國立政治大學應用數學系

碩士學位論文

Transformer 應用於中文文章摘要
Using Transformer for Chinese Article Summarization

指導教授：蔡炎龍　博士

研究生：林奕勳　撰

中 華 民 國 111 年 6 月

DOI:10.6814/NCCU202200797

中文摘要

自從 Transformer 發表後，無疑為自然語言處理領域的立下新的里程碑，許多的模型也因應而起，分別在各自然語言處理項目有傑出的表現。如此強大的模型多數背後依靠巨量的參數運算，但各模型皆以英文為發展主軸，我們很難訓練一個一樣強的中文模型，在缺乏原生中文模型的情況下，我們利用現有的資源及模型訓練機器做中文文章摘要，使用 BERT 及 GPT-2，搭配中研院中文詞知識庫小組的中文模型，並採用新聞資料進行訓練。先透過 BERT 從原文章獲得抽取式摘要，使文章篇幅縮短並保留住重要資訊，接著使用 GPT-2 從抽取過的摘要中再進行生成式摘要，去除掉重複的資訊並使語句更平順。在我們的實驗中，我們獲得了不錯的中文文章摘要，證明這個方法是有效的。

關鍵字：Transformer、BERT、GPT-2、中文文章摘要、抽取式摘要、生成式摘要、深度學習

DOI:10.6814/NCCU202200797

Abstract

Since the publication of Transformer, it has undoubtedly set a new milestone in the field of Natural Language Processing, and many models have also been released depending on it and performed outstandingly in various Natural Language Processing tasks. Most of such powerful models rely on a large number of parameter operations, but most of them are developed in English, and it is difficult for us to train a Chinese model that is equally strong. In the absence of native Chinese models, we use existing resources and model to train the machine to make Chinese article summaries: using BERT and GPT-2 model, with the Chinese model of the Chinese Knowledge and Information Processing of the Academia Sinica of Taiwan, and using news datasets for training. First, use BERT to obtained an extractive summarization from the original article, so that the length of the article is shortened and important information is retained, then use GPT-2 to generate a summarization from the extracted summary to remove duplicate information and make the sentence smoother. In our experiments, we obtained decent Chinese article summaries, proving that this method is effective.

Keywords: Transformer, BERT, GPT-2, Chinese article summarization, Extractive summarization, Abstractive summarization, Deep learning

DOI:10.6814/NCCU202200797

Contents

中文摘要 i

Abstract ii

Contents iii

List of Figures v

1 Introduction 1

2 Deep Learning 2

2.1 Neurons and Neural Networks 4

2.2 Activation Function 6

2.3 Loss Function 8

2.4 Gradient Descent Method 10

3 Word Embeddings 12

3.1 Word2Vec 12

3.2 GloVe 13

3.3 FastText 14

4 Transformer 16

4.1 Embeddings 16

4.2 Encoder 18

4.3 Decoder 22

DOI:10.6814/NCCU202200797

5 Contextualized Word Embeddings **24**

5.1 ELMo 24

5.2 BERT 25

5.3 GPT-2 27

6 Summarization **28**

6.1 Two methods of summarization 28

6.2 TextRank 29

6.3 BERTSUM 30

7 Experiments **33**

7.1 Data Preparation 34

7.2 Extractive Summarization with BERTSUM 34

7.3 Abstractive Summarization with GPT-2 35

7.4 Result 36

8 Conclusion **39**

Bibliography **40**

DOI:10.6814/NCCU202200797

List of Figures

2.1 Three steps of deep learning 3
2.2 (a)Biological Brain; (b)Architecture of NN 4
2.3 The Operation of a Neuron 4
2.4 Fully Connected Feedforward Network 5
2.5 Sigmoid function 7
2.6 Hyperbolic tangent (tanh) 7
2.7 Rectified linear unit (ReLU) 8

4.1 Difference of without or with teacher forcing 16
4.2 Encoder in Transformer 18
4.3 Decoder in Transformer 22

5.1 The architecture of pre-trained model in BERT 26

6.1 Two methods of summarization 28
6.2 Structure of Bert for Summarization 31

7.1 Pipeline of experiments 33
7.2 Pipeline of abstractive summarization with GPT-2 35
7.3 Example result of experiments 36
7.4 Example result of experiments 37
7.5 Example result of experiments 37

DOI:10.6814/NCCU202200797

Chapter 1

Introduction

In recent years, artificial intelligence is used in various fields, whether in image recognition, semantic analysis, recommendation systems, image creation, etc., and has achieved impressive results even better than humans. For example, Go is an abstract strategy board game with innumerable ways to play, even computer cannot completely simulate all the steps with the development now. However, in 2017, the artificial intelligence of playing Go named AlphaGo [28], which is created by DeepMind, defeated Ke Jie the number one ranked Go player in the world at the time, and AlphaGo cost only about two years to surpassed human performance. With the help of artificial intelligence, people can save a lot of time to do some tedious things, or even humans can't do it.

Transformer is a relatively new model in deep learning released by in 2017. The difference from Recurrent Neural Networks (RNN) model and Long Short-Term Memory (LSTM) model, it only uses self-attention mechanism, which make model recognize things more like a human. Nowadays, there are lots of model have been developed depending on Transformer: BERT, GPT, XLNet, BART, T5, etc., each with its own field of expertise.

Because of most models only support English. In this paper, we try to make a summary of Chinese article with two kinds of model, BERT and GPT-2, in extractive and abstractive ways, then train the model with Chinese news data.

DOI:10.6814/NCCU202200797

Chapter 2

Deep Learning

How to let the computers work smarter or more like human being than before? It's a big issue we want to figure out. Machine learning which represent that machine learn by itself. More accurate, Machine learning is to find the best function for the training data. Deep learning is a method of machine learning which depends on neural networks (NN) [7] [11] [12]. It can be roughly divided into three types, supervised, semi-supervised, and unsupervised learning [3] [27]. Here are some examples of how deep learning works.

1. Image Recognition

   $$f(\text{“ Image of a dog ”}) = \text{“ dog ”}$$

   When we use deep learning to recognition images [8], the image will be restructure into matrices represented by number. Then, the machine will find the characteristic of the image and classify it into the correct category. The image of dog is the input of the function, and will become the category “dog” as the output of the function.

2. Stock Price Prediction

   $$f(\text{“ Stock price for the past 20 days ”}) = \text{“ Stock price for the next day ”}$$

   If we want to use deep learning to predict the future price of stock, we will give the machine the target date, then the machine will find some relations or trends of prices and news from past time. Actually, machine get the date as the input, but learn from the past information, and give the prediction of price as the output.

DOI:10.6814/NCCU202200797

3. Playing Game

$$f(\text{“ Current chess game ”}) = \text{“ Knight moves to D-3 ”}$$

We use deep learning to predict the next step of game. Give the situation of game as input, then the machine will calculate numerous steps as it can and find the way with highest probability of winning. Now, the most famous deep learning model of playing games is AlphaGO created by DeepMind which beat Lee Sedol one of the best players at Go.

4. Translation

$$f(\text{“ 你好世界 ”}) = \text{“ Hello World ”}$$

When we want to solve the problem of languages with deep learning, the most difficult part is getting the machine to understand what texts or sentences mean. In translation [2], it is most important to preserve the original meaning in the assigned language. In the above case, we give the sentence in Mandarin as input and let machine find the corresponding sentence in English as output.

Now, there are lots of applications with deep learning in our life. To realize how deep learning works, we can take the following three steps in the figure below to understand.

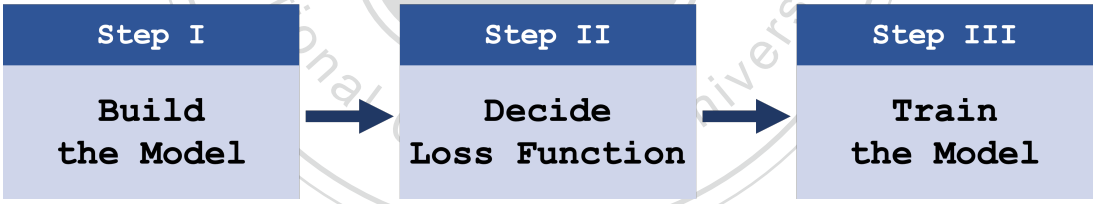

Figure 2.1: Three steps of deep learning

We can divide the process of deep learning into three steps. First, construct a neural network model and set a function determined by the structure of network. Then, select a suitable loss function which can identify the most proper function by evaluating all of the candidates. Finally, let the machine train by itself, the machine will learn from training data and be modified itself to get the best function.

DOI:10.6814/NCCU202200797

2.1 Neurons and Neural Networks

Neural network (NN) is the most important core of deep learning. Neural network is similar to human brain neurons, it is constructed by many connected computing units, like biological neurons, that send signals to others. The following section will explain how neural networks work.

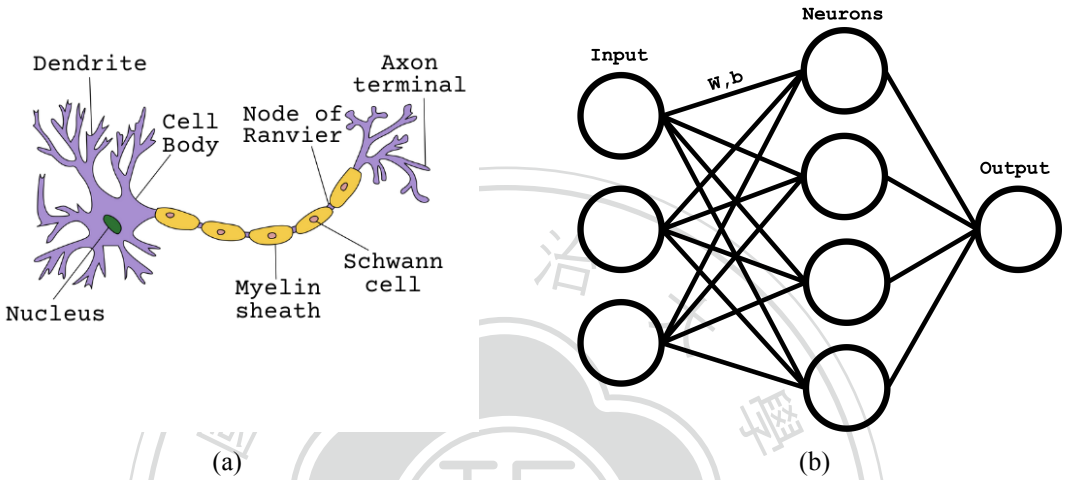

Figure 2.2: (a)Biological Brain; (b)Architecture of NN

Every computing unit which called neuron is a simple function, evaluate inputs and then gives an output to other linked neurons as an input. Weights represent how strong the connection of each two linked neurons are.

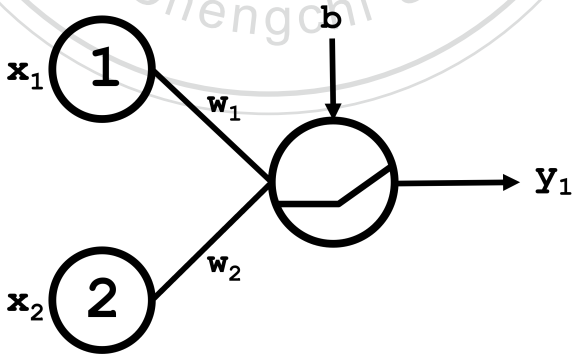

Figure 2.3: The Operation of a Neuron

The operation of each neuron can be divided into two components, the affine transformation and the activation function. Calculating the input data with corresponding weights and biases,

DOI:10.6814/NCCU202200797

then use the activation function on the result value. The activation function can be chosen by different network architectures. There are some popular choices of activation functions such as logistic function, the hyperbolic tangent function, the sigmoid function and rectified linear unit function (ReLU).

As the following example with Figure 2.3, we can easily understand the operation of a neuron. Suppose $x_1 = 1$ and $x_2 = 2$ as the input of the neuron in Figure 2.3, weights $w_1 = 3$ and $w_2 = -1$, bias $b = 4$, and let the activation function be ReLU which will get the same value as the input if positive. The result value of affine transformation is $[1 \times 3 + 2 \times (-1)] + 4 = 5$. Applying the activation function ReLU. Hence, the output is 5. Deciding the parameters, the values of weights and bias, is the main purpose on the learning algorithms, and the backpropagation algorithm is a popular learning algorithm.

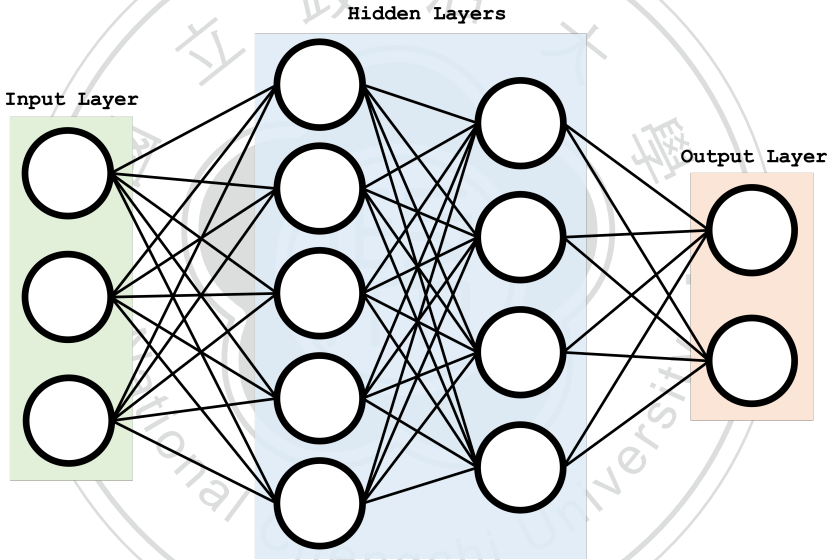

Figure 2.4: Fully Connected Feedforward Network

The simplest type of structure in deep learning models is a fully connected feed-forward network. The network can be divided into three parts, input layer, hidden layers and output layer. We give an example in Figure 2.4. There are three neurons in input layer, five and four neurons in each hidden layer, and two neurons in output layer. Note that there may be more than one layer in hidden layers, depends on the structure of neural network. In this structure, each neuron receives the inputs which are the outputs from all neurons of the previous layer, then pass down its output to the next layer in the same forward direction. There is no cycle or loop in this network. Every neuron connects to all other neurons from previous one layer and next

DOI:10.6814/NCCU202200797

one layer except the layer it belongs.

Each neuron in the network can be written as

$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l)$$

- a_j^l: the output of the j^{th} neuron in the l^{th} layer
- w_{jk}^l: the weight from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer
- b_j^l: the bias of the j^{th} neuron in the l^{th} layer

Given a neural network structure, the structure will define a function set. According to the training process, machine will decide the most suitable parameters to match the function set we provide. Hence, deciding the structure of a neural network beforehand in deep learning is still necessary. The function set decided not well might not have the ability to find a suitable function. For different types of problems, there are different suitable model to deal with.

2.2 Activation Function

In real world, most results of problems are not linear. Without activation function [14], usually a non-linear function, in the neuron network, all inputs of neurons is a linear combination of outputs from the previous layer which is linear, and its output is still linear. Therefore, activation function can let them have non-linear relationship. Followings are some famous activation function:

1. Sigmoid function

   Equation:

   $$f(x) = \frac{1}{1+e^{-x}}$$

   Range: $(0, 1)$

   Graph:

DOI:10.6814/NCCU202200797

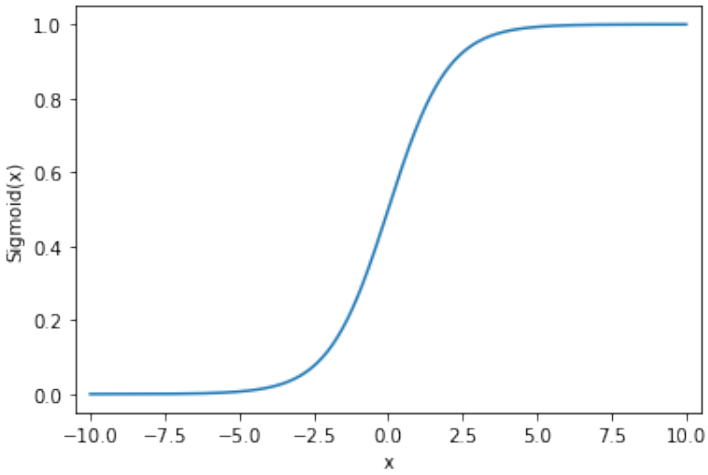

Figure 2.5: Sigmoid function

We usually use sigmoid function as the activation function when predicting the probability as outputs.

2. Hyperbolic tangent function (tanh)

   Equation:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

   Range: $(-1, 1)$

   Graph:

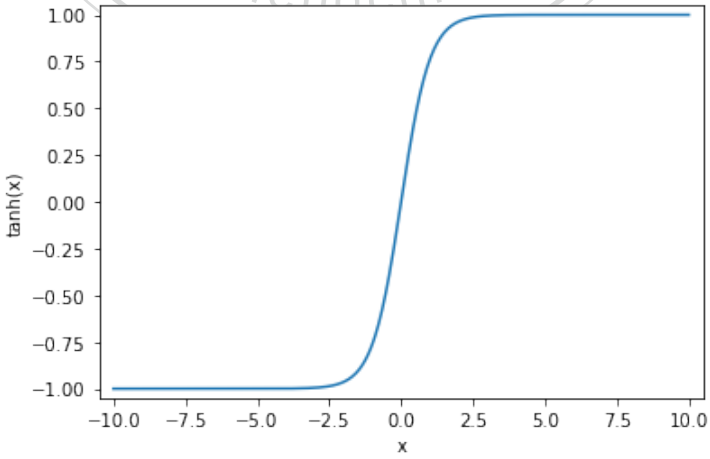

Figure 2.6: Hyperbolic tangent (tanh)

DOI:10.6814/NCCU202200797

Hyperbolic tangent function is similar with sigmoid function, but they have different range. The advantage is that it can put more emphasis on negative and zero inputs

3. Rectified linear unit function (ReLU) [19]

   Equation:

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

   Range: $[0, \infty)$

   Graph:

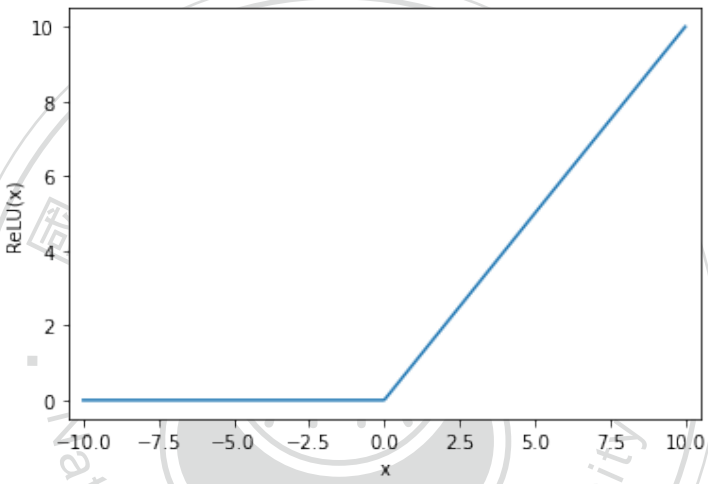

Figure 2.7: Rectified linear unit (ReLU)

   ReLU function is the most popular activation function now since it is widely used in many famous deep learning model.

2.3 Loss Function

The loss function is the key point of deep learning. After deciding the structure of a neural network, we have to adjust parameters, including weights and biases. Denote $D = \{(x_i, y_i)\}_{i=1}^{k}$ as a set of training data and $\theta = \{w_1, w_2, ..., w_p, b_1, b_2, ..., b_q\}$ as the set of parameters. Let Θ be the set of collection of all parameters θ. For a certain neural network structure, we want to find the optimal function $f_{\theta^*} \in \{f_\theta | \theta \in \Theta\}$ so that $f_{\theta^*}(x_i) = y_i$ for all $1 \leq i \leq k$

DOI:10.6814/NCCU202200797

To achieve our goal, denote loss function as $L(\theta)$. Our purpose is to make the predicted values (outputs) as close as the real values (input), which means we have to minimize the loss function L. Therefore, deciding a suitable loss function is important. Followings are some common used loss functions:

1. Mean absolute error (MAE)

$$L(\theta) = \frac{1}{k}\sum_{i=1}^{k} ||y_i - f_\theta(x_i)||$$

where k is the total number of data, and $y_i \in \mathbb{R}^n$.

2. Mean squared error (MSE)

$$L(\theta) = \frac{1}{k}\sum_{i=1}^{k} ||y_i - f_\theta(x_i)||^2$$

where k is the total number of data, and $y_i \in \mathbb{R}^n$.

3. Cross-entropy

$$L(\theta) = -\frac{1}{m}\sum_{i=1}^{m} \log(f_\theta(x_i))$$

where m is the total number of data, and $f_\theta(x_i) \in \mathbb{R}$ is the predicted value corresponding to $y_i \in \mathbb{R}$.

4. Binary cross-entropy

$$L(\theta) = -\frac{1}{m}\sum_{i=1}^{m} [y_i \log(f_\theta(x_i)) + (1 - y_i) \log(1 - f_\theta(x_i))]$$

where m is the total number of data, each $y_i = 0$ or 1, and $f_\theta(x_i) \in \mathbb{R}$ is the predicted value corresponding to y_i.

DOI:10.6814/NCCU202200797

2.4 Gradient Descent Method

No matter in real life or building software products, optimization is out main goal. There are some differences between deep learning and normal cases. Normally, we can know how the data exactly looks like and where we want to improve. But in deep learning, we have have no idea about how the new cases looks like, so we need to optimize with training data and use validation data to evaluate its performance. To achieve the goal, we use "Gradient Descent" which is one of the most common used method to optimize the values of the parameters in the neural network that can minimize the loss function $L(\theta)$.

Gradient Descent is an algorithm to reach a local minimum of differentiable function. It is easy to imagine how gradient descent work. Imagine that if we want to move to the ground from the top of mountain, we will look for somewhere nearby that is lower than where we stand now, then we move to the lower place and repeat the previous process. Same as gradient descent, assume that the parameters of the neural network is $\theta = \{w_1, w_2, ..., w_n, b_1, b_2, ..., b_m\}$, and choose randoms values for w_h and b_k, denote by $w_h^{(1)}$ and $b_k^{(1)}$. Evaluate the first-order derivative $\frac{\partial L}{\partial w_h^{(1)}}$ and $\frac{\partial L}{\partial b_k^{(1)}}$, then update the parameter as $w_h^{(2)} = w_h^{(1)} - \eta \frac{\partial L}{\partial w_h^{(1)}}$ for $1 \leq h \leq n$, and $b_k^{(2)} = b_k^{(1)} - \eta \frac{\partial L}{\partial b_k^{(1)}}$ for $1 \leq k \leq m$, where η is called learning rate and will introduced later. Repeating this process until it reaches the parameters that $L(\theta)$ is small enough. The gradient ∇L and the new θ will be as following:

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial w_m} \\ \frac{\partial L}{\partial b_1} \\ \frac{\partial L}{\partial b_2} \\ \vdots \\ \frac{\partial L}{\partial b_n} \end{bmatrix}, \theta^{new} = \begin{bmatrix} w_1^{new} \\ w_2^{new} \\ \vdots \\ w_m^{new} \\ b_1^{new} \\ b_2^{new} \\ \vdots \\ b_n^{new} \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} - \eta \nabla L = \begin{bmatrix} w_1 - \eta \frac{\partial L}{\partial w_1} \\ w_2 - \eta \frac{\partial L}{\partial w_2} \\ \vdots \\ w_m - \eta \frac{\partial L}{\partial w_m} \\ b_1 - \eta \frac{\partial L}{\partial b_1} \\ b_2 - \eta \frac{\partial L}{\partial b_2} \\ \vdots \\ b_n - \eta \frac{\partial L}{\partial b_n} \end{bmatrix}$$

The learning rate η is like how big the steps we move, which controls the speed of the

DOI:10.6814/NCCU202200797

movement. Usually, learning rate is not quite large. If the learning rate is too large, it may miss the minimum values or move back and forth around the minimum values. Otherwise, if it is too small, it costs lots of time to reach the minimum values or stuck in some relatively high minimum values. Hence, decide the learning rate is important to our training of neural network.

DOI:10.6814/NCCU202200797

Chapter 3

Word Embeddings

In Natural Language Processing (NLP) tasks, words are usually used as inputs, but machine can only compute with numbers. How to transfer words into numerical data to use in model is the main problem. Moreover, not only transfer words into number, but also keep meaning and unique fo words. There is a method called word embedding, which can represent complicated language by numbers. Intuitively, use one-hot encoding, a n-dimensional vector, to embed the word. For example, we can count how many different words appear in the data we use, supposed n. Then we denote every word as n-dimensional vector like $[1,\ 0,\ 0,\ \cdots,\ 0]$ or $[0,\ 1,\ 0,\ \cdots,\ 0]$, which let each different word correspond to a independent vector. In this way, we have done word embedding for n-dimensional one-hot encoding to every words. However, this approach lacks utility. In English, there are more than a hundred thousand words in total and counting. Twenty to thirty thousand words are typically used by native speaker. If there is a text with millions of words, the input will be too large and the embedding result may not useful for other texts. Most important, words are represented as one-hot encoding which don not capture the relationship between them. Hence, there are many research try to make the dimensions of word vector lower and embed by the relation between words. Followings are some introduction of famous embeddings of word vectors:

3.1 Word2Vec

The Word2Vec [18] model captures both syntactic and semantic similarities between the words. It use two model to train: Continuous Bag-of-Words Model (CBOW) and Continuous

DOI:10.6814/NCCU202200797

Skip-gram Model (Skip-gram), CBOW model predict the middle word when given previous words and future words, and Skip-gram model predict the previous and future words with middle word.

One of the well known examples of the vector algebraic on the trained Word2Vec vectors is $Vector(``King") - Vector(``Man") + Vector(``Woman")$ results in a vector that is closest to the vector representation of the word "Queen". This shows that the model not only knows the similarity of "King" and "Queen", but also the opposing relationship of "Man" and "Woman".

There is a single hidden layer in Word2Vec, it has weights like other neural networks. Word2Vec uses these hidden weights as the word vectors. In the original paper, it uses vector of dimension 640 as default, and as the dimension increases, the accuracy also increases.

3.2 GloVe

GloVe, abbreviation of Global Vectors for Word Representation [21], take the co-occurrence count of words to train. The main idea is: the relationship between two words (W_i and W_j) is examined by finding co-occurrence probability with some probe words (W_k). Denote X as the matrix of word-word co-occurrence counts, X_{ij} as the number of times word j occurs in the context of the word i, $X_i = \sum_k X_{ik}$ as the number of times any word appears in the context of the word i, and $P_{ij} = P(j|i) = X_{ij}/X_i$ as the probability that word j appear in the context of word i. For example in the paper, we have two words W_i as "Ice" and W_j as "Steam" and some probe words W_k as "Solid", "Gas", "Water", "Fashion". We know that "Solid" is more related to "Ice" (W_i) and "Gas" is more related to "Steam" (W_j) while "Fashion" is not related to both "Ice" and "Steam", "Water" is related to both "Ice" and "Steam". When W_k is "Solid", the probability P_{ik} will be higher and P_{jk} will be lower, so P_{ik}/P_{jk} will be higher as 8.9 in the paper. Conversely, when W_k is "Steam", P_{ik}/P_{jk} will be lower as 8.5×10^{-2}. For W_k is "Water" or "Fashion" which is almost same related to W_i and W_j, P_{ik}/P_{jk} will be close to 1. Actually, it uses a decreasing weighting function, so that word pairs contribute $1/d$ to the total count as they are d words apart.

To take advantage of the co-occurrence matrix without calculating it directly, GloVe

DOI:10.6814/NCCU202200797

consider a function

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

where the ratio P_{ik}/P_{jk} depends on three words W_i, W_j, and W_k. w_i, w_j and w_k represent the word vectors, $\tilde{w}$ denotes the separate context word vector. Since vector spaces are inherently linear structures, the most natural way to do this is with vector differences. The function F is modified to

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

Due to the operation of homomorphism of F,

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}$$

$$F(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}$$

then let $F = \exp$, so that

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i)$$

However, $\log(X_i)$ is independent of k, so it can be converted to bias b_i for w_i, and add an additional bias $\tilde{b}_k$ for $\tilde{w}_k$ to satisfy the symmetry,

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

Finally, use the Mean Square Error to calculate the error in the ground truth and the predicted co-occurrence counts. After training, use w_i to be the word vector representing the word W_i.

3.3 FastText

The structure of FastText [4] is similar to Word2Vec. Different from CBOW in Word2Vec, in FastText, each word is represented as a bag of character n-grams in addition to the word itself. For example, for the word "<matter>" which is added the special symbols "<" and ">", with

DOI:10.6814/NCCU202200797

$n = 3$, the FastText representations for the character n-grams is "<ma", "mat", "att", "tte", "ter", "er>". Then sum up the character n-grams of the word to represent itself. This helps preserve the meaning of shorter words that may show up as n-grams of other words, and this also capture meaning for suffixes and prefixes. Hence, this make some out-of-vocabulary (OOV) words have word vectors even if they are not present in the training set. To reduce computing time, FastText uses negative sampling and hierarchical softmax. Negative sampling only updates a small number, use 5 in the model, of negative words in Skip-gram model. For one actual context word, 5 random negative words are sampled. Hierarchical softmax use the binary tree, where leaves represent probabilities of words, more specifically, leaf with the index i is the i^{th} word probability and has position i in the output softmax vector. Each of the words can be reached by a path from the root through the nodes, which represent probability along that way, instead of calculating all nodes. These makes FastText can be trained faster and has better performance in text classification.

DOI:10.6814/NCCU202200797

Chapter 4

Transformer

"Transformer", which is the whole new model in deep learning, was announced in 2017 [29]. This is a major breakthrough by using nothing about the architecture of Recurrent Neural Networks (RNN) [26] or Convolutional Neural Network (CNN) [13]. Transformer uses self-attention mechanism with its encoder and decoder structure. At that time, it became the most advanced model in machine translation task and a popular construct in NLP. Following is the introduction of framework.

4.1 Embeddings

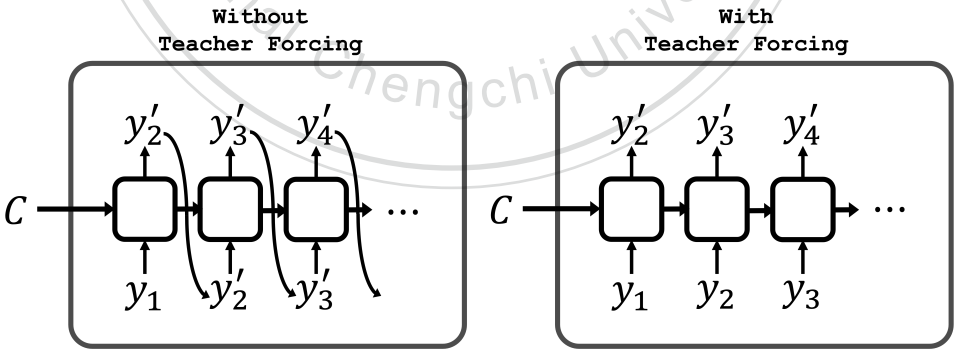

Figure 4.1: Difference of without or with teacher forcing

First, introduce the input and output of Transformer. As the other tasks for machine translation, we input the original sentence into encoder and get the translation be input of decoder. While model predicting, the output word at time t will be the input at time $t+1$. We call this property as auto-regressive. However, this makes training to be inefficiency because

DOI:10.6814/NCCU202200797

the wrong result of prediction will make the next incorrect one. Therefore, we train the model with teacher forcing [30]. That is, no matter what model output at time t, we always input the correct result to be input at time $t+1$, so model will be trained with the right things.

Transformer uses the method called wordpiece embedding which divides every word into sub-word units [31] and switch them to index sequence. For example, the word "attention" will be split to "atten" and "tion" and the sequence will be $[1, 2]$. To use this method, we can constitute any word easily. Then we give them the index "begin of sequence" ([BOS]) in the beginning and "end of sequence" ([EOS]) in the end. Since the lengths of each batch of input sequence are usually different, we will decide a maximum number of sequence length and wipe all the part of sequence which is longer than the number out in the next step. We also make other sequence be longer to the maximum number, by filling up the index of "padding" ([PAD]). Finally, every token will be embedded to d_{model} dimension. For sure, there are several ways of embedding for each language. Transformer cam also generate multiple words at the same time.

Except word embedding, the Transformer also keep the information about position, which can tell the model where a word in the sentence. Word embedding and position embedding will be summed as the final input. Each value of positional embedding will be

$$\text{PE}(pos, 2i) = \sin(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}})$$
$$\text{PE}(pos, 2i-1) = \cos(\frac{pos}{10000^{\frac{2i-1}{d_{\text{model}}}}})$$

where $1 \leq pos \leq d_{\text{model}}$ and $1 \leq i \leq \frac{d_{\text{model}}}{2}$. The combination of two embedding make the model pay attention to not only the word value but also the order, which let model understand the sentence more like human.

DOI:10.6814/NCCU202200797

4.2 Encoder

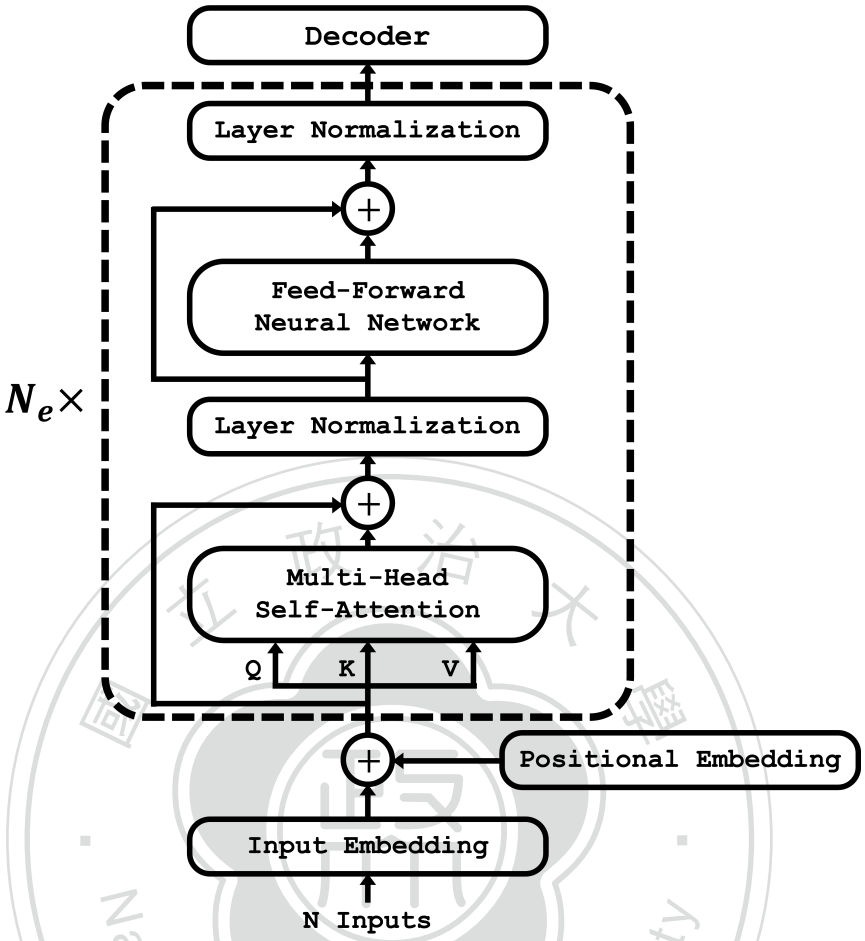

Figure 4.2: Encoder in Transformer

Next, referring to Figure 4.2, we introduce layers in the encoder. There two sub-layers in the encoder, first is multi-head self attention and second is a fully connected feed-forward neural network (FFN) noted $\mathscr{F}_D$. There is still a residual connection [9] and layer normalization [1] after each sub-layers. For each input vector x_i in self-attention layer, it will be separated into three parts: query q_i, key k_i and value v_i by multiplying different parameter matrices. For example, suppose x_1, x_2, ..., x_n are n input word vectors of dimension d_{model}. Then we get the queries, keys and values by:

Q : query (to match others)

$$q_i = x_i \cdot W^Q$$

K : key (to be matched)

DOI:10.6814/NCCU202200797

$$k_i = x_i \cdot W^K$$

V : value (information to be extracted)

$$v_i = x_i \cdot W^V$$

for $1 \leq i \leq n$ where W^Q and $W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$. Q and K have same dimension, d_k and d_v are the dimension of key and value. Set q_1, use each key, then make the scaled dot-product attention $\alpha_{1,i}$ by

$$\alpha_{1,i} = \frac{q_1 \cdot k_i}{\sqrt{d_k}} \text{ for } 1 \leq i \leq n$$

We use softmax function to get the attention scores $\hat{\alpha}_{1,i}$ from $\alpha_{1,i}$ for $1 \leq i \leq n$, which the coefficients are related to v_i for $1 \leq i \leq n$. Then we compute output b_1 by the summation of these weighted values. The equations are:

$$\hat{\alpha}_{1,i} = \frac{\exp(\alpha_{1,i})}{\sum_{j=1}^{n} \exp(\alpha_{1,j})}$$

$$b_1 = \sum_{i=1}^{n} \hat{\alpha}_{1,i} \cdot v_i$$

The summation of $\hat{\alpha}_{1,i}$ which is calculated via softmax function is 1, so the output b_1 is a convex combination, a linear combination with all coefficients are non-negative and sum to 1, of v_i. By the same method, we can compute the other outputs b_2, b_3, ..., b_n. This process is called self-attention, which the outputs come from the information of itself different from the method in RNN case. Obviously, the model can finish the computation of all inputs at the same time.

Let there be N word vector of dimension d_{model} be the input denoted by $X \in \mathbb{R}^{N \times d_{\text{model}}}$. The matrix computation can be expressed as following:

$$Q = X \cdot W^Q$$

$$K = X \cdot W^K$$

$$V = X \cdot W^V$$

$$\text{Attention}(Q,\ K,\ V) = [\text{softmax}(\frac{QK^T}{\sqrt{d_k}} - M \cdot P)]V$$

where Q, K,V are query, key and value which include all q_i, k_i, v_i together $\forall\ 1 \leq i \leq N$.

DOI:10.6814/NCCU202200797

$W^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are parameter matrices for query, key and value, respectively. d_k is the dimension of query and key. According to the token sequence, make a padding mask matrix $P \in \mathbb{R}^{N \times N}$ which the i-th column of matrix is 1 if the i-token is [PAD], others are all 0. Then multiply a huge positive number M like 10^9 and let $Q^T K$ minus $M \cdot P$ such that the attention weights of those padding position will be 0 with softmax function. This method can help the model skip those padding position. After all, the model will output a weighted values vector of dimension d_v. GPU can speed up the training progress effectually based on outstanding ability in parallel computing.

The research team think only one attention is not enough, they found that it is useful to compute several attention at the same time. Each attention function has its Q, K, V which same as above, they will give different weights to these matrices. We denote each output as a head. Suppose there are h heads, then concatenate them to become one matrix and reshape the matrix to d_{model} dimension by multiplying a parameter matrix, which is:

$$\text{MultiHead}(Q,\ K,\ V) = [\text{concatenate}(\text{head}_1,\ \text{head}_2, \cdots,\ \text{head}_h)] \cdot W^O$$
$$\text{where head}_i = \text{Attention}(Q \cdot W_i^Q,\ K \cdot W_i^K,\ V \cdot W_i^V)$$

where the parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ for $1 \leq i \leq h$. Multi-head self-attention let model understand sentence more, each head can focus on different characters, as humankind.

Before we import the output into FFN, we need to do some preprocessing. First, let the output from multi-head attention add the original input vector, then execute layer normalization $\text{LayerNorm}(X + \text{MultiHead}(Q,\ K,\ V))$:

$$\mu_j = \frac{1}{d_{\text{model}}} \sum_{i=1}^{d_{\text{model}}} x_{ij}$$
$$\sigma_j = \sqrt{\frac{1}{d_{\text{model}}} \sum_{i=1}^{d_{\text{model}}} (x_{ij} - \mu_j)^2}$$
$$\hat{x}_{ij} = g_{ij} \frac{x_{ij} - \mu_j}{\sigma_j} + b_{ij} \quad \text{for all} 1 \leq i \leq N,\ 1 \leq j \leq d_{\text{model}}$$

where x_{ij} is the element of $X + \text{MultiHead}(Q,\ K,\ V)$ matrix. After layer normalization, we obtain $\hat{x}_{ij}$, where g_{ij} and b_{ij} are parameters learned during training.

DOI:10.6814/NCCU202200797

To the next sub layer, fully connected feed-forward neural network, which dimension of input and output is d_{model}. Use activation function ReLU in the middle of a only hidden layer of dimension d_{ff}. Input each vector $\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \cdots, \hat{\mathbf{x}}_N$. Then $\mathscr{F}_D(\hat{\mathbf{x}}_i)$ can be write as:

$$\mathscr{F}_D(\hat{\mathbf{x}}_i) = \text{ReLU}(\mathbf{x}_i \cdot W + B) \cdot W^{'} + B^{'}$$

$$\text{where ReLU}(\mathbf{x}_i \cdot W + B) = \max(0, \sum_{j=1}^{d_{\text{model}}} x_{nj} w_{nj} + b_n)$$

$\forall\ 1 \leq i \leq N,\ 1 \leq j \leq d_{\text{model}},\ 1 \leq n \leq d_{\text{ff}}$ where $W \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}},\ B \in \mathbb{R}^{1 \times d_{\text{ff}}},\ W' \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}},\ B' \in \mathbb{R}^{d_{\text{model}} \times 1}$. The n-th element in $\mathbf{x}_j,\ W$ and B are denoted as $x_{nj},\ w_{nj}$ and b_n, respectively. Like above, execute residual connection and layer normalization after FFN. Hence, we obtain output of this layer and will input to the next layer. There are N_e layers in encoder. In the original paper, $\text{N}_e = 6,\ d_{\text{model}} = 512,\ h = 8,\ d_v = d_k = d_{\text{model}}/h = 64$ and $d_{\text{ff}} = 2048$. We can see the structure in Figure 4.2.

DOI:10.6814/NCCU202200797

4.3 Decoder

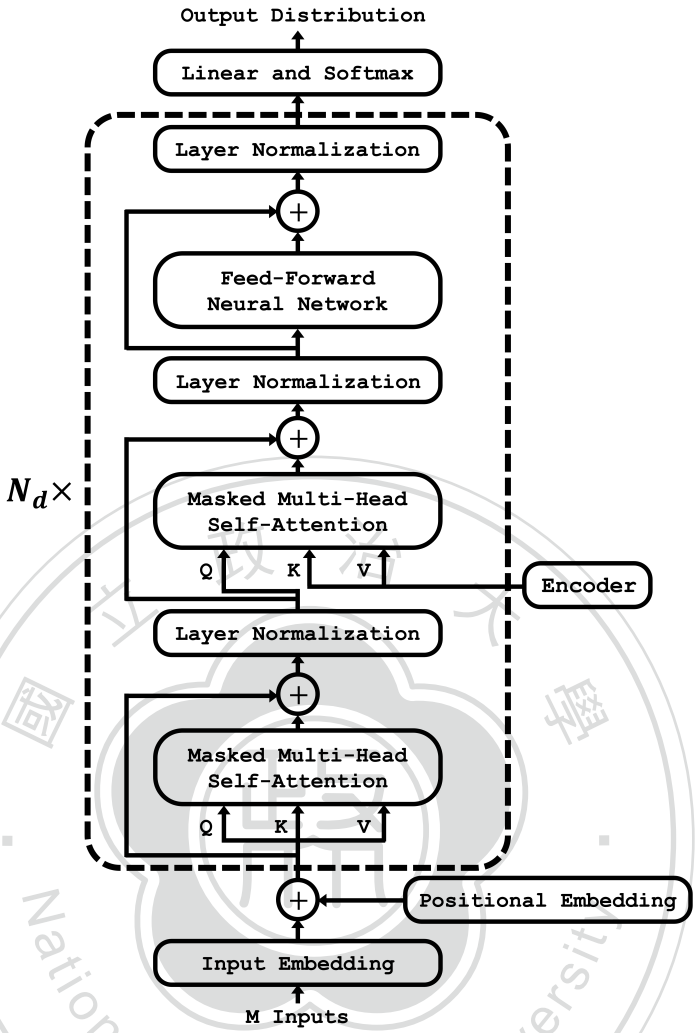

Figure 4.3: Decoder in Transformer

Moving forward to decoder. Figure 4.3 is the structure of decoder. Like encoder, there several layers in decoder, each layer has three sub-layers in itself. The first one is called masked multi-head self-attention, the second one is multi-head self-attention, the third one is feed-forward neural network, which second and third one are similar to encoder. Residual connection and layer normalization will be executed in each sub-layers. Multi-head self-attention obtains keys and values from the output of encoder and query from the previous sub-layer. This can let decoder learn from the base of encoder. Last, adjust masks in the self-attention in each first sub-layers to let model cannot know the answer. We need to confirm that the i-th token can only participate to the token before i. Otherwise, from input, the model can know the answer of $i+1$-th word. Therefore, we hide the information by using a look-ahead mask. The mask denoted

DOI:10.6814/NCCU202200797

$L \in \mathbb{R}^{M \times M}$ is an lower triangular 0-1 matrix. Multi-head self-attention with combination of mask L and padding mask $P \in \mathbb{R}^{M \times M}$ is called masked multi-head self-attention which is in first sub-layer. In the original paper, there are $\mathrm{N}_d = 6$ layers in decoder.

We will input the output of last layer in decoder to a linear layer which can transfer it to the shape in $\mathbb{R}^{d_{\text{dict}} \times M}$ where d_{dict} is the number of all sub-words in the dictionary constructed by our embedding way. By the softmax function, we choose the relative index which has the maximum values in the distribution. Calculate the sum of cross entropy between output and the correct answer, then use optimizer called Adam to update the model.

Transformer uses self-attention structure different to CNN or RNN and can be trained quickly by parallel computing. Hence, transformer have good performance in NLP tasks.

DOI:10.6814/NCCU202200797

Chapter 5

Contextualized Word Embeddings

A word can be multiple senses depending the context. For example, “It is safest to deposit your money in the bank.”, and “They stood on the river bank to fish”, same word “bank” have different meaning. To solve this problem, here are some model that can realize the meaning of words in different context.

5.1 ELMo

ELMo, abbreviation of Embeddings from Language Model [22], is a RNN-base language model which trained from lots of sentences. It uses bidirectional LSTM [10] not only to learn from the start to the end of sentences but also form the end to the start. Then combine the two ways information of the surrounding words as a context-dependent word vector, which a word can have different embeddings according to the context.

Compared with the traditional word embeddings which the word vector is fixed that can only express one meaning, the ELMo model focuses on solving that a word may have different meanings in different context. Since the word vector generated by ELMo uses contextual information. According to the downstream tasks, the word vector can be adjusted by weights to suit different tasks.

DOI:10.6814/NCCU202200797

5.2 BERT

BERT, abbreviation of Bidirectional Encoder Representations from Transformers [6], is a new embedding model with structure of encoder in Transformer. BERT has its own embedding way which depends on the context. BERT can fine-tune the parameters depend on the different NLP tasks with pre-trained model. These things makes BERT become a good model to handle several tasks. Following, We will introduce more detail about BERT.

In unsupervised NLP model, there two common methods: feature-based and fine-tuning. The same character of them is that using a huge amount of unlabeled language data to pre-train the model. There are two methods to train the model: In feature-based, we will set all the parameters in language model (LM) and add an output layer to become the task-specific model as final model to solve out task. In fine-tuning, all the parameters will be trained to be the best for our task. We also have to adjust input and output. Therefore, BERT can be applied to various tasks. If the pre-trained model is powerful, the model will be easy to have better performance.We train the model to know about language in BERT by two tasks: Masked Language Model (MLM) and Next Sentence Prediction (NSP). Suppose the input of pre-trained model ia a pair of sentences A and B. Mask some tokens in each sentence randomly, then what model have to do is find the best choices for the correct words be masked at these positions, which is like fill-in-the-blank questions. At the same time, let model predict whether sentence B is the next sentence of sentence A, which is like a yes-no question, to realize the relationship between sentence A and B.

DOI:10.6814/NCCU202200797

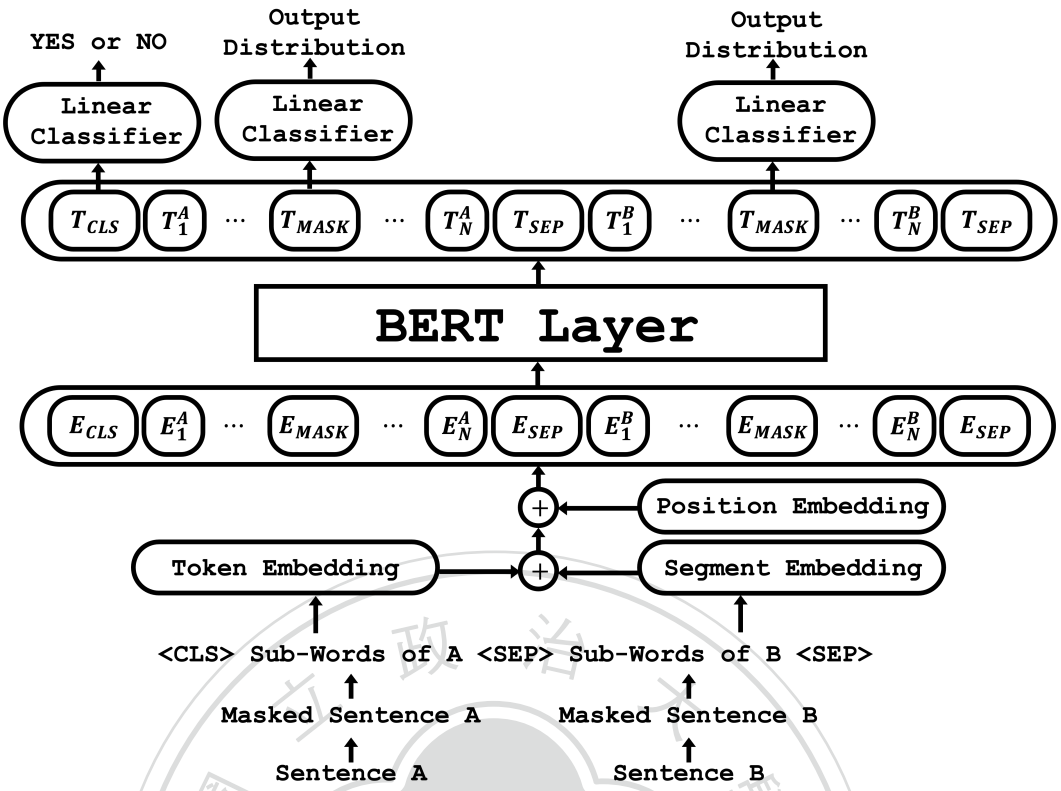

Figure 5.1: The architecture of pre-trained model in BERT

We have to reform the input embedding of BERT for these two tasks. In the beginning, separate the sentence into sub-words units just like in Transformer. We set some special tokens: [CLS] will be put in the beginning of each input sequence, [SEP] locate in the end of two sentences and [MASK] is used to hide tokens. Then, embedding the tokens, there three kinds of embedding method in BERT. Token embedding, which transfers each word become an corresponding index. Segment embedding, which distinguishes the word in different sentence. Position embedding, which provides the relative position of words, same as in Transformer. Last, summarize these embedding to be the final input. Actually, the BERT layer is the encoder in Transformer.

We use the final output vector of BERT layer to be the embedding of input in next step. This representation include the information of context different from the one-to-one word embedding. Add a linear classifier on each relative position of [CLS] and [MASK] tokens. The classifier for [CLS] will evaluate the relation between sentences by using sigmoid function. Other classifiers for [MASK] are same as the final output in Transformer, the classifiers will output a distribution of all possibility of all words in the dictionary, then we choose the corresponding word with

DOI:10.6814/NCCU202200797

highest possible value as the output. Also, evaluate loss and optimize the model. Then we can fine-tune this pre-trained model by setting the input and output to fit our task and adjust the parameters.

5.3 GPT-2

GPT, abbreviation of Generative Pre-Training Transformer [23] [24] [5] , is a language model (LM) with structure of decoder in Transformer. Natural Language Tasks include various tasks such as natural language inference, question answering, semantic similarity assessment and text classification. Previously to train models on these tasks, special data set curated for such tasks were required. But, the datasets are very difficult to acquire and even if we have such datasets, we can't have large corpus. The other problem is that such models cannot be used on the other NLP tasks as they were trained for specific tasks.

GPT is a transformers model pre-trained with very large corpus of English data in a self-supervised method which the model takes input data and tires to generate an appropriate response without any human labeling, and then allowing users to fine-tune the language model so that it can perform downstream tasks. More precisely, the model was trained to predict the next word in sentences.

As a result of its pre-training, one of the significant achievements of GPT model is it can reach few-shot learning even zero-shot learning on various tasks. It means that we can give model an example we want it to predict or just give the target of the task, then GPT model can finish the task. This capability shows that language model served as an effective pre-training objective which could help model generalize well. With Transfer learning, GPT becomes a powerful model to perform NLP tasks with very little fine-tuning. It generates paths for other models that can further enhance its potential for generative pre-training on larger datasets and parameters.

Currently, there are three generations of GPT. With the evolution of generation, the number of parameters has also increased a lot. GPT-1 has 117 millions parameters. GPT-2 has 1.5 billions parameters which is about 5 times as many as BERT and 10 times more than GPT-1. GPT-3 even has exaggerated 170 billions parameters ,so that is almost impossible for us to use GPT-3 model in our personal computer. Therefore, we will use GPT-2 in our experiment.

DOI:10.6814/NCCU202200797

Chapter 6

Summarization

6.1 Two methods of summarization

To shorten the time required for reading, we summarize the article and extract the key points. Here are two methods of summarization methods:

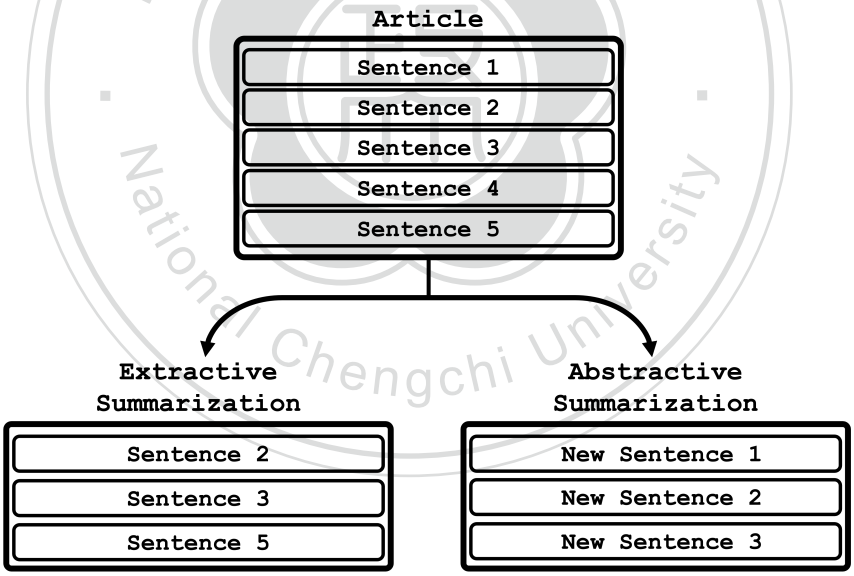

Figure 6.1: Two methods of summarization

1. Extractive Summarization:

   In extractive summarization, the summary was created by selecting a subset of all sentences in article. Easily to say, we split the article into sentences, then identify and choose sentences based on a score of importance in whole article. Finally, connecting the chosen sentences together. In model training, it is used as the task of sentence

DOI:10.6814/NCCU202200797

classification. Obtain the sentences as a summary by classifying whether each sentence is important.

2. Abstractive Summarization:

   Abstractive summarization is a method of feeding the original article to the model and generate a summary directly, which is a sequence to sequence task. First, machine analyzed the article and get interpretation, then predicts a summary based on this interpretation.The difficulty of abstractive compared to extractive is that abstractive needs to be able to generate words or sentences that do not originally exist in the article, and must conform to the rules of the language.

6.2 TextRank

TextRank [17] is a graph-based ranking model for graphs extracted from natural language text. TextRank is a keyword or key-sentence extraction method based on PageRank [20]. Its essence is to construct nodes and edges according to the window for the text token, actually the co-occurrence relationship of nodes within a certain window range. The basic idea implemented is "voting" or "recommendation". When one vertex links to another, it is basically casting a vote for that other vertex. The more votes that are cast for a vertex, the more importance of the vertex. Formally, let $G = (V, E)$ be a directed graph with the set of vertices V and set of edges E which is a subset of $V \times V$. Select a vortex V_i, denote $In(V_i)$ as the set of vertices that point to V_i and $Out(V_i)$ as the set of vertices that V_i points to. The score of a vertex V_i is defined as follows:

$$S(V_i) = (1-d) + d \times \sum_{j \in In(V_i)} \frac{1}{Out(V_j)} S(V_j)$$

where $0 \leq d \leq 1$ is a damping factor ,which is the probability of jumping from V_i to another random vertex. Denote w_ij as the "strength" of the connection between two vertices V_i and V_j. The formula of weighted score is as following:

$$WS(V_i) = (1-d) + d \times \sum_{V_j \in In(V_i)} \frac{w_{ji}}{\sum_{V_k \in Out(V_j)} w_{jk}} WS(V_j)$$

However, in key-sentences extraction in TextRank which used in natural language texts,

DOI:10.6814/NCCU202200797

instead of counting incoming links, the similarity between sentences is used. First we build a graph of the sentences in the article, which the vertices denoted the sentences. The co-occurrence relation used for keyword extraction can not be used here, since the sentences are significantly larger than one or few words, and co-occurrence is meaningless for such large contexts. Instead, we use "similarity" to determine the connection between two sentences, where"similarity" is measured as a function of their content overlap. This relationship can be see as "recommendation": a sentence with some concepts gives the reader a "recommendation" to refer to other sentences with the same concepts, so we can link two such sentences which have common content. Formally, let two sentences S_i and S_j. S_i is represented by the set of N_i words that appear in the sentence: $S_i = w_1^i, w_2^i, ..., w_{N_i}^i$, so is S_j. The similarity of S_i and S_j is defined as :

$$\text{Similarity}(S_i, S_j) = \frac{|\{w_k | w_k \in S_i \& w_k \in S_j\}|}{\log(|S_i|) + \log(|S_j|)}$$

Finally, the sentences with the highest rank are selected for inclusion in the summary.

6.3 BERTSUM

Recall the BERT model, we used [CLS] token in the beginning and [SEP] in the end of two sentences, then we did token embeddings, segment embeddings and position embeddings, and input these into BERT model. Applying BERT directly to summarization is obviously problematic. First of all, when doing summary, we usually use sentence embeddings to do related tasks, but the training method of BERT is MLM (Masked Language Model), and the focus of learning is on word, not sentence. Another issue is that only sentence pairs are used for training In the original BERT. To do summarization, more than two sentences are usually used as required input.

The extractive summary is created by identifying the most important sentence in the article then concatenating subsequently. Neural network models consider extractive summarization as a sentence classification problem that the classifier predicts which sentences should be selected as summaries.

DOI:10.6814/NCCU202200797

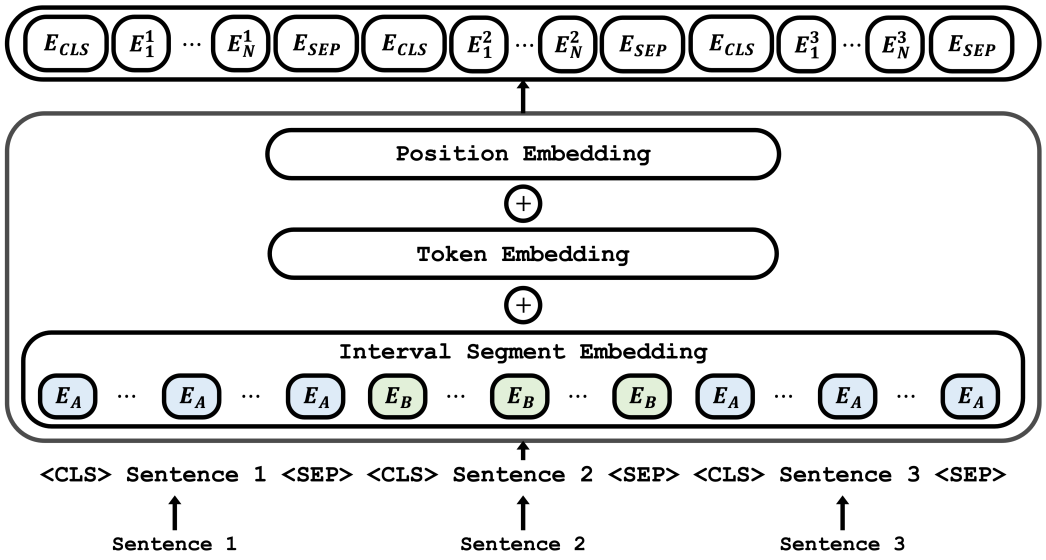

Figure 6.2: Structure of Bert for Summarization

In BERTSUM [16], BERT architecture for Summarization, an extra [CLS] token is inserted before each sentence in the article. This is done here for two reasons: one is to highlight individual sentences, and the other is to use these [CLS] tokens as the outputs of the sentence embeddings of each sentence which is used for subsequent prediction tasks. Segmentation embeddings was also bee modified to "interval segment embeddings" to distinguish multiple sentences better. The method is that separate sentence $sent_i$, where $sent_i$ is the i-th sentence, into E_A and E_B depending on whether i is odd or even. For example, an article contains [$sent_1$, $sent_2$, $sent_3$, $sent_4$, $sent_5$], the interval segment embeddings of these sentences are [E_A, E_B, E_A, E_B, E_A]. With these embeddings, the model can learn about representation of sentences, adjacency of sentences and meaning of entire article. Let an article containing sentences [$sent_1$, $sent_2$, ..., $sent_m$]. Extractive summarization can be defined as the task of assigning a label $y_i \in \{0, 1\}$ to each $sent_i$, indicating whether the sentence been selected in the summary. With BERTSUM, the vector of the i-th [CLS] token from the top layer denoted as t_i can be used as the representation for $sent_i$. Then input these t_i into several Transformer layers to captain the features for extracting summaries:

$$\tilde{h}^1 = \text{LayerNorm}(h^{l-1} + \text{MultiHead}(h^{l-1}))$$

$$h^l = \text{LayerNorm}(\tilde{h}^1 + \text{FFN}(\tilde{h}^1))$$

where $h^0 = T$ denotes the sentence vectors output by BERTSUM with position embeddings.

DOI:10.6814/NCCU202200797

The final output layer is a sigmoid classifier :

$$\hat{y}_i = \sigma(W_o h_i^L + b_o)$$

where h_i^L is the vector for $sent_i$ from the top layer of Transformer. Finally, use greedy algorithm to obtain an oracle summary from given summary of each article, and maximize the similarity between extractive summary and the oracle summary to train the model.

DOI:10.6814/NCCU202200797

Chapter 7

Experiments

In this chapter, we will use BERTSUM model and GPT-2 model to make a summarization of articles without human reading that can reduce much time to get the key point. Since there are already many pre-trained models support for English article summarization originally like BART [15], T5 [25], etc., and it is difficult to train a Chinese BART or T5 model personally which requires lots of hardware resources to compute, we will focus on how to use existing pre-trained model to achieve goals in Chinese data. The main concept is to extract the article first and then do the abstract. Following will introduce how our method do.

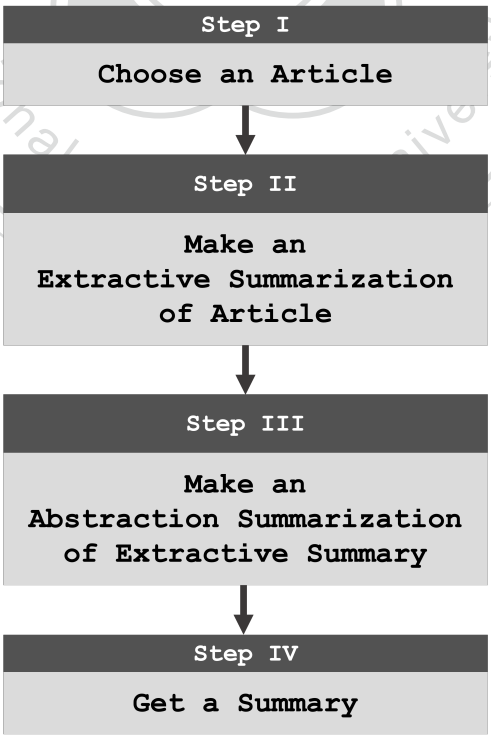

Figure 7.1: Pipeline of experiments

DOI:10.6814/NCCU202200797

7.1 Data Preparation

Because of the lack of Chinese corpus, we found that there are several labeled datasets in a competition which called Natural Language Processing and Chinese Computing in China. This competition provides a lot of training materials about Chinese like emotion detection, question answering, etc., including document summarization surely. After checking the training data, we found that the data is a large collection of news articles and headlines, where the headlines is a kind of summary of articles.

Before using these data, we still have to do some data preprocessing. Since we have to input these data into BERT model, we only choose the articles, including the addition of [CLS] and [SEP] tokens, less than 512 words. There are 15,906 articles with average length 306.5 words left in the end.We reverse 10 articles for test, then split remaining data into 15,101 for training and 795 for validating, which the ratio is twenty to one. Compared with English punctuation, Chinese punctuation is more complex and also includes halfwidth and fullwidth forms, we have organized the data by cleaning up some strange words or punctuation and replacing punctuation from halfwidth form into fullwidth form ,then split article into paragraphs with some punctuation like ，, ；, ！, ？, 。, etc. There is sill some useless information in articles like (圖片), (記者), etc., so we remove these from articles. Finally, these data can be used for training.

7.2 Extractive Summarization with BERTSUM

The research team have released the pre-trained Chinese BERT model named “BERT-Base-Chinese”, thus we have the pre-trained Chinese BERT model and tokenizer, so that we don’t have to train a Chinese BERT model to use by ourselves which is almost impossible. There is also a pre-trained BERTSUM model which is trained in English, we have to change the segmentationer from English into Chinese, then we use BERT-Base-Chinese to be the base BERT model and tokenizer. After modifying article as the input we want, we set the ratio of extractive as 0.3 which means we will extract at most 30% of article as a summary to use in next step. If we extractive too many sentences, it is almost as if we used the whole article. Then we let the selected sentences must be more than 5 words, which can remove some sentences that are too short or meaningless, and also let the selected sentences must less than 100 words. Hence, we have the extractive summarization that can be used in the next step.

DOI:10.6814/NCCU202200797

7.3 Abstractive Summarization with GPT-2

Because of extracting from the segment of article, the consecutive sentences may be incoherent, so we use generation model GPT-2 to improve this situation. As we did in previous step, we use BERT-Base-Chinese model as the tokenizer, but we use "ckiplab/ gpt2-base-chinese" which is released by Chinese Knowledge and Information Processing team of Taiwan Academia Sinica as the base GPT-2 model .

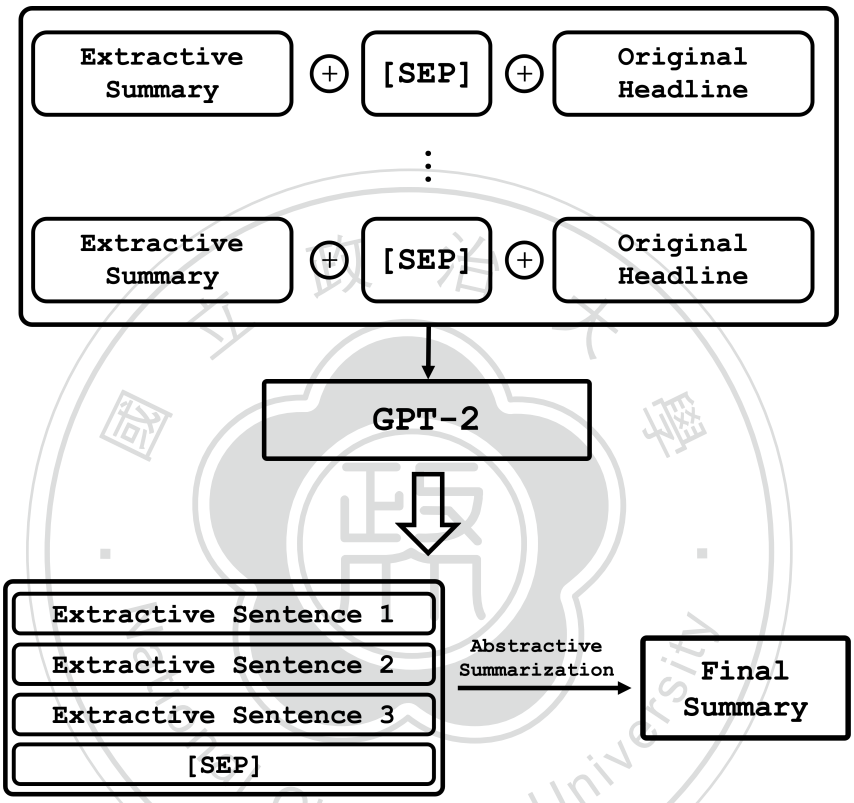

Figure 7.2: Pipeline of abstractive summarization with GPT-2

Since ckiplab/ gpt2-base-chinese model was created by Taiwanese who used to use traditional Chinese, we have to transfer words in simplified Chinese into traditional Chinese before using ckiplab/gpt2-base-chinese model. Next, we connect the extractive summary from previous step and the original headline of each article form original datasets with a [SEP] token as the training data, this trains the machine to generate the abstractive summary after seeing [SEP] token. With training, the default weights of ckiplab/gpt2-base-chinese will be adjusted to fit the condition for summarization tasks. After some parameters adjusting and trying, we set the epochs to 50 instead of 100 or bigger number to prevent overfitting, and set batch size to 256 with learning rate to $5 \cdot 10^{-5}$, train the model with 100 warm-up steps and 2950 total steps.

DOI:10.6814/NCCU202200797

7.4 Result

With this method, we found some news articles, then make summary, followings are three example results:

```
Article:
蘋果開發者大會WWDC將於6月7日凌晨1時開始，此次除了預期中會發布的iOS16等作業系統之外，
也傳出蘋果可能推出硬體更新，新款的MacBookAir可能會隨之亮相。此次盤點了五大WWDC重點。
首先，關於iOS16其實並沒有很多爆料流出，根據先前彭博社記者MarkGurman爆料指出，iOS16
可能會有重新設計的螢幕桌面，在小工具上會有所改變，而健康功能也會有更新。除此之外，iOS1
6還將加入隨顯螢幕AlwaysonDisplay，可以在手機處於待機狀態時顯示有限的資訊，目前在And
roid手機上廣泛可用，不過目前預計只會在高階機型上出現此功能。蘋果可能在WWDC大會上推出新
的Mac電腦，預計可能推出新款MacBookAir，但由於新冠疫情影響產能，上市時間可能較晚。新款
MacBookAir預計將採用類似MacBookPro的外形，不再是錐形機身，將更薄更輕，預計將有多種
顏色，尺寸可能在13.6吋左右。MacBookAir可能會配備下一代M2晶片，預計將擁有與M1相同數量
的計算核心，但速度更快，圖形處理能力也有所提高，有9到10個GPU圖形內核，而目前的M1晶片是
7到8個GPU核心。目前透露的資訊，iPadOS16將會在性能上更加強化，讓iPad更像筆記型電腦而
不是智慧型手機，這在系統界面上預計會有重新設計的多功能界面，多功能交換更加容易，允許用
戶調整視窗的大小，蘋果還計劃為用戶實現同時處理多個應用程式的新方式。更新將圍繞日常操作
邏輯，也針對手錶推出新版省電模式，目前AppleWatch省電模式較為無感，改進後或許更類似iPh
one，不限制功能但降低電池的使用強度。而在心室顫動檢測功能會更細化，例如支援一個人在某段
時間內心室顫動狀態的頻率。其他跟健康有關辜能包含改善健康和活動跟踪，更多的運動方式，跑
步的額外數據記錄等。不斷有消息說AR / VR頭戴式裝置可能會在2022年全球開發者大會上推出，但
目前認為可能性不高，由於多名分析師，爆料者提到，蘋果目前還沒有準備好推出這款產品，發熱
之類的問題沒解決，且軟體內容還尚未研發完成。

Extractive Summary (from Article):
此次除了預期中會發布的iOS16等作業系統之外，也傳出蘋果可能推出硬體更新，iOS16可能會有
重新設計的螢幕桌面，在小工具上會有所改變，不過目前預計只會在高階機型上出現此功能。但由
於新冠疫情影響產能，將更薄更輕，預計將擁有與M1相同數量的計算核心，但速度更快，多功能交
換更加容易，蘋果還計劃為用戶實現同時處理多個應用程式的新方式。目前AppleWatch省電模式
較為無感，例如支援一個人在某段時間內心室顫動狀態的頻率。更多的運動方式，由於多名分析
師，發熱之類的問題沒解決，

Abstractive Summary (from Extractive Summary):
蘋果或加入，將更便利軟體更新，速度更快，附加試運動系統或高階乘用次數。此外，蘋果還可能
選擇高階機型升級，例如等；目前現在已經有多名分析師考慮降低的速度
```

Figure 7.3: Example result of experiments

DOI:10.6814/NCCU202200797

Article:
新北市五股區今天早上8點半起部分路段傳出停電，有民眾在臉書社團「我是五股人」表示，聽到「碰」一聲後就停電了，新北市消防局表示，消防人員已經抵達現場，因成泰路三段及新五路二段路口的電桿電線上方的變電箱爆炸，明顯有火，已通知台電人員到場處理，目前已全數復電。五股區公所表示，根據台電初報，目前停電路段包含凌雲路三段、直坑路、中直路崩山巷、中直路、凌雲路二段、成泰路三段、新城五路、新城三路、新城八路、成泰路及成泰路二段一帶都停電，總計影響1564戶。對此，台電台北西區營業處表示，今天上午8時28分，新北市五股區成泰路三段與新五路二段交叉路口，電線桿冒出火煙，疑因鳥類碰觸引起桿上線路故障，台電立即派員至現場搶修，先行於8時31分轉供復電358戶，另1206戶則於上午9時46分全數復電。

Extractive Summary (from Article):
新北市五股區今天早上8點半起部分路段傳出停電，聽到「碰」一聲後就停電了，因成泰路三段及新五路二段路口的電桿電線上方的變電箱爆炸，明顯有火，台電台北西區營業處表示，先行於8時31分轉供復電358戶，

Abstractive Summary (from Extractive Summary):
今天早上8時許，五股區今早8時許發生部分路段傳出部分路段傳電，致交通中斷。據悉，事發現場明顯有火。據悉，事發時此路段正採取停電措施。此前其它路段均有停電引發。由南向東按照南北方向營業處的模式辦理停電事宜

Figure 7.4: Example result of experiments

Article:
大S在離婚4個月後閃電宣布和酷龍具俊曄結婚，男方透露、就是找到20年前的電話號碼聯繫上大S促成這段姻緣。但您知道嗎？過去要換電信公司，電話號碼就必須跟著換，不過在17年前，我國工研院研發出手機攜碼技術，整合14家電信業者，才讓大S可以一直維持同個號碼。讓網友驚、原來工研院是兩人的媒人！不過原來我國工研院是這段浪漫佳話的媒人，17年前研發的手機攜碼技術，才讓大S可以一直維持同個號碼。大S、具俊曄20年情復燃媒人竟是工研院，韓國酷龍成員具俊曄和大S分手20年，靠著一通電話舊情復燃，不過這段愛情故事的最大媒人，竟然是工研院，工研院幫了什麼？17年前超前部署的門號可攜系統，具俊曄在社群媒體寫下，找到了20多年前的那個號碼，幸好門號沒變才能重新連結起來，20年不變的門號瞬間成為話題，不過如果兩人再早幾年相遇，可能沒那麼幸運，工研院資通所新創長周勝鄰說：「以前這個號碼是綁訂電信業者的，你就沒辦把用這個號碼，轉到其他電信業者去。」其實這先進技術當年新加坡、馬來西亞就馬上來台取經，台灣境內串連14家電信業者資料庫，整理3000萬筆資料，促使門號可攜系統誕生，讓大S具俊曄20年情復燃，正是這系統的傑作，民眾說：「因為號碼有時候跟名字一樣，如果換號碼的話，你以前的朋友，包括前男友都找不到你啊。」民眾說：「可攜碼的話對我們比較有保障，買手機也比較便宜。」現在可以直接攜碼，不但省下重印名片的成本，也讓過去感情能夠透過一通電話，重新接線。

Extractive Summary (from Article):
但您知道嗎？不過在17年前，不過原來我國工研院是這段浪漫佳話的媒人，17年前超前部署的門號可攜系統，幸好門號沒變才能重新連結起來，可能沒那麼幸運，工研院資通所新創長周勝鄰說：轉到其他電信業者去。台灣境內串連14家電信業者資料庫，整理3000萬筆資料，讓大S具俊曄20年情復燃，包括前男友都找不到你啊。」民眾說：買手機也比較便宜。

Abstractive Summary (from Extractive Summary):
臺灣19年前門號未更新，門號更換結構，讓網友更便宜；資通所網站建構剛剛完成，網友可以連結到其他電信業者去。詳細信息請瀏覽原所有者您帳號

Figure 7.5: Example result of experiments

These three news are technology news, entertainment news, and local news, respectively. We use different kinds of news to show that the model can handle different kinds of articles. After reading the article and summary, we can see the extractive summary contains most of information in the article. There are still some inconsistencies and repetitive information. With

DOI:10.6814/NCCU202200797

the second summarization by abstracting, we got a more smooth summary. However, some important information shown in extractive summary was dropped, and some irrelevant words were generated in the abstractive summary.

DOI:10.6814/NCCU202200797

Chapter 8

Conclusion

In this paper, we restudy the methods of summarize Chinese article. We remove the less useful information, then we generate better summaries with the remaining useful information. Honestly, the results shown in previous chapter are better results after model training. In extractive summarization, we can get a common summary most of time, but in abstractive summarization, we sometimes get a weird summary which even less related to the article. To compare with doing abstract summary from article directly which may generates disorganized summary, our method still work much better. However, our experiments still have some advantages and disadvantages. The datasets with lots of news make training data have more variety, let our model can be applied in more fields. Variety brings another problem, the model may be confused about what it should generate, then generates a strange summary. For example, computer might generate a summary with some technology segments from a weather news.

To get the better performance and application, there are several points we can discuss in the future. One is to try more combinations of models to summarize the article. The other is to use more kinds of articles for training, such as novels, poems, lyrics, etc. that can make model more generally used in any articles. Last is trying to increase the length of data that model can read, so that the model can do longer article summary.

DOI:10.6814/NCCU202200797

Bibliography

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information, 2016.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] Kunihiko Fukushima. Neural network model for a mechanism of pattern recognition unaffected by shift in position-neocognitron. *IEICE Technical Report, A*, 62(10):658–665, 1979.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

DOI:10.6814/NCCU202200797

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[11] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[15] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[16] Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*, 2019.

[17] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 404–411, 2004.

[18] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[19] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

DOI:10.6814/NCCU202200797

[20] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[21] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[22] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[23] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

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

[25] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

[26] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[27] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, Jan 2015.

[28] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[30] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

DOI:10.6814/NCCU202200797

[31] W Yonghui, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

DOI:10.6814/NCCU202200797