UNIVERSITY OF SCIENCE
ADVANCED PROGRAM IN COMPUTER SCIENCE

TUAN-ANH NGUYEN - DUY-TUE TRAN-VAN

EXTRACTIVE SUMMARIZATION WITH BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

HO CHI MINH CITY, 2019

UNIVERSITY OF SCIENCE
ADVANCED PROGRAM IN COMPUTER SCIENCE

TUAN-ANH NGUYEN - 1451007
DUY-TUE TRAN-VAN - 1551044

EXTRACTIVE SUMMARIZATION WITH BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

THESIS ADVISOR
DR. NGHIÊM QUỐC MINH

HO CHI MINH CITY, 2019

ACKNOWLEDGEMENTS

We would first like to express our deep appreciation to our thesis advisor Dr. Nghiem Quoc Minh, Faculty of Information Technology at Vietnam National University - University of Science for guiding us during the making of this thesis.

We would like to extend our thanks to all lecturers and teacher assistants in Advanced Program in Computer Science, VNU-HCMUS, who instructed us during our four years of Bachelor Degree of Computer Science. We would like to thank Faculty of Information Technology for providing us with the hardware equipments to carry out experiments and develop our thesis.

Finally, we would like to thank our family, university staffs, and friends in APCS for supporting us.

Ho Chi Minh City, 01/08/2019
Signature(s) of student(s)

Tuan-Anh Nguyen Duy-Tue Tran-Van

Contents

List of Figures

List of Tables

ABSTRACT

The abundance of information on the internet has become more and more problematic. The need for a efficient yet reliable way of conveying information from news articles and research papers is increasing as we have so much data to cover. Automatic text summarization is an area of research which investigates the problems and potential solutions for the act of automatically conveying important information from text documents. Researchers have been tackling the task of automatic summarization since the late 1950s. Multiple novel approaches have been proposed with great success. Moreover, since the introduction of various large-scale datasets on automatic summarization and the blooming successes in the field of deep learning, we are gradually bridging the gap between human and computer in text summarization.

In this thesis, we examine one particular problem in this field, extractive summarization of news articles. First, we try to combine the powerful Bidirectional Encoder Representations from Transformers for contextualized sentence embeddings with an auto regressive decoder for jointly sentence scoring and selecting. We then carry out experiment on oracle summaries and recognize the drawbacks of current approaches in extracting important information towards the end of the input articles, which leads to a huge loss in ROUGE evaluation scores. We proceed to propose an approach of utilizing the whole article in Bidirectional Encoder Representations from Transformers for better sentence representations encoding.

On the CNN/DailyMail dataset, we show the results and lessons learned from our first method as it turns out to be not a feasible option. With our proposed method of using the whole article, we achieve state-of-the-art results with 43.31 and 39.77 on ROUGE-1 and ROUGE-L scores respectively and with 34.42 average ROUGE score in extractive summarization task.

Chapter 1

Introduction

1.1 Overview

1.1.1 Text Summarization

Text summarization is the process of creating a concise summary of the most important information from a source or multiple sources such as pieces of text or news articles. The main objective of any automatic summarization system is to extract the important information from the source document and create a summary out of those information. Different types of summaries focus on specific aspects of the source document. For example, extractive summarization system concentrates on solving the problem of deciding which sentences are important. Other summaries focus on extraction of information instead of getting the whole sentences depending on the needs of the summary.

Summarization is a simple task for human because we have developed the capacity to understand the meaning and retrieve the related information from a text document to generate a summary using our own words and style. Automatic text summarization, on the other hand, still struggles to create a coherent and fluent summary for documents. However, we cannot denied the crucial role of automatic summarization in today's world as the amount of data is increasing exponentially day by day and we need a efficient way to convey information from such huge sources.

In this thesis, we review some of the techniques and approaches in text summarization from early traditional methods to complex deep neural network systems. We also provides some basic information and observations on related subjects as well as dataset. Finally, we show our method of tackling extractive summarization on the CNN/DailyMail [49] dataset,

explain our rationale for our decisions, and comment on the experiment results that we carried out.

1.1.2 Approaches in Text Summarization

1.1.2.1 Classical approaches

Extractive methods

Early works in extractive summarization concentrate on the task of ranking the importance of information at sentence level. They used statistical tests to rank each sentence according to their relation with other sentences. Notable works in this categories are Luhn et al. with frequency of high descriptive words [1], Edmundson with multiple statistical features to indicate sentence importance [2], TF-IDF weighting of words by Salton et al. [4] and Jones et al. [18], log-likelihood ratio test for identifying topic signatures by [14], *Latent Semantic Analysis* by Gong and Liu [15], sum of weighted similarities with graph-based approach by Mihalcea and Tarau [20].

Machine learning algorithms were also utilized to train on appropriate features to decide which sentence should be included in the summary. Wong et al. trained supervised and semi-supervised models (Support Vector Machine and Naïve Bayes) on sentence features (surface, content, event, and relevance features) to decide the importance of sentences. A ranking algorithm will then revise the order of sentences accordingly [25].

Abstractive methods

Non-extractive or abstractive methods mainly focus on identifying the most salient information in the source document and generating a coherent summary. Scripts and templates for slot fillings were used to produce abstractive summaries [3, 10]. Techniques such as sentence revision [12, 11], sentence fusion [24], and sentence compression [13, 16, 22] are usually used in generating summary. These techniques create summaries by modifying, substituting, removing, or combining words and information from the input sentences using prior or learnt knowledge.

1.1.2.2 Deep learning approaches

The rapid development of deep learning has enabled computers to achieve incredible results on multiple tasks across different fields of research. With the appearance of multiple

large summarization datasets such as English Gigaword [39] and CNN/DailyMail [49], complex deep learning models have shown significant improvement over traditional methods. The availability of large amount of data and novel techniques are pushing the boundary of automatic summarization to further close the gap with human performance.

In general, neural-based summarization systems utilize the following process (Figure 1.1). The source document is extracted into sentences which consist of word embeddings for each token. These sentences are then encoded using the embeddings to extract the information from the source document. Finally, a summary will be produced by a model that takes document representation from previous step as its input. This model is decided based on the type of output summary that we want (e.g. extractive or abstractive).

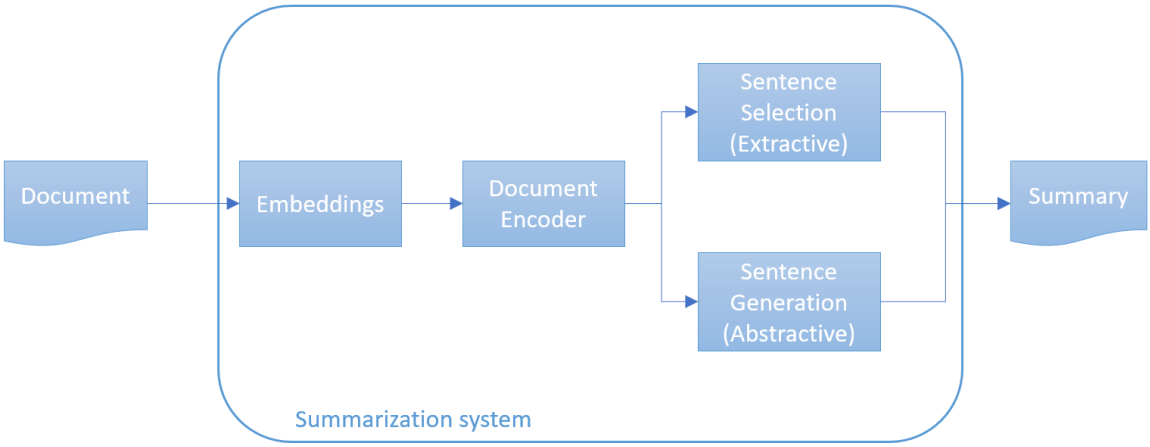


Figure 1.1: General pipeline of a text summarization system.

Since the introduction of some large summarization datasets, researchers have come up with more innovative techniques for automatic text summarization. In Chapter 3, we show some of the significant works that set the foundation and new directions for neural-based summarization models.

1.2 Motivation

In this thesis, we focus on extractive summarization as it is more computationally efficient than abstractive approach and it can generate summaries with better grammatical and coherent quality [47].

Recent works [62] [76] investigate different factors of model architecture to the performance of extractive summarization, especially the choice of sentence encoder, document encoder, and decoder (details can be found at section 3.3.5). However, they do not carry out experiments on encoders with transfer knowledge from BERT (BERT encoders) [58]. Due to the prevalence of BERT application in various NLP tasks [70][73][75], we believe that it is necessary to try BERT encoders along with different decoder architectures consisting of auto-regressive and non-regressive model (detailed can be found at section 3.3.5). Recently, L. Yang, 2019 [69], tries BERT encoders with non-regressive approach in extractive summarization. However, this raises the question whether auto-regressive decoder on the same encoder can improve the performance in the same task. This motivates us to try this option on BERT encoders from the above work of L. Yang.

Furthermore, Liu [69] and Zhou et al. [74] use BERT encoders have to truncate input articles to the first 512 work tokens due to the architectural constraint of BERT [58]. This approach do not use full input articles, thus potentially losing important and salient information in the tail of the articles. For example, the truncated version of one input article in Table 1.1 does not have the information "*up to 70 % of nigerians live on less than a dollar a day*", which is the last sentence of golden summary. Therefore, we propose our model to adapt BERT encoders from L. Yang 2019 [69] to learn full information from input articles without truncation.

In conclusion, we have two objectives:

- Seek for a better understand how auto-regressive and non-regressive decoders benefits BERT encoders.
- Propose a model to take use of full input articles without truncating first token words like existing works.

1.3 Contributions

With above goals, we make following contributions:

- We find that auto-regressive decoder lags behind non-regressive decoder with the same BERT encoder of L.Yang 2019 [69].
- For extractive summarization task, we found that utilizing the whole article improves oracle evaluation scores by a large margin.

Table 1.1: An example to show that truncated input article do not have the information in the golden summary

Truncated article (the first 512 tokens): Lagos, Nigeria (CNN)A day after winning Nigeria's presidency, Muhammadu Buhari told CNN's Christiane Amanpour that he plans to aggressively fight corruption that has long plagued Nigeria and go after the root of the nation's unrest. Buhari said he'll "rapidly give attention" to curbing violence in the northeast part of Nigeria, where the terrorist group Boko Haram operates. By cooperating with neighboring nations Chad, Cameroon and Niger, he said his administration is confident it will be able to thwart criminals and others contributing to Nigeria's instability. For the first time in Nigeria's history, the opposition defeated the ruling party in democratic elections. Buhari defeated incumbent Goodluck Jonathan by about 2 million votes, according to Nigeria's Independent National Electoral Commission. The win comes after a long history of military rule, coups and botched attempts at democracy in Africa's most populous nation. APC Party: Nigerian Pres. Jonathan has conceded 03:20 In an exclusive live interview from Abuja, Buhari told Amanpour he was not concerned about reconciling the nation after a divisive campaign. He said now that he has been elected he will turn his focus to Boko Haram and "plug holes" in the "corruption infrastructure" in the country. "A new day and a new Nigeria are upon us," Buhari said after his win Tuesday. "The victory is yours, and the glory is that of our nation." Earlier, Jonathan phoned Buhari to concede defeat. The outgoing president also offered a written statement to his nation. "I thank all Nigerians once again for the great opportunity I was given to lead this country, and assure you that I will continue to do my best at the helm of national affairs until the end of my tenure, " Jonathan said."I promised the country free and fair elections. I have kept my word."Buhari, 72, will be sworn in on May 29. He will take the helm at a critical time, as Nigeria grapples with Boko Haram, serious economic woes and corruption. A leader again This isn't Buhari's first time leading Nigeria, but it's his first time in nearly 30 years. Who is Nigerian President-elect Muhammadu Buhari? A military coup brought Buhari to power in late 1983, closing a brief period of popular rule by Shehu Shagari. But Buhari himself was ousted by another military coup in August 1985. Read more: Who is Nigeria's Muhammadu Buhari? His presidential win is the result of his fourth attempt to lead the country since he was ousted 30 years ago. Buhari is a Sunni Muslim from Nigeria's poorer North, while Jonathan comes from a Christian and animist South that is rich with oil. Buhari praised voters for exercising their right peacefully. "Your vote affirms that you believe Nigeria's future can be better than what it is today," he said in his statement. "You voted for change, and now change has come." Buhari campaigned as a born-again democrat to allay fears about his strict military regime. He stressed that Nigeria's security needs to be the next government's focus.
Golden summary: muhammadu buhari tells cnn 's christiane amanpour that he will fight corruption in nigeria . nigeria is the most populous country in africa and is grappling with violent boko haram extremists . nigeria is also africa 's biggest economy , **but up to 70 % of nigerians live on less than a dollar a day .**

- We proposed our model that utilizes the full length of articles to extract more sentence vectors, hence having better document context. We achieve the state-of-the-art performance in extractive summarization on the CNN/Daily Mail dataset in terms of ROUGE-1 and ROUGE-L scores.

1.4 Structure of this thesis

- **Chapter 2** introduces some basic knowledge in summarization types as well as Natural Language Processing, and Deep Learning for our summarization system.
- **Chapter 3** shows some of the works that are fundamental and closely related to our main approach for the summarization problem. The chapter contains 3 sections:
  - Section 3.1 presents some of the early approaches in extractive summarization by using traditional methodologies.
  - Section 3.2 presents several works that tackle extractive summarization through the mean of machine learning and hand-crafted features.
  - Section 3.3 presents multiple recent studies on extractive summarization which utilize the help of deep neural networks and large scale datasets.
- **Chapter 4** shows our proposed methods for automatic text summarization. We presents our choice of model and training methods. We additionally explain the reasons behind our decisions.
- **Chapter 5** introduces the dataset that was used in this thesis. It also provides information on statistics, and in-depth detail of the dataset that we found during the process of working on the system. We also introduce and explain the main metrics that are commonly used in evaluating automatic summarization system. We also present and explain our rationale for all of our experiments. We then analyze and discuss the result of the model with respect to other related models in extractive summarization task.
- **Chapter 6** concludes this thesis with our method and results as well as shows our future works on automatic text summarization.

Chapter 2

Background

2.1 Types of summaries

There are multiple types summaries. These types can be categorized based on different factors: type of input, type of output, purposes, output formats, etc [9]. Each type of the summaries serves a unique objective for the summarization task.

Input factors

The summarization tasks in the early day mostly had to handle a *single input document* such as a news article or scientific research paper. However, in the recent years, with the exploding amount of information on the internet, summarization systems have to create a summary from *multiple input documents* on a common topic. The multi-document summarization task provides a feasible solution for clustering large number of news articles based on a specific event or a user interest.

Moreover, based on the language of the input document(s), there are also *monolingual*, *multilingual*, and *cross-lingual* summarization tasks. *Monolingual summarization* system generates summaries that are in the same language as the input documents. *Multilingual summarization* system generates summaries that are also in the same language as the inputs, but it can accomplish that across several different languages. *Cross-lingual summarization* system, on the other hand, works on input-output pairs that are in different languages.

Output factors

There are two types of output summaries: extractive and abstractive. *Extractive summaries* consist of the exact sentences taken directly from the source documents. *Abstractive*

summaries contain the information from the source document but expressed in the words of the summary author. These two types of summaries are the main focus of most researches recently because of the introduction of the large-scale CNN/DailyMail dataset [49].

In this thesis, we will focus on abstractive summarization techniques. In abstractive summarization, given a document, the system must distill the core ideas of the document and encode them into feature representations. A decoder will then use these feature representations to sequentially generate words in summaries accordingly. One of the main problems in abstractive summarization is that the model has to generate coherent summaries without copying all the words from the source document. In addition, it has to deal with different sub-problems depending on the datasets [49, 39].

Purpose factors

The summary can either act as a replacement for the source document as it contains all important facts (*informative summary*), or enable the reader to quickly grasp the overall contents of the document but cannot be a substitution for the source (*indicative summary*). Furthermore, most of the time, summaries serve a generic purpose of conveying all important information from the input document(s) without knowing who's the audiences and what information they may need. There are also *user-oriented summarization* system that takes a query from the user and generates summary that is only relevant to the user's need. Summarization systems can also be categorized into *general purpose* systems and *domain-specific* systems.

Output format

The format of the output is also considered to differentiate types of summaries. The summarization system can generate *paragraph-length* summaries, *keywords and phrases* only, or a *headline* (single sentence).

2.2 Network architectures for automatic text summarization

2.2.1 Recurrent Neural Networks

Traditional neural networks such as Multi-layer Perceptron and Convolutional Neural Networks have achieved impressive results and even surpassed human performance on several tasks across different research fields. These architectures excel at their respective tasks when the inputs and outputs are independent of each others. However, when it comes to modelling sequences of data, they cannot capture the nature of this type of data. Recurrent Neural Networks (RNNs) are designed perfectly for this type of problem. RNNs have a special hidden state that can memorize the previous computed information. This is the key for RNNs to effectively remember and represent data sequences.

In Natural Language Processing, sequences of data are the most common type of inputs and outputs. That is the reason why RNNs are usually the first choice when it comes to choosing an architectures for a problem in this field of research. Researches in different tasks such as machine translation [46, 33], speech recognition [27, 38], question answering [43, 41], sentiment analysis [40, 51], text summarization [39, 49], etc. have proved the significance of RNNs in these problems.

2.2.1.1 Definition

RNNs, in the simplest way, can be thought of a normal Multi-layer Perceptron with loops. Apart from moving inputs from an input layer through a hidden layer to an output layer, RNNs also pass information sideway between hidden units. With this flow of information, RNNs are able to take into account information from previous steps which is the key to understand data in sequences. In general, to process a sequence of vectors x, at each time step t, we apply a recurrence formula:

$$h_t = f_W(h_{t-1}, x_t) \tag{2.1}$$

where h_t and h_{t-1} is the hidden state at time step t and $t-1$ respectively, x_t is the input vector at time step t, and f_W is the activation function that is used at every time step.

A simple illustration of a RNN architecture is shown in Figure 2.1. The recurrent part of an RNN on the left of Figure 2.1 is shown in detail on the right side of the figure. The output

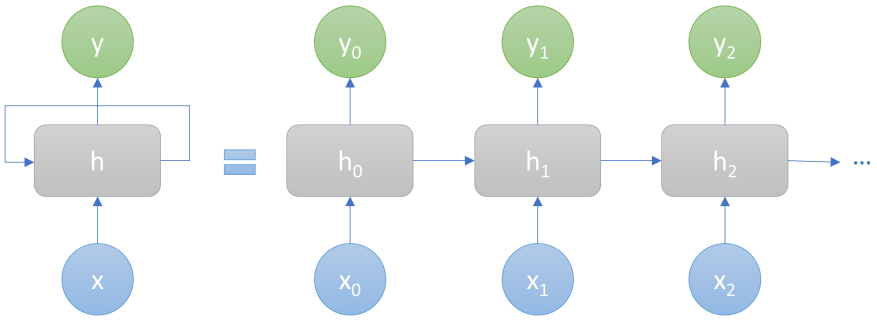


Figure 2.1: Simple illustration of a RNN on an input sequence.

of a hidden state at a time step will be the input of the next step along side with the input from the sequence. This architecture allows RNNs to pass information from previous steps to later parts of the network, which enable the networks' ability to utilize the sequence nature of data. Moreover, the network can also takes variable-length sequence of inputs, which is another great advantage of RNNs over other architectures especially in Natural Language Processing.

2.2.1.2 Backpropagation in RNNs

RNNs are a variation of feed-forward networks. Hence, backpropagation is the technique to train this type of network. However, if we keep the network with the loop as in the concept, backpropagation cannot happen. Therefore, to perform backpropagation on RNNs, the network has to be unfolded into series of neurons which share the same parameters. In other words, instead of having a cell with a connection to itself (C $\longrightarrow$ C), we use two cells with the exact same weight values to represent the recurrent property of RNNs (C1 $\longrightarrow$ C2). By unrolling the network like in Figure 2.1, we can train the network with backpropagation like other feed-forward networks.

Since RNNs can have a large number of steps depending on the input sequence of the dataset, backpropagation will take forever if we do normal backpropagation through all the units. If we want to compute the gradient of a hidden cell in the beginning of the network, like h_0, we need to backpropagate through every other RNN cells. This raises the computational cost of the network by a large amount, especially ones for long input sequences. In order to do backpropagation in a reasonable manner, we run the forward and backward passes through chunks of the sequence instead of the whole sequence. This means that hidden states will be carried forward as long as the input sequence, but we only backpropagate for a smaller

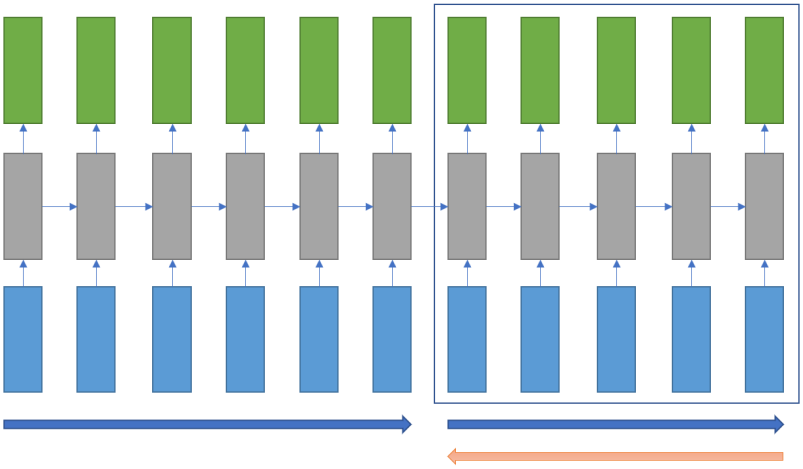


Figure 2.2: Truncated Backpropogation through time. Reproduced from image of Fei-Fei Li et al.

number of steps (Figure 2.2). This technique is call Truncated Backpropagation through time [21].

Moreover, RNNs can have several layers as well as a lot of neurons to handle long sequences. But as the network gets bigger, the problems of vanishing gradients and exploding gradients become more and more significant [5]. Exploding gradients cause the training unstable due to large values of loss and gradients. One solution to mitigate this issue is gradient clipping [26] which takes into account the L_2norm of the computed gradient and adjusts according to a threshold. On the other hand, vanishing gradients happen when the gradient becomes very small, which will make it really hard for the model to know which direction to improve the cost function [77]. Fortunately, with the introduction of Long Short-Term Memory architecture [7], vanishing gradients no longer pose a problem in training RNNs.

2.2.1.3 Long Short-Term Memory units

In the original vanilla RNN cell, the value of hidden state h_t is computed using the equation 2.1. Specifically, it uses a $tanh$ activation function on the matrix multiplication of weight matrix W with previous hidden state value h_{t-1} and the current input x_t at time step t (biases are excluded for clarity):

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t) \tag{2.2}$$

When performing backpropagation from h_t to h_{t-1}, we have to multiply the gradient with

the same transposed weights W_{hh}^T repeatedly through the whole network. This repeating matrix multiplication and also repeated $tanh$ activation is the main reason why exploding (multiply with value larger than 1) or vanishing (multiply with value less than 1) gradients happen [5].

In 1997, Hochreiter and Schmidhuber proposed a new cell architecture called Long Short-Term Memory (LSTM) which alleviates the problem of vanishing gradients in training RNNs [7]. LSTM contains gates that decide how they handle the information outside the original flow of RNNs. These cell gates are input gate (denoted i), forget gate (denoted f), output gate (denoted o), a special gate to "block" input (denoted g). These gates have the following formulas (biases are excluded for clarity):

$$i = \sigma(W_{ih}h_{h-1} + W_{ix}x_t) \tag{2.3}$$

$$f = \sigma(W_{fh}h_{h-1} + W_{fx}x_t) \tag{2.4}$$

$$o = \sigma(W_{oh}h_{h-1} + W_{ox}x_t) \tag{2.5}$$

$$g = tanh(W_{gh}h_{h-1} + W_{gx}x_t) \tag{2.6}$$

Each LSTM cell has two important states: cell state and hidden state. The cell state is private to the memory cell only (Equation 2.7). It decides how the cell deals with storing and reading information. Meanwhile, the hidden state is what the memory cell shows to the outside (i.e. passes on to the next cell). The formulas of these states are as follows:

$$c_t = f \odot c_{t-1} + i \odot g \tag{2.7}$$

$$h_t = o \odot tanh(c_t) \tag{2.8}$$

In equation 2.7, the first element-wise $\odot$ product between forget gate f and previous time step cell state c_{t-1} decides what information to remember, what to forget from previous time step by using the sigmoid function of the forget gate (which produces value range from 0 to 1). The input gate i from equation 2.3 tells us whether we want to write to each element of the cell state. The block gate (equation 2.6) decides on how much information we want to write to cell state because of the values ranged from -1 to 1 from the $tanh$ activation function. The cell equation 2.7 shows us how the memory cell remember or forget previous state as well as the amount of memory information that we want to write to a cell. Now that we got the cell memory from equation 2.7, we need to decide how much we want to reveal the memory to the next step of the network.

During the forward pass, each LSTM cell passes on its hidden state and cell state. Be-

cause in LSTM cell, these values are computed using simple element-wise $\odot$ product (complex matrix multiplication in vanilla RNN cell), the control of gradient flow of theses states is much nicer and easier. In addition, the sigmoid activation functions keep the values at a reasonable range which also helps ease the process of computing local gradient for W. Therefore, this design of the LSTM cell keep the gradient flow uninterrupted which minimizes the effect of vanishing gradients in training RNNs.

2.2.1.4 Gated Recurrent Units

Gated Recurrent Unit (GRU) was first introduced as a new type of hidden unit that "adaptively remembers and forgets" in 2014 by Cho et al. [30]. GRU is considered to be a variation of LSTM cell as it is designed with similar gates and also solves the problem of vanishing gradients. The new architecture of GRU is simpler and requires computational cost than LSTM. GRU has only two gates: update gate z and reset gate r. The update gate z decides whether the new hidden information is worth remembering while the reset gate r determine when to forget previous memory. An illustration of an GRU can be found in Figure 2.3.

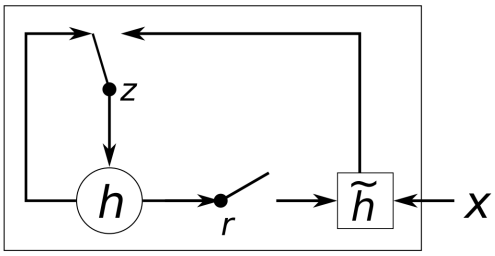


Figure 2.3: Architecture of an GRU. Image of Cho et al., 2014 [30].

Each GRU will have the following equations (biases are excluded for clarity):

$$z_t = \sigma(W_{zh}h_{t-1} + W_{zx}x_t) \tag{2.9}$$

$$r_t = \sigma(W_{rh}h_{t-1} + W_{rx}x_t) \tag{2.10}$$

$$\tilde{h_t} = tanh(W_h(r_t \odot h_{t-1}) + W_{xh}x_t) \tag{2.11}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h_t} \tag{2.12}$$

The first two equations for z_t and r_t just multiply the input and previous hidden state with their respective weight matrices and squash them down to 0-1 range by using a sigmoid activation function. The equation for $\tilde{h_t}$ computes the current hidden state of the memory

cell. If the output of the reset gate r_t is close to 0, it means that the GRU will forget the previous hidden memory h_{t-1}. Finally, to compute the hidden state h_t that the cell will pass on to the next block, we use the update gate to decide whether the current information need to be remembered or ignored. Specifically, when z_t is close to 0, the unit will drop most of the current memory in $\tilde{h_t}$ and keep the past information as it is still relevant to the context. With each unit has their own update and reset gates, the GRU has the capability of capturing both short and long term dependencies as it can decide how long it keeps the previous memory [30].

2.2.1.5 Bidirectional Recurrent Neural Networks

In certain Natural Language Processing tasks such as speech recognition, there may exist ambiguity at an input step given just the previous ones. In order to better understand the context at a given point in time, apart from what has been seen, we also need to see what is coming next in the sequence [27]. Bidirectional Recurrent Neural Networks (introduced in 1997 by Schuster et al. [8]) in general, bidirectional LSTM specifically have shown significant performance improvement over traditional RNNs in the speech recognition task [27].

Bidirectional Recurrent Neural Networks in general is just a simple basic RNN with an additional hidden layer. This additional hidden layer takes input in the reversed order of the sequence. Let $X = \{x_0, x_1, \ldots, x_n\}$ be the input sequence of the left-to-right hidden layer, then the right-to-left hidden layer of the bidirectional RNN will take $\overleftarrow{X} = \{x_n, x_{n-1}, \ldots, x_0\}$ as their respective input. The network will then take the concatenation or summation of the outputs of these two hidden layers to be its outputs. Figure 2.4 shows the diagram that illustrate the simplified architecture and information flow of a bidirectional RNN.

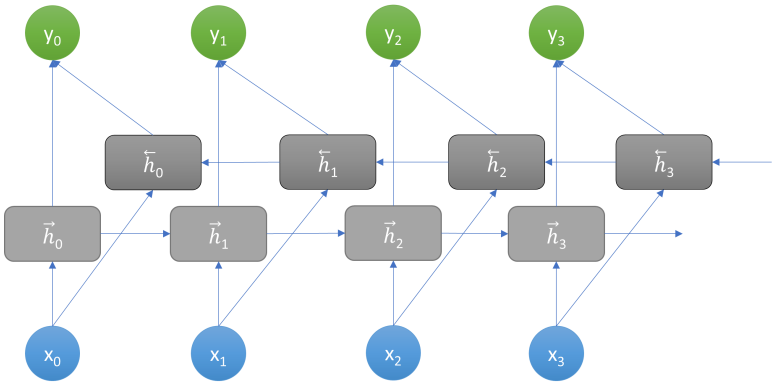


Figure 2.4: Architecture of a Bidirectional Recurrent Neural Network.

2.2.2 Encoder-decoder architectures

The first sequence-to-sequence models using neural networks (specifically RNNs) were introduced by Sutskever et al., 2014 [33] and Cho et al., 2014 [30]. They have shown comparable and even better results on Machine Translation tasks. They also pointed out that this architecture can perform better on limited vocabulary and long sentences [33] and also generating "well-formed" phrases [30]. In text summarization problem, the main objective is to produce an output summary with length from a few words to a paragraph from a source document. This means that the system has to take in a sequence of tokens (i.e. words) from the input and generate another sequence of tokens for the summary. This is why sequence-to-sequence models are commonly used in text summarization.

In fact, below the surface, a text summarization sequence-to-sequence model consists of mainly an encoder and a decoder. The encoder takes in the input sequence, compiles its information to creates a fixed size vector that represents the context of the input. The context vector will then be used as the decoder's input for the next step. The decoder generates output sequence according to the context of the input sequence which was encoded earlier. An illustration of an encoder-decoder system can be found in Figure 2.5.

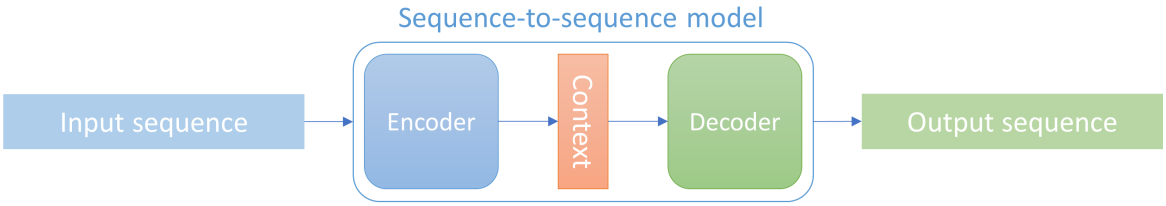


Figure 2.5: A simple illustration of an encoder-decoder system.

In case of an RNN encoder-decoder model, the encoder and decoder are usually RNNs or sometimes bidirectional RNNs (mostly the encoder) in a text summarization system. The encoder forwards input tokens one by one in the traditional RNN fashion. We take the hidden state of the RNN after feeding the last token (usually a special end token) in the sequence as the context vector of the whole sequence. The decoder iteratively generates tokens y_t at time step t for the output by using the context c of the input document from the encoder, the previous hidden state h_{t-1}, and the previous generated token y_{t-1}. The hidden state for time

step t of the decoder is computed by:

$$h_t = f(h_{t-1}, y_{t-1}, c) \tag{2.13}$$

With the new hidden state, the decoder uses a probability function $g(h_t, y_{t-1}, c)$ (e.g. softmax) to calculate the distribution of the next tokens.

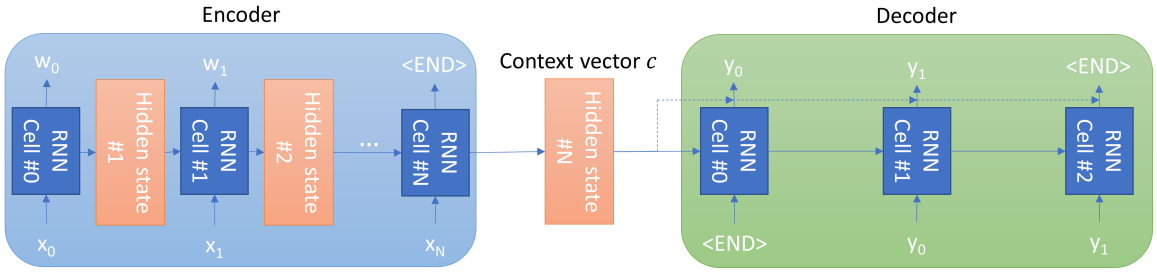


Figure 2.6: Detail information flow of a RNN encoder-decoder system.

As in Figure 2.6, the encoder distills the main information of the source document and stores it in a fixed size context vector c. The decoder then uses this context vector as both the initial hidden state of its RNN as well as concatenates it (dashed line) with the output at each time step for better word prediction. Moreover, the decoder takes the last output token of the encoder ($< END >$ in this case) to be its initial input. The generated token of the decoder at each time step t will be the input of the RNN cell. The decoder stops when an $< END >$ token is generated.

However, as pointed out by Bahdanau et al., 2014, the squashing of all the context information of a sentence into a fixed size vector during the encoding step is the bottleneck of the encoder-decoder model [34]. For long sentences, the model struggles to capture all the information into a single vector, hence, results in poor performance in its respective tasks [29]. In Figure 2.6, only the hidden state of the last step in the encoder RNN is used as the context vector for the decoder. This can cause information loss if the input sequence is too long. In order to overcome this issue, Bahdanau et al. proposed a novel attention mechanism which will be explained in § 2.2.3.

2.2.3 Attention mechanism

Attention mechanism was first introduced by Bahdanau et al. in 2014 [34] as an simple yet effective solution to convey more context information from the source input in Machine

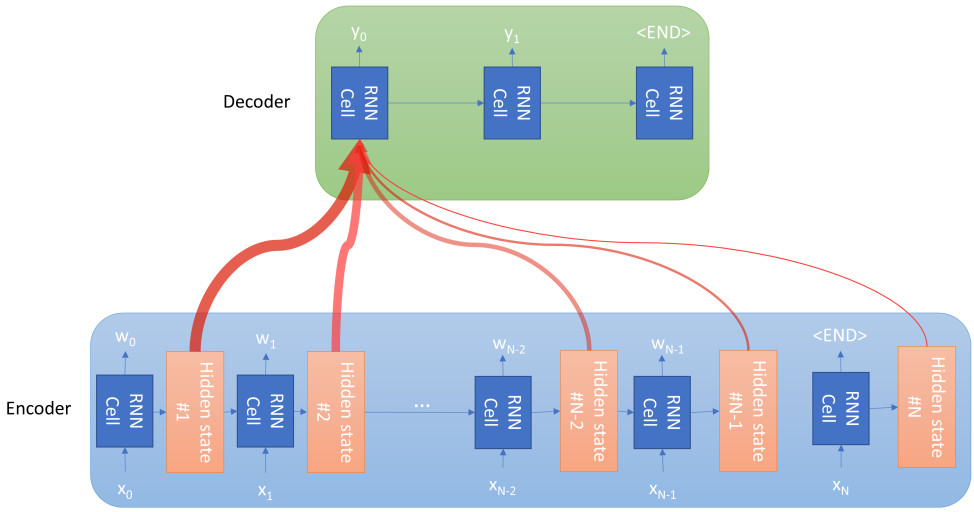


Figure 2.7: Attention mechanism in an encoder-decoder sytem.

Translation task. The mechanism has been successfully adapted and achieved significantly better performance in text summarization tasks (earliest approach is by Rush et al., 2015 [39]). In general, attention allows the decoder to focus on the most relevant parts of the input sequence while generating words at each time step. Instead of squeezing all the context into one single fixed size vector, it attends information at all encoding steps and chooses the most relevant part for each decoding step. This mechanism mitigates the drawback of single fixed size context vector in traditional encoder-decoder model.

In a normal encoder-decoder system, the context information is encoded into a single vector (which is the last hidden state of the encoder RNN). An attention-based encoder-decoder system, on the other hand, passes all hidden states of the encoder RNN to its attention decoder. The attention decoder receives a sequence of hidden vectors from the encoder and scores them according to its current input at time step t. The most relevant hidden vector will have the highest score while the others get lower scores. This highest score hidden vector is used to compute a context vector c_t. This context vector c_t along with the current hidden vector h_t of the decoder RNN are concatenated and used to compute the output token of the time step t by passing through a feed-forward network.

Figure 2.7 represents the attention mechanism in an encoder-decoder system at a high level abstraction. At each time step t of the summarization process, the decoder attends to all the context vectors (hidden states) of the encoder. It will decide which context vector is the most relevant to the current input being fed. This decision is made according to the output

scores of a softmax layer. The most relevant vector is assigned with a high score (thicker line in the figure), which contributes significantly to what word the decoder will generate. Other unrelated parts will have smaller or close to zero scores, represented by thin line in the figure.

The attention mechanism plays an essential role in helping the summarization system correctly produce summary from the most important information of the source document. Moreover, it also dramatically increases the efficiency in which the network aggregates information while generating tokens. The Transformer network, introduced in 2017 by Vaswani et al., has shown the significant impact on performance and speed when utilizing attentions in Machine Translation task and multiple other tasks [55].

2.2.4 Pointer-generator network

In text summarization problem, dealing with out-of-vocabulary (OOV) words is always a challenging obstacle that the model has to overcome, especially in abstractive summarization task. In the traditional encoder-decoder system (sequence-to-sequence system), the model encodes the whole input sequence into one (or multiple) vector(s) for the decoder to generate output tokens. This process of compressing information makes it extremely difficult for the model to copy a word from the source document. For example, when encountering a rare word which usually has weak word embedding during training, the network cannot distinguish it among other out-of-vocabulary words since these are all likely to be clustered together. Therefore, it maybe impossible to reproduce correctly while generating tokens.

In 2017, See et al. proposed a network architecture called Pointer-Generator network based on the pointer network by Vinyals et al., 2015 as a solution to this OOV problem. The pointer-generator architecture allows the network to decide whether to copy from source or generate a new token when producing summary. This alleviates the issue of OOV words as the network can choose to copy the word it does not know directly from the source via pointing. To control the balance between copying and generating, a generation probability p_{gen} is used as a soft switch to choose whether to generate a new word from the vocabulary or to copy a word by sampling the distribution from attention scores.

Figure 2.8 shows the process of a Pointer-Generator network in the decoding stage. At each time step, p_{gen} is computed using the context vector and the information from the current decoding step. Besides, an attention distribution and a vocabulary distribution will also be computed for the current input. These three components are then combined together to generate a word distribution that will result in an output word at that time step.

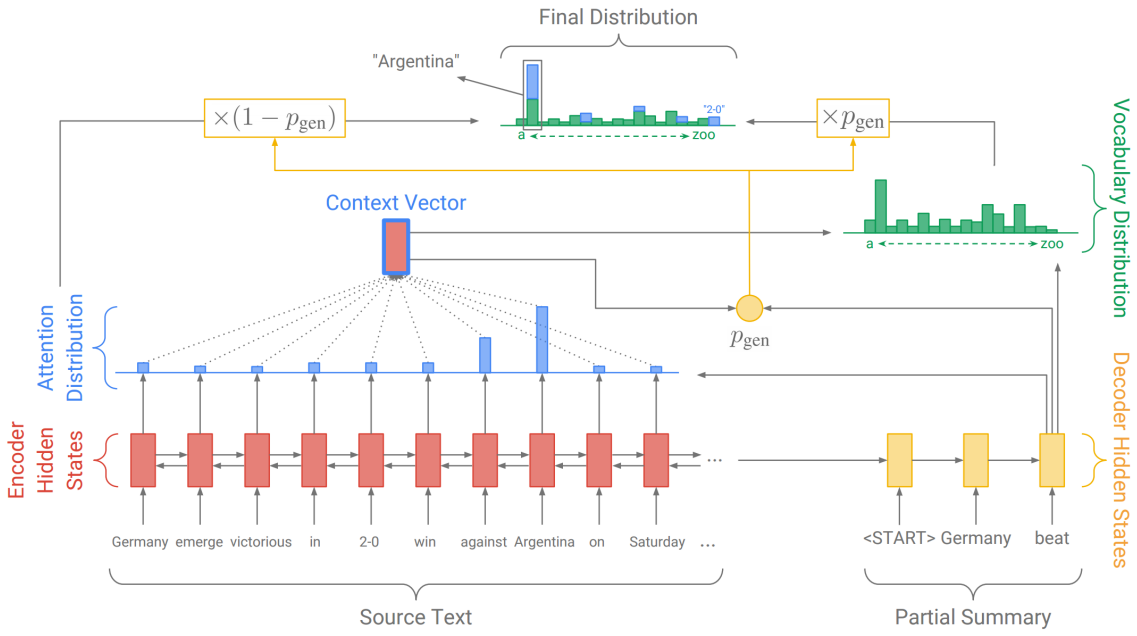


Figure 2.8: Architecture of a Pointer-Generator network. Image of See et al., 2017 [54].

The p_{gen} probability is defined as follows:

$$p_{gen} = \sigma(w_{h_t^*}^T h_t^* + w_{s_t}^T s_t + w_{x_t}^T x_t + b) \tag{2.14}$$

where $w_{h^*}, w_{s_t}, w_{x_t}$ are weights for the context vector h_t^*, decoder state s_t, and current decoder input x_t at time step t. σ is the sigmoid function to compute the probability p_{gen}. The probability of the model generating the word w is the weighted sum of the attention distribution a and the vocabulary distribution P_{vocab}. The generation probability p_{gen} is used as the weight between two distributions. The final distribution $P(w)$ is calculated as follows:

$$P(w) = p_{gen} P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i} a_i \tag{2.15}$$

With the pointer-generator mechanism, the summarization system will have some 2 major advantages over traditional sequence-to-sequence models:

- The model is better at handling OOV words while training and generating summaries. This allows us deal with unseen words while maintaining small memory footprint by using a much smaller vocabulary.
- It is easier to copy words directly from the source document.

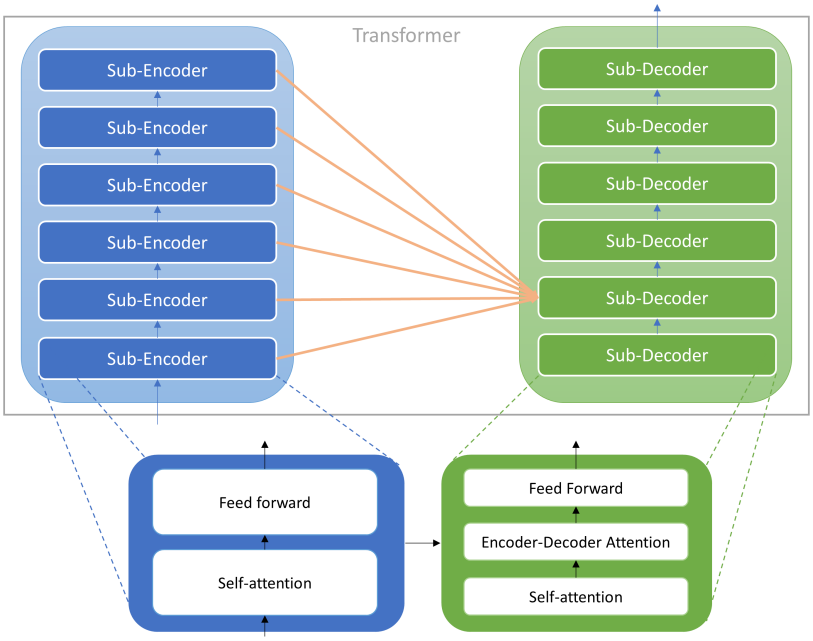


Figure 2.9: Architecture of a Transformer network.

2.2.5 Transformer network

Attention mechanism has been shown to improve performance of machine translation models and text summarization significantly [37, 34, 39, 54]. It allows the model to quickly gather information from the whole input sequence while generating tokens. This has inspired Vaswani et al. to create Transformers, an encoder-decoder model mainly based on the self-attention mechanism, to not only achieve great results but also at a impressive speed in machine translation task [55]. In the following part, we will try to explain how a Transformer network works as it plays an important role in our network architecture later in **Chapter 4**.

Underneath a Transformer is basically an encoder-decoder network. However, the encoder here consists of many other small encoders (6 in the paper) and so does the decoder with 6 sub-decoders. All sub-encoders are identical in architecture but they are not sharing the same weights. Each of them contains 2 modules: a self-attention module and a feed-forward network. The sub-decoders also have these modules and an additional attention layer that helps them connect and gather information from the encoder (similar to that in Section 2.2.3).

As in Figure 2.9, the Transformer consists of a stack of 6 encoders on the left and another stack of 6 decoders on the right. Zoomed in details of the sub-encoder and sub-decoders are at the bottom of the figure. The encoder receives a list of word embeddings of size 512 and

pushes through its self-attention layer. The self-attention module helps the encoder look and attend information at other parts of the input sequence while it is encoding a specific word. The outputs are then fed through a feed-forward network to get another list of vectors of the same size for the next encoder.

Self-attention mechanism

What makes Transformer special is that it can models the relationship between words in a sentence just in a small number of steps. To achieve this, Transformer uses self-attention mechanism while encoding a sentence. Let us explain self-attention through a simple example. Suppose we have an input sentence to encode: *"The animal didn't cross the street because it was too tired."*. For us, we immediately refer the word **"it"** to **"The animal"** because of the word **"tired"** at the end of the sentence. Computers, however, may have problem deciding this. For the traditional RNNs architecture, at the encoding step for **"it"**, the network has not seen the word **"tired"** yet, hence it has difficulty in deciding which word **"it"** related to the most (**"The animal"** or **"the street"**).

Specifically, with self-attention mechanism, when computing the vector representation for the word **"it"**, Transformer looks at every other word in the sentence. It gives each other word an attention score, which determines how related or how much that word contributes to the representation of the current word **"it"**. The attention scores becomes the weights for their respective words and the weighted sum of all the word representations is fed through the feed-forward layer of the encoder to compute the representation scores for the word **"it"**. In this case, the new word representation indicates that the word **"it"** strongly reflects **"The animals"**.

Figure 2.10 illustrates the attention scores given for each other word in the sentence while encoding **"it"**. The stronger color intensity of the words **"The animal"** indicates their high attention scores. Therefore, the newly generated representation will be close to the meaning of **"The animal"** instead of **"the street"**.

In self-attention mechanism, while encoding a specific word, the self-attention layer will give each other word its attention score with respect to the current word being encoded. The attention score for a word is calculated as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.16}$$

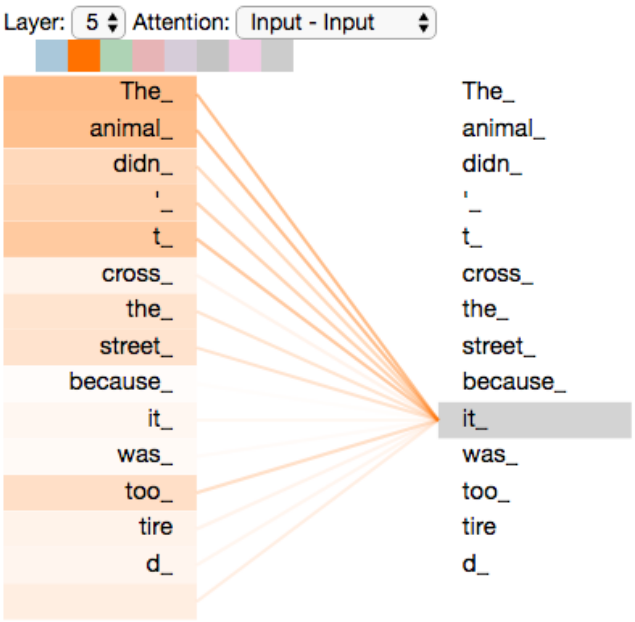


Figure 2.10: Visualization of self-attention of Transformer while encoding word representation for the word **"it"**. Image from visualization tool of Vaswani et al. [66].

where Q, K, V are matrix of query vectors, key vectors, and value vectors of the words in the current sentence; $d_k = 64$ is the dimension of the key vectors. The three vectors Q, K, V are calculated from the input embedding vector by multiply it with the respective weight matrices W_Q, W_K, W_V. These weights are optimized through the training process.

In equation 4.6, first we have to give a score for each word in the input sentence against the current word by using dot product on the query matrix Q with the key matrix K. In other words, for every words in the sentence, we take it key vector k_i and multiply it with the query vector q of the current word being encoded. The result of the multiplication above is the score of that word with respect to the current word. For a sentence of 3 words, if we are computing the self-attention score for the word at the #1 position, we need to take the q_1 query vector and get the score for the first, second, and third word with $q_1 \odot k_1, q_1 \odot k_2, q_1 \odot k_3$. This computation is turned into matrix multiplication of QK^T in the equation 4.6. The computed scores are then divided by 8 (squared root of d_k = 64) and go through a softmax layer to normalize the outputs. The outputs now act as the weights for other words in the sentence. We multiply them with the values matrix and sum up the weighted vectors to get the output values of the self-attention layer.

In RNNs, the way the network retrieve information from other part of the input sequence is iterative. It has to go step by step from 1 direction (or 2 for Bidirectional RNNs) to know

whats on the other end of the sentence. The self-attention mechanism allows the Transformer to quickly aggregate all the information of the input sequence in a single step to generate the appropriate representations. The self-attention layers are also used in the decoders for the same purpose.

Multi-headed Attention

By using self-attention mechanism, the Transformer is able to instantaneously retrieve encoding information from all the words in the sentence. The representation of a word will contain a little bit of all other words, hence, there is a chance that it is dominated by the word itself which is useless in the encoding process. To further extend the ability of the model to focus on different parts, the authors proposed to use multi-headed attention mechanism instead of a single attention function. This helps the model explores and produces the q, k, v vectors in multiple representation subspaces, thus increases the overall understanding of the sentence context.

More specifically, in the paper, the authors use 8 attention heads for each self-attention layer. Each individual attention head consists of its own set of weight matrices w_q, w_k, and w_v for they query, key, and value vectors respectively. Therefore, for each input word embedding, we will receive 8 different representations with respect to 8 attention heads. These attention representations of a word is concatenated and multiply with a weight matrix W^O) to create a combined vector representation for the feed-forward network in the next step.

$$MultiHead(Q, K, V) = Concat(head_1, head_2, ..., head_8)W^O \tag{2.17}$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.18}$$

Positional Encoding

One of the most important factor when retrieving the context of a sentence is the ordering of words or their positioning. However, Transformer neither has recurrence nor convolution because it is constructed with only self-attention layers and feed-forward neural network in all their encoders and decoders. This renders the model useless in terms of capturing ordering of words in the sentence. The authors addresses this problem using a technique called Positional Encoding.

For each word, apart from its input word embedding, the Transformer also includes an encoded vector to provide information on its positioning in the sentence. The intuition behind this technique is that with these values, the model can embed more meaningful information,

especially on positioning and distances between words in the sentence, into the encoding and decoding process. These vectors have the same dimension with the input vectors and follow a pattern. The positioning vector is divided into 2 halves and modeled according to the sin and cos functions of position pos and dimension i as follows:

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.19}$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.20}$$

According to the authors, there are multiple others functions for positioning encoding. However, they argued that with the sin and cos functions, the model is able to adapt sequences that are much longer than ones during training.

Decoder

The process of the decoder is pretty much the same as the encoder in the self-attention and the feed-forward layers. For the Encoder-Decoder Attention layer, it is also implemented with multi-headed attention mechanism. All of these layers takes input from a set of key and value vectors coming from the top sub-encoder. For each word vector, the decoder generate its respective representation vector. The stack of decoders generates a list of vectors and these are turned into words (indexes) by a final block (a linear layer followed by a softmax layer).

2.2.6 Bidirectional Encoder Representations from Transformers

In Natural Language Processing, the benefits of general purpose language representation models are enormous. Since the introduction of pre-trained word embeddings such as GloVe (Global Vectors for Word Representation) [32] by Pennington et al., 2014 and word2vec by Mikolov et al., 2013 [28], the results of multiple tasks have increase substantially by using these pre-trained models instead of training from scratch. Despite their success, GloVe and word2vec are just word based models. The output of these two models of a word is just its numeric vector representation which was combined together from all of its different instances. In other words, regardless of the position and what meaning a word may have, all of these words will have the same exact vector representation when retrieving from GloVe or word2vec. For example, the word "**cell**" from "It is my **cell** phone" and **cell** from "It is my blood **cell**" have different meaning based on their sentence context. However, the

conventional word embedding models will only return exactly one vector representation for these two "**cell**" words. This is the major drawback of using normal word embeddings model as human languages contains lots of words that have multiple meaning depending on their context.

2.2.6.1 Bidirectional Encoder Representation from Transformers

The era of pre-trained neural networks for word embedding generation

Due to the significant drawbacks of plain word-based embedding methods, researchers need a new way to produced contextualized word embeddings for better language understanding. In early 2018, Peters et al. introduced ELMo for this particular task. ELMo utilizes task specific bidirectional LSTMs to look at the whole sentence before generating word embeddings [64]. The LSTMs which pre-trained for language modeling (predicting the next word in the sentence) is used to generate hidden states for the final concatenated output of ELMo. However, LSTMs show weaknesses in capturing long-term dependency in long sequence. Transformer, on the other hand, has the edge over LSTMs in these long-term dependencies.

OpenAI's Transformer utilizes the vanilla Transformer's decoder architecture to create a model that is mainly used for transfer training language models [65]. With 12 decoders stacked together, their Transformer model is heavily trained in language modeling (predicting the next word). We can use this pre-trained decoder stack for multiple downstream tasks such as classification, entailment, similarity, and multiple choices (as suggested in the original paper [65]). Unfortunately, OpenAI only use decoders in their model which eliminates the Transformer strengths of modeling context in both direction. BERT is the Transformer-based language model that can look in both directions because it uses the encoder stack instead of decoder like OpenAI's one.

BERT

Bidirectional Encoder Representation from Transformers (BERT), introduced in late 2018 by Devlin et al. [58], is a pre-trained neural network which produces contextualized word embeddings from an input sentence. This approach of using a pre-trained neural network to produce embeddings has been explored earlier: ULM-FiT by Howard and Ruder [60], Peters et al., 2018 with their ELMo model [64], and OpenAI's Transformer [65]. From the inside, BERT uses Transformer (§ 2.2.5) [55] as their backbone architecture while ELMo uses LSTMs. Moreover, unlike word-based models such as GLoVe and word2vec, BERT

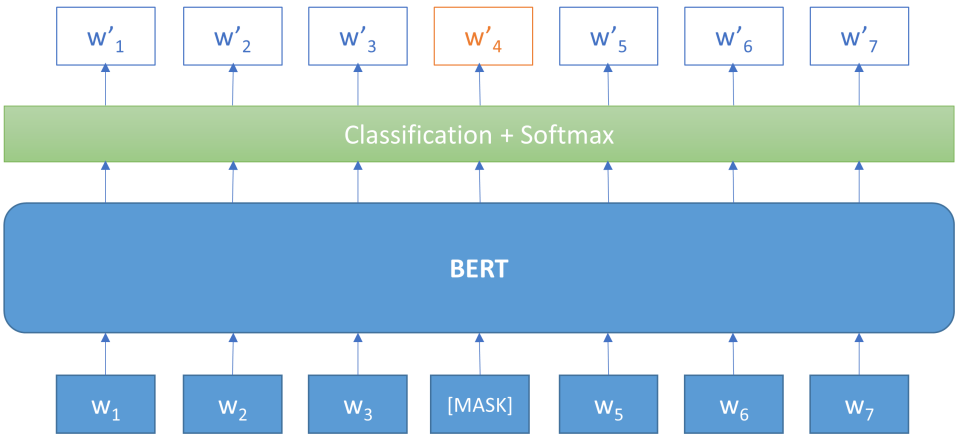


Figure 2.11: Training BERT as a Masked Language Model

takes word ordering (i.e. sentence context) into consideration when generating word embeddings, hence, it is able to generate different word representations for the same word token given their context. As the ambiguous factor when producing word representations by using word-based models has been alleviated, BERT has achieved state-of-the-art results on multiple NLP tasks and opened up the possibilities of using it in other downstream task via transfer learning [58].

BERT - Masked Language Model

In order to get a working pre-trained BERT model, it has to be trained with the right task(s) to unlock the Transformer's full potential. A normal language model uses previous tokens to predict the next unknown token. However, as BERT is implemented with Transformer's encoders, this means that given an input sentence, BERT can uses all tokens in the input to guess the predicting token, including the token itself. This becomes the problem for training BERT as a language model since it will always much or less know the word being predicted.

The authors proposed to use partially masked input sentence to force BERT to guess the words as a task for learning. For each input sequence, BERT masks 15% of the words with $[MASK]$ token. However, the actual BERT implementation uses random words and the word itself sometimes to mitigate the difference between training and later fine-tuning tasks. Through training, BERT has to learn to predict what words are being masked. The learning process is quite simple: (1) the masked input sequence is fed through the encoder stack of

BERT to get the encoder's output vectors, (2) these vectors go through the classification layer which is on top of the stack, (3) the outputs are multiply with the embedding vectors to acquire the vocabulary matrix, (4) a softmax function is applied to the vocabulary matrix to get the probability of every words (see Figure 2.11). Loss is calculated with the prediction of only the masked words with a cross entropy function.

BERT - Next sentence prediction

Masked Language Model training helps BERT get the understanding of correlation between words. In order to make BERT better at modeling sentence-to-sentence relationships, the authors applies next sentence prediction training to BERT. In other words, given a pair of sentences (A and B), BERT has to determine whether B is likely to be the sentence that follows A. In particular, during training, the authors give BERT 50% of the time pairs of sentences where the second sentence is the actual subsequence sentence of the first one from a source document and the other 50% is negative samples.

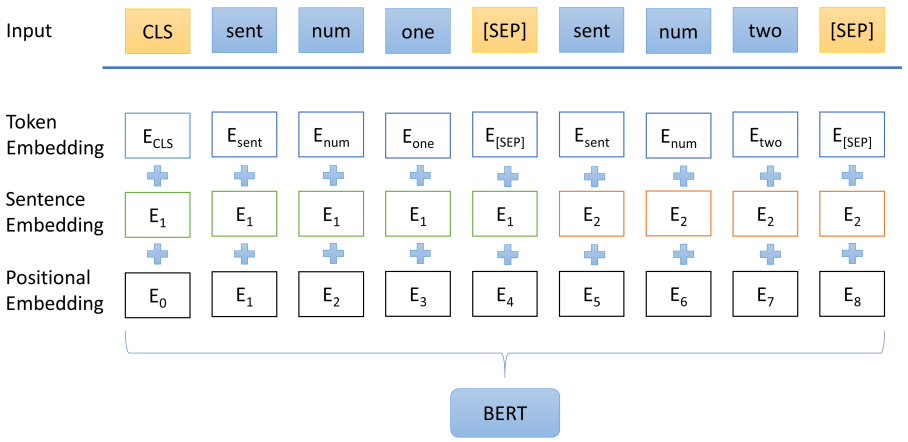


Figure 2.12: Input processing for training NSP in BERT.

The pre-processing input for BERT is illustrated in Figure 2.12. To train BERT for such task, we have to pre-process the input sentences in the following way:

- A classification token $[CLS]$ is added to the beginning of the input sequence that BERT receive. This special token will be used as the classification result of predicting next sentence after going through the network.
- After the last token of each sentence, a special token $[SEP]$ is added as the indication for the end of sentence.

- Sentence embeddings are also added to the token embeddings to indicate which sentence a word belongs to.
- Positional embeddings (explained in § 2.2.5) are also added to the embeddings to enhance the positional knowledge.

After the input sequence is processed, they will be fed through BERT to get the output of the $[CLS]$ token. This output vector is transformed into a (2, 1) vector and goes through a softmax layer to get the classification result for the final prediction. In practice, BERT was trained for both tasks simultaneously. It means that with the input as in Figure 2.12, some tokens are replaced with the $[MASK]$ tokens and then go through the pre-processing with other type of embeddings.

In this thesis, we use BERT in a similar way to extract features and classify sentences in an input document to decide what sentences to be retrieved for a summary. This will be explained more clearly in the **Chapter 4**.

Chapter 3

Related works

Extractive summarization has been under the scope of researchers for a long time. The techniques that had been used over half a decades ago still prove to be effective these days. Sentence scoring and sentence selection remain as the key approaches for extractive summarization systems even in the era of machine learning and deep learning. Sentence scoring gives scores to each sentence in a document and ranks them in term of saliency and semantic importance. Meanwhile, sentence selection examines and determines which sentence the system should pick for the summary based on the sentence scores.

3.1 Classical approaches

Classical approaches utilize sentence and word features to analyze and score sentences in an unsupervised manner. Early works like Luhn et al., 1958 use frequency of high descriptive words [1], Edmundson in 1969 with additional features like position of sentence and cue words to indicate sentences' importance [2], Salton et al. in 1988 and Jones et al. in 1998 incorporated TF-IDF weighting of words [9]. Gong and Liu, 2001, used matrix calculation to represent sentence importance according to several extracted concepts [15]. Others like Mihalcca and Tarau, 2004 [20], Erkan and Radev, 2004 [17], turn document sentences into a connected graph. The graph has edges which contain weights of the similarity between two sentence vertices and the importance score of each sentence is based on its neighbors.

3.2 Machine learning approaches

With the blooming success of learning models, machine learning algorithms were also applied extensively in sentence scoring for summarization. Kupiec et al., 1995 [6], proposed a Naïve Bayes classifier to score sentences using multiple features of words, phrases, sentences, and paragraph. Wong et al., 2008 [25], works on their defined features such as surface, content, event, and relevance features with Support Vector Machine and Naïve Bayes to train supervised and semi-supervised models.

3.3 Deep learning approaches

Deep neural networks enable researchers to further extend the possibilities in extractive summarization systems by proposing multiple novel techniques.

3.3.1 Attention-based Summarization System

Attention-based Summarization (ABS) system [39] is the first system to utilize sequence-to-sequence with attention support. The model was introduced in 2015 by Rush et al. along with the first large-scale dataset for text summarization, English Gigaword [39]. ABS consists of a sequence-to-sequence model for encoding and generating summary as well as an attention module that support the encoding process. With the attention-based encoder, the model is able to attend the source document for an appropriate word during each time step of the decoding process. This allows the model to generate novel words according to the context of the source sentence. At that time, the ABS model achieved state-of-the-art results for automatic text summarization on two datasets (English Gigaword [39] and DUC-2004 [23]).

In 2016, another large-scale dataset for summarization named CNN/DailyMail was released by Nallapati et al. [49]. While the English Gigaword dataset was mostly for headline generation task, the CNN/DailyMail dataset aims at paragraph-length summarization from news articles. The dataset set a new challenge for automatic summarization as it contains much longer source documents and requires the model to generate summaries from 3 to 4 sentence on average. Furthermore, the authors also proposed several improvements on baseline sequence-to-sequence model such as: feature-rich encoder, switching generator-pointer, and hierachical attention. The research reached state-of-the-art results on DUC and English Gigaword dataset as well as established a baseline for the newly introduced dataset.

3.3.2 Pointer-Generator networks

After the release of CNN/DailyMail dataset, researchers have developed more novel techniques in producing more fluent and relevant summaries. By taking advantage of pointer network [42], See et al. proposed a new method that enables model to copy words from the article text via pointing and generate words from a given vocabulary. The model chooses between copying a word from source text or generating a new word according to a generation probability. This helps the model deal with out-of-vocabulary (OOV) words and generate more readable summary.

Apart from OOV problem, sequence-to-sequence models also suffer from repeating issue [50]. See et al. also proposed to use coverage mechanism [50] in summarization to notice the model of what has been summarized, hence, reduce the repetitions. This method produces summaries with less inaccuracies and repetitions, which outperforms state-of-the-art abstractive model by a significant amount in the challenging CNN/DailyMail dataset.

3.3.3 Deep reinforced models

Attention-based encoder-decoder system has been shown to give high performance results on text summarization task [49, 39]. Moreover, this type of system also generating summaries with a lot of repeating words. In 2017, Paulus et al. introduced a new attention mechanism called intra-attention [52] which boosts the readability of the output summaries. They also proposed a new combined training method that incorporates reinforced learning to further enhance the evaluation performance.

The intra-attention mechanism consists of two different attention methods: intra-temporal attention for the encoder and intra-decoder attention for the decoder. These two attentions combined will respectively make sure the encoder utilizes the whole input sequence and the decoder to receive more information, hence reduce the repetition of words. Furthermore, the authors use a mixed training objective function that make the model produce more human-like summaries by using an additional reward function.

By using these two approaches, the authors achieved improved result on the CNN/DailyMail dataset over previous state-of-the-art extractive models [52]. Additionally, the authors also proved that their approach creates more readable summaries through human evaluation. This research created a foundation for later automatic text summarization systems to incorporate different attention mechanisms as well as more novel mixed objective functions.

3.3.4 Other significant works in Extractive summarization

In 2015, Cao et al., proposed to use context-independent features of summaries called prior to decide whether a sentence should be included in the summary [35]. The authors applied convolutional neural networks to learn how to reflect these document-independent features from variable length phrases. Later, researches from Cheng and Lapata, 2016 [45], Nallapati et al., 2017 [49] defined extractive summarization as a sentence labeling task where the system gives each sentence a label of whether they should be in the summary.

Specifically, in their research, Cheng and Lapata encode each sentence in a document using an encoder-decoder network which composes of RNNs to retrieve its context [45]. The system then uses the encoded features to classify sentences into two classes of to be extracted or not. The decoder, which uses the attention mechanism, acts as a classifier while the encoder turns sentences into context vectors. Nallapati et al. [48] also treats the problem as sentence labeling with the similar approach as Cheng and Lapata. However, they only implemented a model without a decoder. The system encodes sentences using RNNs and then label each sentences into two classes in the same manner. The authors proposed to use additional features such as relative positions and sentence absolute to further increase the learning capability of the network.

These methods of sentence regression mostly based on multiple aspects which have no semantic property of the sentences. Ren et al., 2017, proposed Contextual Relation-based Summarization neural model which leverages more contextual features from the input document to generate better extractive summaries [53]. They retrieves the contextual information at word-level by using attentive convolutional neural network. Then, an attentive RNN is used to gather contextual representation of the document at sentence level. These multiple levels of contextual information are learnt to give similarity scores between sentences and generate appropriate summaries.

Early 2018, Narayan et al. extended the task of extractive summarization to reinforce learning by using their novel training technique which optimizes the ROUGE score directly [63]. They demonstrated that for extractive summarization, cross-entropy loss alone is not the perfect solution for this type of task. They argued that training summarization system with that can yield unnecessary information and proposed to use a new reinforcement learning objective which optimizes the ROUGE metric. With the new combined learning objective of cross-entropy loss and ROUGE score policy, a sentence is given a high score for selection when it appears in summaries with high ROUGE score.

3.3.5 Common frameworks

Generally, most of existing extractive summarization systems can be abstracted into the following processes: sentence encoder, document encoder and decoder. Here we mention some of the works that apply different methods in each of the categories.

3.3.5.1 Sentence encoder

Sentence encoders have the role of producing meaningful representations for each respective sentence. These vectors can either be used for direct sentence scoring and selection or become the input for the next document encoding step for a global approach. Most of recent approaches in summarization utilize RNNs and CNNs to encode sentences to a fixed vector.

For RNNs, or specifically bidirectional RNNs, the final sentence vector is the result of the concatenation between the last outputs of both the forward and backward pass. GRU by Cho et al., 2014 [30] and LSTM orginally proposed by Hochreiter et al., 1997 [7] are the main choices of RNN units of the sentence encoders. Recent works such as Cheng and Lapata, 2016 [45], Nallapati et al., 2016 [49], and also Zhou et al., 2018 [67] works with RNNs for sentence encoders and yield impressive results.

On the other hand, CNN encoders are also used in several works for sentence representation extraction. Notable works are Cheng and Lapata, 2016, [45] and Zhong et al., 2019 [76]. In CNN encoders, multiple convolutional filters are applied over word embeddings to create vectors for each sentence. All outputs of those filters go through a max pooling layers to reduce the size and then are concatenated for the final output vector.

Recently, Liu proposed a new approach in using BERT for sentence encoding [69] in extractive summarization. BERT learns and produce embeddings through its Transformer encoders. The author also incorporates multiple features such as word embeddings, positional encoding, and also sentence interval embedding. With the help of contextualized word embeddings, the system was able to achieve state of the art performance on the extractive summarization task.

3.3.5.2 Document encoder

In order to get the whole context of the input document, the summarization system has to not only understand each sentence's meaning, but also figure out how sentences correlate with each other. A document encoder has the main objective of generating the contextual rep-

resentations for every sentence in the document. At document level, RNNs and Transformer are usually used as the decoder layer.

Cheng and Lapata, 2016 [45], Zhou et al., 2018 [67], Narayan et al., 2018 [63], many others implemented a document encoder in their systems. Some works like Liu [69], 2019 and Zhong et al. [76], 2019 tested both LSTM recurrent decoder as well as Transformer one. These two approaches yield similar performance in terms of ROUGE evaluation scores on the CNN/DailyMail dataset.

3.3.5.3 Decoder

Decoders in general can be specified into two categories: auto-regressive decoder and non auto-regressive decoder.

Auto-regressive decoder

An auto-regressive decoder which usually represented by RNNs is a decoder that generate the next prediction based on the result of the previous steps. Pointer Network orginally utilized by See et al., 2017 [54] is used extensively for a auto-regressive decoder in summarization systems. Notable works are Chen and Bansal, 2018 [57] and Jadhav and Rajan, 2018 [61] as they produce impressive performance on the extractive summarization task. Moreover, the joint sentence scoring and selecting module based on RNNs from Zhou et al., 2018 [67] achieves state-of-the-art results on the CNN/DailyMail dataset.

Non-regressive decoder

Meanwhile, a non auto-regressive decoder predicts the next output without the need of the last token. For non auto-regressive decoders, their jobs are similar to the task of sentence labelling. Given a list of sentences, these decoders produce outputs in $0, 1$ to decide which sentence should be selected. A simple linear classifier or Transformer decoder from Liu [69] or the SeqLab module from Zhong et al. [76] are some of the examples for this particular decoder type.

3.4 Neusum - Neural Document Summarization by Jointly Learning to Score and Select Sentences [67]

Neusum is an extractive summarization system proposed by Zhou et al. in 2018. It incorporates the help of ROUGE evaluation metrics in scoring and selecting sentences for extractive summarization [67]. With the help of the jointly learning module, the model was able to achieve state-of-the-art results on the CNN/DailyMail dataset at the time of the paper.

The authors introduce an end-to-end summarization system that utilizes the relative relationship between sentences in terms of their ROGUE scores. The model scores each sentence it selects by both the sentence's contextual information as well as its relative ROUGE score with respect to the currently generating summary. In other words, a sentence is selected if it has the highest gain in ROUGE score over the picked sentences.

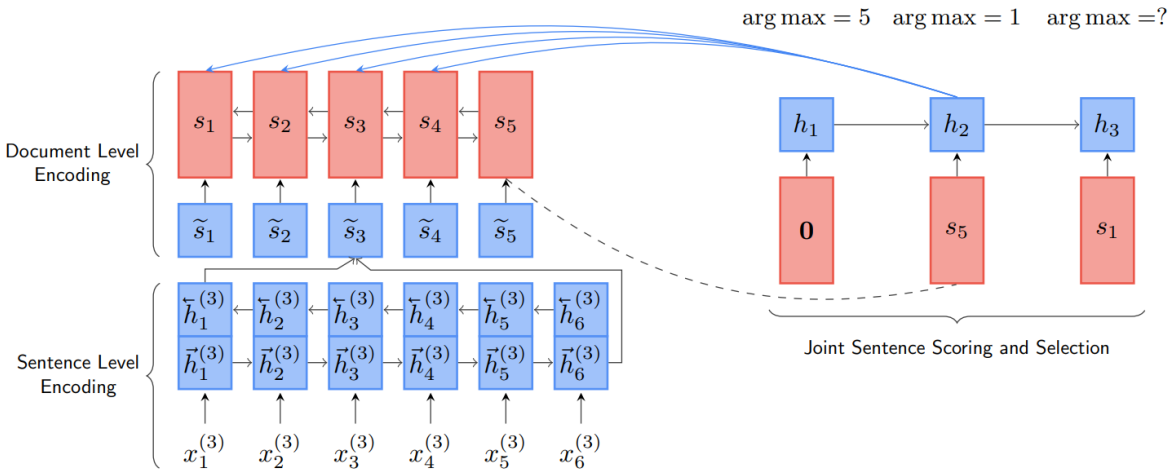


Figure 3.1: Original architecture of Neusum. Image of Zhou et al., 2018 [67]

.

In particular, from the list of word embeddings, a bidirectional GRU sentence encoder creates sentence representation vectors for each sentence in the input document. These sentence vectors now become the input of the next document encoding phase. Another bidirectional GRU is used for document level encoding. These steps create a hierarchical method for encoding information from text, which may yield better information output.

For the decoding part, the authors proposed a joint sentence scoring and selection module to take advantage of auto-regressive decoder in predicting sentences. First, a RNN is used as a baseline for the prediction steps. A pointer network is implemented as an attention mechanism to consider current possible candidates while the RNN looks at the previous

selection.

Another important idea of their proposed method is the usage of relative ROUGE gains between current sentence and selected sentences during training. For each sentence being considered for selection, the system calculates the amount of ROUGE scores that can be increased for choosing that candidate with respect to the generated summary. This targets evaluation scores directly which benefits not only in quantitative but also qualitative results because ROUGE-2 and ROUGE-L partially help choosing sentences that are more closely related to ground truth sentences.

However, the drawback of this system is the use of conventional biRNN to retrieve sentence vectors. We believe that if we replace the current sentence encoder with a more powerful contextualized encoder, e.g. BERT [58] or ELMo [64], the system can achieve even higher results. Therefore, we decide to make some changes to the network and apply BERT encoder into the model for our experiment. Moreover, we plan to keep the joint sentence scoring and selection module because of its promising aspects on the summarization task.

3.5 BERTSUM - Fine-tune BERT for Extractive Summarization [69]

With the introduction and the undeniable success of incorporating contextualized embedding of words into multiple tasks of Natural Language Processing, BERT (proposed by Devlin et al., 2018 [58]) has also showed its ability for extending into downstream tasks such as text summarization. BERTSUM was proposed by Yang Liu in 2019 with the purpose of applying BERT encoder into downstream summarization task. The system achieved a significant improvement (1.65 score gain in ROUGE-L) on the extractive summarization state-of-the-art leaderboard by fine-tuning BERT for summarization [69]. Liu proposed a method to use BERT for sentence representation extraction and use it for sentence scoring and selecting from the input documents. Apart from contextualized word embeddings given by BERT, the author also introduced additional information such as positional embeddings for words and interval segment embeddings for distinguishing sentences. These features are trained together to create sentence representation vectors for the summarization layer. It is shown that even with a simple linear classifier, the system can produce state-of-the-art result and beat the previous system Neusum by Zhou et al., 2018 [67] by a large margin.

One of the important contribution of this approach is the method of turning article into data that can be used with BERT, especially multi-sentence handling. Because the origi-

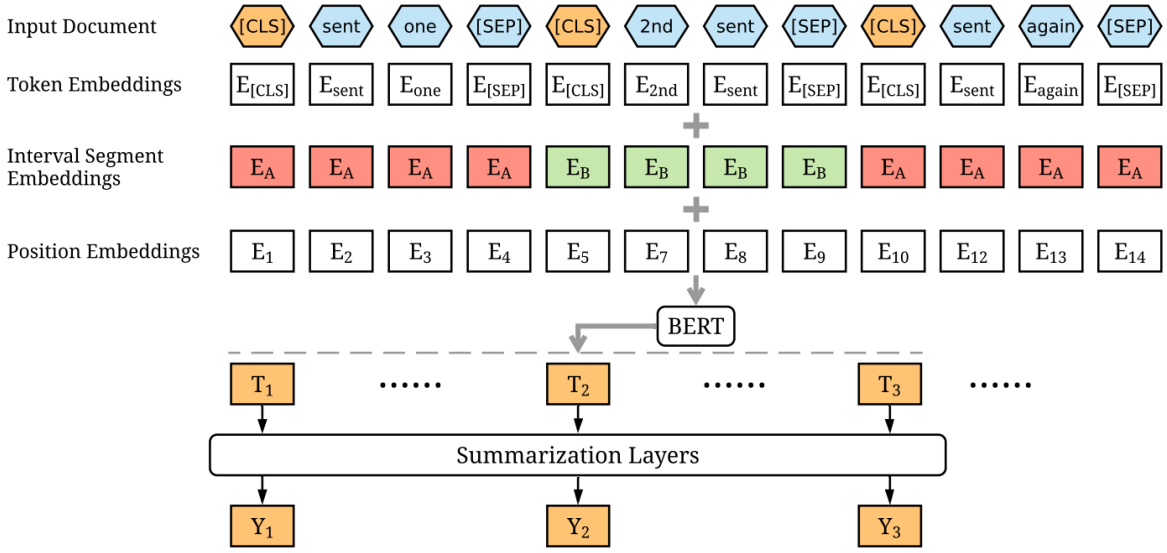


Figure 3.2: Original architecture of BERTSUM. Image of Yang Liu, 2019 [69]

.

nal BERT was pretrained for specific tasks (one of them is Next Sentence Prediction, see more in Section 2.2.6.1), it can only handle two sentences at a time. By introducing a new type of embeddings, i.e. Interval segment embeddings, the author was able to make BERT distinguish multiple sentences in the input articles.

All token embeddings from every input token accompanied by their positional embeddings and interval segment embeddings are fed through BERT to acquire a list of sentence representation vectors. These vectors are at the position of the corresponding $[CLS]$ vectors in the input (details are explained in Section 2.2.6.1). Now the system uses this list of sentence vectors to be the input of the summarization layer where its main objective is to select the most important sentences.

The author tested three different types of architecture for this summarization layer: linear classifier, Transformer, and LSTM. Even with the linear classifier, BERTSUM already surpass previous state-of-the-art results of Neusum by Zhou et al. [67] by a large margin. When predicting with Transformer, the model was able to achieve state-of-the-art on the extractive summarization task.

However, the system was only able to support input articles with the maximum of 512 tokens including special ones. The problem comes from the capability of BERT architecture. This means that for longer input articles, information is going to be truncated, hence the loss of important sentences.

Another part of the model that we think can be improve is the summarization layer.

Currently, the author uses a non-auto regressive decoder to predict sentences which means predictions are independent. In other words, the decision of choosing a sentence is not based on what have been chosen and what have not. We think that if we integrate BERT with an auto-regressive decoder, it can further improve the ability to generate better summaries.

Chapter 4

Method

In this chapter, we want to test whether auto-regressive decoder on the same BERT encoders from [69] can outperform non-regressive decoder (the original decoder of that paper [69]). Furthermore, we propose our model which has modifications of BERT encoders to take use of all input articles of CNN/Daily Mail dataset.

4.1 Problem statement

Let D denotes a document containing several sentences $d = [sent_1, sent_2, ..., sent_m]$, where $sent_i$ is a ith sentence in D and m is the number of sentences in d. Extractive summarization can be defined as the selection of important sentences which are included in the summary.

4.2 Auto-regressive decoder on BERT Encoders

4.2.1 BERT Encoders

4.2.1.1 Sentence representations with BERT

Since BERT originally produces word embedding for each token, we have to modify BERT to get sentence embeddings. As in § 2.2.6.1, we explained how the vanilla BERT was pre-trained as a masked-language model and also Next Sentence Prediction. BERT assigns a special $[CLS]$ token at the beginning of the input sentence (or 2 sentences in the case of Next Sentence Prediction) to hold the representation vector of that sentence. However, the vanilla

BERT can only take up to a maximum of 2 sentences which is unpractical for summarization tasks as articles usually consist of multiple sentences.

We adopt the approach of Liu, 2019 [69] in using BERT to create contextualized sentence embeddings. For every sentence in an input article, we give it a representation token $[CLS]$ at the beginning and a separator token $[SEP]$ that marks when the sentence ends. A maximum of 512 tokens is fed through BERT to get their respective contextualized token embeddings. The sentence embeddings in the vanilla BERT are extended to interval segment embeddings which deal with distinguishing extra sentences of articles. Specifically, sentences at odd positions will have interval segment embedding of E_A and sentences at even positions will have E_B as their embeddings. The vector of token $[CLS]$ before each sentence is chosen as its sentence representation, as illustrated in Figure 4.1. Such sentences vectors are denoted as s_i for each sentence $sent_i$.

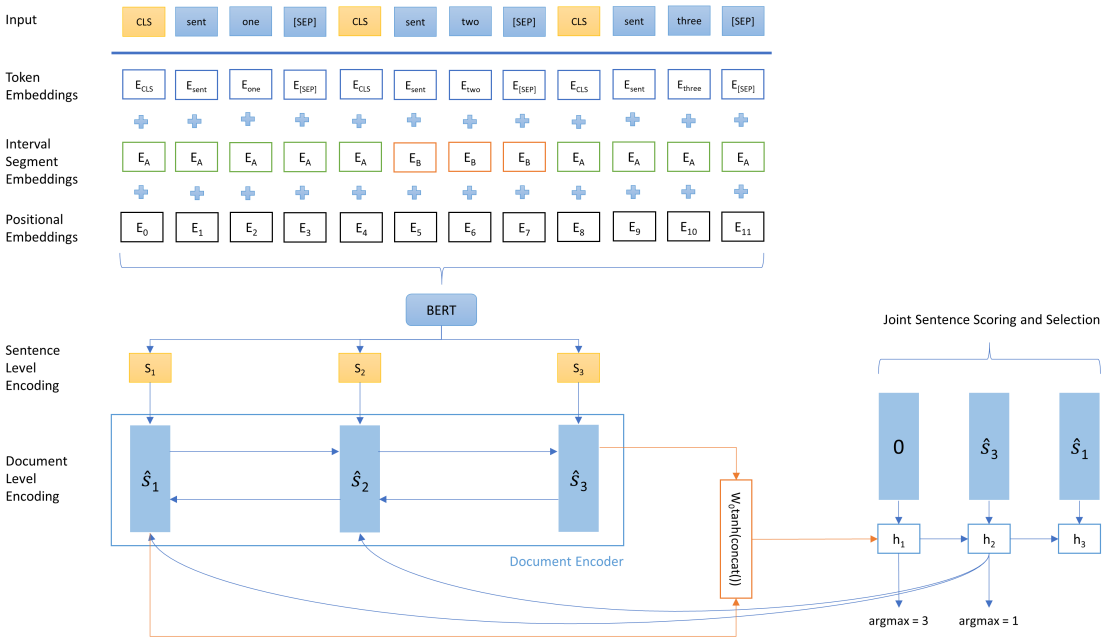


Figure 4.1: Bertjoin: BERT encoder and LSTM + Pointer network decoder

4.2.1.2 Document-level representation

In the previous step, we encode sentences of input article into sentence representation vectors using BERT as the encoder. With the list of sentences vectors, we want to further enhance the ability to retrieve contextual information from the document by pushing them through another LSTM network for document level encoding. We only implement a single

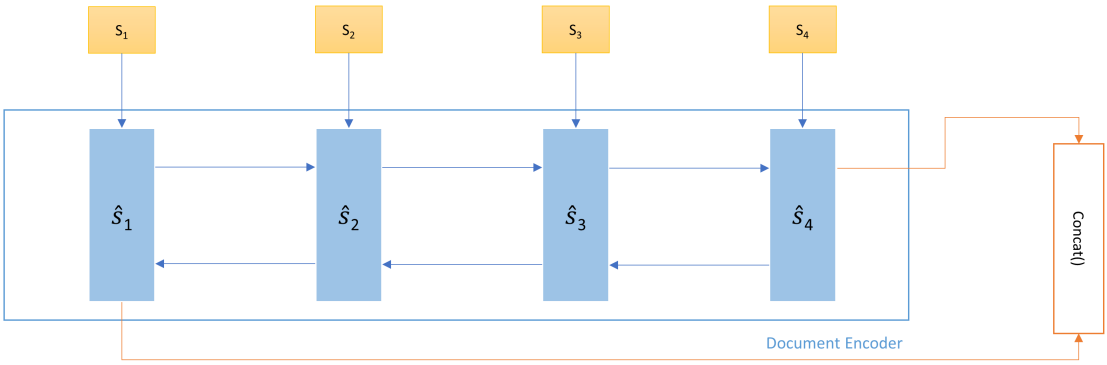


Figure 4.2: The bidirectional LSTM for document encoding from list of sentence vectors from BERT.

layer bidirectional LSTM for the document encoder because we argue that we do not need create a deeper network for document embedding as BERT has already embedded some context information into these sentence vectors.

After obtaining sentence vector s_i which is the output of BERT for the $[CLS]$ token with respect to sentence $sent_i$, we feed them into a vanilla biLSTM with forward hidden states $\vec{\hat{s}_i}$ and backward hidden states tfa$\overleftarrow{\hat{s}_i}$. The initial vectors of the biLSTM are set to zeroes, i.e $\vec{\hat{s}_0} =$ tfa$\overleftarrow{\hat{s}_m} = \vec{0}$. By feeding sentence vector s_i to this biLSTM, we can get the global-dependent sentence vectors $\hat{s}_i$ which is the concatenation of $\vec{\hat{s}_i}$ and tfa$\overleftarrow{\hat{s}_i}$. More specifically:

$$[\hat{s_1}, ..., \hat{s_m}] = BiLSTM([s_1, ..., s_m]) \tag{4.1}$$

where

$$\hat{s}_i = concat(\vec{\hat{s}_i}, \text{tfa}\overleftarrow{\hat{s}_i})$$

Figure 4.2 illustrates the biLSTM document encoder that is implemented in the system. The list of vectors [S_1, S_2, S_3, S_4] is the sentence representation vectors we get from BERT in the last step. After all sentences are fed into the network, we take the concatenation of the last hidden state in the forward pass, i.e. $\vec{\hat{s}_4}$, and the last hidden state in the backward pass, i.e. tfa$\overleftarrow{\hat{s}_1}$, to be the document representation vector for prediction.

4.2.2 Auto-regressive decoder

Using a LSTM for joint sentence scoring and selecting

Now that we have an encoder that encodes the document in a hierarchical manner from token to sentence level to document level. We move on to the decoding stage of the model where it learn to score sentences based on their importantness and select the appropriate ones for the final summary.

Zhou et al., 2018, proposed an approach of jointly scoring and selecting sentences using bidirectional GRU [67]. Their method of jointly scoring and selecting sentences shows a lot of potential improvements and we decide to adapt this approach. In particular, we use a bidirectional LSTM for the decoder with additional multi-head attention mechanism for sentence scoring step. The LSTM helps the system in considering previous selected sentences as input for the next prediction while the attention heads enable the model to take into consideration all other unselected sentences from the input article.

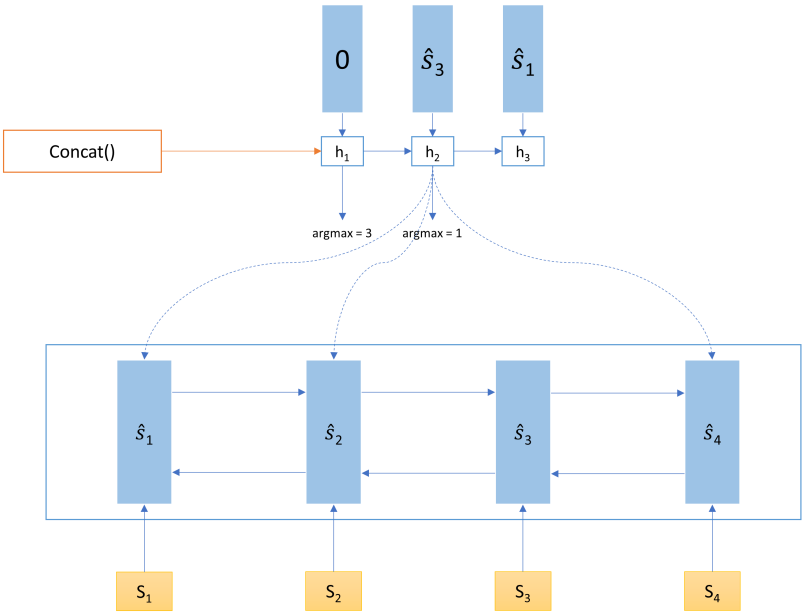


Figure 4.3: The attention module aids the decoder in attending to all remaining sentences in the input articles.

In the first decoding time step, the attention module attends to all sentences to compute the probability for the most suitable sentences with respect to the initial hidden vector from the document encoder. The best sentence chosen for time step t_1, for example $sent_3$, will

have its sentence vector be the input for the next time step. Now that we have the hidden vector and the sentence vector from the previous prediction, we proceed to calculate the next prediction. In the time step t_2, the attention module now looks at all the sentence but the selected sentence in the last step, i.e. $sent_3$ as illustrated in Figure 4.3. This prevents the network from re-picking sentences that have high saliency.

We also follow their steps of creating additional training data for the decoder module. Apart from using oracle sentence IDs as the ground truth labels for the sentence selection part, we also use the relative gains of ROUGE evaluation scores over previously selected sentences as a guideline for the next prediction. By utilizing ROUGE scores during training, the system is able to learn how to select sentence to directly benefit ROUGE score as we are comparing our summaries with abstractive ground truth in the evaluation step later.

The auto-regressive aspect of the decoder means that for every decoding time step, we take into consideration two information: what sentences have been selected, what is the relative relationship between the selected and the remaining sentences. At time step t, we use the selected sentence vector s_{t-1} and the hidden state h_{t-1} of the RNN at the previous time step $t-1$ to be the next input of the LSTM:

$$h_t = LSTM(\hat{s}_{t-1}, h_{t-1}) \tag{4.2}$$

We use a pointer network that captures the hidden state of current time step t h_t and all the document-level sentence vectors that have not been selected in the final summary to predict the attention probability and we can pick the argmax of this distribution as chosen sentence. More specifically,

$$score_{t_i} = score\_function(h_t, \hat{s}_i) \tag{4.3}$$

$$normalized\_scores_t = softmax(scores_{t_i}) \tag{4.4}$$

We use the score function as in [55], as follows:

$$Attention(Q, K) = softmax(\frac{QK^T}{\sqrt{d_k}}) \tag{4.5}$$

where Q is the hidden state from the LSTM at time step t h_t, K is the document-level

sentence vectors $\hat{s}_i$. Noted that we do not have V matrix as we only care about the attention probability in this problem.

The predicted sentence $pred_i$ is the sentence which has the maximum probability in the distribution from Equation 4.4:

$$pred_i = \arg\max_{\hat{s}_i}(normalized\_scores_t) \tag{4.6}$$

4.3 Adaptation of BERT encoders to take use of the whole input articles

4.3.1 Utilizing multiple BERTs to extract sentence representations

Our idea is dividing document D into many segments and then using BERT encoder from section 4.2.1.1 to encode information of each part. Finally, we aggregate all sentence vectors, feed them into a document-level encoder and score sentences for summary selection.

Document splitting

We divide our document D into several segments such that each segment have maximum 512 word tokens and no sentence is splitted into two segments. In particular, if the last sentence in a segment exceeds the limit of 512 word tokens, we move it to the next one. By splitting this way, we ensure the linguistic structure of every sentence in the document D. For example, if D has 4 sentences $[sent_1, sent_2, sent_3, sent_4]$ and the length of word tokens in each one is $[250, 250, 250, 250]$, D is splitted into two segments $([sent_1, sent_2], [sent_3, sent_4])$ instead of $([sent_1, sent_2, sent_3[:12], [sent_3[12:], sent_4])$.

Encoding sentences with BERT

As illustrated in Figure 4.4, we use a single BERT encode from section 4.2.1.1 to take turn encoding each segment of document D into sentence vectors for each sentence $sent_i$. After that, we can have the list of all sentence vectors: $[s_1, s_2, ..., s_m]$ where m is the length of sentence in document D. Following the example of the previous sub-section on Document Splitting, initially we feed document D with two segments into BERT encoder to get sentence vectors $([s_1, s_2], [s_3, s_4])$ and $[s_1, s_2, s_3, s_4]$ after that.

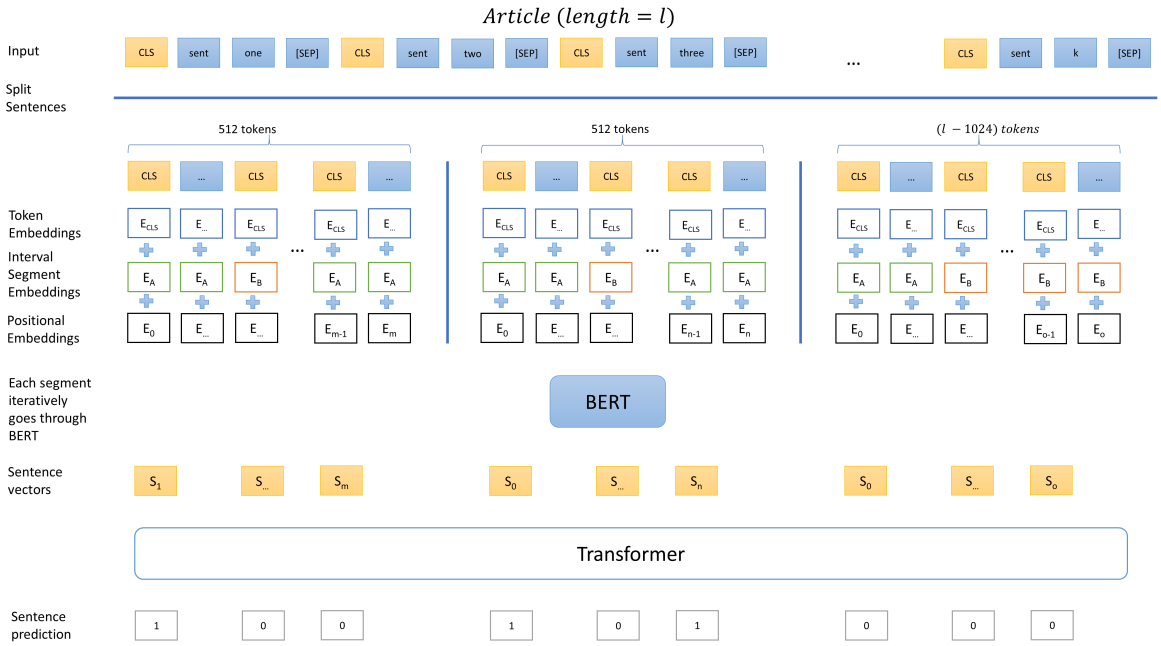


Figure 4.4: Our proposed approach for utilizing all sentences from an input article with BERT encoder.

4.3.2 Sentence scoring and selecting

After obtaining sentence vectors from BERT encoder, we build Transformer summarization-specific layer on top of BERT output in order to capture document-level feature for extractive summarization. Each sentence vector s_i is fed into these layers and assigned a score $prob_{s_i}$. The loss of the model is binary classification entropy between all $prob_{s_i}$ and the golden summary label Y_i (0 or 1) where $i \in [1..m]$.

Transformer layer

We adopt Transformer layer from Vaswani et al., 2017 [55], as the encoder for document-level sentence vectors h_i. This step encodes the sentence vectors s_i from BERT to extract document context which is then used to score and predict the likelihood $prob_{s_i}$ of each sentence $sent_i$ of document D being in the final summary.

The reason for our decision is that we want to take the full advantage of the Transformer network because of their faster training time and its self-attention mechanism. Training time on a LSTM is significant longer than on a Transformer. Moreover, with the self-attention mechanism being embedded in the network itself which is proved to be greatly effective in capturing semantics dependencies [55]. Therefore, we think that it could further pushing the

performance of our method.

This transformer layer has several sub-layers in which each one contains a feed-forward layer and a self-attention layer just like the original Transformer from Vaswani et al. [55], specifically:

$$\hat{h}^l = LayerNorm(h^{l-1} + MultiHeadAtt(h^{l-1}) \tag{4.7}$$

$$h^l = LayerNorm(\hat{h}^l + FeedForward(\hat{h}^l) \tag{4.8}$$

where h^0 is $concat([PosEmb(S), S]$: the concatenation of sentence vectors $S = \{s_i\}$ and their positional embeddings $PosEmb(S)$ which indicates their sentence positioning; $FeedForward$ is the 1-layer feed forward network; $MultiHeadAtt$ is the multi-head attention operation; $LayerNorm$ is the layer normalization operation [44].

The Transformer receives a list of sentence representation vectors from BERT and forwards through its sub-encoders. Each output vector h_i of the Transformer is the corresponding to the final sentence vector for s_i of the input article. We use these vectors to predict the probability of a sentence being in the final summary by using a sigmoid function σ as follows:

$$prob_{s_i} = \sigma(Wh_i + b) \tag{4.9}$$

An illustration of the whole process of the proposed approach is available in Figure 4.4. In particular, in Figure 4.4, we initially have an article with length l that is supposedly longer than 512 tokens. We divide the article into three segments using our soft splitting method that we explained earlier in this section. The token embeddings along side with other basic additional BERT information (interval segment embedding and positional embeddings) are packed together for each sentence segments. Each package then becomes the input for BERT to extract a number of sentence vectors with respect to the number of sentences in each segment. The sentence scoring module uses these sentence vectors for predicting in the next step.

Chapter 5

Experimental setups and results

5.1 Overview

We perform two different experiments on CNN/Dailymail dataset to test our hypotheses.

First, we propose auto-regressive approach in extractive summarization by combining the decoder: LSTM and pointer network with the BERT encoder. In this experiment, we hope to test, if we change decoding strategy from non-regressive to auto-regressive approach, whether our proposed model 1)improves the ROUGE score and 2) is consistent with findings from recent work [76] claiming that auto-regressive model outperforms non-regressive one.

Second, by analyzing results from the first experiment, we propose new adaptation of BERT in non-regressive approach and the full article is utilized instead of its truncated version. By learning more information on articles, we hope our model can make the best use of all input information and therefore can predict important sentences with higher ROUGE score.

5.1.1 Dataset

5.1.1.1 Original dataset

Initially, the CNN/DailyMail dataset was introduced in 2015 by Hermann et al. [36] as a question answering dataset for their research on reading comprehension of machines. The dataset contains news articles from CNN[1] and Daily Mail[2]. There are a total of 92,579 documents from CNN and 219,506 documents from DailyMail. Moreover, the dataset also

[1] www.cnn.com
[2] www.dailymail.co.uk

provides corresponding human generated abstractive summaries of each article [36]. These abstractive summaries are supplementary parts that are given by news providers through bullet points in the articles (Figure 5.1).

Figure 5.1: Example of bullet point summaries taken directly from DailyMail website. Initially, each summary bullet was used as question and the corresponding story was the training passage for the system to find a hidden entity in question answering task.

5.1.1.2 Modified versions for text summarization tasks

Abstractive version

In this thesis, we mainly focus on automatic text summarization on the publicly available CNN/DailyMail text summarization dataset [49]. In 2016, Nallapati et al. repurposed the CNN/DailyMail dataset by Hermann et al. for abstractive document summarization task. From the script released by Hermann et al., the authors modified it and retrieved the summary bullets in the original order to construct a multi-sentence abstractive summary for each story [49]. Since these are human generated summaries, they are gold standard of the abstractive summary task.

According to the authors, their corpus consists of 311,672 summary-passage pairs: 286,817 training pairs, 13,368 validation pairs, and 11,487 test pairs [49]. Each document contains an average of 29.74 sentences with 766 words and the summaries are about 53 words long spanning over 3.72 sentences.

Furthermore, the dataset has two versions: anonymized and non-anonymized versions. For the anonymized version, all entity names have been replaced with specific ids for each particular documents. For example, the original sentence from DailyMail *"Denis Suarez scores late winner for Sevilla as they take control of quarter-final tie with Zenit."* will be replaced with *"@entity0 scores late winner for @entity1 as they take control of quarter-*

Non-anonymized version	Anonymized version
Denis Suarez scores late winner for Sevilla as they take control of quarter-final tie with Zenit.	@entity0 scores late winner for @entity1 as they take control of quarter-final tie with @entity2.

Figure 5.2: Example on anonymized and non-anonymized versions of the CNN/DailyMail dataset.

final tie with @entity2." as all named entities are hidden. For the non-anonymized version, all named entities are kept the same as the original new articles (Figure 5.2).

Extractive version

Because the current dataset by [49] only had abstractive summaries as its gold standard, it could only be used for abstractive summarization task. In *Neural Summarization by Extracting Sentences and Words* by Cheng and Lapata (2016), the authors proposed a method to create an extractive summarization dataset from existing CNN/DailyMail corpus [45]. They created two large-scale datasets for sentence extraction and word extraction. They followed the method by Hermann et al. in [36] to retrieved news articles as well as their corresponding abstractive summaries from DailyMail. In order to generate data for extractive task, the authors created a system that marks sentences in the source document that should be in the summary. The system determines each sentence based on its semantic correspondence with the gold standard abstractive summary.

In total, there are approximately 200,000 documents that are labeled using this approach. However, this corpus only has articles from DailyMail. Specifically, for the DailyMail corpus, there are 196,557 training documents, 12,147 validation documents, and 10,396 test documents. For the joint CNN/DailyMail dataset, there are 287,227 training documents, 13,368 validation documents, and 11,490 test documents [48].

5.1.1.3 Statistics of the dataset

The dataset we used in this thesis is based on the abstractive non-annonymized version that follows preprocessing steps by Hermann et al. in [36]. Originally, the dataset was tokenized using Stanford CoreNLP[3] tokenizer and lowercased all letters. For the training set,

[3]Open source tool: https://stanfordnlp.github.io/CoreNLP/

Table 5.1: Statistics of the CNN/DailyMail dataset.

Subset	Attribute	Train	Validation	Test
	Articles	287,227	13,368	11,490
Source	Average tokens	791.4	769.3	778.3
	Max tokens	2,882	2,134	2,380
	Min tokens	10	45	59
	Average sentences	39.0	32.9	33.5
	Articles with <= 400 tokens	14.00%	17.08%	16.64%
	Articles with <= 600 tokens	36.17%	39.80%	39.34%
	Articles with <= 800 tokens	58.18%	60.52%	59.85%
Target	Average tokens	62.7	69.7	66.1
	Max tokens	2,352	1,752	738
	Min tokens	6	12	11
	Average sentences	4.6	4.9	4.7
	Articles with <= 100 tokens	93.08%	88.44%	91.53%

the average number of tokens is 791.4 (maximum of 2,882 and minimum of 10) for source articles and 62.7 tokens (maximum of 2,352 and minimum of 6) for target summaries. There are 114 pairs that have no source article for the summaries so we excluded them from the dataset. For the validation set, source articles have 769.3 tokens on average (range from 45 to 2,134 tokens) while summaries have an average of 69.7 tokens (12 - 1,752 tokens). The test set has an average of 778.3 tokens (59 - 2,380 tokens) for articles and 66.1 tokens (11 - 738 tokens) for summaries.

We measure the percentage of articles that have the number of tokens in a specific range. There are only around 15.09% of articles with 400 tokens or less across all three sets. These numbers for articles with less than 600 and 800 tokens are 38.44% and 59.52% respectively. For summaries, there are 91.01% summaries with 100 words or less on average. All three set are quite similar in terms of this measurement.

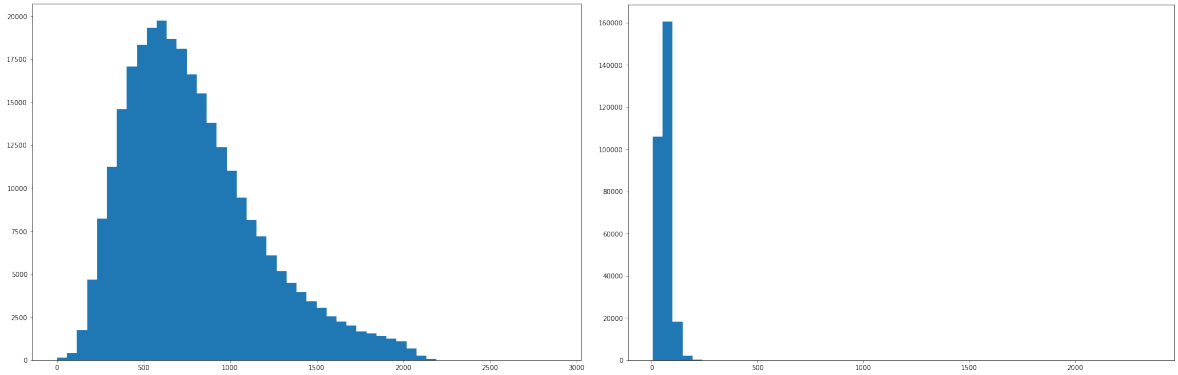
See et al. found that for the CNN/DailyMail dataset, using the first 400 words of the articles to train the model produces much higher results than using 800 tokens [54]. They argued that the reason behind this is the fact that journalists often write news articles with the most important information placed at the start. With the dataset trimmed down to 400 tokens per article, they achieved a comparable result with the state-of-the-art extractive model in 2017. This suggests that we can reduce training time and memory by using just around 400-500 words per articles without loosing performance of the model.

There are a few summaries that have a large amount of words (e.g. 2,352, 1,752, ... words). We check these summaries and find that only the first 4-5 highlights are actual

summaries from the site. All later bullet points are just the article's bullets in the passage that the crawler had mistaken. Therefore, according to the statistics, using a limit of 100 words in the target summaries is a good practice since it's already covered over 90% of the dataset. Other longer summaries will be truncated to 100 tokens without losing much information since there should be only 4-5 highlights per article.

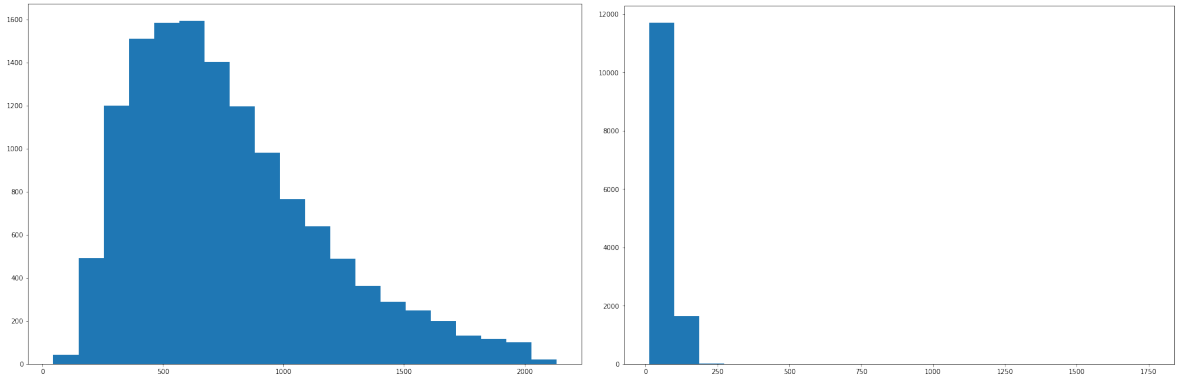
Distribution of token numbers in CNN/Daily Mail

We plot the length histogram of articles and summaries in the dataset to see the distribution of these values across the training, validation, and testing sets. Overall, three sets are quite similar in terms of distribution of articles lengths.



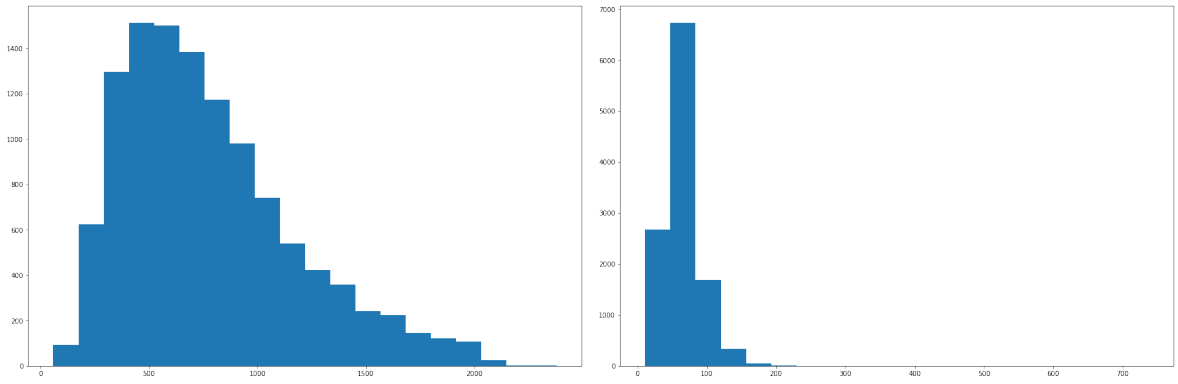
(a) Article tokens (b) Summary tokens

Figure 5.3: Distributions of the number of tokens in training set.



(a) Article tokens (b) Summary tokens

Figure 5.4: Distributions of the number of tokens in development set.



(a) Article tokens (b) Summary tokens

Figure 5.5: Distributions of the number of tokens in test set.

In particular, we find that both training and validation sets have approximately 88.3 percent of article with 1,500 tokens or less. This figure is 87.4 percent for the test set. Therefore, if we are using just a truncated version of the CNN/DailyMail dataset, there is a possibility that we are missing out a lot of information, especially for extractive summarization. In extractive summarization, each sentence missing is a huge loss for the system in terms of ROUGE evaluation scores.

With the above observation, we set to investigate the differences between the truncated version and the full article version of the dataset on extractive summarization task in § 5.1.1.6 below.

5.1.1.4 Manual analysis

We sample 100 article-summary pairs from the dataset to examine manually. Most of the time, the content related to the first highlight will be at the beginning of the article. We observe that in some cases the order of later highlights is not presented in the same order as the related information in the source article. In other words, second bullet summary may sometimes come from the middle part of the article, while the third highlight may be summarized from first few sentences of the source.

Most of the training-summary pairs will be the same in the ordering of information: first summary sentence is extracted from the first few sentences, second sentence will likely to contain information in the following segments, so on. For example, in the following source-target pair, the two highlights of the article were rewritten with the information extracted from the first (line 2) and third (line 4) sentences of the source articles.

```
1  Source:
2      paris , france -- world number three novak djokovic crashed out of
       the paris masters after being trounced in his opening match by veteran
       frenchman fabrice santoro on wednesday .
3      fabrice santoro returns the ball during his shock second-round
       victory against novak djokovic .
4      the 34-year-old santoro , who beat world no. 5 andy roddick at the
       lyon grand prix last week , again rose to the occasion in front of his
       home fans as he stormed to a 6-3 6-2 second-round victory against the
       serb .
5      it was the first time the two players had met , with djokovic making
       his first outing since losing to david nalbandian in the semifinals of
       the madrid masters two weeks ago .
6      djokovic , like the american roddick , has already qualified for the
       season-ending masters cup and will now have extra time to prepare for
       the event in shanghai starting next month
7      . the 20-year-old said he was struggling following dental surgery to
       remove two wisdom teeth .
8      `` i could n't give my 100 percent , not even 30 percent of my
       possibilities , '' djokobvic said . `` he deserved to win .
9      [...]
10
11 Target:
12     world no. 3 novak djokovic beaten in the second round of the paris
       masters .
13     the serb , who had a first-round bye , lost 6-3 6-2 to veteran
       fabrice santoro .
14     [...]
```

However, the ordering of summaries sometimes are not the same as the information in the source article. The example below show that the first summary line mention the head banging that is located in the 7th sentence of the source article (line 8). Meanwhile, the second highlight is related to information of *immorality* stated in line 5.

```
1  Source:
2      -lrb- cnn -rrb- -- polygamist sect leader warren jeffs tried to hang
       himself earlier this year while he was in jail awaiting trial ,
       according to court documents unsealed by a utah judge on tuesday .
3      sect leader warren jeffs arrives in court to hear the verdict against
       him september 25 in st. george , utah .
4      jeffs , the leader and so-called prophet of the 10,000-member
```

```
    fundamentalist church of jesus christ of latter day saints , is now
    awaiting sentencing after being convicted on two counts of being an
    accomplice to rape .
5    the documents , released by fifth district judge james shumate at the
     request of the media , also indicate that jeffs confessed to ``
    immorality '' with a `` sister '' and a daughter more than 30 years
    ago .
6    among the documents is a competency report on jeffs completed in
    april , in which social worker eric nielsen wrote that throughout the
    month of january , jeffs refused food and drink and developed ulcers
    on his knees from kneeling in prayer for hours .
7    on january 28 , the report said , he attempted to hang himself in his
     cell .
8    in the days following the suicide attempt , while he was on suicide
    watch , jeffs on separate occasions threw himself against the wall and
     banged his head on the wall .
9    [...]
10
11 Target:
12    documents say after suicide attempt , jeffs repeatedly banged head on
     cell wall .
13    transcripts say jeffs confessed to `` immorality , '' said he is not
    `` the prophet ''.
14    jeffs ' attorneys say he has recanted statements .
15    [...]
```

We think that human factor is the reason for this phenomenon. Because these are news articles and they were written by different authors, each person will have a unique writing style which leads to various ways of ordering information. Some people summarize the articles section by section, other scan the whole articles. This inconsistency of information order may raise a challenging problem for the text summarization systems.

5.1.1.5 pyROUGE

Following recent works in text summmarization [69, 67, 63, 53], we evaluate the performance with the standard ROUGE-1, ROUGE-2 and ROUGE-L F1 scores [19] by using the pyrouge package [4].

[4] https://github.com/andersjo/pyrouge

5.1.1.6 Summary oracle for extractive summarization

We investigate different factors that we think may have some effects on the oracle summary of an article. An oracle summary is basically a ground truth summary of an article. Since the original version of the CNN/DailyMail dataset was provided with abstractive highlights as their summaries [49], we have to extract context-related sentences from the original articles for the extractive labels.

In particular, we adapt the method of Liu, 2019, in choosing the sentences based on the combined score of ROUGE-1 and ROUGE-2 F1 scores. In the original paper, the author proposed two ways of selecting sentences for ground truth of the extractive summary task: greedy selection and combination selection.

For the greedy approach, to obtain the oracle summaries for extractive summarization task, first we compute the ROUGE-1 and ROUGE-2 F1 scores to get one sentence with the highest combined score, which means closest to the ground truth. The second sentence will be picked according to the highest combined ROUGE scores of it and the selected sentence, so on. For this greedy approach, only 3 sentences will be picked to be the ground truth for extractive task.

In the combination approach, we first generate all possible combinations of sentence with lengths from 1 to 4. For sentence length of 1, if the ROUGE score for that sentence is 0, then we list it as impossible sentence which means it cannot be in the ground truth summary. We compute the combined ROUGE scores for all generated combinations and pick the best combination that has the best combined score as the oracle sentence.

We experiment with the length of articles, methods of choosing label sentences, and the objective score while selecting sentences.

5.1.2 ROUGE

5.1.2.1 Definition

ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. It is a set of metrics mostly used in evaluating machine translation and automatic text summarization systems introduced by Lin in 2004 [19]. It consists of multiple measurements based on recall (i.e. overlapping of words) between a set of automatically generated summaries and a set of human-annotated (ground truth) summaries. There are 4 metrics: ROUGE-N, ROUGE-L, ROUGE-S, ROUGE-W which we will explain in the following section.

5.1.2.2 Types of ROUGE

ROUGE-N: N-gram Co-Occurrence Statistics

ROUGE-N refers to the overlaps of n-grams between the gold summary and the system generated one. Specifically, ROUGE-1 is the unigram recall of the reference and system summaries. ROUGE-2 is the bigram recall between those summaries. The formula for computing ROUGE-N is as follows:

$$ROUGE - N = \frac{\Sigma_{ReferenceSummaries}\Sigma_{gram_n}Count_{match}(gram_n)}{\Sigma_{ReferenceSummaries}\Sigma_{gram_n}Count(gram_n)} \tag{5.1}$$

where n is the length of the n-grams ($gram_n$ which we are computing). The inner summation Σ_{gram_n} loops through all $gram_n$ (unigrams or bigrams) in a single pair of system-reference summaries and calculate the total number of matching n-grams between those two summaries. The $\Sigma_{ReferenceSummaries}$ goes through every pair of system-reference summaries to repeat the same process over all reference summaries. The final ROUGE-N score is averaged over all computed scores above. This score represents the overlap of n-grams between the system generated summaries and the reference ones.

For example, consider these two summaries:

```
1  the dog is playing in the garden
2  the dog is in the garden
```

The first sentence is what a summarization system had generated while the second one is the reference summary created by human. If we consider the recall of words over unigram only (i.e. the overlapping of separate words) between these two sentences, the $ROUGE - 1_{recall}$ will be:

$$ROUGE - 1_{recall} = \frac{number\_of\_overlapping\_words}{total\_words\_in\_reference\_summary} = \frac{6}{6} = 1$$

It means that our system has successfully captured and generated all the necessary words as the reference summary. We then proceed to measure how relevant the system summary is compared to reference one by calculating the precision. The formula is straightforward as follows:

$$ROUGE - 1_{precision} = \frac{number\_of\_overlapping\_words}{total\_words\_in\_system\_summary} = \frac{6}{7} = 0.86$$

However, if we are using only unigram for measuring the performance of a summarization system, it is quite unreasonable because we are just considering summaries based on word level. In other words, if every word in our system summary in the example above is scrambled, we still get the exact same ROUGE score for a sentence that is ungrammatical and basically has no meaning. Therefore, we have to consider higher n-grams to apply more constraints into the summarization system.

Now in order to compute the ROUGE-2 score for the two sentences above, we need to list out all the bigrams from both of them. For the system summary, we have the following bigrams:

```
the dog, dog is, is playing, playing in, in the, the garden
```

And for the reference summary, here are all the bigrams:

```
the dog, dog is, is in, in the, the garden
```

Now we apply the same precision and recall formulas to compute on the bigrams that we have listed:

$$ROUGE - 2_{precision} = \frac{4}{6} = 0.67$$
$$ROUGE - 2_{recall} = \frac{4}{5} = 0.8$$

ROUGE-1 and ROUGE-2 are usually used together to show the fluency of the summaries which is extremely important in some cases. A system summary is considered to be more fluent if it has similar word orderings (bigram in this case) as the reference one. For example, in abstractive summarization, one wants the system to generate summaries that both are fluent and also contain good novel words. And the reference summaries that were created by human are the perfect examples for a summarization system to follow.

ROUGE-L: Longest Common Subsequence

One particular problem of using only ROUGE-N (especially ROUGE-1 and ROUGE-2 only) is that it only considers the summary in word level order. Suppose we have two completely opposite sentences in terms of meaning, but somehow they score the same ROUGE-1 and ROUGE-2 scores, we cannot consider that system summary a good one because it does not reflect the original semantic meaning of the reference one. To make thing clear, we have the following example:

```
S1: dogs ate the steak
```

```
2     S2: dogs eat the steak
3     S3: the steak eats dogs
```

We have S1 as the reference summary, while S2 and S3 are two summaries generated by two summarization systems. If we consider ROUGE-2 on these two sentences with respect to the reference one, they both have the same score because of the only bigram "**the steak**". Now if we consider those on a sentence level, it is obvious that S2 is the more suitable one for the reference S1. We can easily see that S2 sticks closely to the original meaning of S1, while S2, however, says complete nonsense. ROUGE-L can help us decide these type of situations.

ROUGE-L examines longest matching sequences between sentences based on Longest Common Subsequence (LCS). In the example above, S2 will have a score of $3/4$ because it matches the first and the last two words ("**dogs ... the steak**") with the reference one. S3, on the other hand, only matches a single bigram "**the steak**" with S1, therefore, it only scores 2/4. The great thing about LCS is that it does not consider consecutive subsequence matches but in-sequence matches. This means that ROUGE-L works on sentence level word order instead of just plain word order, which means semantic aspect of the summary is somewhat measured in evaluating automatic summarization system.

ROUGE-S and ROUGE-W

ROUGE-S (Skip-Bigram Co-Occurrence Statistics) and ROUGE-W (Weighted Longest Common Subsequence) were also introduced by Lin [19] in the original package. However, these two metrics are not used in current evaluating leaderboard of automatic text summarization systems. Therefore, we are just going to give brief information about these metrics on what they work with when evaluating summaries.

ROUGE-W works exactly like ROUGE-L as it uses LCS to evaluate summaries. The different is it gives weights to different LCS matches. By doing this, it allows us to choose sequences with even more correct in terms of semantics as we could set it to favors consecutive matches instead of skipped one like the example in the section above.

ROUGE-S focuses on bigram, or more specifically skip-bigram. Skip-bigram is any pair of words in a sentence that is in order and has arbitrary gaps in between. For example, with the previous example "**dogs eat the steak**", we can have the following skip-bigrams: "dogs eat, dogs the, dogs steak, eat the, eat steak, the steak".

5.2 Experiments

5.2.1 Experiment A: Auto-regressive model by jointly scoring and selecting important sentences

In this experiment, we perform extractive summarization on CNN/DailyMail dataset with BERT encoder in truncated version (first 512 tokens) of the input articles. We want to investigate the performance of different decoding schemes with BERT encoder. Specifically, we adapt auto-regressive decoder with LSTM and pointer network from the work of Zhou el al., 2018 [67] to BERT encoder of this work [69].

Result

In Table 5.2, we show the results of our above experiment as BERT+AutoReg. We also provide the results for some of the most recent works on CNN/DailyMail dataset for comparison:

- POINTER-GEN or Pointer Generator Network also proposed by See et al. [54] is an abstractive summarization system which utilizes pointer network for better word generation.
- DCA by Celikyilmaz et al., 2018 stands for Deep Communicating Agents which is also an abstractive summarization system. The model incorporates reinforcement learning by using multiple agents for document encoding, attention mechanism, and also decoding.
- REFRESH (Narayan et al., 2018 [63]) is also an abstractive system that uses reinforcement learning. It learns by maximize the ROUGE evaluation scores.
- BOTTOM_UP (Gehrmann et al., 2018 [59]) is a two-stage abstractive summarization system that consists of a pointer generator network for baseline summary and a content selector to re-select words for the final summary.
- LEAD3 is a strong baseline for extractive summarization created by See et al. [54]. It simply takes the first three sentences of an input document as its summary.
- CRSUM by Ren et al., 2017 extracts sentence relations for a more global contextual approach [53].

- NEUSUM from Zhou et al., 2018 [67] is an extractive summarization system that use a GRU recurrent network as a auto-regressive decoder for jointly score and select sentences.

- BERTSUM by Liu, 2019, is the state-of-the-art approach in extractive summarization [69]. It includes BERT as sentence encoder and different non-regressive decoders.

The first three systems are abstractive summarization models. The next three are works in extractive summarization tasks. The methods of using BERT in extractive summarization are showed separately in the third section. These are currently the state-of-the-art systems at the time of this thesis. In the final section, we show our result of the auto-regressive decoder module in combined with BERT sentence encoder.

Table 5.2: ROUGE F_1 evaluation results on the CNN/DailyMail test set.

Model	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**	**Average ROUGE**
POINTER-GEN [54]	39.53	17.28	37.98	31.60
DCA [56]	41.69	19.47	37.92	33.03
REFRESH [63]	41.0	18.8	37.7	32.50
BOTTOM_UP [59]	41.22	18.68	38.34	32.75
LEAD3 [54]	40.42	17.62	36.67	31.57
CRSUM [53]	40.52	18.08	36.81	31.80
NEUSUM [67]	41.59	19.01	37.98	32.86
BERTSUM+Classifier [69]	43.23	20.22	39.60	34.35
BERTSUM+Transformer [69]	**43.25**	**20.24**	**39.63**	**34.37**
BERTSUM+LSTM [69]	43.22	20.17	39.59	34.33
BERT+AutoReg (Ours)	41.80	19.20	38.10	33.03

In our experiment with combining BERT sentence encoder with a LSTM auto-regressive decoder, the result is not as we expected. The model only achieve scores of 41.8 on ROUGE-1, 19.2 and 38.1 on ROUGE-2 and ROUGE-L respectively on test dataset. This is approximately the same as results produced by Zhou et al. in their Neusum model which only encodes sentences with a traditional bidirectional RNN [67]. We hypothesize that since BERT is pre-trained by masked language model and next sentence prediction in which results are predicted in non-regressive approach, thus there is a mismatch when using encoding information from BERT and decoding them in auto-regressive way.

5.2.2 Experiment B: Non-regressive model on BERT by utilizing full length of input articles

In this experiment, we perform extractive summarization on CNN/DailyMail dataset with full length of input article with BERT encoder instead of truncated versions (only first 512 tokens of input articles) in previous works using the same BERT encoder [69] [72].

5.2.2.1 Oracle on full input articles

Extractive summarization requires the extraction of the exact sentences of the most important information from the source article. This means that each sentence extracted could be responsible for approximately 25-33% of the final ROUGE scores, assuming that a normal summary contains about 3-4 sentences. Previously, the study of See et al. (2017) showed that training model with only the first 400 words of an article achieved a better results than using 800 words [54] on abstractive summarization task. Liu in 2019 with their fine-tuning BERT approach also uses only up to 512 tokens for the extractive model [69]. However, from the observations that we made in Section 5.1.1 about some summaries that are related to the end of the articles, we argue that using the whole article in extractive summarization can give better evaluation scores in ROUGE metrics.

We test the effect of full length articles versus the 512 tokens truncated version that was used by Liu in their model [69]. Moreover, we also use the combination approach which was not used in the original paper. The result of the test is as what we expected from the beginning as for the greedy approach, there are a 3.95 and a 3.8 gain in ROUGE-1 and ROUGE-L scores respectively in the validation subset. ROUGE-2 score also increases from 31.71 to 34.34 (details are in Table 5.3).

Table 5.3: ROUGE evaluation scores on different oracle settings of the CNN/DailyMail validation and test sets.

Method		Validation			Test		
		R-1	R-2	R-L	R-1	R-2	R-L
Truncated (512 tokens)	Greedy ROUGE-2[F1] only	51.62	32.34	48.52	50.99	31.84	47.88
	Greedy	53.09	31.71	49.45	52.59	31.23	48.87
	Combination	53.09	31.71	49.45	52.59	31.23	48.87
Full article	Greedy	57.04	34.34	53.25	56.45	33.82	52.62
	Combination	**57.76**	**34.8**	**54.04**	**57.05**	**34.2**	**53.28**

From Table 5.1, for all subsets, there are approximately 25-30% of source articles that have less than 500 tokens. With over 70% of source articles that are longer than 500 tokens,

when considering truncated versions of these articles, we are missing out a lot of information. When the source articles are limited to just 512 tokens, the proposed approach is only capable of retrieving a ROUGE-1 score of 53.09 and 52.59 for validation and test set respectively. Surprisingly, both extracting approaches give the exact same score in this setting. We run the greedy retrieval approach on the whole article and gain almost 4 ROUGE-1 points on both subsets. Furthermore, there is an improvement in the combination method compared to the greedy one. These information confirms our hypothesis on the potential improvement that is being missed out from shortening the training article in extractive summarization.

Distribution of oracle sentences in different BERT segments

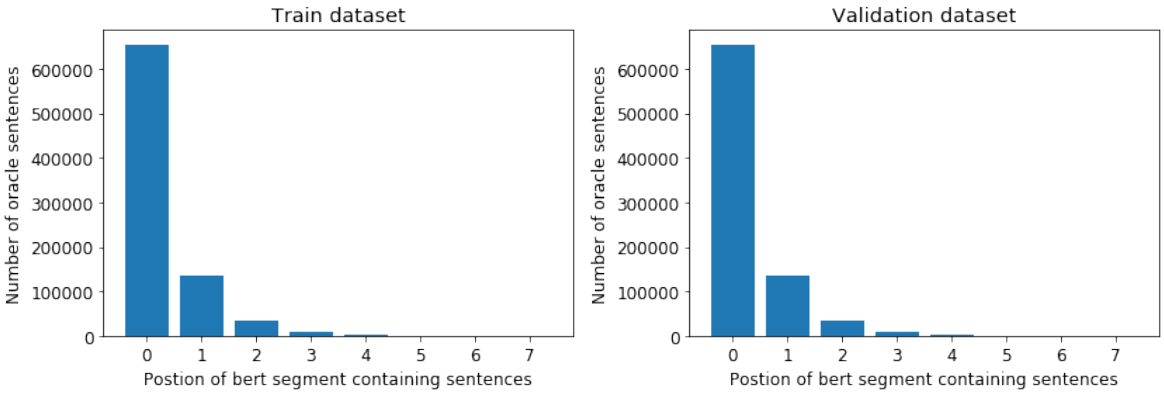
We want to investigate how oracle sentences (sentences that are labeled "1" by our oracle) distributes in different BERT segments if we split article mentioned in section 4.3.1. As shown in Figure 5.6, there are a considerable amount of oracle sentences in the second and the third segments which account for substantial portion in total oracle sentences. This observation proves that using only one BERT segments which contains only the first 512 tokens in input article is not enough for generating oracle.

Particularly, we generate the oracle summaries for different lengths of input articles and compute the ROUGE scores for each setting. Due to limited time, we were only able to get the results for the greedy approaches of 5 different lengths and 2 for the combination approach. Each additional BERT increases the total number of tokens by 512 including the special $[CLS]$ and $[SEP]$ tokens. All the results are reported in Table 5.4.

With more information being covered, there is a significant improvement in all ROUGE scores in 3 BERTs setting. It's a 2.97 increase in average ROUGE score of the validation set and 2.87 in the test set if we use 3 BERTs (i.e. 1536 tokens). There is a slightly higher score with 4 BERTs setting, however we think that it is not a considerable gains for another 512 tokens, hence, we will stick with 3 for our experiments. Moreover, for the combination approach, with only 3 segments, the oracle results are just under full length oracle results. We can safely claim that 3 BERTs segments is the sweet spot in information coverage.

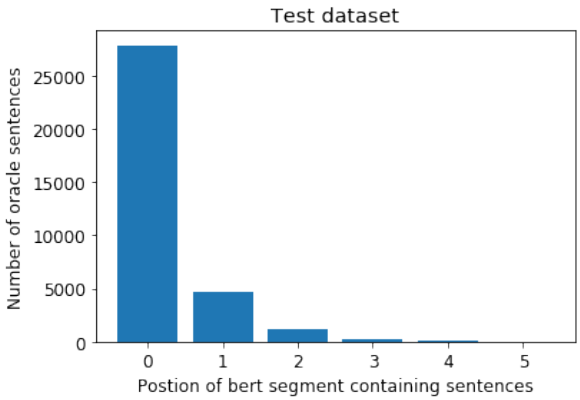
5.2.2.2 Implementation details

We adopt the training process *stack-and-finetune* from Wang et al. [72] in which we undergo two steps. First, we freeze the weights of BERT sentence encoder and only train the top document-level encoder layer (Transformer). Second, we train the whole model (encoder along with decoder). We also adopt the hyperparameters from this works, i.e. we set learning



(a) Train

(b) Validation



(c) Test

Figure 5.6: Number of oracle sentences in each BERT segments by soft splitting

rate to $lr = 0.001$ in the first stage of training the document-level layer and set it to $lr = 5e^{-5}$ to the later stage. We use Adam [31] as the optimizer with the default arguments except from L_2 weight decay of 0.01 since such configuration has excellent performance in the original BERT [58]. We use gradient clipping if gradient norm of a tensor exceeds the value of $[-5, 5]$.

Trigram blocking

Adopting from Y. Liu, 2019 [69], trigram blocking is used during inference phrase to reduce redundancy of sentences. Specifically, given the selected summary S and a predicted candidate sentence c, we select c if there is not any trigram overlapping between c and S.

Table 5.4: ROUGE scores on oracle summaries of 5 different length configurations on the CNN/DailyMail dataset. Best scores of each column are bolded.

		Validation			Test		
		ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-1	ROUGE-2	ROUGE-L
Greedy	1 BERT	53.09	31.71	49.45	52.59	31.23	48.87
	2 BERTs	54.87	34.66	51.70	54.25	31.17	51.09
	3 BERTs	55.55	35.13	52.35	54.91	34.65	51.73
	4 BERTs	55.67	**35.22**	52.48	55.04	**34.75**	51.85
	Full article	57.04	34.34	53.25	56.45	33.82	52.62
Combination	3 BERTs	57.62	34.71	53.91	56.92	34.11	53.16
	Full article	**57.76**	34.80	**54.04**	**57.05**	34.20	**53.28**

Trucated BERT segments

As illustrutated in Figure 5.6 and Table 5.4, to save memory and boost the training speed, we only use the first 3 BERTs segment in the combination approach to split the document.

5.2.2.3 Results

Here we show the results of our experiment on utilizing the whole articles with BERT for sentence representation extraction and predicting sentence labels with the modified Transformer network. We are able to achieve state-of-the-art results on ROUGE-1 with the score of 43.31 and ROUGE-L with the score of 39.77 on the ROUGE evaluation metrics; on average ROUGE score of 34.43 for extractive summarization task (improving the previous state-of-the-art result of BERTSUM by 0.06 average ROUGE).

Table 5.5: ROUGE F_1 evaluation results on the CNN/DailyMail test set.

Model	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**	**Average ROUGE**
POINTER-GEN [54]	39.53	17.28	37.98	31.60
DCA [56]	41.69	19.47	37.92	33.03
REFRESH [63]	41.0	18.8	37.7	32.50
BOTTOM_UP [59]	41.22	18.68	38.34	32.75
LEAD3 [54]	40.42	17.62	36.67	31.57
CRSUM [53]	40.52	18.08	36.81	31.80
NEUSUM [67]	41.59	19.01	37.98	32.86
BERTSUM+Classifier [69]	43.23	20.22	39.60	34.35
BERTSUM+Transformer [69]	43.25	**20.24**	39.63	34.37
BERTSUM+LSTM [69]	43.22	20.17	39.59	34.33
BERT+AutoReg (Ours)	41.80	19.20	38.10	33.03
MultiBERT+ Combination (Ours)	**43.31**	20.20	**39.77**	**34.43**
- interval segments	43.29	20.17	39.75	34.40
- trigram blocking	42.68	20.00	39.2	33.96

Oracle choice: greedy or combination?

Table 5.6: ROUGE F_1 evaluation results on the CNN/DailyMail test set with greedy oracles and combination oracles

Model	ROUGE-1	ROUGE-2	ROUGE-L	Average ROUGE
Combination	43.31	20.20	39.77	34.43
Greedy	43.30	20.18	39.76	34.41

We show how greedy and combination oracle help our model to learn the task. As shown in Table 5.6, combination oracle do help our model learn better than greedy one with the improvement of 0.02 average ROUGE score. This can be explained due to the fact that combination oracle can take more sentence combinations into consideration, thus expanding our search space of suitable sentences.

Ablation studies

Ablation studies are conducted to show the contribution of different components of our model. As illustrated in Table 5.5, trigram blocking is used in inference phrase can boost the performance from 33.96 to 34.43 average ROUGE (improving by +0.47). Interval segments meanwhile only improve the average ROUGE score by 0.03 (from 34.40 to 34.43).

Performance on articles with smaller and larger than 512 tokens

Because our method focuses on dealing with article with much longer article lengths, we want to make sure that with the new segment system, the model is still able to correctly capture and predict sentences in shorter articles. In particular, we divide the test set of CNN/DailyMail dataset into two subsets: one with all the articles less than 512 tokens, and one for the rest. We measure the ROUGE performance on two subsets between BERTSUM [69] and our model to see how well the model deals with longer articles and is it as good as BERTSUM in capturing in shorter version. We use pyRouge to measure the evaluation scores of two systems. All parameters are the same between two models. Initial seeds are also identical.

In Table 5.7, we show the evaluation scores between our model and BERTSUM when summarizing from articles with less than or longer than 512 tokens. If input articles are shorter than 512 tokens, our model still keeps the same performance level as BERTSUM. However, when dealing with articles that are longer than 512 tokens, we outperform BERTSUM with 0.1 average ROUGE score. In particular, there is a 0.15 increase in ROUGE-1,

Table 5.7: Performance comparison between our model and BERTSUM on articles with less than or equal to 512 tokens and on articles with larger than 512 tokens

		R-1	R-2	R-L	Average ROUGE
Less than 512 tokens	BERTSUM	45.64	23.20	42.19	37.01
	Ours	**45.65**	**23.22**	**42.20**	**37.02**
More than 512 tokens	BERTSUM	42.65	19.42	39.00	33.69
	Ours	**42.80**	**19.49**	**39.10**	**33.80**

0.07 increase in ROUGE-2, and 0.1 improved score on ROUGE-L. These results prove that our model is working well with longer input and able to capture sentences at later section of the articles.

Sample results

We sample some predicted results in the test data from our model in Table 5.8. We observe that our model do take into accounts of the text context in BERT segments other than the first one. Specifically, in Table 5.8, the extractive oracle chooses the sentence "*The shocking footage comes after it has emerged that one in 12 truckies are high on illicit substances when they are pulled over by Victorian police and many are rorting the system so they can drive for up to 16 hours a day.*" in the second BERT segment as the sentence summary. In previous approaches, this sentence is discarded because we have to truncate the first 512 (word-piece) tokens when we feed the article input into BERT, thus the final summary does not have this important sentence. However, our model takes longer input (up to 3 BERT segments with approximately $512 * 3 = 1,536$ word-piece tokens), hence it can detect this sentence as one that should be in the summary.

Table 5.8: An sample predicted result of our model from test dataset

Source article The first 512 tokens *Truck drivers in Victoria are dealing drugs from the behind the wheel of 60-tonne rigs, using secretive codes over their radio systems.* In a report issued to the Herald Sun, Victoria police revealed that there is a distribution network of speed, ice and marijuana operating in the trucking industry and one in every 12 truckies tested positive for drugs while out on the road. [...] One truck driver filmed himself lighting up a pipe filled with Ice (methamphetamine) and taking to the Hume Highway in an A Current Affair report. The next 512 tokens A worrying video appeared online in February exposing a truck driver (above) snorting drugs from behind the wheel of a heavy vehicle before taking to the road [...] *The shocking footage comes after it has emerged that one in 12 truckies are high on illicit substances when they are pulled over by Victorian police and many are rorting the system so they can drive for up to 16 hours a day.* [...]
Golden summary Truck drivers in Victoria are dealing drugs using codes over their radios. It has been reported that 1 in 12 truck drivers are high when pulled over. According to a former truckie, the trend is more common than you think. A video has appeared online showing a man snorting ice behind the wheel He is driving a 40-tonne rig and says it is for his fatigue. Many are rorting the system so they can drive for 16 hours a day.
Extractive oracle Extractive oracle chooses italics sentences in the full article
Predicted sentences Truck drivers in Victoria are dealing drugs from the behind the wheel of 60-tonne rigs, using secretive codes over their radio systems. Speaking to the publication, a former Melbourne truck driver said his colleagues were taking drugs regularly, and meeting areas were often set up between drivers over radios using codes, where drugs were then handed out. The shocking footage comes after it has emerged that one in 12 truckies are high on illicit substances when they are pulled over by Victorian police and many are rorting the system so they can drive for up to 16 hours a day.

Chapter 6

Conclusion

6.1 Conclusion

With the goal of finding new approach that can improve the performance of automatic extractive summarization system, we have tried several approaches regarding combining state-of-the-art contextualized word embeddings model BERT. Firstly, we combine BERT with a jointly sentence scoring and selecting module. We then proceed to investigate the effect of article lengths on extractive summarization. Finally, we propose a new method of utilizing all sentences in input articles with BERT to increase the performance of the system on the CNN/DailyMail dataset.

For our combining method of BERT and joint module, we come to the conclusion that using an auto-regressive decoder with BERT is irrational. Because originally BERT was used for Next Sentence Prediction task which only requires it to predict one sentence right after the first one. Therefore, BERT cannot learn to predict in such orderly matter approach where it has to take multiple sentences and their orders into account.

We also examine the impact of longer input documents on both generating ground truth summaries as well as training the model. We find that when considering the whole article for producing oracle summaries, there are large improvements on ROUGE evaluation scores across the dataset. This proves that we are missing out a lot of important information when truncating the article to a smaller size.

In our attempt at utilizing all the tokens in the articles, we achieve state-of-the-art results on the extractive summarization task of the CNN/DailyMail dataset. This result once more proves our claim at information being missing out on the truncated version of the CNN/DailyMail dataset.

6.2 Future works

There are several experiments that we did not have enough time to conduct. In particular, following the recent work of Wang et al., 2019 [71], we intend to try out BERT in different pretrained tasks, especially Switching method, with our current model as the author showed promising result in their paper. We think that with this technique, we could further improve the performance of our model on the extractive summarization task.

Moreover, at the moment, our method of splitting does not take positional embedding into consideration when dividing document into segments. It means that when encoding segments, there is a chance that a sentence in one segment is more related to another sentence in a different segment as we split. This is indeed a problem during encoding because BERT attends to all sentences at once to encode a vector, hence, there could be a loss of contextual information in the current approach. We think that with the new Transformer XL from Dai et al. (2019) [68], the limit of length will be discarded which may be the solution for our method of using the whole input articles.

Bibliography

[1] Hans Peter Luhn. The automatic creation of literature abstracts. *IBM Journal of Research and Development*, 2(2):159–165, 1958.

[2] H. P. Edmundson. New methods in automatic extracting. *J. ACM*, 16(2):264–285, 1969.

[3] G. DeJong. An overview of the FRUMP system. In W. Lehnert and M.H. Ringle, editors, *Strategies for Natural Language Processing*, pages 149–176. Lawrence Erlbaum, 1982.

[4] Gerard Salton and Chris Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, 1988.

[5] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, March 1994.

[6] Julian Kupiec, Jan O. Pedersen, and Francine Chen. A trainable document summarizer. In *SIGIR'95, Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Seattle, Washington, USA, July 9-13, 1995 (Special Issue of the SIGIR Forum)*, pages 68–73, 1995.

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

[8] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Trans. Signal Processing*, 45(11):2673–2681, 1997.

[9] Karen Sparck Jones. Automatic summarising: Factors and directions. In *Advances in Automatic Text Summarization*, pages 1–12. MIT Press, 1998.

[10] Dragomir R. Radev and Kathleen R. McKeown. Generating natural language summaries from multiple on-line sources. *Computational Linguistics*, 24(3):469–500, 1998.

[11] Inderjeet Mani, Barbara Gates, and Eric Bloedorn. Improving summaries by revising them. In *27th Annual Meeting of the Association for Computational Linguistics, University of Maryland, College Park, Maryland, USA, 20-26 June 1999.*, 1999.

[12] Hongyan Jing and Kathleen R. McKeown. Cut and paste based text summarization. In *6th Applied Natural Language Processing Conference, ANLP 2000, Seattle, Washington, USA, April 29 - May 4, 2000*, pages 178–185, 2000.

[13] Kevin Knight and Daniel Marcu. Statistics-based summarization - step one: Sentence compression. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, pages 703–710, 2000.

[14] Chin-Yew Lin and Eduard H. Hovy. The automated acquisition of topic signatures for text summarization. In *COLING 2000, 18th International Conference on Computational Linguistics, Proceedings of the Conference, 2 Volumes, July 31 - August 4, 2000, Universität des Saarlandes, Saarbrücken, Germany*, pages 495–501, 2000.

[15] Yihong Gong and Xin Liu. Generic text summarization using relevance measure and latent semantic analysis. In *SIGIR 2001: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, September 9-13, 2001, New Orleans, Louisiana, USA*, pages 19–25, 2001.

[16] Kevin Knight and Daniel Marcu. Summarization beyond sentence extraction: A probabilistic approach to sentence compression. *Artif. Intell.*, 139(1):91–107, 2002.

[17] Günes Erkan and Dragomir R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *J. Artif. Intell. Res.*, 22:457–479, 2004.

[18] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 60(5):493–502, 2004.

[19] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[20] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing , EMNLP 2004, A meeting of SIGDAT, a Special Interest Group of the ACL, held in conjunction with ACL 2004, 25-26 July 2004, Barcelona, Spain*, pages 404–411, 2004.

[21] Herbert Jaeger. A tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the "echo state network" approach. 2005.

[22] James Clarke and Mirella Lapata. Models for sentence compression: A comparison across domains, training requirements and evaluation measures. In *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17-21 July 2006*, 2006.

[23] Paul Over, Hoa Dang, and Donna Harman. DUC in context. *Inf. Process. Manage.*, 43(6):1506–1520, 2007.

[24] Emiel Krahmer, Erwin Marsi, and Paul Pelt. Query-based sentence fusion is better defined and leads to more preferred results than generic sentence fusion. In *ACL 2008, Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, June 15-20, 2008, Columbus, Ohio, USA, Short Papers*, pages 193–196, 2008.

[25] Kam-Fai Wong, Mingli Wu, and Wenjie Li. Extractive summarization using supervised and semi-supervised learning. In *COLING 2008, 22nd International Conference on Computational Linguistics, Proceedings of the Conference, 18-22 August 2008, Manchester, UK*, pages 985–992, 2008.

[26] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.

[27] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649, 2013.

[28] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances*

in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States., pages 3111–3119, 2013.

[29] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, pages 103–111, 2014.

[30] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734, 2014.

[31] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. Technical report, 2014.

[32] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[33] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[34] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[35] Ziqiang Cao, Furu Wei, Sujian Li, Wenjie Li, Ming Zhou, and Houfeng Wang. Learning summary prior representation for extractive summarization. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers*, pages 829–833, 2015.

[36] Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *CoRR*, abs/1506.03340, 2015.

[37] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1412–1421, 2015.

[38] Yajie Miao, Mohammad Gowayyed, and Florian Metze. EESEN: end-to-end speech recognition using deep RNN models and wfst-based decoding. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2015, Scottsdale, AZ, USA, December 13-17, 2015*, pages 167–174, 2015.

[39] Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 379–389, 2015.

[40] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *CoRR*, abs/1503.00075, 2015.

[41] Ming Tan, Bing Xiang, and Bowen Zhou. Lstm-based deep learning models for non-factoid answer selection. *CoRR*, abs/1511.04108, 2015.

[42] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2692–2700, 2015.

[43] Di Wang and Eric Nyberg. A long short-term memory model for answer sentence selection in question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers*, pages 707–712, 2015.

[44] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[45] Jianpeng Cheng and Mirella Lapata. Neural summarization by extracting sentences and words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[46] Orhan Firat, KyungHyun Cho, and Yoshua Bengio. Multi-way, multilingual neural machine translation with a shared attention mechanism. *CoRR*, abs/1601.01073, 2016.

[47] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. SummaRuNNer: A Recurrent Neural Network based Sequence Model for Extractive Summarization of Documents. Technical report, 2016.

[48] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. *CoRR*, abs/1611.04230, 2016.

[49] Ramesh Nallapati, Bowen Zhou, Cícero Nogueira dos Santos, Çaglar Gülçehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*, pages 280–290, 2016.

[50] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[51] Xingyou Wang, Weijie Jiang, and Zhiyong Luo. Combination of convolutional and recurrent neural network for sentiment analysis of short texts. In *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11-16, 2016, Osaka, Japan*, pages 2428–2437, 2016.

[52] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *CoRR*, abs/1705.04304, 2017.

[53] Pengjie Ren, Zhumin Chen, Zhaochun Ren, Furu Wei, Jun Ma, and Maarten de Rijke. Leveraging contextual sentence relations for extractive summarization using a neural attention model. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 95–104, 2017.

[54] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1073–1083, 2017.

[55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6000–6010, 2017.

[56] Asli Çelikyilmaz, Antoine Bosselut, Xiaodong He, and Yejin Choi. Deep communicating agents for abstractive summarization. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1662–1675, 2018.

[57] Xiuying Chen, Shen Gao, Chongyang Tao, Yan Song, Dongyan Zhao, and Rui Yan. Iterative document representation learning towards summarization with polishing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 4088–4097, 2018.

[58] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[59] Sebastian Gehrmann, Yuntian Deng, and Alexander M. Rush. Bottom-up abstractive summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 4098–4109, 2018.

[60] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 328–339, 2018.

[61] Aishwarya Jadhav and Vaibhav Rajan. Extractive summarization with SWAP-NET: sentences and words from alternating pointer networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 142–151, 2018.

[62] Chris Kedzie, Kathleen R. McKeown, and Hal Daumé III. Content selection in deep learning models of summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1818–1828, 2018.

[63] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Ranking sentences for extractive summarization with reinforcement learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1747–1759, 2018.

[64] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 2227–2237, 2018.

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

[66] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N. Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. Tensor2tensor for neural machine translation. *CoRR*, abs/1803.07416, 2018.

[67] Qingyu Zhou, Nan Yang, Furu Wei, Shaohan Huang, Ming Zhou, and Tiejun Zhao. Neural document summarization by jointly learning to score and select sentences. In

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers, pages 654–663, 2018.

[68] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. 2019.

[69] Yang Liu. Fine-tune BERT for extractive summarization. *CoRR*, abs/1903.10318, 2019.

[70] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune BERT for text classification? *CoRR*, abs/1905.05583, 2019.

[71] Hong Wang, Xin Wang, Wenhan Xiong, Mo Yu, Xiaoxiao Guo, Shiyu Chang, and William Yang Wang. Self-Supervised Learning for Contextualized Extractive Summarization. 2019.

[72] Ran Wang, Haibo Su, Chunye Wang, Kailin Ji, and Jupeng Ding. To Tune or Not To Tune? How About the Best of Both Worlds? 2019.

[73] Hu Xu, Bing Liu, Lei Shu, and Philip S. Yu. BERT post-training for review reading comprehension and aspect-based sentiment analysis. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2324–2335, 2019.

[74] Haoyu Zhang, Yeyun Gong, Yu Yan, Nan Duan, Jianjun Xu, Ji Wang, Ming Gong, and Ming Zhou. Pretraining-based natural language generation for text summarization. *CoRR*, abs/1902.09243, 2019.

[75] Xingxing Zhang, Furu Wei, and Ming Zhou. HIBERT: document level pre-training of hierarchical bidirectional transformers for document summarization. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 5059–5069, 2019.

[76] Ming Zhong, Pengfei Liu, Danqing Wang, Xipeng Qiu, and Xuanjing Huang. Searching for effective neural extractive summarization: What works and what's next. *CoRR*, abs/1907.03491, 2019.

[77] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.