Abstractive Text Summarization Using Deep Learning

TRAN, Chien Xuan

School of Information Science
Japan Advanced Institute of Science and Technology
June, 2017

Master's Thesis

Abstractive Text Summarization Using Deep Learning

1510032 TRAN, Chien Xuan

Supervisor : Associate Professor NGUYEN, Minh Le
Main Examiner : Associate Professor NGUYEN, Minh Le
Examiners : Professor Satoshi Tojo
Associate Professor Kiyoaki Shirai

School of Information Science
Japan Advanced Institute of Science and Technology

May, 2017

Abstract

Text summarization is one of the most active research in natural language processing. Even though the history of text summarization dates back to 1950s, a majority of research focuses on extractive summarization in which we select some sentences from the input document as the summary. Abstractive summarization is considered as closer to human style, but it has not got enough attention from the community over the years due to its difficulty and complexity.

With the development of deep learning lately, we have witnessed many impressive results in various fields ranging from computer vision to natural language processing. Using deep learning for abstractive summarization has also yielded promising results in recently published papers. However, it is still in an early stage, and more research needs to be done in this field. This thesis presents our study on abstractive text summarization and our contributions: (1) we implement two deep learning models for summarization with different encoding mechanism, then we compare their performance in terms of ROUGE score and computational time; (2) we improve the quality of the summary by employing diverse beam-search decoding and we propose a method to deal with the problem of word redundancy in the output.

First of all, we introduce a widely used deep learning model for abstractive summarization, the encoder-decoder model using recurrent neural network (RNN). In this model, the document is simply viewed as a sequence of words. Based on that, we implement a second model for comparison which treats the input document as a hierarchical structure. To be specific, we use Convolutional Neural Network (CNN) to get the representation of each sentence and use Bidirectional RNN to encode these sentence vectors. By doing this, we hope to reduce the computational complexity of the whole network while still achieving a competitive performance comparing to the standard model. Our experiments show that the hierarchical encoding model achieves comparable full-length ROUGE-1 and ROUGE-L scores while having the smallest computational time. But its ROUGE-2 score is much lower than the standard model. Because ROUGE-2 is an important metric for evaluation, it partly indicates the weakness of hierarchical model in generating a good summary. Our additional experiments also demonstrate the effectiveness of using Part-of-Speech feature for improving the system performance as well as the minor effect of removing stop words from the input document.

Our second work presented in this thesis focuses on the decoding process of neural summarization. The summaries in our deep learning model are generated by using beam-search, a frequently used method in neural text summarization model. By using beam-search with beam size K, we can generate K-best list of outputs and the output with the highest score assigned by the model is selected as the summary. However, this score might not reflect the quality of the summary, i.e. a summary with a lower score can be a better summary. This can be fixed by using a re-ranking stage. The main issue with beam-search is that it produces K-best list with very similar lexical content, which consequently reduces the effectiveness of re-ranking model. To deal with this issue, we

chose to adopt a technique proposed recently to help generating diverse outputs. This method controls the dissimilarity between beams at each step by assigning lower priority to tokens which were already chosen in other beams. The combination with an re-ranker shows the effectiveness of this technique since it yields higher Full-length F1 ROUGE score than the default decoding method.

Another issue found in our summarization model is the repetition of words and phrases, a common problem which has been recognized in other similar works. In this research, we proposed a simple solution which can be incorporated easily into the decoding process. More specifically, we guide the word expansion process in beam search algorithm by assigning lower score to words which can cause the repetition issue. For this purpose, we designed a function to tell how much a word is likely to be repeated based on the unigrams and bigrams of previously decoded tokens in that beam. After performing the evaluation, we found that our proposed method can increase the baseline ROUGE score by up to 40% or more when combining with an Oracle re-ranker. Thus, this method can give researchers another way to improve their existing model without modifying the training architecture.

Despite there are several limitations, our work at least provides more aid for other future research in this field. Our findings and results not only can be applied to abstractive text summarization problem but also for other similar research such as image caption generation, machine translation or spoken dialog generation.

Keywords: abstractive text summarization, deep learning, natural language processing, diverse beam search, re-ranking.

Acknowledgement

Studying at JAIST is a wonderful part of my life. I would like to thank my main supervisor, Associate Professor Nguyen Le Minh, for his continued support. I always admire him for his vast knowledge of artificial intelligence and his care towards all students. Without his guidance, I would not have finished my program smoothly. I am also in debt to Pham Thi Vuong and Nguyen Viet Hung for introducing me to this study opportunity.

I appreciate the feedback offered by Professor Satoshi Tojo and Associate Professor Kiyoaki Shirai, their comments have been a great help in improving the quality of my research. I also have greatly benefited from my minor research supervisor, Associate Professor Beuran Razvan. His dedication and detail-oriented skills are truly impressive. In addition, I received a great support from Professor Ken Satoh of NII. He gave me a chance to participate in a big project and I learned a lot of things from it.

I would like to show my greatest appreciation to many people supporting me during the time I stayed at JAIST. I learned a lot from Professor Kawanishi through his interesting classes about Japan. I also would like to show my gratitude to my Japanese language teachers: Tsutsui-sensei, Horiguchi-sensei, Yukari-sensei, Nakai-sensei, Fujimoto-sensei, and all other teachers in NIFA classes. They were very kind to me and always helped me to improve my Japanese skills.

Special thanks to all members in Nguyen's Laboratory. It is hard to find a lab with many Vietnamese staying in the same room like this. Their knowledge and comments were all helpful to my research.

I would like to thank all members of Vietnamese Football Club (VIJA). Thanks to this club, I could maintain my physical health and did not feel stressed out while doing my research. Additionally, I want to thank JAIST Futsal Club for letting me play with them. I met many Japanese friends there and learned more words outside of my Japanese class.

I owe my deepest gratitude to my family who always gives me the freedom to make my own decision. Especially I would like to thank my dear, M.T, you have always stayed beside me, encouraged me and waited for me. Last but not least, I would like to express the deepest appreciation to my colleagues at NUS Technology. They worked so hard to grow the company when I was not with them.

There are many other people I want to mention here, but because of space limitation, I could not list out all the names. I hope for your understanding.

Tran Xuan Chien
May 2017

Contents

Abstract **i**

Acknowledgement **iii**

1 Introduction **1**
1.1 Background . . . 1
1.2 Motivation . . . 2
1.3 Goals . . . 4

2 Literature Review **5**
2.1 Theoretical frameworks . . . 5
2.1.1 Encoder-Decoder model . . . 5
2.1.2 Recurrent Neural Network . . . 6
2.1.3 Convolutional Neural Network . . . 9
2.1.4 Attention mechanism . . . 10
2.2 Related Work on Neural Summarization . . . 12

3 Summarization Models **16**
3.1 Attention-based Sequence-to-Sequence . . . 16
3.1.1 Encoder . . . 16
3.1.2 Decoder . . . 16
3.1.3 Beam-search decoding . . . 19
3.2 Hierarchical Encoding . . . 20
3.3 Diverse beam-search decoding with Re-ranking . . . 23
3.3.1 Diversity between beams . . . 23
3.3.2 Diversity within a beam . . . 24
3.3.3 Re-ranking . . . 25

4 Evaluation **27**
4.1 Dataset and Preprocessing . . . 27
4.2 Experimental settings . . . 28
4.3 Results . . . 29
4.3.1 Experiment 1: Decoding with standard beam-search . . . 30
4.3.2 Experiment 2: The effect of beam size . . . 31

4.3.3 Experiment 3: Diverse beam-search decoding with reranking 35
4.4 Performance comparison with other systems 38

5 Discussion **43**

6 Conclusion **45**

References **47**

List of Figures

1.1 Illustration of a summarization system. 2

2.1 Encoder-Decoder model. 5
2.2 RNN architecture. 7
2.3 An RNN cell. 7
2.4 Bidirectional RNN. 9
2.5 Illustration of the convolution layer (kernel size is 2, two feature maps are used). 10
2.6 Illustration of MAX pooling layer. 11
2.7 Sentence classification using CNN. 11
2.8 Attention mechanism. 12
2.9 A recurrent convolutional document reader with a neural sentence extractor. 13

3.1 The process of decoding at time step t. The predicted token is the index with highest probability. 18
3.2 Difference between decoder input when training and testing. 19
3.3 Illustration of beam-search decoding with $K = 3$. 21
3.4 Hierarchical encoding for summarization 22
3.5 Diversity by grouping beams. 24
3.6 Illustration of beam-search with local diversity. 25

4.1 A sample document with anonymous entities. 28
4.2 Outputs from different models when we fixed beam size K = 2. 32
4.3 Comparison of ROUGE score when changing the beam size. 34
4.4 A graph showing the variation of decoding time when changing the beam size. 35
4.5 Similar beam outputs generated by the *AttnSeq2Seq* model. Beam size is 5. 37
4.6 Comparison of Full-length F1 score when using different methods for decoding. 39
4.6 Sample outputs generated by different decoding methods. Beam size is 20. 41

List of Tables

4.1 Number of documents in the experimental dataset. 27
4.2 Experimental settings. 29
4.3 Number of epochs, average training time and decoding time on each models. 30
4.4 Performance comparison of different models with fixed beam size K = 2. . 31
4.5 Performance comparison with different beam sizes. 33
4.6 Comparison of decoding time when changing the beam size. 35
4.7 Performance comparison of different decoding methods on various beam sizes. 36
4.8 First-75b F1 comparison with other work. 42
4.9 Full-length F1 comparison with other work. 42

Chapter 1

Introduction

In this chapter, we will give an overview to the research problem, our motivation to do the research and our goals when doing this research.

1.1 Background

In these days, text is one of the most common formats used for exchanging information. There are many different sources of text data such as emails, reports, newspaper, blogs, messages, etc. Millions of these electronic documents are created daily by end-users, reporters or organizations. The amount of information contained in these electronic documents is enormous, hence it poses many challenges to process these documents efficiently. It is too expensive to store and process the entire document content while in many cases, only the main content is needed. Due to this reason, there is a need to extract the most important information from text documents.

Automatic text summarization is a research topic in natural language processing (NLP) which began in the middle of 20th century [1]. Its aim is to extract a summary representing the main content of a single document or a collection of documents. The extracted summary can be used in a variety of ways to solve different problems in NLP like information retrieval (IR) [2, 3], question answering [4] or text classification [5]. For example, an IR system can utilize the extracted summary to quickly retrieve relevant documents instead of checking the entire content. Another example is when comparing documents, we can simply compare their main content to know if they are similar or not. These are some examples of how we can use automatic text summarization for different purposes.

Text summarization systems can be categorized into two groups depending on their input: *single-doc summarization* and *multi-doc summarization*. Single-doc summarization system extracts a summary from a single document meanwhile multi-doc summarization system is able to extract a summary representing the content from multiple relevant input documents. Summarization can also be *query-focused* or *generic*, depending on whether we want to generate a summary related to the user query or a summary for general purpose.

In terms of techniques, we can divide text summarization into two main approaches:

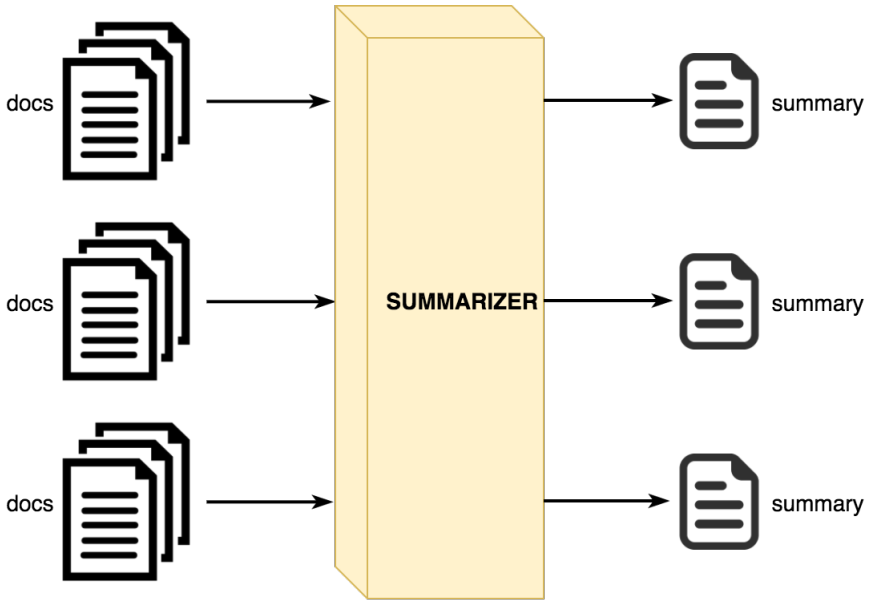


Figure 1.1: Illustration of a summarization system.

extractive and *abstractive* [6]. In the first approach, the system tries to assign a salience score to each sentence in the input documents and then rank these sentences accordingly. The top few sentences will be selected as a summary. The advantage of this approach is that we do not have to deal with the grammar of the generated summary. Each sentence in the extracted summary is assumed to be grammatically correct as long as the input is grammatically correct [7]. In the case of abstractive summarization, we need to analyze the documents and employ an advanced language generation technique for generating the summary, using words or phrases which do not exist in the input documents[1]. This approach is considered to be closer to our human-way of writing the summary [8]. As a human, we do not simply select existing sentences to construct the summary but instead, we probably use our own words or phrases to describe the main content of the text.

1.2 Motivation

Even though the abstraction-based approach is theoretically closer to the way a human writes a summary, its difficulties and complexities make it less interesting to research community over the years. First, this approach requires a deeper analysis on the input documents and advanced techniques for generating the summary. The unsatisfactory performance of available parsing and generation tools can affect the performance of the summarization system badly. Second, evaluation in abstractive summarization is considered to be harder than extractive summarization. In extractive summarization, we can compute the overlapping of the output summary with a gold standard summaries

[1]Criteria to distinguish extractive and abstractive summarization is not clearly defined and controversial.

(written by humans) to evaluate if a system output is good or not. But in abstractive summarization, this is not a good method since there might be different ways to express a certain fact or event. Due to these two main reasons, most of the researchers have focused their effort on extractive summarization and only a few research addressed abstractive summarization.

Nevertheless, there are still several works trying to solve the problem of abstractive summarization. These works are mainly based on *structure-based approaches* (templates, rules, ontology, etc) or *semantic-based approaches* (semantic model, semantic graph)[9]. Recently, with the advance of deep learning, a newer approach for abstractive summarization has been proposed and received a great deal of attention from researchers in this field. Deep learning has achieved great success in many areas like computer vision or speech processing [10, 11]. NLP has also been benefited from this trend with many state-of-the-art results reported in different tasks such as parsing [12], named entity recognition [13] or machine translation [14]. This encouraged researchers to experiment deep learning with new tasks in NLP, including the challenging task of abstractive text summarization. Recent publications show that abstraction-based summarization systems using deep learning can yield promising results.

Despite that, abstractive summarization using deep learning is still in an early stage of development, and there are rooms for improvement. Current summarization models generate understandable summaries but they still contain undesirable grammatical mistakes or redundant words. Also, starting from a standard model used in machine translation, researchers propose new models by adding more features or developing more complex architecture. This comes with the cost of computational complexity. Training a deep learning model is now quite expensive and it can take days, weeks or months to complete training. Using less computational deep learning models while still maintaining the quality of generated summaries is also a tough challenge. In this master research, we would like to study several deep learning architecture for abstractive summarization to get a deeper understanding. In particular, we want to compare the standard neural summarization model with another architecture to learn more about their cons and pros in terms of running time and output quality.

Another important aspect in neural summarization that is worth mentioning is the decoding phase. Currently, most neural summarization models employ a technique called *beam-search* for finding the best summary. Though this method has been used extensively, its default algorithm has a limitation. The K-best list of outputs generated by beam-search are very similar in terms of lexical content. This is an issue if we want to employ a re-ranking model for selecting a good summary. Another issue affecting the quality of summary is the repetition of words or phrases, which has been mentioned in many publications about neural text generation [15, 16, 17, 18]. Therefore, we think that improving the decoding process by tackling these issues is an important and interesting research in abstractive summarization. The solution will be very helpful because it can be integrated into any neural summarization easily without changing the architecture of the whole model.

Our last motivation to do this master research is the lack of publicly accessible imple-

mentation of an end-to-end abstractive summarization system using deep learning. This technical issue might discourage researchers from developing their novel ideas in this field. Many of the published results do not have a corresponding public implementation due to copyright issue or other reasons. Because of that, we want to implement the deep learning summarization model and release its source code to the research community. We believe that another public implementation of an abstractive summarization system will be a great help for other researchers in their research and ease them from implementing a system from scratch.

1.3 Goals

In this research, we limit our research on the *generic single-doc abstractive summarization* problem. Our goals are as follows:

- We would like to implement two deep learning models for summarization: a standard model and a new model with a simpler architecture. We want to run different experiments to compare these two models to get a deeper understanding on them. Our hope is to achieve a competitive performance with a less complex network architecture.

- We want to improve the quality of the summary by addressing different problems in decoding phase. We hope to be able to use a technique to generate diverse beam search and test its capability with a re-ranking model. We also want to propose a method to fix the word repetition issue by directly integrating it into the beam-search algorithm.

- Finally, we would like to implement an end-to-end deep learning system and release the source code to the research community.

Chapter 2

Literature Review

In this chapter, we first present several theoretical frameworks of deep learning related to our research (other types of abstractive summarization models like graph-based or ontology-based are not included in this thesis). Then, we introduce several deep learning models for abstractive summarization which have been proposed in recent years.

2.1 Theoretical frameworks

2.1.1 Encoder-Decoder model

Most of the current approaches to abstractive summarization using deep learning are based on the Encoder-Decoder model. This model works as illustrated in Figure 2.1. The basic work flow of Encoder-Decoder model consists of two steps:

- **Encoding**: In this phase, the model reads the input and uses a neural network to convert it into an intermediate representation, usually a vector. This vector can be considered as a compressed box containing information about the input.
- **Decoding**: From the representation vector, the model uses another neural network to generate the output.

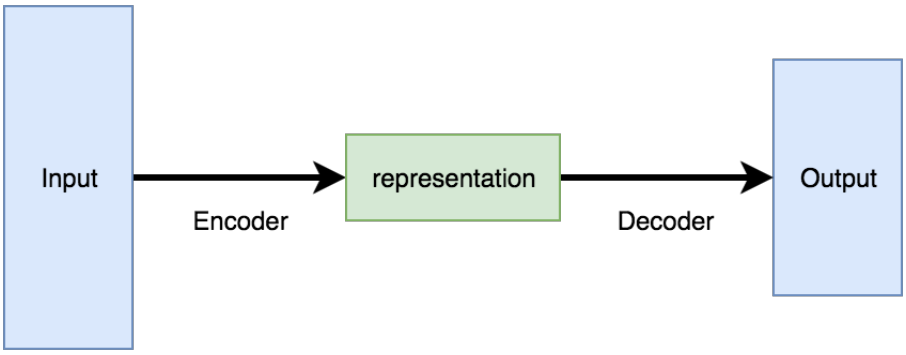


Figure 2.1: Encoder-Decoder model.

The input and the output can be different types of data depending on the task. For example, in the image caption generation task, the input is an image and the output is a sentence. In many cases, both the input and the output are also a sequence of words. That is the reason why in NLP this model is often called *Sequence-to-Sequence* model or *Seq2Seq* for short.

This model has been used successfully in the Machine Translation task [14, 19] and was adopted to text summarization due to their similarity. We can view the input document as a source language and the summary as a target language. The main difference is their length and their vocabulary. For example, machine translation task usually works with a single sentence and their output is also a single sentence. Meanwhile, text summarization task accepts a document consisting of multiple sentences and generates a multi-sentence summary. Additionally, machine translation is one-to-one mapping task (lossless compression) while text summarization is a lossy compression, only the most relevant content is kept.

The main concern when designing this model is how to encode the input and generate the output. In the scope of this research, we introduce two common neural networks for that purpose: Recurrent Neural Network and Convolutional Neural Network.

2.1.2 Recurrent Neural Network

Recurrent Neural Network (RNN) is a neural network which is used to model an input with temporal behavior. In other words, the input is a sequence of items and there are temporal dependencies between two consecutive items. Sentence is a good example of this type of input, it consists of many words placed in a specific order to show a certain meaning.

Given the input sequence $X = \{x_1, x_2, ..., x_n\}$ with $x_t \in \mathbf{R}^d$ is the d-dimension input vector at time step t and n is the number of time steps, RNN works on this input sequence by applying the following formula n times:

$$h_t = \Theta(W_h h_{t-1} + W_x x_t + b) \tag{2.1}$$

In above equation, $h_t \in \mathbf{R}^r$ is called the *hidden state* at time step t with r is the dimension of the hidden vector, $W_h \in \mathbf{R}^{r \times r}$ and $W_x \in \mathbf{R}^{r \times d}$ are parameters of the network, $b \in \mathbf{R}^r$ is bias value and Θ is a non-linear transformation function such as *sigmoid* or *tanh*. As we can see from the formulation, at each time step, the hidden state h_t is computed by using not only the input value x_t but also the hidden state of the previous time step h_{t-1} (the first hidden state h_0 is initialized randomly). This mechanism enables the network to transfer the information from previous time step to current time step. Applying above formula n times will give us a list of n hidden states. We normally use the last hidden state as the representation of the whole sequence because it carries information of all previous time steps. A simple illustration of RNN architecture is shown in Figure 2.2.

In RNN, the component to compute the new hidden state using the current input and previous hidden state is called a *cell*. A typical RNN cell is illustrated in Figure 2.3. As

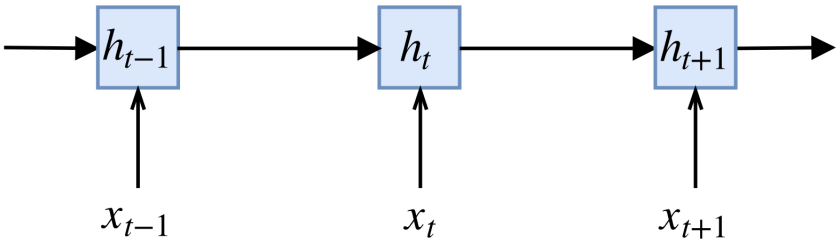


Figure 2.2: RNN architecture.

we show in the next part, the internal computation of a cell can be changed to address certain problems.

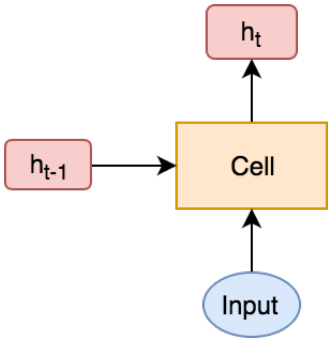


Figure 2.3: An RNN cell.

The property of RNN allows it to work on any sequence regardless of their length. However, as explored by Bengio et al. [20], vanilla RNN suffers from vanishing gradient issue, in which the network is unable to learn much when there are long-term dependencies in the sequence. To deal with this issue, several extensions of RNN cell were proposed such as Long Short-Term Memory or Gated Recurrent Unit.

Long Short-Term Memory

Long Short-Term Memory (LSTM) [21] is an RNN with a special cell. Unlike the normal cell, each LSTM cell consists of 3 gates: input, forget and update. These gates will help the network to learn which information should be kept, forgotten or updated. The

formulation used inside each LSTM cell is as follows[1]:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.2}$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.3}$$
$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{2.4}$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.5}$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.6}$$
$$h_t = o_t * \tanh(C_t) \tag{2.7}$$

with $[h_{t-1}, x_t]$ represents the concatenation of two vectors h_{t-1} and x_t; $W_f \in \mathbf{R}^{r \times (r+d)}$, $W_i \in \mathbf{R}^{r \times (r+d)}$, $W_c \in \mathbb{R}^{r \times (r+d)}$, $W_o \in \mathbf{R}^{r \times (r+d)}$ are parameters in LSTM; b_f, b_i, b_c and b_o are bias values. Using these equations, the LSTM is shown to be better at capturing the long-term dependencies [19, 22].

Gated Recurrent Unit (GRU)

Similar to LSTM, Gated Recurrent Unit (GRU) proposed in [19] also employs the gating mechanism to compute the hidden state at each time step. However, the number of parameters in GRU is less than LSTM because it does not have the output gate. Its formulation is as follow:

$$z_t = \sigma(W_z h_{t-1} + U_z x_t + b_z) \tag{2.8}$$
$$r_t = \sigma(W_r h_{t-1} + U_r x_t + b_r) \tag{2.9}$$
$$\tilde{h}_t = \tanh(W x_t + U(r_t * h_{t-1}) + b_h) \tag{2.10}$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{2.11}$$

GRU has also been shown to achieve performance comparable to LSTM [22]. For simplicity, we denote the operation of the cell to compute new hidden vector from previous hidden as *RNN*. The exact formulation depends on the cell we specify.

Bidirectional RNN

In a typical RNN, each hidden state h_t is computed using the previous hidden state h_{t-1}. Sometimes, however, an input value has dependencies with both previous and next input value. In that case, it makes more sense to capture the information from both directions: left-to-right and right-to-left. Bidirectional RNN (Bi-RNN) [23] was proposed to deal with this problem. It basically consists of two conventional RNN: one *forward RNN* processes the sequence from left to right and another *backward RNN* processes the sequence from right to left. Finally, we concatenate the forward and backward hidden states at each step to construct the final hidden state h_t.

[1]Detailed explanation can be found in the original paper or at the following blog post: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

$$\overrightarrow{h}_t = RNN(\overrightarrow{h}_{t-1}, x_t) \tag{2.12}$$

$$\overleftarrow{h}_t = RNN(\overleftarrow{h}_{t+1}, x_t) \tag{2.13}$$

$$h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t] \tag{2.14}$$

The architecture of Bi-RNN is shown in Figure 2.4. Many researchers have used Bi-RNN in their models and reported good results [14, 24, 25].

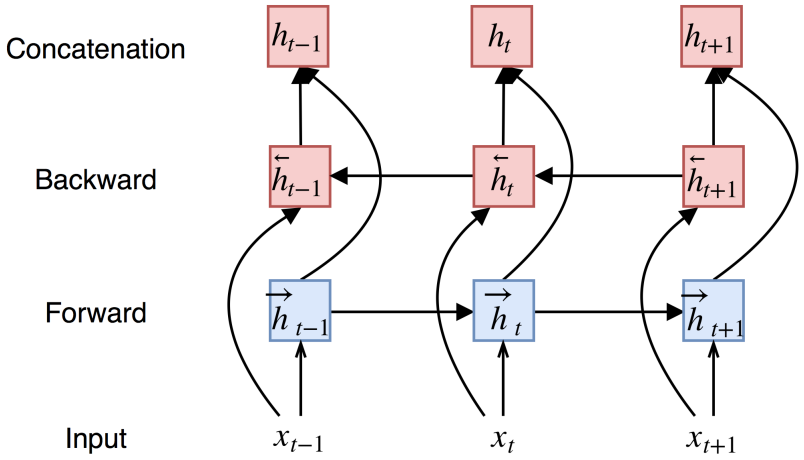


Figure 2.4: Bidirectional RNN.

2.1.3 Convolutional Neural Network

Convolutional Neural Network (CNN) is a feed-forward neural network which uses convolution filters to capture local features [26]. CNN is designed to extract and combine local features of the input, thus it can not only extract many different features but also it can learn the correlation between these features better. A typical CNN has two basic layers:

- **Convolution layer**: CNN uses a matrix called *kernel* or *filter* to map each small sub-region (also known as *receptive field*) in the input matrix into a value. This kernel matrix is replicated at every location of the input data to cover the entire input matrix. The distance when replicating this across the entire input is called *stride*. These replicated units have the same weights and they form a *feature map*. The size of each sub-region is called *kernel size*. Typically, to obtain a richer representation of the input data, people use many kernels with different kernel sizes and multiple feature maps on each kernel, with each feature map represents a certain feature of the input. The number of kernels, kernel sizes, feature maps and strides are varied depending on the problem. This convolution process is illustrated in Figure 2.5.

- **Pooling layer**: The purpose of this layer is to down-sample each feature map to progressively reduce the dimension of the feature matrix and retain only the most

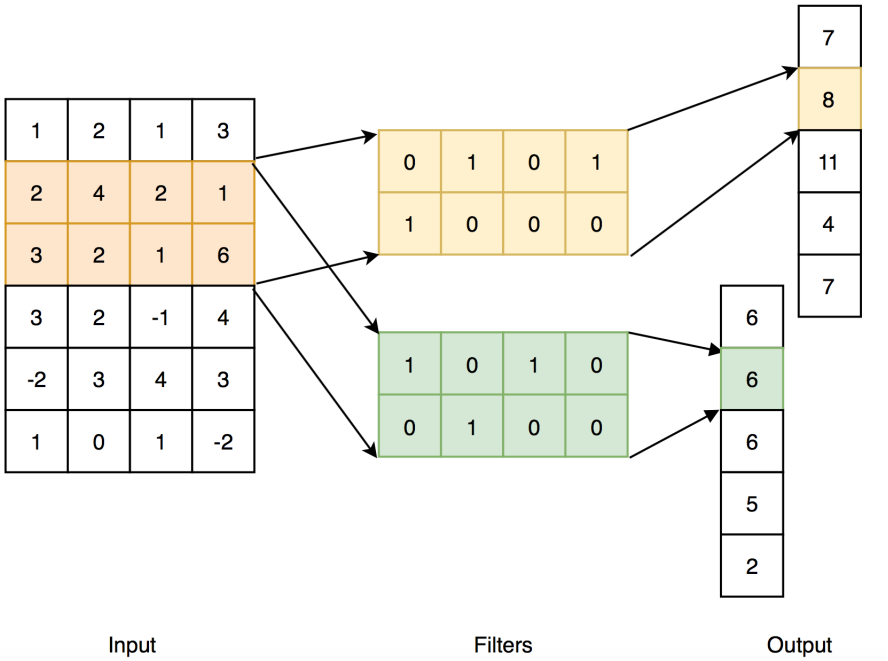


Figure 2.5: Illustration of the convolution layer (kernel size is 2, two feature maps are used).

important values. There are many different types of pooling operations such as *MAX pooling*, *AVERAGE pooling* or *DYNAMIC pooling*. Figure 2.6 illustrates the *MAX pooling* operation for down-sampling the matrix.

Originally, CNN was designed and applied to Image processing. But later it was adapted to use in NLP and many excellent results were published over the years showing the effectiveness of CNN in this field [27, 28, 29]. Even though the input data in NLP like sentence or document do not have the same format as an image, we still can transform and treat them similarly as image data. For example, a sentence can be represented as a 2-dimensional matrix where each row represents a word vector. This matrix is viewed as an image with a single channel and CNN can be applied as usual. Figure 2.7 taken from [29] shows an example of how CNN architecture is used for sentence classification task.

2.1.4 Attention mechanism

In the conventional encoder-decoder model, the decoder only looks at the representation vector provided by the encoder to generate the output. This, however, limits the capability of decoder because it might not have enough information for generating correct output. One popular approach for dealing with this issue is to use the *attention mechanism*. Instead of providing only a single vector to the decoder for generation, we can use additional vectors generated by the encoder to support this decoding process. This idea is based on the intuition that when generating a certain output, we might look at the certain region in the input to get some ideas about what to generate.

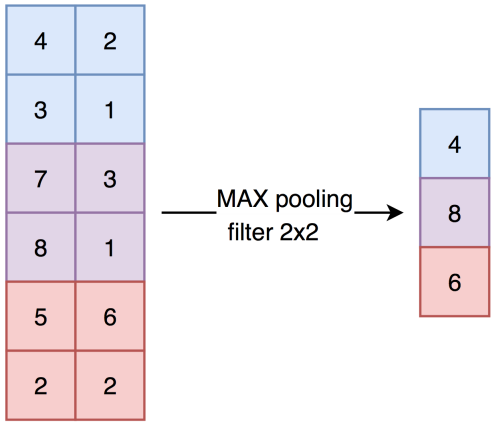


Figure 2.6: Illustration of MAX pooling layer.

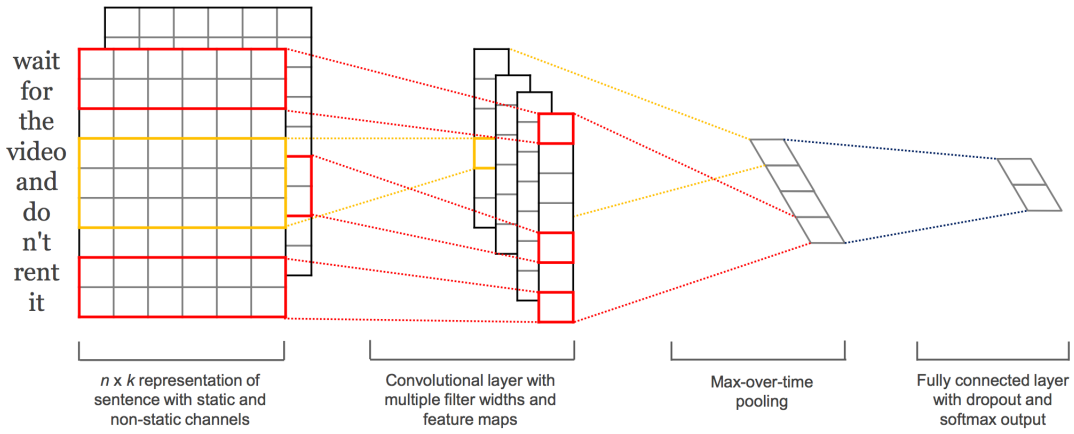


Figure 2.7: Sentence classification using CNN.

In attention-based model, the decoder is given an additional vector called the *context vector* at each step. This vector carries the information of the entire input and it is computed as weighted sum of all encoder states. Given a list of states generated by the encoder $\{h_1, h_2, ..., h_n\}$ and the previous hidden state of the decoder $\hat{h}_{t-1}$, we compute the context vector c_t at this time step as follows:

$$e_{t_i} = \tanh(W_a \hat{h}_{t-1} + U_a h_i) \tag{2.15}$$

$$a_{t_i} = \frac{\exp(e_{t_i})}{\sum_{j=1}^{N} \exp(e_{t_j})} \tag{2.16}$$

$$c_t = \sum_{i=1}^{N} a_{t_i} h_i \tag{2.17}$$

This computation is illustrated in Figure 2.8. After having the context vector c_t, we

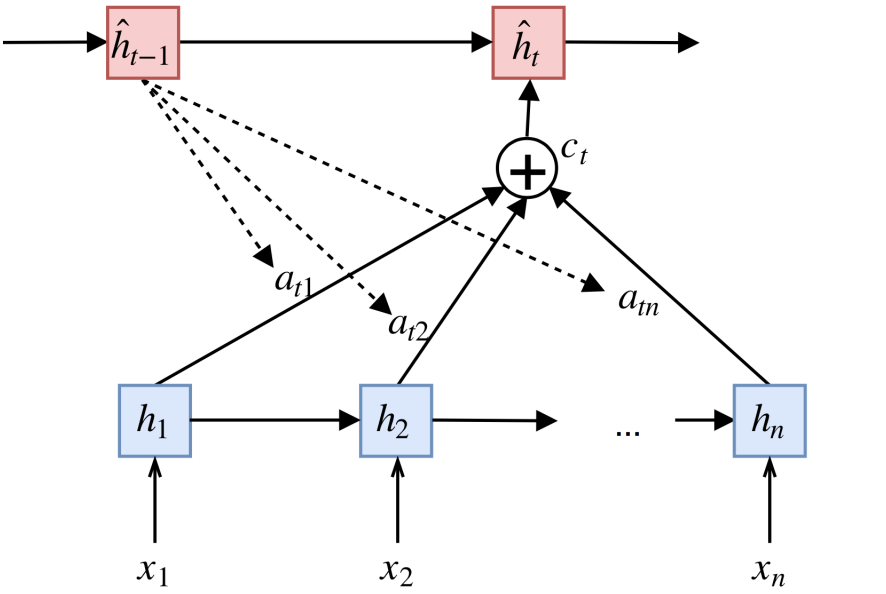


Figure 2.8: Attention mechanism.

combine it with the previous hidden state $\hat{h}_t$ to compute the new hidden state for the decoder.

$$\hat{h}_t = RNN(\hat{h}_{t-1}, c_t, x'_t) \tag{2.18}$$

At the moment, attention mechanism has yielded impressive results in many tasks [30, 31] and it now seems to be used extensively in most models for text generation.

2.2 Related Work on Neural Summarization

With the success of using deep learning in machine translation task, many researchers have attempted to apply similar models for abstractive sentence summarization task. Rush et al. [32] proposed a neural attention model which combined an attention-based contextual encoder with a neural language model for automatically learning the alignment between the input with the generated summary. They trained their model on Gigaword dataset [33] and evaluated it on the DUC-2003 and DUC-2004 dataset [34]. The experimental results showed that their model performed better than the other baselines. Chopra et al [35] later extended Rush's model by using an *attentive encoder* and a conditioned RNN decoder for this task. Their model namely Recurrent Attentive Summarizer outperformed the original model on similar datasets. Takase et al. [36] proposed a model which used the syntactic structure of the sentence for summarization. Specifically, they represented the sentence using Abstract Meaning Representation and used Tree-LSTM for encoding. Their model also got better results comparing to the baseline.

All these systems are different to our research in which they were used for sentence compression while our work is to summarize a full-text document and generating a summary

containing multiple sentences.

Cheng et al. [37] proposed a deep learning model which can be used for summarizing the entire document. The authors used a hierarchical document encoder with an attention-based decoder for extracting sentences and words as a summary. First, CNN was used for constructing the representation vector for each sentence in the document. After that, RNN was applied over the list of sentence vectors to generate the encoder hidden states. At the decoding phrase, their model assigns a label to each sentence by training the network to maximize the likelihood of all sentence labels given the input document vector. This process is illustrated by Figure 2.9 taken from [37]. The author also proposed a word extractor with a hierarchical attention architecture to softly visit all words in the original document and compute the probability of generating summary word-by-word. We consider the word extraction model in this work as a hybrid between extractive and abstractive approach because even though the output sentences are not copied from the original document, the summary only contains words or phrases included in the input document[2]. This model, therefore, is different from our system in this research since we use an open vocabulary for generating each token at each time step. We are interested in the hierarchical architecture the authors used and we would like to apply this into our research.

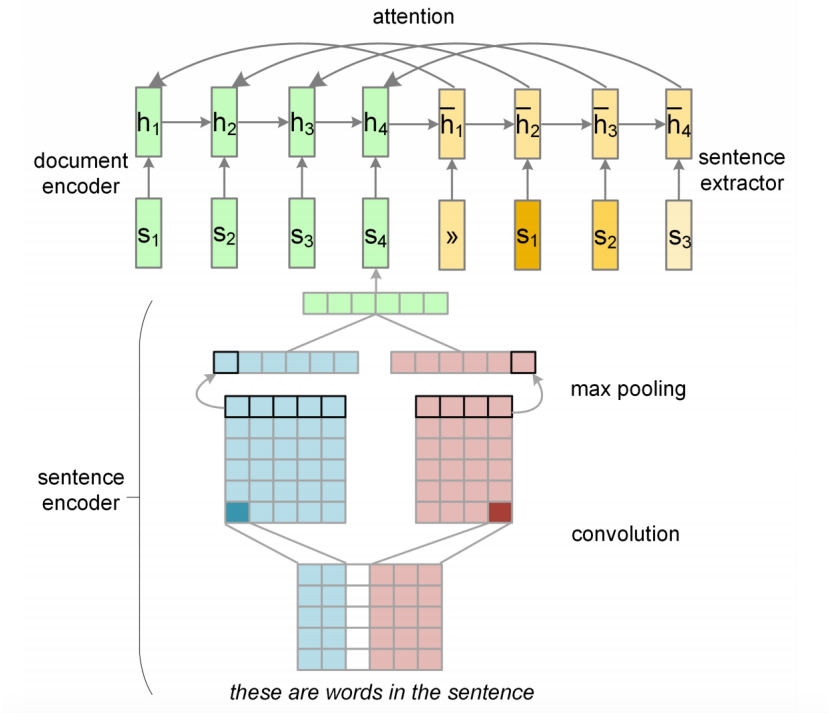


Figure 2.9: A recurrent convolutional document reader with a neural sentence extractor.

An important contribution of [37] is that they provided a free big DailyMail dataset for training an abstractive summarization model. This dataset is important to our research

[2]The authors actually added frequent words to the vocabulary for generation.

because Gigaword dataset is not publicly free at the time we conduct this research. Also, Gigaword dataset is mainly used for training to generate a single sentence while we are interested in generating multiple-sentence summaries. Other datasets such as DUC are for extractive summarization or simply not big enough for training a deep learning system like our research.

The experimental results in this paper show an interesting point. Even though their sentence extractor seems to perform better than other baselines, the word extractors yielded very low score. It was only slightly better than the default neural summarization baseline they implemented[3]. One reason for this result is that in their word extraction model, the grammar of the summary is not enforced due to the vocabulary constraint. Besides, they said that the ROUGE evaluation metric [38] they used was not suitable for measuring the quality of the abstractive summaries.

One of the most recent and complex systems is developed Nallapati et al. [15]. They also treated abstractive text summarization as a sequence-to-sequence problem, but they addressed many problems in it. Starting from the baseline Sequence-to-Sequence model [14], the authors proposed to apply several recent advanced techniques from machine translation to overcome the issues presented in summarization:

- **Large Vocabulary Trick** [39]: they utilized this technique from Machine Translation to reduce the vocabulary size of the decoder and hence save the computation when training. However, they also proposed to expand the vocabulary with 1-nearest-neighbors to reduce the effect of Large Vocabulary Trick.

- **Feature-rich Encoder**: they enriched the input to their model by extending the default word embedding vector with other feature vectors such as term-frequency (TF) and inverted-document-frequency (IDF) vectors, part-of-speech vector, named entity vector.

- **Pointer Network**: when generating the summary, some words might be unknown because they do not exist in the vocabulary. One reason is that they are not common words and the vocabulary size is limited to frequent words only. This is especially true with named entities such as person names, location names, etc. To deal with this issue, the authors proposed a switching generator/pointer architecture to decide whether to copy a word from the original document or to generate a new word.

- **Hierarchical Encoder with Hierarchical Attention**: In this architecture, the author employed two levels of attentions: sentence-level and word-level. The aim of this architecture is to capture the key sentences and keywords at the same time.

- **Temporal Attention**: the authors noticed repetitive words and sentences in the summary, so they utilized *temporal attention* technique [40] to let the decoder remember which part of the document was attended in the previous step. They reported significant improvement when using this technique.

[3]They mentioned that this baseline has similar architecture to their word extraction model except it uses an open vocabulary during decoding.

The author conducted their evaluation on three corpora: Gigaword, DUC and DailyMail. The results show that their models outperformed other baselines on Gigaword and DailyMail datasets, and achieved comparable performance on DUC-2004 dataset. However, their model is expensive to train (hierarchical attention model took more than 12 hours per epoch). In addition, they could not completely solve the repetition issue, especially on the DailyMail dataset.

In this research, we do not aim to build a system which incorporates all these techniques due to our limited time and resources. Instead, we would like to compare the standard sequence-to-sequence neural summarization with a hierarchical encoding model. Furthermore, we also would like to improve summarization quality by focusing on the decoding process. Currently, beam-search decoding is often employed for generating text in neural summarization model. But as explored by Li et al. [41], the issue of beam-search is the lack of diversity on the output. Resolving this issue is important in case we want to re-rank the generated texts and select the best one according to some criteria. Li et al. [41] proposed a very simple technique to force the beam search to generate more diverse outputs at each time step. Vijayakumar et al. [42] went further by dividing the beams into multiple groups and enforcing the diversity between groups when decoding. All these methods are interesting and useful because they help to improve the performance of corresponding systems without re-training the whole model. In our research, we also apply these techniques to solving the single-doc abstractive text summarization problem.

Chapter 3

Summarization Models

In this chapter, we present the models we implemented in our research as well as the techniques we employed for improving the beam-search decoding process.

3.1 Attention-based Sequence-to-Sequence

3.1.1 Encoder

The first model we implemented is inspired from the model proposed by Bahdanau et al. [14], which was used for machine translation task. In this model, we treat the input document as a sequence of words. We use Bi-RNN with GRU cell for encoding the entire content of the document. The last hidden vector is used as the document vector.

In order to map each word in the vocabulary to a vector, we use a *word embedding* layer. This layer has a matrix $W_x \in \mathbf{R}^{d \times V}$ where each column corresponds a word vector with dimension d. Let say we have a word w represents by a one-hot vector $v = [0, 0, ..., 1, 0, ..., 0] \in \mathbf{R}^V$ (all elements in v are 0 except the *i-th* element, which is the index of w in vocabulary list). We convert v into a word vector x as follow:

$$x = W_x v \tag{3.1}$$

Even though we can directly provide the one-hot vector v as an input to the RNN network, it is better to use the embedding layer. The first reason is that we cannot capture the semantic relationship between words using one-hot vectors (like synonyms, antonyms). Secondly, we want the network to be able to adjust the word vectors during its training to find the best word representation for our problem.

3.1.2 Decoder

The decoder is also an RNN with GRU cell, it has quite similar formulation to the encoder:

$$x'_t = W_x w_t \tag{3.2}$$
$$\hat{h}_t = RNN(\hat{h}_{t-1}, x'_t, c_t) \tag{3.3}$$
$$y_t = softmax(\hat{h}_t W_y + b_y) \tag{3.4}$$
$$\tag{3.5}$$

with:

- $w'_t \in \mathbf{R}^V$: a one-hot vector represents the input word at time step t.
- $W_x \in \mathbf{R}^{d \times V}$: embedding matrix.
- $x'_t \in \mathbf{R}^d$: decoder input at time step t.
- $c_t \in \mathbf{R}^r$: context vector computed by the attention mechanism from the hidden states of each words in the input document. r is the dimension of the hidden state vector from the encoder.
- $W_y \in \mathbf{R}^{r \times V}$: output embedding matrix for transforming the hidden vector $\hat{h}_t$ to the V-dimension output vector.
- $y_t \in \mathbf{R}^V$: predicted probability vector, each component in this vector is computed using *softmax* function as follow:

$$softmax(o_{t_i}) = \frac{\exp(o_{t_i})}{\sum_{j=1}^{V} \exp(o_{t_j})} \tag{3.6}$$

From predicted vector $\hat{y}_t$, we can find the token by getting the index of the component with the highest probability:

$$token_t = argmax(\hat{y}_t) \tag{3.7}$$

The work flow of predicting a token from the hidden vector is described graphically as in Figure 3.1. Noted that even though we can infer the highest probability without going through the *softmax* layer, we need to use *softmax* function in order to get a probability vector $\hat{y}_t$ and then use this vector for computing the loss function.

The initial hidden state of the decoder is set to the last hidden state of the encoder and the first input token is a special token representing the start of the output sequence (denoted as *GO* token). There is a difference between the decoder inputs for training phase and testing phase. During the training phase, we assume that the previously decoded token is correct and therefore use the gold token to feed into the decoder. But when we run the model for generating the summary on an actual document, the gold tokens are not available, so we instead use previously decoded token as the input to the

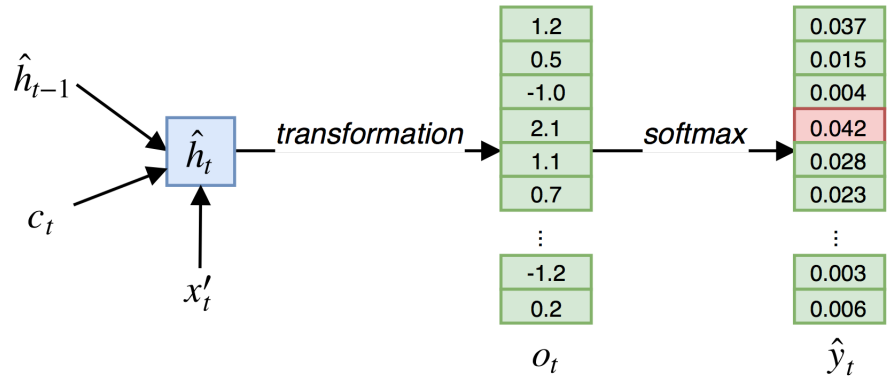


Figure 3.1: The process of decoding at time step t. The predicted token is the index with highest probability.

next step. This is illustrated in Figure 3.2. The decoding process continues until we reach the End-Of-Document (EOD) token or we reach the maximum number of pre-defined time steps.

The goal of this model is to maximize the log probability of the generated summary $\hat{\mathbf{Y}} = \{\hat{y}_1, \hat{y}_2, ..., \hat{y}_M\}$ conditioned on the input document $\mathbf{X}$

$$\log p(\hat{\mathbf{Y}}|\mathbf{X}) = \sum_{i=1}^{M} \log p(\hat{y}_i|\mathbf{X}, \hat{\mathbf{Y}}_{1:i-1}) \tag{3.8}$$

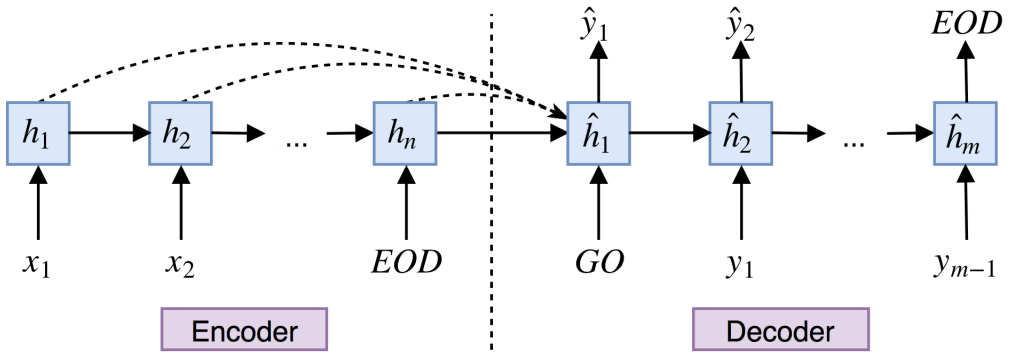
This is achieved by training the model to generate a sequence of words as similar as possible to the gold summary. The loss function is defined as the average categorical cross entropy between the gold sequence and the predicted sequence

$$H(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{M} \sum_{i=1}^{M} H(y_i, \hat{y}_i) \tag{3.9}$$

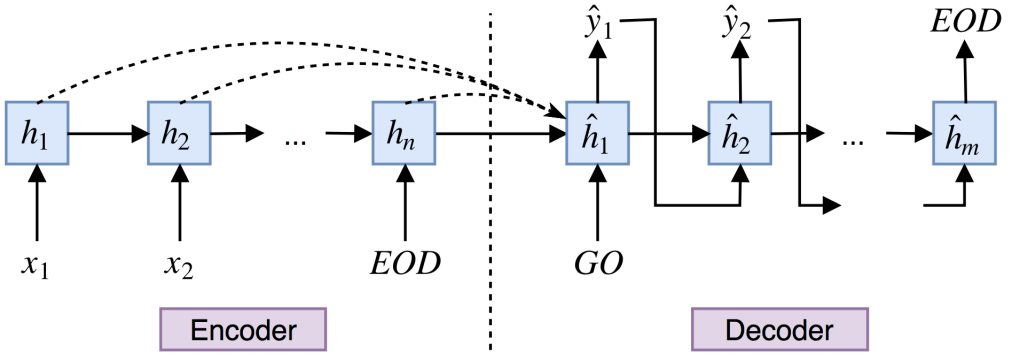
$$H(y, \hat{y}) = -\sum_{i=1}^{V} y[i] \log \hat{y}[i] \tag{3.10}$$

with $y \in \mathbf{R}^V$ is a one-hot vector representing the gold token, $y[i]$ represents the *i-th* component of the vector y. Assuming that the gold token is the *m-th* word in the vocabulary, the cross entropy between y and $\hat{y}$ can be written as follow:

$$H(y, \hat{y}) = -\sum_{i=1}^{V} y[i] \log \hat{y}[i] = -\log \hat{y}[m] \tag{3.11}$$



(a) Decoder inputs are gold tokens when training.



(b) Decoder inputs are from previously decoded tokens when testing.

Figure 3.2: Difference between decoder input when training and testing.

The most expensive computation in the decoder is the softmax layer since we need to multiply $\hat{h}_t$ with every column in the matrix W_y. In text generation, this vocabulary can be quite big and thus it dramatically slows down the whole system when training. There are several techniques for dealing with this issue such as hierarchical softmax [43], differentiated softmax [44], etc. In this research, we employ a technique called *sampled softmax* [39] to reduce the computational complexity. In particular, for each time step, we construct a smaller decoder vocabulary by sampling a subset S_t of the large vocabulary L, and then compute the probability that our predicted token is equal to the true token in the smaller vocab S_t[1]. This sampled softmax technique is applied at training phase only, the inference at testing time still uses full softmax as usual.

3.1.3 Beam-search decoding

After training the model, the decoding process to generate the summary is, in fact, a search problem. At each time step t, the model outputs a list of words with their corresponding probabilities. Our job is to find the best summary by selecting good candidates from the first step to the final step. The easiest way is to employ a *greedy search* in which

[1] S_t is constructed in a way that it always contains the true token. More details can be found at: `https://www.tensorflow.org/extras/candidate_sampling.pdf`

we simply select the word with the highest probability at each time step and discard all other words. This, however, might prevent the best output to be found later in the search process. A better and widely used search strategy in this problem is *beam-search*. It works by keeping a fixed number of candidates at each time steps to increase the chance to discover a better summary. The beam-search decoding process is illustrated in Figure 3.3. With K beams at time step t, we expand each beam to select new K candidates and get $K \times K$ beams after this expansion. We then assign a score to each beam as follow:

$$score(B_i) = \frac{1}{M} \sum_{t=1}^{M} \log p(w_t | \mathbf{X}, W_{t-1}) \tag{3.12}$$

with:

- B_i: the *i-th* beam in our decoding process.
- M: number of words in this beam.
- $\mathbf{X}$: the input document.
- w_i: the predicted token at time step t.
- $W_{t-1} = \{w_1, w_2, ..., w_{t-1}\}$: all words previously predicted in the beam.
- $\log p(w_t | \mathbf{X}, W_{t-1})$: the log probability of new predicted word given its history and input document. This value is given by the model.

We sort these $K \times K$ beams by their score and keep only top K beams for the next step. When an *EOD* token is generated on a beam, we add that beam to the *result list*. We repeat the search process until the number of results is equal to the beam size or when we reach the maximum length of the output. Setting $K = 1$ is equivalent to greedy search while setting K too big will slow down the decoding process.

3.2 Hierarchical Encoding

RNN is effective at capturing the information of a sequence but the longer the sequence gets, the slower the network becomes. This is because we have dependencies between time steps in the inputs, so the computation needs to be performed sequentially. More specifically, to calculate the output values, RNN needs to compute the output value starting from time step $t = 1$ first and repeat its calculation until it reaches the final time step $t = T$. Additionally, to update the weights in the network, it has to apply a technique called *backpropagation through time (BPTT)*, in which we have to backpropagate the gradients from the last time step $t = T$ to the first time step $t = 1$. In our summarization problem, a document can have up to a thousand words this can result in a long training time.

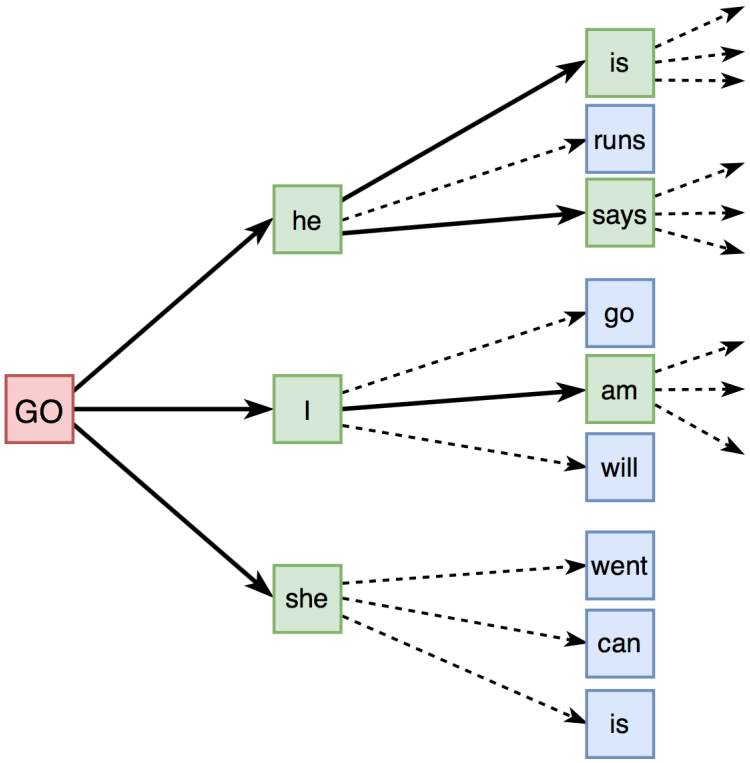


Figure 3.3: Illustration of beam-search decoding with $K = 3$.

To reduce the computational time of this process, we can decrease the number of time steps performed by the encoder. In this case, we treat a document as a hierarchical structure, each document D has n_s sentences $\{s_1, s_2, ..., s_{n_s}\}$ and each sentence s_i has k words $\{x_1^i, x_2^i, ..., x_k^i\}$. We first encode each sentence into a vector v_{s_i} using CNN:

$$v_{s_i} = CNN(x_1^i, x_2^i, ..., x_k^i) \tag{3.13}$$

and then we feed this sentence vector into RNN to compute the hidden state:

$$h_t = RNN(h_{t-1}, v_{s_t}) \tag{3.14}$$

Using this encoding method, we can significantly reduce the number of time steps in the RNN and hence expect to get a faster training time. CNN is chosen to encode a sentence because it has been shown to perform pretty well in some sentence-level tasks [29, 45] while its computation is not too expensive due to its architecture (it is a feed-forward network with fewer parameters than a fully connected network).

We illustrate this hierarchical model in Figure 3.4. This design is very similar to the model proposed by Cheng et al. [37]. However, we use this model for generating summary using an open vocabulary instead of extracting words or sentences. The decoder is the same as the default model presented in Section 3.1. The only difference is when computing the context vector c_t, we use the hidden states calculated from the sentence vectors.

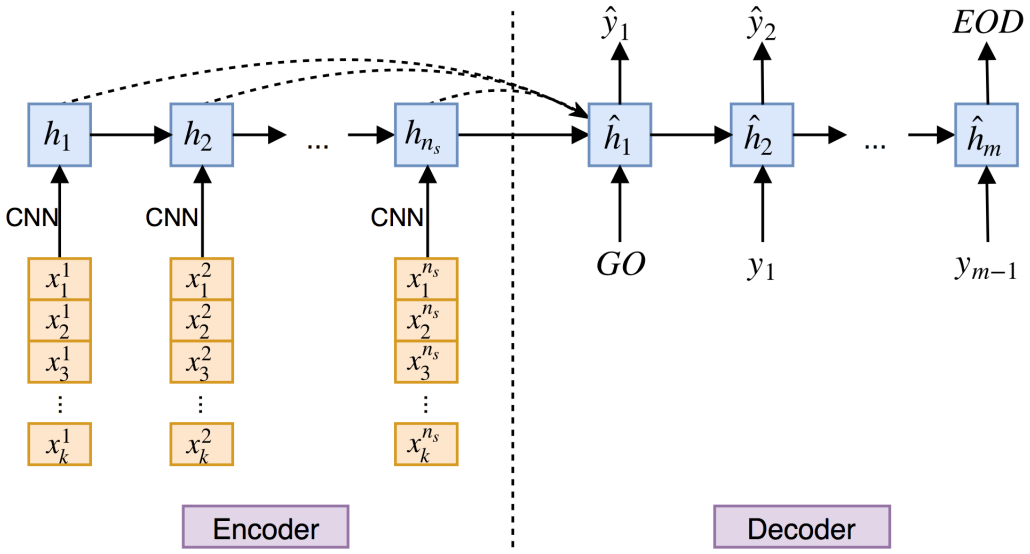


Figure 3.4: Hierarchical encoding for summarization

Beside this new hierarchical encoding architecture, we also perform two additional experiments to study the abstractive summarization model. In particular, we are interested in seeing how the input features can affect the performance of the system. In [15], Ramesh et. al. incorporated many additional features besides the word embedding to help the model capture the key concepts and entities. However, we limit our study into two aspects due to limited time and resources:

1. We enrich the input features by providing the Part-of-speech (POS) of each word in addition to the default word vector. We build a POS vocabulary[2] similar to the word vocabulary and map each POS into a vector using the POS embedding matrix. Word vector and POS vector are concatenated before feeding into the network. We hope that the network can learn better using this information.

2. We discard stop words from the input document. We use a list of top 25 stop words common in Reuters-RCV1[3]. In our view, stop words are redundant information and it might not be needed in a summarization model.

We could have performed these experiments on the standard attention-based Seq2Seq, but due to the expensive training time, we decided to conduct these two experiments on the hierarchical encoding model.

[2]We use the POS list from `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html` and some additional special POS tokens

[3]https://nlp.stanford.edu/IR-book/html/htmledition/dropping-common-terms-stop-words-1.html

3.3 Diverse beam-search decoding with Re-ranking

In beam-search decoding, the beam outputs are ranked using the score computed from the log probabilities which generated by the model. But these log probabilities only tell us how good our model is at learning to minimize the loss function, they do not guarantee to generate a good summary according to our custom evaluation, such as the similarity with the input document or the grammar quality. A method to deal with this issue is to employ a re-ranker to select the best summary from all the beam outputs. A good re-ranker can be designed to prefer summaries which are highly relevant to the input document or simply have better grammar. Utilizing a re-ranking is also an easy method to improve the performance of a summarization system without modifying the training model, which is an expensive process.

There are two important factors affecting the performance of the re-ranking process: the difference between candidates and the quality of each candidate. If most candidates are similar, there is not much chance for the re-ranker to select a best one. While if the quality of each candidate is bad, the final output after re-ranking is expected to be bad as well. Unfortunately, our summarization model possesses both of these issues. We would like to describe these issues in greater details and how we tackle them by modifying the beam-search decoding process.

3.3.1 Diversity between beams

Beam-search decoding has been known to have the *lack of diversity* issue [41]. It means that most of the generated output are lexically similar and they are different at only some small parts of the text, such as punctuation. This makes the re-ranker have fewer options to rank and therefore decrease the chance to find the best output from the candidate list. A solution to this problem is to force the beam-search decoder to generate more diverse outputs during its generation process. One of the common methods is to modify the log probability of each predicted words before expanding a beam to prevent similar words to be selected in different beams.

In this research, we use a technique proposed by Vijayakumar et al. [42] for enforcing the diversity between beams. The authors divide K beams into G groups and control the diversity between these groups while expanding each beam. They modify the log probability of each predicted word on group G_i (except the first group G_1) as follow:

$$score(w_t^i|\mathbf{X}, W_{t-1}^i) = \log p(w_t^i|\mathbf{X}, W_{t-1}^i) + \lambda\Delta(P_{i-1}) \tag{3.15}$$

with:

- w_t^i: candidate word on group G_i at time step t.
- P_{i-1}: list of the last generated tokens on previous groups G_1, G_2, ..., G_{i-1}.
- $\Delta(P_{i-1})$: the dissimilarity of current group with previous groups when token w_t is selected. The choice of this function can be varied.

- λ: the diversity factor.

This algorithm is illustrated in Figure 3.5 taken from [42]. Because the score in the first group is not modified, their algorithm performs at least as good as the K/G-size beam search. The final outputs are collected from each groups.

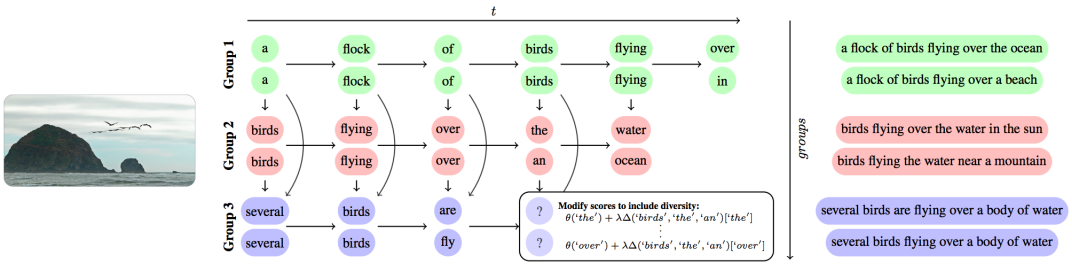


Figure 3.5: Diversity by grouping beams.

As reported by the authors on several tasks such as image captioning, machine translation and visual question generation, this method improves both the diversity of the output and the performance of the system. We, therefore, expect to see an improvement in our system as well because of its similarity with machine translation task. Regarding the dissimilarity term $\Delta(\cdot)$, we use *Hamming Diversity* due to its simplicity and efficiency[4].

3.3.2 Diversity within a beam

Even though the technique described in Section 3.3.1 enforces the diversity between different beams, it does not take into account the diversity within each beam. In other words, the output of each beam can also contain repetitive words and phrases. This issue is quite common in the sequence-to-sequence model for neural generation task [15, 16, 17, 18]. This is especially severe in our summarization problem since the output is pretty long and contains multiple sentences. There have been several attempts to deal with this issue. For example, Sakaran et al. [40] tackled this issue in machine translation by incorporate a *temporal attention* into the decoder to let it memorize the information which was used for generation. Suzuki et al. [17] proposed a method to jointly estimate the frequency and control the output words by their estimated frequency. These methods need to be integrated into the model during the training architecture, and it could potentially increase the training time significantly [15]. We instead would like to tackle this issue without changing the training architecture and we prefer a technique which can be applied to any pre-trained models using beam-search for text generation. Therefore, we propose to add a simple update in beam-search to avoid word redundancy issue in summarization task.

Our main idea is to utilize the previously decoded tokens on each beam to guide its expansion at each step. More specifically, the word which is likely to cause the issue of repetition should have lower probability than other words. Following this idea, we modify the log probability of each candidate word in a way that it discourages previously decoded

[4]The authors also reported the best oracle performance with Hamming Diversity in their paper [42]

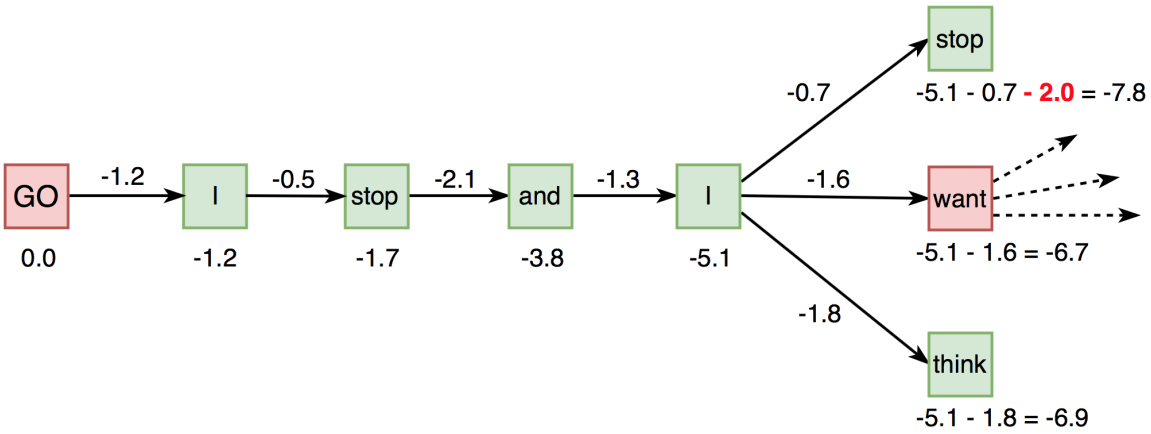


Figure 3.6: Illustration of beam-search with local diversity.

words from being chosen in beam search. The formulation for adjusting the score of each candidate word is as follow:

$$score(w_t|\mathbf{X}, W_{t-1}) = \log p(w_t|\mathbf{X}, W_{t-1}) - \beta\Theta(w_t, W_{t-1}) \tag{3.16}$$

with β is called *local diversity factor* which controls how much you want to avoid repetition. A small β is not very helpful while big β might force the beam search to generate incoherent text (as the very low log probability word might be selected). $\Theta(\cdot)$ is a function which returns a value indicating the *degree of repetition* when w_t is selected for this beam. We define $\Theta(\cdot)$ as follow:

$$\begin{aligned}\Theta(w_t, W_{t-1}) = &|\{x \in unigrams(W_{t-1})|x = w_t\}| \\ &+ |\{y \in bigrams(W_{t-1})|y = [w_{t-1}, w_t]\}|\end{aligned} \tag{3.17}$$

In other words, $\Theta(\cdot)$ is sum of the frequency of the unigram w_t and the bigram $w_{t-1}w_t$ in the decoded sequence $W_{t-1} = \{w_1, w_2, ..., w_{t-1}\}$ of the beam. This process is illustrated in Figure 3.6. In that figure, *"stop"* is supposed to be chosen but we decrease its score since the unigram *"stop"* and bigram *"I stop"* are already in the beam. Thus, *"want"* becomes the chosen token for this beam. A better $\Theta(\cdot)$ function can be studied in future work. In our research, we combine both the diverse group beam-search decoding in Section 3.5 and this technique to produce multiple diverse output, each output is likely to contain less repetitive words.

3.3.3 Re-ranking

Since we are interested in showing the effectiveness of our diverse beam-search decoding, designing a sophisticated re-ranker is not a focus of our research. Thus, we develop a simple re-ranker which ranks the generated summaries based on their similarity with the first 5 sentences in the input document. Similarity value is the Jaccard Index between two sets:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.18}$$

with A and B is the set of n-grams of the generated summary and first 5 sentences respectively. In this research, we use only unigrams and bigrams for comparison.

We also design an Oracle re-ranker in which the summary candidates are ranked based on their similarity to the gold summary. This is just to show how much we can achieve if we have a good re-ranker. In practice, we need to analyze the problem as well as the input document to choose an appropriate re-ranker. There are already many research about this topic such as linear re-ranker or learning-to-rank. We leave this as future work.

Chapter 4

Evaluation

4.1 Dataset and Preprocessing

The dataset we used in this research is provided by Cheng et. el. [37][1]. It contains articles collected from two major news media: DailyMail[2] and CNN[3]. Each article includes the content of the article plus highlights written by a human. The statistics of this dataset are shown in Table 4.1. For training the model, we used all 277,554 articles from both sources, but for validation and testing, we only used a subset of the total dataset to reduce the running time. The numbers of documents for validation and testing are 3,000 and 500 respectively.

Table 4.1: Number of documents in the experimental dataset.

	Training	**Test**	**Validation**
DailyMail	193986	10350	12147
CNN	83568	1093	1220
Total	277554	11443	13367

In each article, all named entities are replaced with anonymous entities in order to reduce the vocabulary size. An example is shown in Figure 4.1. It can also be seen from the example that similar entities (*Michelle MacLaren* and *MacLaren*) are mapped to a single key (*@entity5*). Each sentence is padded to the same length if their word length is smaller than a certain threshold. If their word lengths exceed the threshold, they will be truncated. We also append *end_of_sentence (EOS)* token to every sentence. The *end_of_document (EOD)* token is appended at the end of the last sentence to mark the end of the document. We also replace all numeric tokens with a single <DIGIT> token. Any words that are not included in the vocabulary are set to <UNK> token.

[1] http://homepages.inf.ed.ac.uk/s1537177/resources.html
[2] http://www.dailymail.co.uk/
[3] http://edition.cnn.com/

Michelle MacLaren is no longer set to direct the first "Wonder Woman" theatrical movie
MacLaren left the project over "creative differences"
Movie is currently set for 2017

Original text

@entity5 is no *longer* set to direct the first " @entity9 " theatrical movie
@entity5 left the project over " creative differences "
movie is currently set for 2017

Text with anonymous entities

@entity5:MacLaren
@entity9:Wonder Woman

Metadata

Figure 4.1: A sample document with anonymous entities.

4.2 Experimental settings

For word embedding layer, we initialized the matrix using the values provided by a *word2vec* model [46] trained on this DailyMail dataset and we let the network continue to update this matrix during the training phase. All models were trained with Adam optimizer, an adaptive method for stochastic optimization [47]. The encoder vocabulary and decoder vocabulary were limited to 150,000 and 100,000 words respectively. All special tokens like <DIGIT>, <GO>, <PAD>, <EOD> were initialized with uniform distribution in range [-0.5, 0.5].

We did not extensively tune all parameters due to time constraint. Instead, we used the values reported by other relevant work and chose some parameters heuristically. Table 4.2 shows our network configuration for training. CNN parameters were chosen similar to the model in [48]. Based on the statistics provided in [15], we fixed the maximum number of sentences to 35, the maximum number of words per sentence to 50 and the maximum number of words per document to 800. The model was trained to generated maximum 100 words. We also detected the *EOD* token in the generated summary and truncated it to a proper length.

At the beginning of each epoch, we randomly shuffle the entire training examples and we also shuffle the examples inside each batch. This is a small trick to prevent the model from over-fitting because of repetitive input data. All the models were implemented in Python with Tensorflow library[4].

Training deep learning model like this requires a fast and powerful computer with GPU support. We took advantage of the GPU-powered supercomputer at JAIST for running the experiments. This server is equipped with Tesla K40M GPU and our computational time is reduced more than 10 folds compared to when it is on CPU. Nonetheless, it still takes a long time to finish a single epoch. In our experiment, we adopted early stopping

[4]https://www.tensorflow.org/

Table 4.2: Experimental settings.

Batch size	32
Word dimension	100
Part-of-speech dimension	30
Hidden size	256
Learning rate	0.001
Sampled softmax classes	4096
CNN kernels	[1,2,3,4,5,6,7]
CNN feature maps	[50, 100, 150, 200, 200, 200, 200]
Max number of sentence	35
Max sentence length	50
Max input sequence length	800
Max output sequence length	100

strategy to avoid over-fitting. We also handled the cases where the loss of our model might go up a little bit before going down by using *patience* value, i.e. how many epochs we should continue when the loss goes up. Patience is set to 2. Early stopping was only applied when each model had already finished their first 20 epochs. This is to prevent a model from stopping too early in their training due to a jump in validation loss.

4.3 Results

For evaluating the quality of a summarization system, we employ ROUGE score [38], the most common method for checking the performance of summarization systems. Even though ROUGE is mainly used for evaluating extractive summarization systems, it is still widely used in abstractive summarization because of its simplicity. Other methods like Pyramid evaluation [49] requires human annotation, which is not available in our case. ROUGE package computes the Precision, Recall and F1 of the generated summary by comparing it with the gold summaries. In this case, gold summaries are the highlights in each document. Because the generated summaries have variable length, we ran ROUGE to compute full-length F1 score[5] as well as first-75-byte F1 score[6]. This is similar to how the authors did in [37] and [15]. We report ROUGE-1, ROUGE-2 and ROUGE-L for each run.

We performed the following experiments to compare the performance of all systems:

- Experiment with standard beam-search decoding: we fixed the beam width and compare the performance of different models in term of ROUGE score and their computational time.

[5]Full command: `ROUGE-1.5.5.pl -m -n 2`.

[6]with `-b 75` option.

- Experiment with varying beam width: we evaluated each system by changing the beam width and observed the performance of each system.
- Experiment with diverse beam-search decoding and reranking.

To make it easier for presenting the results, we use the following abbreviations:

- *AttSeq2Seq*: the basic attention-based Seq2Seq model described in Section 3.1.
- *HierEnc*: the hierarchical encoding model described in Section 3.4.
- *HierEnc+POS*: the hierarchical encoding model with additional POS information.
- *HierEnc-SW*: the hierarchical encoding model which removes stop words from the input document.

4.3.1 Experiment 1: Decoding with standard beam-search

In this experiment, we fixed beam size to 2 on the decoder and observed the differences between models. In Table 4.3, we report the number of epochs, average training time per epoch and the total decoding time on 500 test files.

Table 4.3: Number of epochs, average training time and decoding time on each models.

	#epochs	**Average training time (hours/epoch)**	**Decoding time (seconds)**
AttnSeq2Seq	24	10.4	342
HierEnc	23	4.8	82
HierEnc+POS	24	6.4	157
HierEnc-SW	25	4.4	121

As we can see from Table 4.3, *AttnSeq2Seq* model took the longest time to finish an epoch, almost double the time required by other models. This is totally in accordance with our expectation when designing the hierarchical encoder. The *HierEnc+POS* model is a little bit slower than *HierEnc* because we increase the number of parameters in the input layer when we add POS information. Removing stop words saved us only a little bit of training time. For decoding, *HierEnc* is the fastest model while *AttnSeq2Seq* is the slowest one. We can also see that *HierEnc-SW* is slower than the *HierEnc* when decoding even though its training time is smaller. This is not caused by the network but because of our process to remove stop words before decoding[7].

The ROUGE score results are presented in Table 4.4. In both evaluation (full-length and first-75b), the *AttnSeq2Seq* achieves the best result in ROUGE-2 score. Other models with hierarchical encoder yield surprisingly low ROUGE-2 scores, but their full-length

[7]When training, we use multiple threads running in background for processing the input before feeding into the network so there almost no input latency.

Table 4.4: Performance comparison of different models with fixed beam size K = 2.

	Full-length F1		
	ROUGE-1	**ROUGE-2**	**ROUGE-L**
AttnSeq2Seq	21.3	**7.0**	19.1
HierEnc	21.8	3.9	19.9
HierEnc+POS	**23.5**	3.9	**21.4**
HierEnc-SW	21.5	3.7	19.8

	First 75b F1		
	ROUGE-1	**ROUGE-2**	**ROUGE-L**
AttnSeq2Seq	**20.9**	**6.6**	**14.6**
HierEnc	20.2	3.6	13.4
HierEnc+POS	**20.9**	3.7	13.7
HierEnc-SW	16.8	2.8	11.3

ROUGE-1 and ROUGE-L are competitive with *AttnSeq2Seq*. This tells us that *HierEnc* model generates summaries with more words appearing in the gold summary, but their bigrams are less relevant than the *AttnSeq2Seq* model. If we view ROUGE-2 as the most important metric for evaluating a summarization system like other works [50], we can say that the performance of *HierEnc* is much worse than the standard architecture.

It can also be seen that providing POS information to the encoder does help to improve the performance of the summarization model. Removing stop words, on the other hand, makes the system worse and this is very obvious when evaluating with first 75 bytes of the summaries. Another thing we notice is that *AttnSeq2Seq* yields the best result on first-75b F1 score in all metrics. This means that the standard attention-based model indeed produces the first few words (or the first sentence) more relevant to the gold summary than other models.

We present several summaries generated by different models from an input document in Figure 4.2. As we can see, all models are capable of producing summaries relevant to the gold summary. However, *AttnSeq2Seq* seems to perform a bit better since its first and second sentences are quite good. But it is also obvious that all models have issues with repetitive words/sentences (except *HierEnc* in this case because it produces only one sentence). *HierEnc+POS* even generates two duplicate sentences.

4.3.2 Experiment 2: The effect of beam size

As we use beam-search for decoding, it is interesting to see how changing the beam size affects the performance of each system. We took the previously trained models of each system and ran the decoding with varying beam size: 2, 5, 10 and 20. Our result is reported in Table 4.5.

We can clearly see very different trends in these models. The *AttnSeq2Seq* benefits the most from bigger beam size while performance of other models gets worse. We believe

Gold summary:
@entity8 boss @entity25 has scouted @entity1
@entity6 forward @entity1 is considering his future at the @entity3
@entity9 could leave @entity8 after rejecting Â£ 100,000 - a - week deal
@entity15 , @entity22 and @entity21 are keen on the @entity8 star
@entity89 : the real problem with @entity9 contract saga at @entity8
read : @entity23 are *monitoring* @entity9 , *reveals* @entity94
click here for all the latest @entity8 news

AttnSeq2Seq:
@entity6 forward @entity1 decision to consider @entity3 future at end of season
@entity11 could see him end up at @entity7 if @entity8 sell @entity9
@entity11 forward @entity1 's decision to consider his future at the end of the season

---

HierEnc:
@entity1 's contract expires at the end of the season

HierEnc+POS:
@entity1 has been linked with a loan move to @entity6
the <DIGIT> - year - old has been linked with a move to @entity6
the <DIGIT> - year - old has been linked with a move to @entity6
click here for more *transfer* news

HierEnc-SW:
@entity1 has already sold more than £ 8million
the <DIGIT> - year - old has already sold more than £ 4m for the club

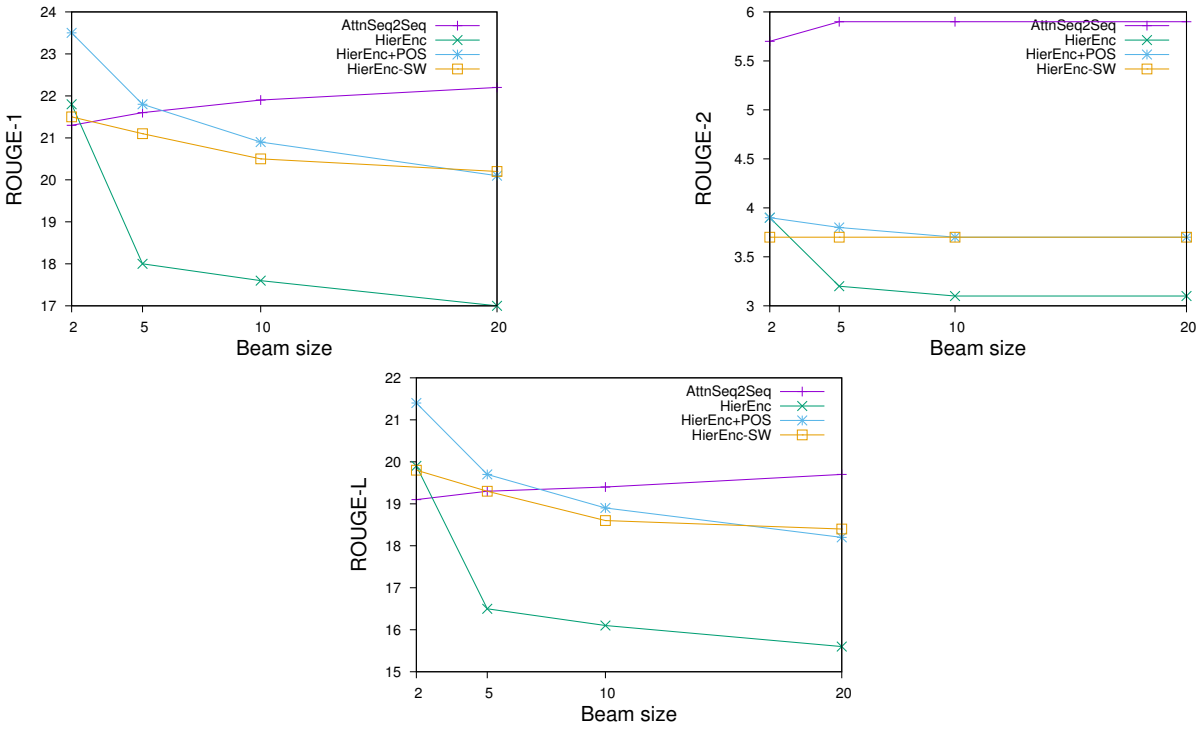
Figure 4.2: Outputs from different models when we fixed beam size K = 2.

Table 4.5: Performance comparison with different beam sizes.

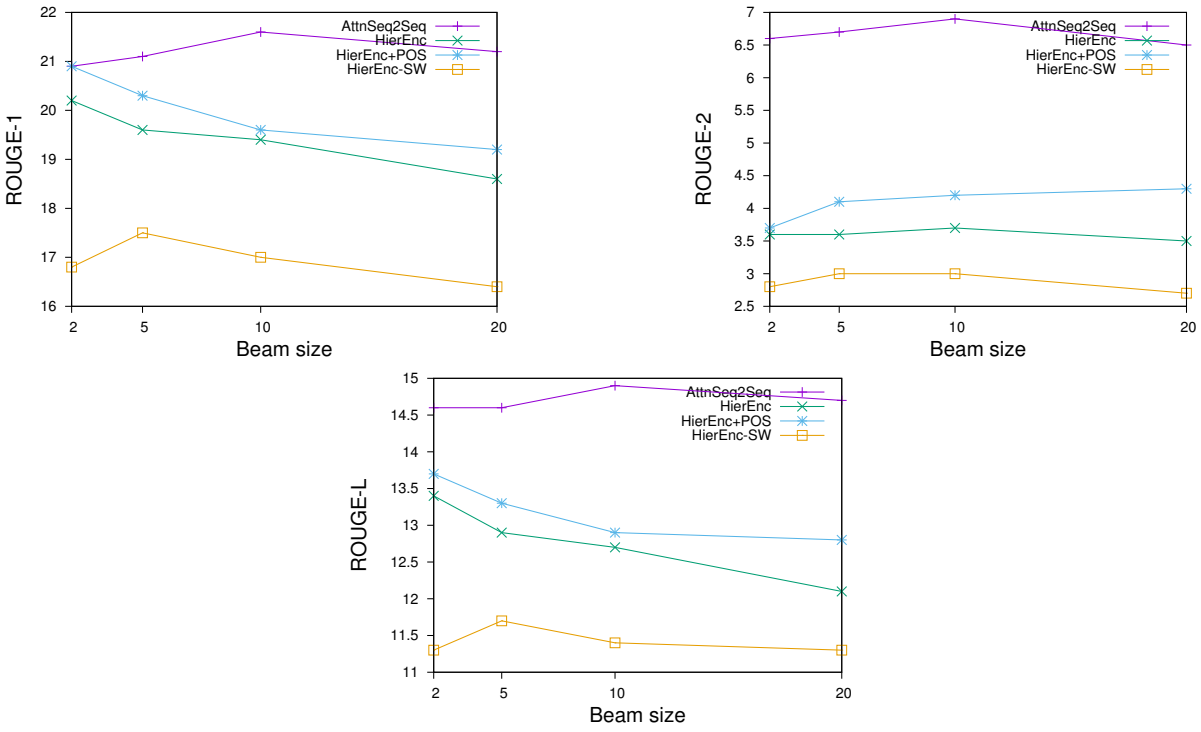
	Full-length F1				**75b F1**			
Beam size	**2**	**5**	**10**	**20**	**2**	**5**	**10**	**20**
	ROUGE-1							
AttnSeq2Seq	21.3	21.6	**21.9**	**22.2**	**20.9**	**21.1**	**21.6**	**21.2**
HierEnc	21.8	18.0	17.6	17.0	20.2	19.6	19.4	18.6
HierEnc+POS	**23.5**	**21.8**	20.9	20.1	**20.9**	20.3	19.6	19.2
HierEnc-SW	21.5	21.1	20.5	20.2	16.8	17.5	17.0	16.4
	ROUGE-2							
AttnSeq2Seq	**5.7**	**5.9**	**5.9**	**5.9**	**6.6**	**6.7**	**6.9**	**6.5**
HierEnc	3.9	3.2	3.1	3.1	3.6	3.6	3.7	3.5
HierEnc+POS	3.9	3.8	3.7	3.7	3.7	4.1	4.2	4.3
HierEnc-SW	3.7	3.7	3.7	3.7	2.8	3.0	3.0	2.7
	ROUGE-L							
AttnSeq2Seq	19.1	19.3	**19.4**	**19.7**	**14.6**	**14.6**	**14.9**	**14.7**
HierEnc	19.9	16.5	16.1	15.6	13.4	12.9	12.7	12.1
HierEnc+POS	**21.4**	**19.7**	18.9	18.2	13.7	13.3	12.9	12.8
HierEnc-SW	19.8	19.3	18.6	18.4	11.3	11.7	11.4	11.3

that this is because the *AttnSeq2Seq* is better at learning to predict the probability of each word in the summary and hence the score of each beam is more likely to reflect the similarity of the output with the gold summary. This means that when we increase the beam size, the *AttnSeq2Seq* model is able to explore other beams and in the end select beam with a better score or more similar to the gold summary. On the contrary, other models may have failed to capture the document information as well as to predict the new token. This leads to a consequence that the beam with a higher score is not necessarily relevant much to the document and the gold summary. These trends can be seen more clearly in the Figure 4.3a. When we compare the models using first-75b F1 score, the patterns are not very obvious because each model changes differently. More specifically, *AttnSeq2Seq* always achieve better ROUGE scores in all metrics, but it gets a bit worse when beam size is larger than 10. This is probably because the model prefers to choose a summary whose relevant content is expressed in several sentences. All *HierEnc*-based models have similar trend as full-length F1 score, except the ROUGE-2 score does not change much in case of *HierEnc* and *HierEnc+POS*.

We also studied the effect of different beam size to the decoding time of each model. The detailed results are presented in Table 4.6 and plotted in Figure 4.4. We can easily see that the decoding time of the *AttnSeq2Seq* increases pretty quickly and the gap between this model and the rest gets bigger when beam size increases. Other models also witness the increase of decoding time but it is not too strong. Similar to the case of the training time, *HierEnc* is the fastest model in decoding phase. Interestingly, it is also the model with the lowest Full-length ROUGE score. In our opinion, there are two factors causing this phenomenon:



(a) Full-length F1.



(b) First-75-byte F1.

Figure 4.3: Comparison of ROUGE score when changing the beam size.

Table 4.6: Comparison of decoding time when changing the beam size.

	Decoding time (seconds)			
Beam size	**2**	**5**	**10**	**20**
AttnSeq2Seq	342	476	688	1188
HierEnc	82	107	162	267
HierEnc+POS	157	206	309	517
HierEnc-SW	121	163	254	430

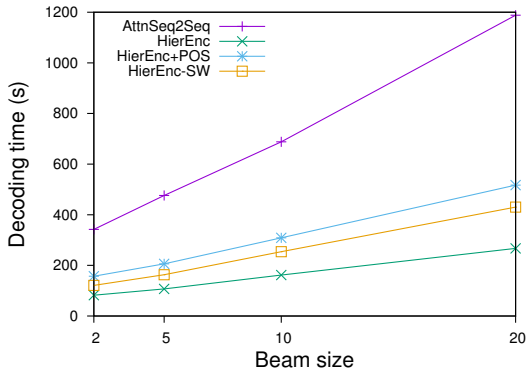


Figure 4.4: A graph showing the variation of decoding time when changing the beam size.

- The CNN we use for encoding each sentence might not capture as much information as the normal RNN. We might have lost some dependencies in the input document and thus the model fails to learn effectively.
- When using hierarchical encoding, we significantly reduce the number of hidden states computed by the encoder. This allows the model to be trained faster, but we also lose information in the attention context vector when decoding.

4.3.3 Experiment 3: Diverse beam-search decoding with reranking

As we discussed in Section 3.3.1, the weakness of beam-search is the lack of diversity. In Figure 4.5, we show an example of this issue when running the decoding process with beam size set to 5. We can see that the words on each beam are quite similar, they only differ at a few locations in the text.

In this experiment, we chose one summarization model and implemented different methods for beam-search decoding on it. Since we focus our study on the difference between these methods, we are not going to compare the models as we did in two previous experiments. To be more specific, we used the *AttnSeq2Seq* as our base summarization model due to its superior performance comparing to other models and we tested different decoding methods to generate the summary. We compare the default decoding, in which we select the beam with the highest score, with two methods:

Table 4.7: Performance comparison of different decoding methods on various beam sizes.

	Full-length F1				**75b F1**			
Beam size	**2**	**5**	**10**	**20**	**2**	**5**	**10**	**20**
	ROUGE-1							
Default	21.3	21.6	21.9	22.2	20.9	21.1	21.6	21.2
OracleReranker	21.7	23.2	24.2	24.8	21.0	21.7	**22.3**	**22.4**
DBS-G+Top5SentReranker	22.2	23.6	24.8	25.6	20.1	19.2	18.8	18.0
DBS-G+OracleReranker	22.3	24.1	26.0	27.6	20.7	20.7	21.1	20.1
DBS-L+Top5SentReranker	25.0	27.4	28.2	28.9	20.7	19.9	18.7	18.9
DBS-L+OracleReranker	**25.5**	**28.4**	**29.7**	**30.9**	**21.5**	**21.6**	20.8	20.7
	ROUGE-2							
Default	5.7	5.9	5.9	5.9	6.6	6.7	6.9	6.5
OracleReranker	5.9	6.6	7.0	7.3	6.7	7.1	**7.3**	**7.3**
DBS-G+Top5SentReranker	6.0	6.5	6.7	6.9	6.3	5.6	5.2	4.6
DBS-G+OracleReranker	6.1	6.7	7.4	8.0	6.6	6.7	6.3	5.7
DBS-L+Top5SentReranker	6.7	7.3	7.4	7.5	6.7	6.1	5.3	5.1
DBS-L+OracleReranker	**7.0**	**7.8**	**10.0**	**10.7**	**7.1**	**7.3**	6.6	6.3
	ROUGE-L							
Default	19.1	19.3	19.4	19.7	14.6	14.6	14.9	14.7
OracleReranker	19.5	20.8	21.6	22.1	14.6	**15.1**	**15.5**	**15.5**
DBS-G+Top5SentReranker	20.0	21.3	22.2	22.9	14.1	13.2	12.8	12.1
DBS-G+OracleReranker	20.2	21.8	23.5	24.9	14.4	14.3	14.4	13.8
DBS-L+Top5SentReranker	22.4	24.4	25.1	25.6	14.3	13.5	13.0	12.8
DBS-L+OracleReranker	**22.9**	**25.5**	**26.5**	**27.5**	**14.9**	14.9	14.4	14.1

- *Method 1 (DBS-G)*: We use diverse beam-search decoding with beam grouping as described in Section 3.3.1. We set *diversity rate* to 1.0 and the number of groups equal to the beam size.

- *Method 2 (DBS-L)*: We use diverse beam-search decoding with local diversity as described in Section 3.3.2, Local diversity rate is heuristically set to 2.0.

For the re-ranking part, it can be either the top-5-sentence-relevance re-ranker (denoted as *Top5SentReranker*) or the Oracle re-ranker (denoted as *OracleReranker*).

The ROUGE scores of these decoding methods are presented in Table 4.7. There are several interesting points we can learn from that table:

- First, re-ranking does help to improve the performance of the summarization model even with the default beam-search decoding. After applying *OracleReranker* to the default decoding, we get better ROUGE score in all cases. The improvement when using a real re-ranker might not be as big as our result, but at least it shows us the room for improvement. This is in accordance with our reasoning that the beam

Gold summary:
@entity9 , of @entity10 , @entity1 , was arrested after she was identified from the video
the cameraman films @entity48 for more than a minute before she notices
he then asks what she 's doing but @entity48 does not respond
the cameraman said he started filming after seeing the woman ' pulling this kid by his hair ' out of a @entity112 's

---

Beam 1:
@entity1 woman was arrested after video posted on @entity3
video posted on @entity3 that showed her striking a young , crying child in face with what appeared to be a @entity8 - like tablet

---

Beam 2:
@entity1 woman was arrested after video posted on @entity3
video posted on @entity3 that showed her striking a young , crying child in the face with what appeared to be a @entity8 - like tablet

---

Beam 3:
@entity1 woman was arrested after video posted on @entity3
she was posted on @entity3 that showed her striking a young , crying child in face with what appeared to be a @entity8 - like tablet

---

Beam 4:
@entity1 woman was arrested after video posted on @entity3
video posted on @entity3 showing her striking a young , crying child in face with what appeared to be a @entity8 - like tablet

---

Beam 5:
@entity1 woman was arrested after video posted on @entity3

Figure 4.5: Similar beam outputs generated by the *AttnSeq2Seq* model. Beam size is 5.

with the highest score returned by the model does not necessarily mean the best summary. Some lower score outputs might contain more relevant content.

- Second, the effect of re-ranking becomes more obvious when we increase the beam size. This is not too difficult to understand because larger beam size means we have more outputs to consider. As a result, we have more chance to select the best output which is closer to the gold summary.

- Third, *first-75b F1* score cannot fully express the efficiency of diverse beam-search and re-ranking. This is because when we have more diverse outputs, we probably prefer the text which has the entire content as relevant as possible to the document and the gold summary, not just the first 75 bytes of the text. That is why we see the scores of models with diverse output tends to decrease as we increase the beam size. *OracleReranker* does not follow this trend because all beam outputs from the *AttnSeq2Seq* have very similar text, especially the beginning part. Thus applying re-ranking to select a different output does not change the first 75 bytes much. The outputs presented in Figure 4.5 are good examples for this case.

If we look at the full-length F1 score when beam size is set to 20, we can clearly see effectiveness of new decoding methods and re-ranking. The *DBS-G* technique allows the model to achieve higher ROUGE scores than the default method in all metrics (ROUGE-1, 2 and L), no matter which re-ranker we use. Our proposed method *DBS-L* is shown to be better than *DBS-G*, even with the simple *Top5SentReranker*. This means that the outputs generated by *DBS-L* are generally better than the *DBS-G*. But *Top5SentReranker* is a very simple ranker and it does not show the full potential of our method. The result when using *DBS-L* in combination with an *OracleReranker* indicates what we can possibly achieve with a well-designed re-ranker. This method yields impressive scores higher than all other methods. Comparing to the standard decoding, it helps the ROUGE-1 score to increase by almost 40%, ROUGE-2 increases by more than 80% and ROUGE-L increases by nearly 40%. We graphically depict the difference between these methods in Figure 4.6.

To illustrate the efficiency of these techniques, we show the summaries generated by each method in the Figure 4.6. Two methods *Default* and *OracleReranker* generate nearly duplicate sentences and repetitive words. *DBS-G+OracleReranker* produces better output but it still repeats the word *"today"* several times. Our proposed method *DBS-L+OracleReranker* does the best job at avoiding repetitive words and phrases (though it cannot remove this issue completely since we the word *pensioner* was duplicated).

4.4 Performance comparison with other systems

We could not find many published systems using the same DailyMail dataset for testing their performance. In this section, we present only 2 systems for comparison:

- The word extraction model namely NN-WE proposed by Cheng et al. [37]. We compare with this model using first-75b F1 score. The author did not report full-length F1 for this model.

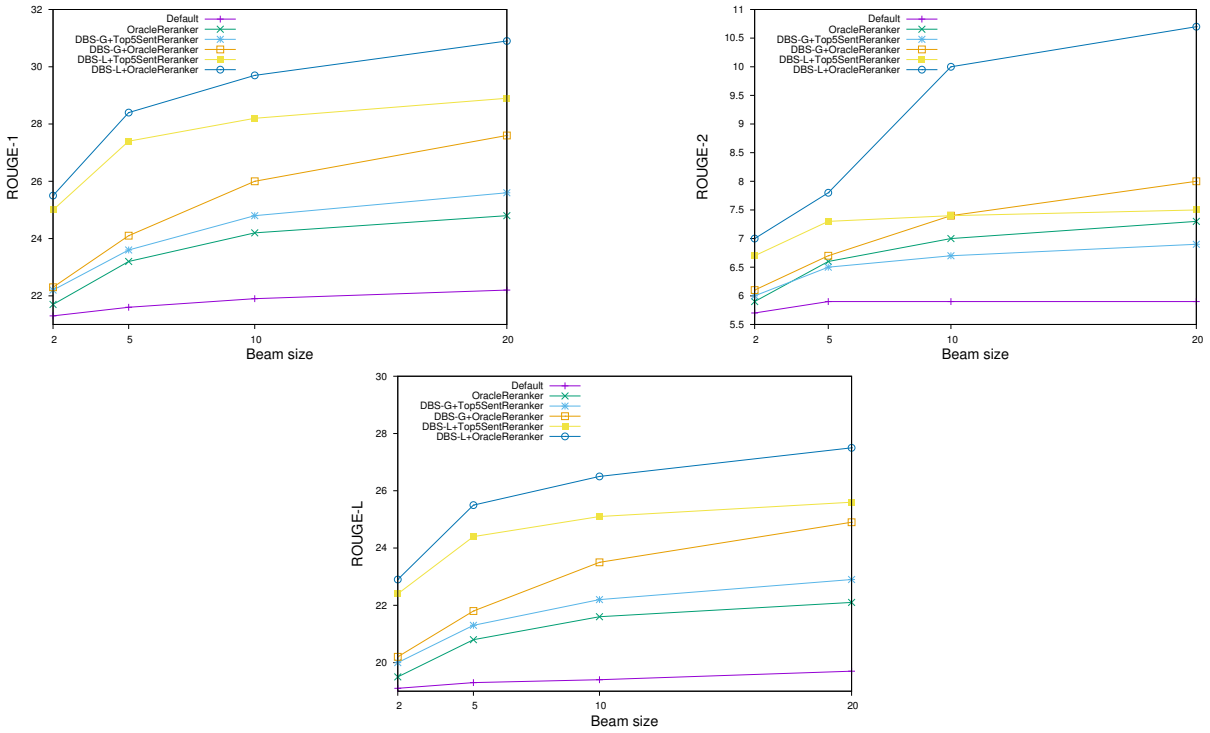


Figure 4.6: Comparison of Full-length F1 score when using different methods for decoding.

- The abstractive summarization model proposed by Nallapati et al. [15]. In particular, we directly take the result of the best model called *words-lvt2k-temp-att* from their paper. We compare our systems with this model using full-length F1 score.

We select two models from our side for comparison. The first one is the *AttnSeq2Seq* model with default decoding method, the other one is the *AttnSeq2Seq* model combined with the *DBS-L+OracleReranker* (both use beam_size = 20 for decoding). The results are presented in Table 4.8 and Table 4.9.

In the first result (Table 4.8), our model achieves better ROUGE-1 and ROUGE-2 score than the *NN-WE* model, ROUGE-2 is not so different. This means the beginning text of our generated summaries has more relevant content than their model. But it should be noted that their model was tested on different test set because the authors randomly selected 500 documents for testing.

The result in Table 4.9 shows that our best model *AttnSeq2Seq+DBS-L* achieves much lower score than *words-lvt2k-temp-att*. This is mainly because:

- We did not use as many techniques as the authors did in their work. Our model is, in fact, similar to their base model without extra techniques they added into it. Therefore the comparison is not fair and lower performance on our model is predictable.

- We did not have time and resources to extensively tune many parameters, we also kept the network size at a reasonable size to make it runnable on the server we used.

Gold summary:
body of woman *believed* to be @entity15 was discovered at her home
her son @entity6 , 73 , has been charged with the 95 - year - old 's murder
police called to the scene in @entity12 on thursday at about 8pm

---

Default:
@entity6 , <DIGIT> , is to appear in court today today charged with the murder of his <DIGIT> - year - old mother at her home
police arrested @entity6 after pensioner pensioner , <DIGIT> , is to appear in court today charged with the murder of his <DIGIT> - year - old mother at her home
police arrested @entity6 after the pensioner pensioner , <DIGIT> , is to appear in court today charged with the murder of his <DIGIT> - year - old mother at her home
police arrested @entity6 after the pensioner pensioner , <DIGIT> ,

---

OracleReranker:
@entity6 , <DIGIT> , is to appear in court today today charged with the murder of his <DIGIT> - year - old mother at her home
police arrested @entity6 after pensioner pensioner , <DIGIT> , is to appear in court today charged with the murder of his <DIGIT> - year - old mother at her home
police arrested @entity6 after the pensioner pensioner , <DIGIT> , is to appear in court today charged with the murder of his <DIGIT> - year - old mother at her home
police arrested @entity6 after the pensioner pensioner was to appear

DBS-G+Top5SentReranker

@entity19 pensioner has been charged with the murder of his <DIGIT> - year - old mother at her home

police arrested @entity6 after pensioner *'s* mother at her home

police arrested @entity6 after pensioner , <DIGIT> , from her home

police arrested @entity6 after the pensioner pensioner is to appear in court today charged with the murder of his <DIGIT> - year - old mother at her home

police arrested @entity6 after the pensioner pensioner , aged <DIGIT> , at her home

police arrested @entity6 after the pensioner pensioner was to appear in court today

---

DBS-G+OracleReranker:

the <DIGIT> pensioner , <DIGIT> , is to appear in court today today charged with the murder of his <DIGIT> - year - old mother at her home

police arrested @entity6 after the pensioner pensioner was to appear in court today today today today

---

DBS-L+Top5SentReranker

@entity122 said pensioner was to appear in court today charged with the murder of his <DIGIT> - year - old mother at her home

police arrested @entity6 after the pensioner pensioner , who is to appear in court today today

---

DBS-L+OracleReranker:

the <DIGIT> pensioner , <DIGIT> , is to appear in court today charged with the murder of his <DIGIT> - year - old mother at her home

police arrested @entity6 after the pensioner pensioner was on @entity28 's home

Figure 4.6: Sample outputs generated by different decoding methods. Beam size is 20.

Table 4.8: First-75b F1 comparison with other work.

	ROUGE-1	**ROUGE-2**	**ROUGE-L**
NN-WE [37]	15.7	6.4	9.8
AttnSeq2Seq	**21.0**	**6.5**	**14.7**
AttnSeq2Seq+DBS-L+OracleReranker	20.7	6.3	14.1

Table 4.9: Full-length F1 comparison with other work.

	ROUGE-1	**ROUGE-2**	**ROUGE-L**
words-lvt2k-temp-att [15]	**35.5**	**16.6**	**32.7**
AttnSeq2Seq	22.2	5.9	19.7
AttnSeq2Seq+DBS-L+OracleReranker	30.9	10.7	27.5

Chapter 5

Discussion

Based on the results we have from the comparison of different summarization models, we can say that the hierarchical encoding model we used is not strong enough to surpass the default RNN encoder in term of ROUGE-2 score. One reason is probably that we have lost some information from the input document after replacing the RNN with the CNN. Another explanation is the attention-based decoder in *HierEnc* model has access only to the information of each sentence, while the *AttnSeq2Seq* is able to attend each word in the input document. The benefit of using hierarchical encoding is we save computational time and achieve competitive ROUGE-1 and ROUGE-L scores.

We also see that providing POS as additional input features can help the deep learning model to work better. We think this is simply because the model has more space to adjust its parameters and therefore it could find better weight values for its network. This can partly explain the reason why the model proposed by Nallapati et al. [15] achieved a good performance. Removing stop words makes the system perform a little worse, but it is not too dramatically.

Our experiments also show that beam-search decoding is a good technique for selecting better output, but only if our model is trained sufficiently well. *HierEnc* did not benefit at all from bigger beam size due to this reason, while all other models showed an improvement when we increase the beam size. One thing we should notice is that increasing the beam size means we spend more time on decoding process. Though it is not a big issue in our case because it takes only a few seconds, some practical applications might not prefer this and we should set beam size to a reasonable number.

Another thing we found in the experiments is that the combination of beam-search and re-ranking gives us a simple way to improve our summarization system. This is useful to be applied in many similar systems because we do not have to modify the training architecture of the model. Yet, re-ranking can only show its full potential when we employ several techniques to generate diverse outputs. Our proposed method *DBS-L* combining with a Oracle re-ranker indicates what we could possibly achieve if we do things properly. Diverse beam-search also benefits a lot from using larger beam-size (which is not obvious in the case of default beam-search).

Our biggest challenge when doing this research is running the experiments. The size of our model was huge and we could only finish those experiments by running it on a

supercomputer with GPU. We did try running our experiments on CPU-only server and we found that it was 10 times slower than GPU. In addition, our training process required more than 10GB of GPU memory (batch size is 32), a typical personal computer is totally not suitable for this. Even when using the supercomputer, we still had to reduce several parameters to make it runnable (by using smaller batch size and smaller hidden size comparing to similar research). Another problem we met is that the server we used had a limited time for each session per user. That is why we had to keep saving the model to disk and restarting it after every 2 days. This process is truly time-consuming and error-prone.

Chapter 6

Conclusion

This thesis presented our study on abstractive text summarization using deep learning, our experiments to compare different summarization models and our proposal for improving the decoding process. There are several points which we consider the most important contributions in this research:

- We implemented two deep learning models for single-document summarization problem. One model uses a standard RNN encoder and the other uses a hierarchical encoder. We ran many experiments to compare these systems in terms of running time and summary quality. The experiments gave us a greater understanding of the network architecture we use for encoding and how it can affect the performance of the system.
- We studied the effect of adding POS information to the encoder as well as removing stop words from the encoder. Although it is a simple experiment, it shows how each feature contributed to the final performance. Our study on the beam-search decoding also showed an interesting result when beam size is varied on each model: increasing beam size is not always beneficial if the model is too weak.
- We observed the output from neural summarization model to understand the current issues in the summary, then employed several techniques to improve the quality of the summaries by modifying the decoding process. In particular, we applied methods to generate more diverse outputs and used re-ranking to select a better summary. Our proposed method achieved the best ROUGE score comparing to the default decoding method.

Even though there are more things we want to do in this research, we could not due to the limited time, resources and knowledge. The main limitations of this research are:

- The reason for the lower performance of hierarchical encoding model was not systematically checked and verified through formal analysis. We could not point out which was the most important cause for this and how to overcome it. More investigation should be done. Besides, network parameters were not tuned enough to achieve the best performance.

- We could not investigate the effectiveness of using other features to support the model. The intention of using linguistic features like syntactic tree could not be accomplished.
- We did not have time to study more functions for maintaining local diversity in our proposed method. Our re-ranker was also a simple one instead of a carefully-designed re-ranker.
- It would be better if we could conduct a manual evaluation for this research. ROUGE is not very suitable for evaluating an abstractive summarization system.

Abstractive text summarization is an interesting research and we think it should deserve more attention from the community. Among all the techniques used for solving this problem, deep learning has shown the most promising results. It is not impossible that in the near future, a deep learning system can generate abstractive summaries which can even be better than our human writing.

Bibliography

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

[2] Min-Yen Kan and Judith L Klavans. Using librarian techniques in automatic text summarization for information retrieval. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 36–45. ACM, 2002.

[3] José M Perea-Ortega, Elena Lloret, L Alfonso Ureña-López, and Manuel Palomar. Application of text summarization techniques to the geographical information retrieval task. *Expert systems with applications*, 40(8):2966–2974, 2013.

[4] Elena Lloret, Hector Llorens, Paloma Moreda, Estela Saquete, and Manuel Palomar. Text summarization contribution to semantic question answering: New approaches for finding answers on the web. *International Journal of Intelligent Systems*, 26(12):1125–1152, 2011.

[5] Dmitry Tsarev, Mikhail Petrovskiy, and Igor Mashechkin. Using nmf-based text summarization to improve supervised and unsupervised classification. In *Hybrid Intelligent Systems (HIS), 2011 11th International Conference on*, pages 185–189. IEEE, 2011.

[6] Udo Hahn and Inderjeet Mani. The challenges of automatic summarization. *Computer*, 33(11):29–36, 2000.

[7] Atif Khan and Naomie Salim. A review on abstractive summarization methods. *Journal of Theoretical and Applied Information Technology*, 59(1):64–72, 2014.

[8] Chao Shen and Tao Li. Learning to rank for query-focused multi-document summarization. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 626–634. IEEE, 2011.

[9] Neelima Bhatia and Arunima Jaiswal. Trends in extractive and abstractive techniques in text summarization. *International Journal of Computer Applications*, 117(6), 2015.

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

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

[12] Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750, 2014.

[13] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

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

[15] Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*, 2016.

[16] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. Modeling coverage for neural machine translation. *arXiv preprint arXiv:1601.04811*, 2016.

[17] Jun Suzuki and Masaaki Nagata. Cutting-off redundant repeating generations for neural abstractive summarization. *EACL 2017*, page 291, 2017.

[18] Haitao Mi, Baskaran Sankaran, Zhiguo Wang, and Abe Ittycheriah. Coverage embedding models for neural machine translation. *arXiv preprint arXiv:1605.03148*, 2016.

[19] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[20] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

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

[22] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

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

[24] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[25] Martin Sundermeyer, Tamer Alkhouli, Joern Wuebker, and Hermann Ney. Translation modeling with bidirectional recurrent neural networks. In *EMNLP*, pages 14–25, 2014.

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

[27] Wen-tau Yih, Xiaodong He, and Christopher Meek. Semantic parsing for single-relation question answering. In *ACL (2)*, pages 643–648. Citeseer, 2014.

[28] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 373–374. ACM, 2014.

[29] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[30] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[31] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, volume 14, pages 77–81, 2015.

[32] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

[33] David Graff and C Cieri. English gigaword corpus. *Linguistic Data Consortium*, 2003.

[34] Paul Over, Hoa Dang, and Donna Harman. Duc in context. *Information Processing & Management*, 43(6):1506–1520, 2007.

[35] Sumit Chopra, Michael Auli, Alexander M Rush, and SEAS Harvard. Abstractive sentence summarization with attentive recurrent neural networks. *Proceedings of NAACL-HLT16*, pages 93–98, 2016.

[36] Sho Takase, Jun Suzuki, Naoaki Okazaki, Tsutomu Hirao, and Masaaki Nagata. Neural headline generation on abstract meaning representation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1054–1059.

[37] Jianpeng Cheng and Mirella Lapata. Neural summarization by extracting sentences and words. *arXiv preprint arXiv:1603.07252*, 2016.

[38] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out: Proceedings of the ACL-04 workshop*, volume 8. Barcelona, Spain, 2004.

[39] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. *CoRR*, abs/1412.2007, 2014.

[40] Baskaran Sankaran, Haitao Mi, Yaser Al-Onaizan, and Abe Ittycheriah. Temporal attention model for neural machine translation. *arXiv preprint arXiv:1608.02927*, 2016.

[41] Jiwei Li, Will Monroe, and Dan Jurafsky. A simple, fast diverse decoding algorithm for neural generation. *arXiv preprint arXiv:1611.08562*, 2016.

[42] Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. Diverse beam search: Decoding diverse solutions from neural sequence models. *arXiv preprint arXiv:1610.02424*, 2016.

[43] Andriy Mnih and Geoffrey E Hinton. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088, 2009.

[44] Welin Chen, David Grangier, and Michael Auli. Strategies for training large vocabulary neural language models. *arXiv preprint arXiv:1512.04906*, 2015.

[45] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

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

[47] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[48] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*, 2015.

[49] Ani Nenkova and Rebecca J Passonneau. Evaluating content selection in summarization: The pyramid method. In *HLT-NAACL*, volume 4, pages 145–152, 2004.

[50] Karolina Owczarzak, John M Conroy, Hoa Trang Dang, and Ani Nenkova. An assessment of the accuracy of automatic evaluation in summarization. In *Proceedings of Workshop on Evaluation Metrics and System Comparison for Automatic Summarization*, pages 1–9. Association for Computational Linguistics, 2012.