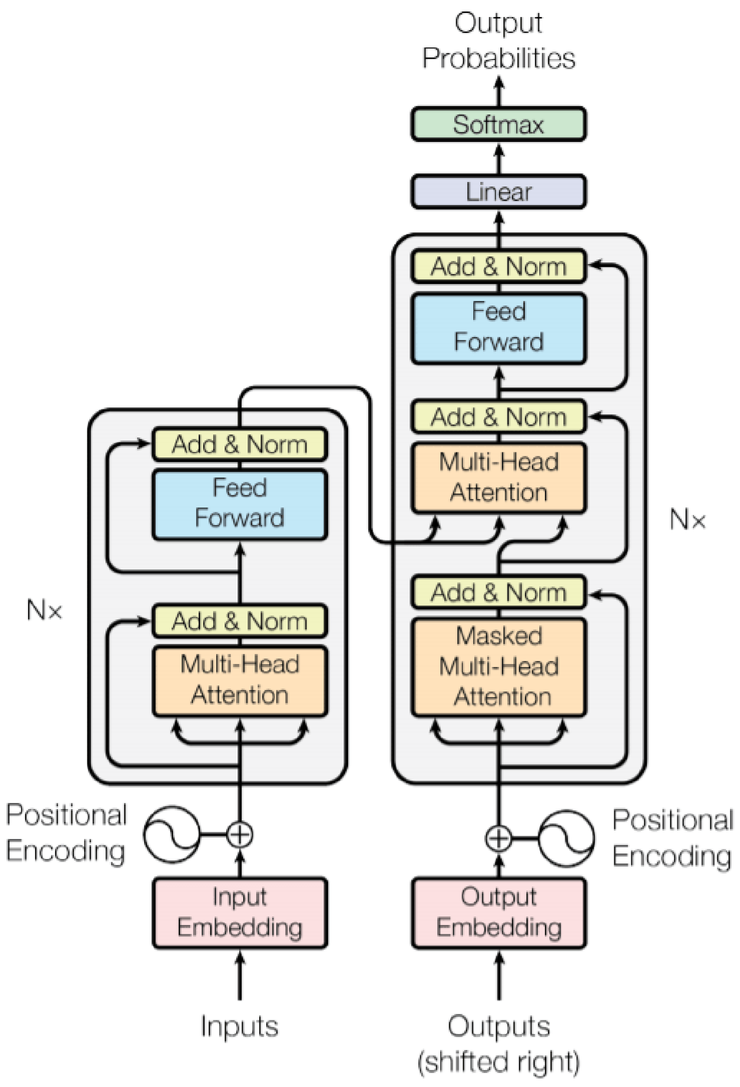


Figure 3.22: The transformer model architecture.

Self-attention

Suppose that there are 4 inputs, now we are going to illustrate how Self-attention work.
1. From x^1 to x^4 is an input sequence. By multiply a matrix, we create the input vectors of the encoder (the embedding of each word) from a^1 to a^4.

$$a^i = Wx^i \tag{3.35}$$

2. Each a^i is multiplied by 3 different weight matrices (W^Q, W^K, and W^V) to create three vectors. Q (Query) refers to matching with other Keys, K (Key) refers to be matched, and V (Value) refers to the extracted information. The dimensions of the three vectors are the same.

$$Q^i = W^Q a^i \tag{3.36}$$

$$K^i = W^K a^i \tag{3.37}$$

$$V^i = W^V a^i \tag{3.38}$$

3. Match Query of each word with Keys of other words to get the "attention" score, then divide each attention by $\sqrt{d_k}$. In self-attention layer, here we adopt the method from the original paper, the algorithm we use to calculate attention score is "scaled dot-product attention". As shown in Figure 3.23, the first word's attention is the result of Q^1 dot-product all Ks.

$$\alpha_{1,i} = Q^1 \cdot K^i/\sqrt{d} \tag{3.39}$$

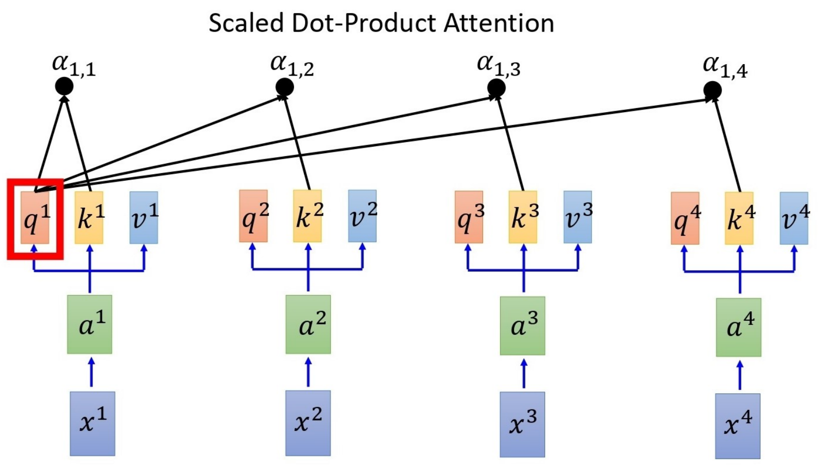


Figure 3.23: Compute self-attention score.

4. Apply a Softmax function to obtain the weights on the values.

$$\hat{\alpha}_{1,i} = \exp\left(\alpha_{1,i}\right) / \sum \exp\left(\alpha_{1,i}\right) \tag{3.40}$$

5. Multiply V^1 to V^4 by $\alpha_{1,1}$ to $\alpha_{1,4}$ respectively as in Figure 3.24, so we get the first output vector of the sequence.

$$b^1 = \sum_i \hat{\alpha}_{1,i} V^i \tag{3.41}$$

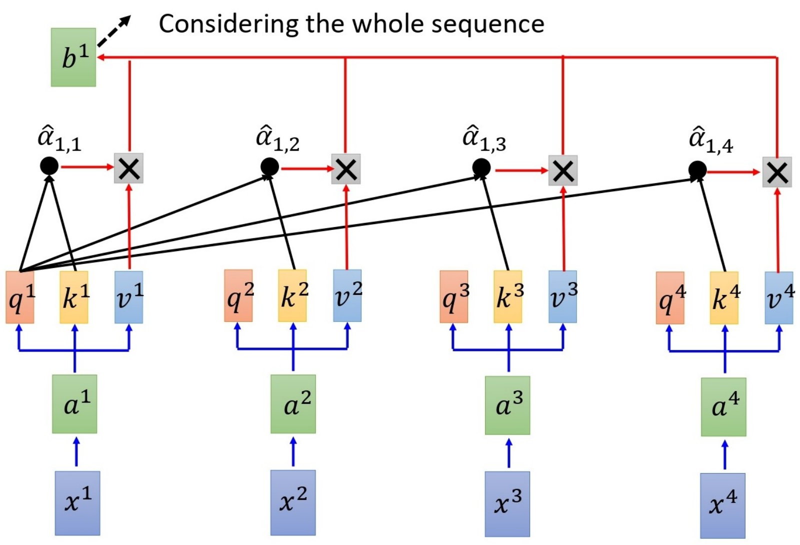


Figure 3.24: Multiply weight and self-attention score to get output.

Conclusively, we compute the matrix of outputs as:

$$\text{Attention}\left(Q, K, V\right) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{3.42}$$

Multi-head self-attention

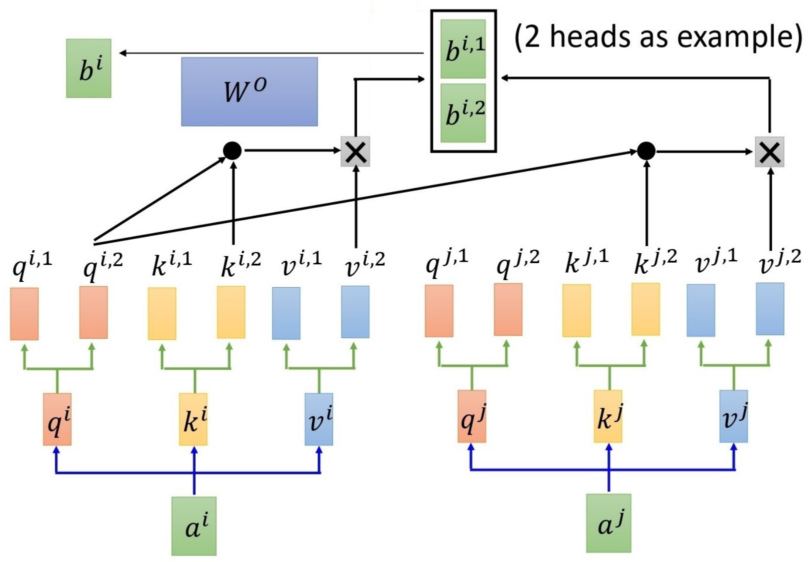
Figure 3.25 shows the architecture of the **Multi-head self-attention**. The reason why we need multi-head is that we have to use the attention mechanism to extract the multiple semantics.Compared to one-head, the computation of multi-head is mostly the same.

1. Each a^i generates Q^i, K^i and V^i. In the case of 2 heads, Q^i will be further split into $Q^{i,1}$ and $Q^{i,2}$, we also implement the same computation to K^i and V^i:

$$\text{head }_i = \text{ Attention }\left(QW_i^Q, KW_i^K, VW_i^V\right) \tag{3.43}$$

2. Do self-attention, only use $Q^{i,1}$ to do dot product for $K^{i,1}$ and $K^{j,1}$, then we get $b^{i,1}$; Similarly, we repeat the same computing of $Q^{i,2}$ with $K^{i,2}$, $K^{j,2}$ to calculate $b^{i,2}$; Finally we concatenate $b^{i,1}$ and $b^{i,2}$ together and multiply by a W^O matrix to output the final result:

$$\text{MultiHead}(Q,K,V) = \text{Concat}_i(\text{head}_i)W^O \tag{3.44}$$



(2 heads as example)
b^i
W^O
$b^{i,1}$
$b^{i,2}$
$q^{i,1}$ $q^{i,2}$ $k^{i,1}$ $k^{i,2}$ $v^{i,1}$ $v^{i,2}$
$q^{j,1}$ $q^{j,2}$ $k^{j,1}$ $k^{j,2}$ $v^{j,1}$ $v^{j,2}$
q^i k^i v^i
q^j k^j v^j
a^i
a^j

Figure 3.25: Multi-head Self-attention.

Positional encoding

So far, there is a lack of a method to indicate the order of words from the input sequence in the Transformer. To solve this problem, the Transformer adds an additional vector called Positional Encoding to the input of Encoder and Decoder layers. The dimensions are setting the same with the embedding dimensions. Each location has a unique positional encoding, but not learned from training. The calculation method is as follows:

$$PE(\text{pos}, 2i) = \sin\left(\text{pos}/10000^{2i/d_{\text{model}}}\right) \tag{3.45}$$

$$PE(\text{pos}, 2i+1) = \cos\left(\text{pos}/10000^{2i/d_{\text{nodal}}}\right) \tag{3.46}$$

Where *pos* is the position of the current word in the sentence, i is the index of each value in the vector. It can be seen that in the even position, *sinusoidalcoding* is used, and in odd positions, *cosinecoding* is used.

Add & Normalize

Figure 3.26 shows a sub-layer (self-attention, feed-forward neural network) in encoder and decoder that has a residual connection, and it is followed by a layer-normalization step.

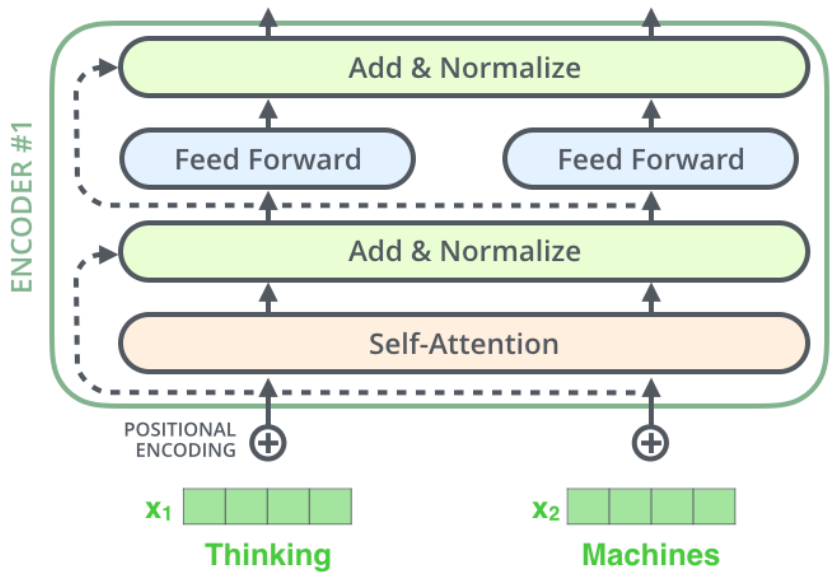


Figure 3.26: A sub-layer of Encoder in Transformer.

Residual connection (Add):

In the previous step, we obtained the V that has been weighted by the attention matrix, which is Attention $(Q,\ K,\ V)$. In the residual layer, we add the output of self-attention with input together to gain $X_{\text{embedding}}$.

$$X_{attention} = A + Attention(Q, K, V) \tag{3.47}$$

Layer Norm:

The purpose of Layer Norm is to normalize the hidden layers in the neural network into the standard normal distribution so that speeding up the training and accelerating the convergence:

1. For $X_{\text{attention}}$,calculate the mean of matrix per row:

$$\mu_i = \frac{1}{m}\sum_{i=1}^{m} x_{ij} \tag{3.48}$$

2. For $X_{\text{attention}}$,calculate the variance of matrix per row:

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_{ij}-\mu_j)^2 \tag{3.49}$$

3. Subtract out the mean per rows, then divide by the standard deviation of this row to get the normalized value:

$$\text{LayerNorm}\,(x) = \alpha * \frac{x_{ij}-\mu_i}{\sqrt{\sigma_i^2+\epsilon}} + \beta \tag{3.50}$$

ϵ here is added for numerical stability (prevent divided by 0). α and β, these two parameters are used to compensate for the information loss during the normalization processing [18].

Architecture analysis of Transformer

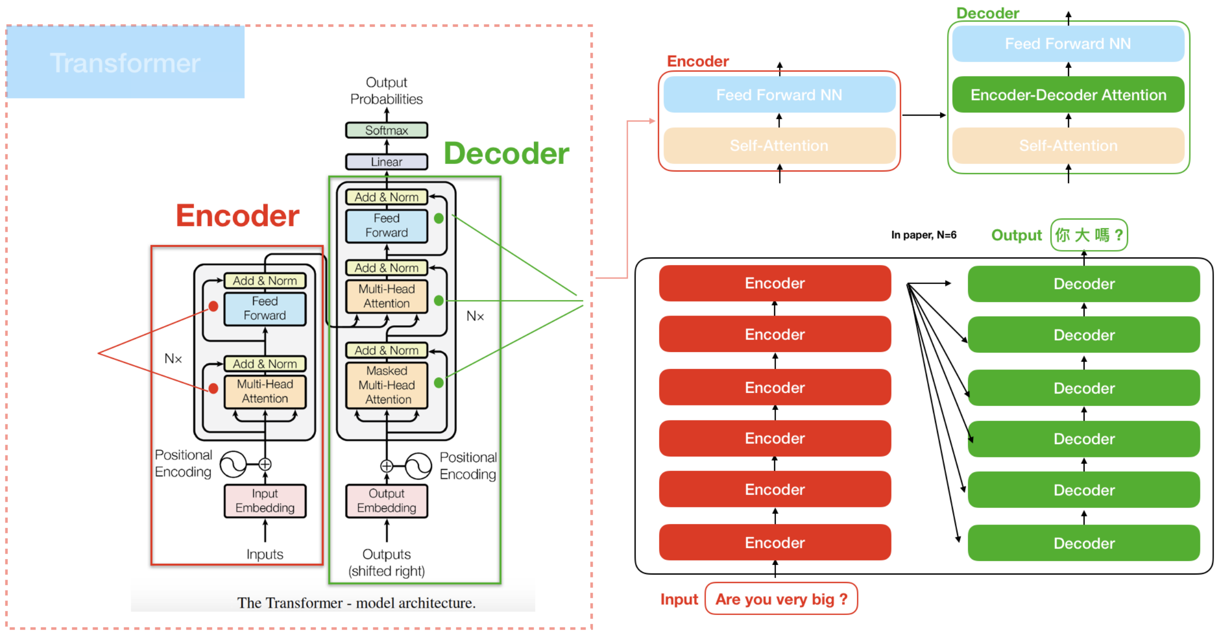


Figure 3.27: Illustration of Transformer model.

Figure 3.27 shows the illustration of Transformer model. This model consists of two parts, an Encoder and a Decoder. Further analysis of this model reveals that each Encoder consists of six encoders (same with the original paper). Each Decoder is also composed of six decoders.

Transformer Encoders:

For each Encoder in Encoders, their structures are the same, but they do not share weights. If we expand each sub layers in the encoders, we can get Figure 3.28:

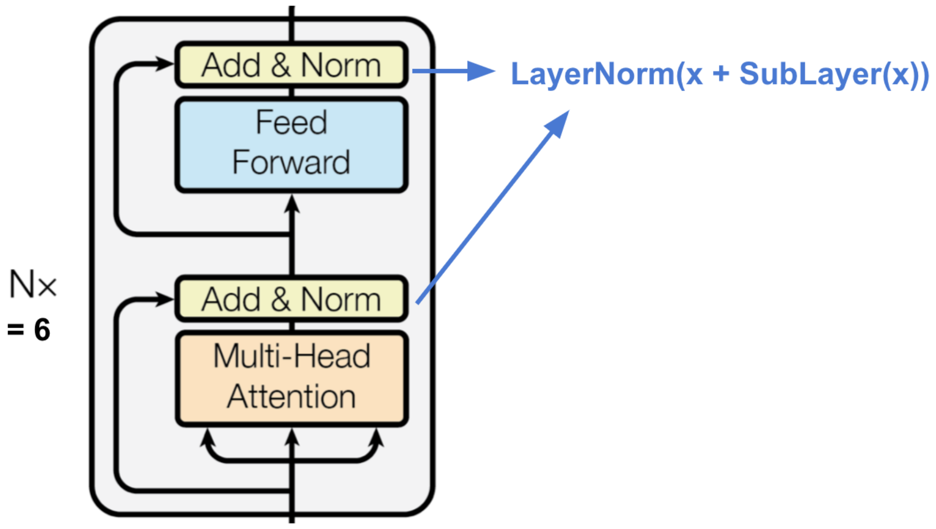


Figure 3.28: Encoder of Transformer.

- sub-layer-1: **Multi-head self-attention mechanism**: To implement self-attention.
- sub-layer-2: **Feed forward Networks**: A simple fully connected neural network, using *ReLU* as activation function:

$$\mathrm{FFN}(x)=\max\left(0, x W_1+b_1\right) W_2+b_2 \tag{3.51}$$

$$\text{LayerNorm}\ (X+\ \text{sublayer}\ (X)) \tag{3.52}$$

Transformer Decoder:

For each Decoder in Decoders, except the first sublayer, the rest of their structures are the same, also they do not share weights. Multi-head Self Attention of the first sublayer is changed into Mask multi-head self-attention. We will illustrate in the following contents. If we expand each sub layers in the encoders, we can get Figure 3.29:

- sub-layer-1: **Masked multi-head self-attention mechanism**: To implement self-attention. The first multi-head attention sub-layer is modified to prevent positions from attending to subsequent positions, which means decoder only implement self-attention to the generated sequence.

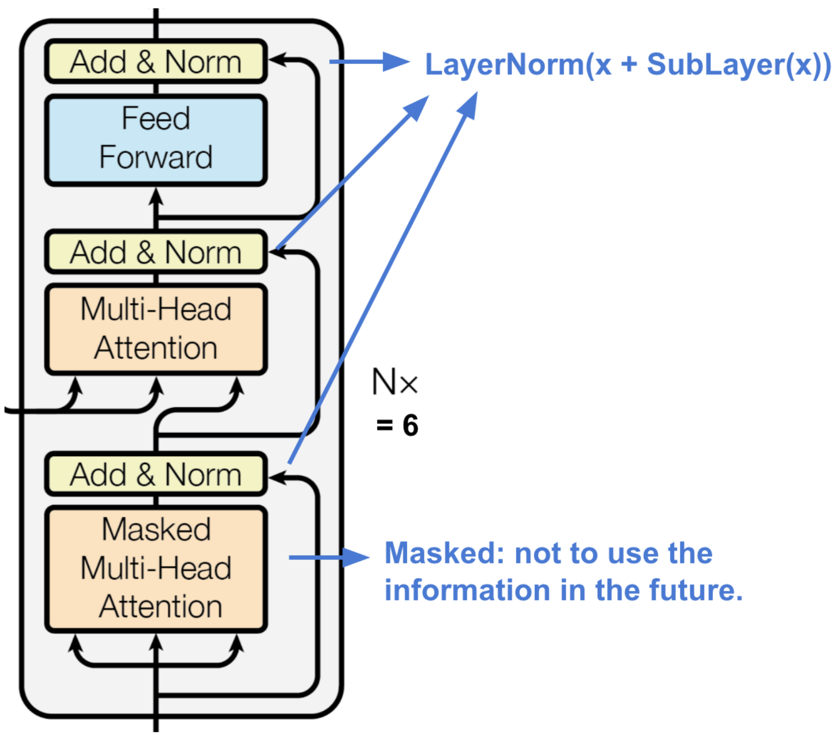

Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx = 6
LayerNorm(x + SubLayer(x))
Masked: not to use the information in the future.

Figure 3.29: Decoder of Transformer.

- sub-layer-2: **Multi-head self-attention mechanism**: same with Encoder.
- sub-layer-3: **Feedforward Networks**: same with Encoder.

Chapter 4

Analysis & Experiments

In the previous sections, we have analyzed the architecture of RNN, CNN and Transformer. This chapter reports the comparison of three feature extractors based on analysis and literature review. Moreover, we adopt published models to reimplement experiments, and the result supports our conclusion of comparison in Section 4.2. About the experiment, We first introduce the baseline dataset for testing the performance of sentiment analysis in Section 4.1. Secondly, we show some experimental settings and configurations of deep learning models on Section 4.3. Lastly in Section 4.4 shows the empirical results of two deep learning models we choose from Table 4.4. Additionally, we use a extra dataset to evaluate their performance.

4.1 Baseline Dataset

1. Task 4 of SemEval-2014 datasets [1] [19] are distributed for Aspect Based Sentiment Analysis (ABSA), which including Laptops and Restaurants two parts:

    - Restaurant dataset consists of 4728 English sentences from the restaurant reviews of Ganu et al. (2009) [20].
    - Laptop dataset consists of 2996 English sentences extracted from customer reviews of laptops.

2. ACL 14 Twitter dataset gathered by Dong et al. (2014) [21]. They are annotated

[1]http://alt.qcri.org/semeval2014/task4/

with the negative, neutral, positive classes account for 25%, 50%, 25%, respectively. Training data consists of 6,248 tweets, and testing data has 692 tweets.

All these datasets above were labeled the aspect terms and polarities by experienced human annotators. Table 4.1 shows the statistics of the datasets all we use in the experiments.

Dataset	**Positive**		**Neural**		**Negative**	
	Train	Test	Train	Test	Train	Test
Restaurant	2164	728	637	196	807	196
Laptop	994	341	464	169	870	128
Twitter	1561	173	3127	346	1560	173

Table 4.1: Statistics of the datasets.

4.2 Comparison of three feature extractors

4.2.1 Empirical evidence

Semantic extraction capabilities

Model	DE→EN				DE→FR		
	PPL	2014	2017	Acc(%)	PPL	2012	Acc(%)
RNNS2S	5.7	29.1	30.1	84.0	7.06	16.4	72.2
ConvS2S	6.3	29.1	30.4	82.3	7.93	16.8	72.7
Transformer	**4.3**	**32.7**	33.7	**90.3**	**4.9**	**18.7**	**76.7**
uedin-wmt17	–	–	**35.1**	87.9	–	–	–
TransRNN	5.2	30.5	31.9	86.1	6.3	17.6	74.2

Table 4.2: The results of different architectures on *newstest* sets and *ContraWSD*. *Acc* means accuracy on the test set.

Table 4.2 compares the accuracy of machine translation in different models. From the perspective of semantic feature extraction capabilities, the current experiment supports the following conclusions: Transformer's performance significantly better than RNN and CNN, and RNN and CNN are not much different [22].

Long-term feature capture capabilities

Model	2014	2017	PPL	Acc(%)
RNNS2S	23.3	25.1	6.1	95.1
ConvS2S	23.9	25.2	7.0	84.9
Transformer	**26.7**	**27.5**	**4.5**	**97.1**
RNN-bideep	24.7	26.1	5.7	96.3

Table 4.3: The results of different NMT models, including the accuracy of long-range dependencies.

Table 4.3 compares the Long-term feature capture capabilities in a specific subject-verb agreement task (such as **I...am..., we...are...**). The experiment supports the following conclusion: The native CNN feature extractor is significantly weaker than RNN and Transformer. The transformer is slightly better than the RNN model (especially when the subject-predicate distance is less than 13); but at a relatively long distance (The subject-predicate distance is greater than 13), RNN is weaker than Transformer. Comprehensively, it can be considered that the Transformer and RNN are not too weak in this respect, while CNN is significantly weaker than the first two [22].

Parallel computing capabilities

We have discussed three models in the previous chapters, and now we only summarize them here. The conclusion is as follows:

The linear sequence dependency of RNN is very suitable for solving NLP tasks, but it is this linear sequence dependency that also limit the parallel computing capability of RNN. It seems difficult. For CNN and Transformer, because they do not have dependencies on the input of different time steps in the intermediate state of the network, they can be very convenient and free to do parallel computing transformation.

4.2.2 Previous research in Sentiment Analysis

We have collected some of the representative models for aspect-based sentiment analysis task used the same dataset from 2015-2019.

	Authors	Year	Models	Laptop(Acc%)	Restaurant(Acc%)
RNN	Tang et al.	2015	TD-LSTM	68.13	75.63
	Wang et al.	2016	ATAE-LSTM	68.70	77.20
	Ma et al.	2017	IAN	72.10	78.60
	Peng et al.	2017	RAM	74.49	80.23
	Binh et al.	2018	BBLSTM-SL	74.90	81.30
CNN	Huang et al.	2018	PF-CNN	72.43	75.73
	Xue et al.	2018	GCAE	69.14	77.28
Transformer	Song et al.	2019	AEN-BERT	**79.93**	83.12
	Song et al.	2019	BERT-SPC	78.99	**84.46**
	Cheng et al.	2018	ASVAET	75.34	81.10

Table 4.4: The result of different models in Task 4 of SemEval-2014 datasets.

Table 4.4 shows some neural network models for aspect-based sentiment analysis task from 2015 to 2019. All these models are tested in Task 4 of SemEval-2014 datasets.

So far, we could not search any standard RNN, CNN or Transformer model, and all of these models are one kind of variant of standard RNN/CNN and Transformer. Here we conclude LSTM/GRU as RNN-based models. Similarly, Gated CNN as CNN-based and BERT or other variants of Transformer Transformer-based models.

From the table above, it suggests show that Transformer-based models have state-of-the-art performance in aspect-based sentiment analysis. Also, we notice that most of the models above adopt Attention mechanism [16] to enhance their capability to capture information from input. Moreover, no matter Attention on RNN/CNN, or Self-Attention, it is obviously that Attention mechanism has significantly affected the development of Natural Language Processing.

4.3 Experiment settings & Configuration

Additionally, we implement the experiment to testify the performance of the BERT-based model. The models we used are AEN-BERT [2] and BERT-SPC [23] above.

We use Python[3] 3.7.5, PyTorch[4] 1.3.0, PyTorch-Transformers[5] 1.2.0 and some build-in libraries for NLP, such as Sklearn, Numpy... as the environment for coding and running the experiments.The configurations of these two models are same and shown as below:

- optimizer: Adam
- learning rate: 2e-05
- dropout: 0.1
- L2 regression: 0.01
- epochs: 10
- batch size: 16
- embed dimension: 300
- pretrained bertname: bert-base-uncased
- max sequence length: 80

4.4 Experiment Results

Table 4.5 and Table 4.6 show our experiment results of Accuracy and F1 score respectively. The values are somewhat lower compared with Table 4.4 above which indicates that we need to extensively test with more hyperparameter settings. However, this confirm the state-of-the-art performance on BERT-based model compared with other non-BERT models.

[2]https://github.com/songyouwei/ABSA-PyTorch
[3]https://www.python.org/
[4]https://www.pytorch.org
[5]https://pytorch.org/hub/huggingface_pytorchtransformers/

Models	Laptop(Acc%)	Restaurant(Acc%)
AEN-BERT	**78.68**	81.79
BERT-SPC	77.74	**83.93**

Table 4.5: Accuracy of AEN-BERT and BERT-SPC

Models	Laptop(F1%)	Restaurant(F1%)
AEN-BERT	73.59	70.94
BERT-SPC	73.33	76.93

Table 4.6: F1 Score of AEN-BERT and BERT-SPC

We further trained these two BERT-based models with the Twitter data, with target-dependent twitter sentiment classification. This is a difficult dataset to deal with not only because the irregularities in human tweets but also the tweets themselves are very short. Figure 4.1 and 4.2 shows the Loss and Accuracy on AEN-BERT and BERT-SPC respectively.

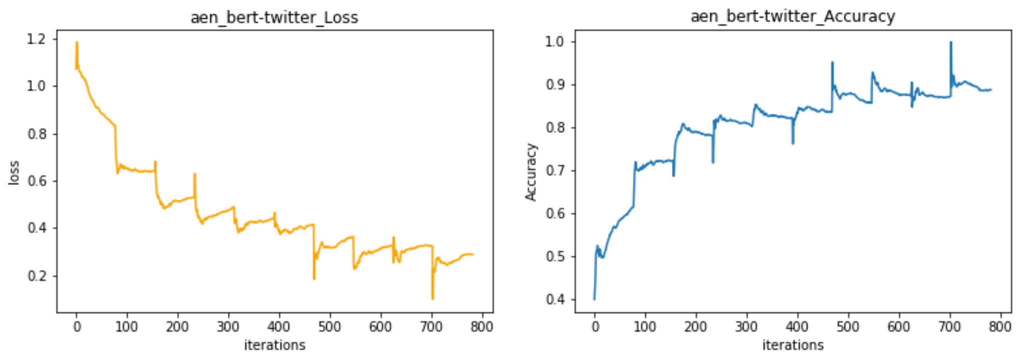


Figure 4.1: Loss and Accuracy of AEN-BERT on Twitter dataset.

The results are shown as in Table 4.7. The BERT-based models present a very strong performance compared with the state-of-the-art for this task such as Dong et al., 2014 (71.1% Acc, 69.9% F1) [21] or Tang et al., 2016 (68.5% Acc, 66.9% F1) [24]:

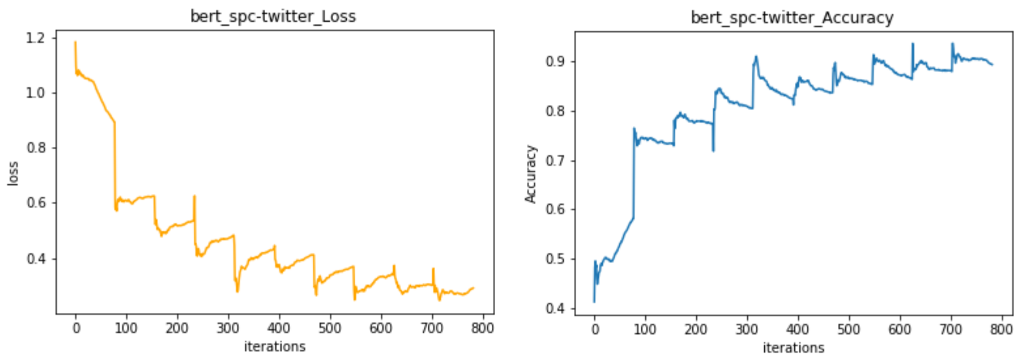


Figure 4.2: Loss and Accuracy of BERT-SPC on Twitter dataset.

Models	Twitter(Acc%)	Twitter(F1%)
AEN-BERT	72.69	70.23
BERT-SPC	73.27	70.77

Table 4.7: Performance of AEN-BERT and BERT-SPC on Twitter dataset

4.5 Conclusion of comparison

We compare the RNN / CNN / Transformer from several different angles and quoted the experimental data in detail. Here is the conclusions:

1. Theoretically, In terms of the comprehensive performance, Transformer is significantly better than CNN, and CNN is slightly better than RNN. In terms of speed, Both Transformer and CNN are is better than RNN. In summary, the Transformer is the best feature extractor among the three models.

2. Practically, we have investigated the effect on aspect-based sentiment classification task. The result also corresponds with our theoretical analysis. Moreover, no matter Attention on RNN/CNN, or Self-Attention, it is obviously that the Attention mechanism has significantly affected the development of Natural Language Processing.

Chapter 5

Conclusions

5.1 Summary

Natural language processing (NLP) is an area cover computer science, artificial intelligence, and linguistics. It focuses on the interaction between computers and human language. Therefore, research of NLP always involves natural language.

The input to NLP is often a sentence or an article, so it has several features:

1. The input is a one-dimensional linear sequence.
2. The length of the input is not stable, maybe short or long.
3. The order of sequence is essential. Sometimes, the interchange of the two words can lead to completely different meanings.

Therefore, the performance in NLP tasks mostly depends on whether the feature extractor can effectively extract features from an input sequence. The evolutionary history of natural language processing also is the evolution of feature extraction techniques.

In this research project, we have surveyed the development of deep learning models in aspect-based sentiment analysis. The survey is done by two parts, the methods to implement DNNs and the DNNs in aspect level sentiment classification. From this survey, we can know in recent years, as the development of the neural network,the performance of deep neural networks are getting better and better than before, it means that the computer can have a better performance on understanding and extracting valuable information from input sequences.

Based on this survey, as a feature extractor, the Transformer is significantly better than RNN and CNN. Recently, the BERT model based on Transformer architecture has profoundly affected the research and application mode of future NLP because of its excellent effect and versatility [25].

In the era of post-BERT, the new approach to solve NLP tasks has become a two-step process:

1. Train language models on large unlabeled text corpora (unsupervised or semi-supervised).

2. Fine-tune this large model to a specific NLP task and train the model (supervised).

From the perspective of model or method, BERT quotes from ELMO, CBOW and other ideas, mainly proposed masked language model and Next Sentence Prediction. However, the most valuable achievement in BERT is that it developed the idea of Pre-training/Fine-tuning training steps in NLP. BERT model itself didn't propose any significant innovation of NLP architecture. It is more similar to the fusion of major achievements of NLP in recent years. Even so, BERT model proved that the effect is good and versatility is high. Most NLP tasks can benefit from a two-step solution, and the effect should be significantly improved. It is foreseeable that Transformer will dominate in the NLP applications in the near future

5.2 Future Work

In our future research in the doctor course, we would like to try to deepen our research on aspect-based sentiment analysis, especially on aspect term extraction. Meanwhile, we will try to apply BERT model to appropriate for unsupervised learning.

Bibliography

[1] B. Liu, "Sentiment analysis and opinion mining," *Synthesis lectures on human language technologies*, vol. 5, no. 1, pp. 1–167, 2012.

[2] L. Zhang, S. Wang, and B. Liu, "Deep learning for sentiment analysis: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1253, 2018.

[3] H. P. Luhn, "A statistical approach to mechanized encoding and searching of literary information," *IBM Journal of research and development*, vol. 1, no. 4, pp. 309–317, 1957.

[4] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, 1972.

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[6] X. Rong, "word2vec parameter learning explained," *arXiv preprint arXiv:1411.2738*, 2014.

[7] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[8] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[9] Y. Bengio, P. Simard, P. Frasconi, *et al.*, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[12] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[13] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[14] R. Johnson and T. Zhang, "Effective use of word order for text categorization with convolutional neural networks," *arXiv preprint arXiv:1412.1058*, 2014.

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

[16] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[18] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[19] M. Pontiki, D. Galanis, H. Papageorgiou, I. Androutsopoulos, S. Manandhar, M. Al-Smadi, M. Al-Ayyoub, Y. Zhao, B. Qin, O. De Clercq, *et al.*, "Semeval-2016 task 5: Aspect based sentiment analysis," in *10th International Workshop on Semantic Evaluation (SemEval 2016)*, 2016.

[20] G. Ganu, N. Elhadad, and A. Marian, "Beyond the stars: improving rating predictions using review text content.," in *WebDB*, vol. 9, pp. 1–6, Citeseer, 2009.

[21] L. Dong, F. Wei, C. Tan, D. Tang, M. Zhou, and K. Xu, "Adaptive recursive neural network for target-dependent twitter sentiment classification," in *Proceedings of the 52nd annual meeting of the association for computational linguistics (volume 2: Short papers)*, pp. 49–54, 2014.

[22] G. Tang, M. Müller, A. Rios, and R. Sennrich, "Why self-attention? a targeted evaluation of neural machine translation architectures," *arXiv preprint arXiv:1808.08946*, 2018.

[23] Y. Song, J. Wang, T. Jiang, Z. Liu, and Y. Rao, "Attentional encoder network for targeted sentiment classification," *arXiv preprint arXiv:1902.09314*, 2019.

[24] D. Tang, B. Qin, and T. Liu, "Aspect level sentiment classification with deep memory network," *arXiv preprint arXiv:1605.08900*, 2016.

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