ANALYSIS OF WORD DEPENDENCY RELATIONS AND SUBWORD MODELS IN ABSTRACTIVE TEXT SUMMARIZATION

by

Ahmet Beka Özkan

B.S., Computer Engineering, Middle East Technical University, 2018

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2021

ACKNOWLEDGEMENTS

I am grateful to my advisor Prof. Tunga Güngör for his continuous support and understanding. His knowledge and guidance throughout my M.S. study were essential. His patience and motivation played a major role when writing the thesis. I also would like to thank my thesis defense jury members, Prof. Fikret Gürgen and Assoc. Prof. Cüneyd Tantuğ, for their valuable comments and feedbacks about the thesis.

I wish to express my gratitude to my parents Hacer and İberya for their endless belief and support. Their encouragement was very crucial, especially during my M.S. study. I remain indebted to them for their help in my life, and I simply cannot thank them enough.

I would like to express my appreciation to my sister Natya for showing support even in her busy times. She was always open to offer any help that she can provide. I also want to thank my cousin Tamila Kılıç for her willingness to assist. Her comments and feedbacks were influential for the integrity of the thesis.

I was awarded the TÜBİTAK 2210-A scholarship for my M.S. study, and I thank them for their support. Most of the models in this thesis were trained on Boğaziçi University TETAM servers.

ABSTRACT

ANALYSIS OF WORD DEPENDENCY RELATIONS AND SUBWORD MODELS IN ABSTRACTIVE TEXT SUMMARIZATION

Abstractive text summarization is an important task in natural language processing. As there are too many textual materials becoming available in the digital world at an unprecedented speed, people begin to need automated text summarization systems to summarize such bulk data in a condensed form that only holds the necessary information. With recent advances in deep learning techniques, abstractive text summarization has gained even more attention. Attention-based sequence-to-sequence models are adapted for this task and achieved state-of-the-art results. On top of it, several additional mechanisms like pointer/generator and coverage were proposed and have become the standard mechanisms to be used for abstractive summarization models. Using these approaches, we integrated word dependency relations and analyzed their effects on the models. We showed that integrating dependency relations increases performance. Recent models for many natural language processing tasks use subwords and achieve state-of-the-art results. We utilized three different subword models in our models and analyze their effectiveness in the abstractive summarization task. We found that subword usage is another viable option to be included for abstractive summarization models as well.

ÖZET

SOYUTLAMALI METİN ÖZETLEMEDE KELİME BAĞLILIK İLİŞKİLERİ VE ALT SÖZCÜK MODELLERİ ANALİZİ

Soyutlamalı metin özetleme, doğal dil işlemede önemli bir alandır. Dijital dünyada daha önce eşi görülmemiş bir hızla çok fazla metin materyalleri oluşturulduğundan, insanlar bu tür metinlerden yalnızca gerekli bilgileri içeren bir biçimde özet elde etmek için otomatik metin özetleme sistemlerine ihtiyaç duymaya başladılar. Derin öğrenme metotlarındaki son gelişmelerle birlikte, soyutlamalı metin özetleme araştırmacılar tarafından daha da fazla ilgi görmeye başlamıştır. Dikkat temelli diziden diziye modeller bu alana uyarlanabilmekte ve bu tip modeller oldukça başarılı sonuçlar vermektedir. Bunlarla beraber, işaretçi/üretici ve kapsam gibi birkaç ek mekanizma sıklıkla kullanılmaktadır. Bu mekanizmalar, başarılarından dolayı soyutlamalı özetleme modelleri için birer standart haline gelmiştir. Bu tezde söz konusu teknikler ile kelime bağlılık ilişkileri kullanımı bütünleştirilmiş ve modellere olan etkileri analiz edilmiştir. Bağlılık ilişkilerini entegre etmenin performansı artırdığı gösterilmiştir. Doğal dil işleme görevleri için tasarlanmış birçok yeni model, alt sözcükler kullanmakta ve oldukça başarılı sonuçlar elde etmektedir. Bu tezdeki modellerde üç farklı alt sözcük modeli kullanılarak soyutlamalı özetleme alanındaki etkileri incelenmiştir. Alt sözcük kullanımının da bu tür soyutlamalı özetleme modellere dahil edilebilecek uygun bir seçenek olduğu gösterilmiştir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . iii
ABSTRACT . . . iv
ÖZET . . . v
LIST OF FIGURES . . . viii
LIST OF TABLES . . . ix
LIST OF SYMBOLS . . . xi
LIST OF ACRONYMS/ABBREVIATIONS . . . xii
1. INTRODUCTION . . . 1
1.1. Abstractive Summarization . . . 2
1.2. Motivation . . . 3
1.3. Thesis Outline . . . 4
2. BACKGROUND INFORMATION . . . 6
2.1. Deep Learning and Neural Networks . . . 6
2.2. Recurrent Neural Networks . . . 7
2.2.1. Long Short-Term Memory . . . 8
2.2.2. Gated Recurrent Unit . . . 9
2.3. Influential Deep Learning Models in NLP . . . 10
2.3.1. Sequence-to-Sequence Encoder-Decoder Models . . . 10
2.3.2. Attention Mechanism . . . 12
2.4. Dependency Parsing . . . 14
2.5. Subword Models . . . 14
2.5.1. Byte-Pair Encoding . . . 15
2.5.2. WordPiece . . . 16
2.5.3. Unigram Language Model . . . 16
2.6. Summarization . . . 16
2.6.1. Extractive Summarization . . . 17
2.6.2. Abstractive Summarization . . . 19
3. RELATED WORK . . . 20

3.1. Studies Using Deep Learning 21
4. DATASETS 31
4.1. Gigaword 31
4.2. CNN/Daily Mail 33
5. MODEL 34
5.1. Baseline Model 34
5.2. Pointer/Generator Mechanism 36
5.3. Coverage Mechanism 39
5.4. Word Dependency Features 41
5.5. Subword Usage 43
6. EXPERIMENTS AND DISCUSSIONS 45
6.1. Preliminaries 45
6.2. Dependency Features on Abstractive Summarization Models 48
6.2.1. Gigaword Experiments 48
6.2.2. CNN/Daily Mail Experiments 51
6.2.3. The Effects of Additional Mechanisms 52
6.2.4. The Effects of Dependency Features 53
6.3. Dependency Features with Different Hyper-Parameter Configurations . 55
6.3.1. Gigaword Experiments 55
6.3.2. CNN/Daily Mail Experiments 57
6.3.3. The Effects of Hyper-Parameters 59
6.4. Integration Position of Word Dependency Features 60
6.5. Subword Models on Abstractive Summarization Models 61
6.6. Proper Usage of Additional Mechanisms 67
6.7. Discussions Regarding the Evaluation Metrics 69
7. CONCLUSION 71
REFERENCES 73
APPENDIX A: A CLOSER LOOK ON GENERATED SUMMARIES 81

LIST OF FIGURES

Figure 2.1. Long Short-Term Memory Cell 9

Figure 2.2. Gated Recurrent Unit Cell 10

Figure 2.3. Sequence-to-Sequence Encoder-Decoder Network 12

Figure 5.1. Sequence-to-Sequence Encoder-Decoder Model with Pointer/Generator Mechanism 39

Figure 5.2. Incorporating Word Dependency Features with Inputs of Encoders 42

Figure 5.3. Incorporating Word Dependency Features with Hidden States of Encoder 43

LIST OF TABLES

Table 6.1. The ROUGE scores of the models with different additional mechanisms over the Gigaword test sets 49

Table 6.2. The ROUGE scores of the models with different additional mechanisms over the CNN/Daily Mail test set 51

Table 6.3. The ROUGE scores of the models with different hyper-parameter configurations over the Gigaword test sets 56

Table 6.4. The ROUGE scores of the models with different hyper-parameter configurations over the CNN/Daily Mail test set 58

Table 6.5. The ROUGE scores of *"Input"* and *"Hidden"* models over the Gigaword test sets 60

Table 6.6. The ROUGE scores of *"Input"* and *"Hidden"* models over the CNN/Daily Mail test set 60

Table 6.7. The ROUGE scores of the models that use byte-pair encoding over the Gigaword test sets 61

Table 6.8. The ROUGE scores of the models that use WordPiece over the Gigaword test sets 62

Table 6.9. The ROUGE scores of the models that use unigram language model over the Gigaword test sets 63

Table 6.10. The ROUGE scores of the models that use byte-pair encoding over the CNN/Daily Mail test set 64

Table 6.11. The ROUGE scores of the models that use WordPiece over the CNN/Daily Mail test set 64

Table 6.12. The ROUGE scores of the models that use unigram language model over the CNN/Daily Mail test set 65

Table 6.13. The ROUGE scores of various models including *"Dep+PG→COV"* over the Gigaword test set 67

Table 6.14. The ROUGE scores of various models including *"Dep+PG→COV"* over the CNN/Daily Mail test set 68

Table 6.15. The METEOR scores of different models over both the Gigaword test sets and the CNN/Daily Mail test set 70

LIST OF SYMBOLS

$*$	Element-wise vector multiplication operation
$\oplus$	Vector concatenation operation
b	Bias term
c_t	Context vector at decoding time t
d	Size of hidden state vector
e_i	Embedding vector corresponding to x_i
e_{ti}	Attention score of x_i at decoding time t
f_i	Embedding vector corresponding to dependency features of x_i
h_i	Hidden state vector at position i in encoder network
n	Number of tokens in an input sentence
$\mathbb{R}$	The set of real numbers
s_t	Hidden state vector at position i in decoder network
U	Weight matrix
V	Size of vocabulary
W	Weight matrix
w_t^*	Predicted token by system at decoding time t
w_t	Output token at decoding time t
x_i	Token at position i in an input sentence
$\mathbf{x}$	Set of all x_i
y_t	Embedding vector of output token at decoding time t
$\mathbf{y}$	Set of all y_i
α_{ti}	Attention value of x_i at decoding time t
σ	Sigmoid function

LIST OF ACRONYMS/ABBREVIATIONS

ABS	Attention-Based Summarization Model
Adagrad	Adaptive Gradient Algorithm
BERT	Bidirectional Encoder Representations from Transformers
biGRU	Bidirectional Gated Recurrent Unit
biLSTM	Bidirectional Long Short-Term Memory
BPE	Byte-Pair Encoding
CNN	Convolutional Neural Network
COV	Coverage
Dep	Dependency
DUC	Document Understanding Conferences
exp	Natural exponential function
GloVe	Global Vectors for Word Representation
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
NFD	Normalization Form Canonical Decomposition
NLP	Natural Language Processing
OOV	Out-Of-Vocabulary
PG	Pointer/Generator
PTB	Penn Treebank
R-1	ROUGE-1
R-2	ROUGE-2
R-L	ROUGE-L
RAS-Elman	Recurrent Attentive Summarizer with Elman RNN
RNN	Recurrent Neural Network
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SEASS	Selective Encoding for Abstractive Sentence Summarization model
softmax	Softmax function

tanh	Hyperbolic tangent function
UNK	Unknown token
Vocab	Vocabulary
w/	with

1. INTRODUCTION

Summarization is the task of acquiring a condensed text considering the main information of an original longer text. It is frequently done in many areas of our everyday lives. It is an important subfield in Natural Language Processing (NLP). A system that does summarization itself is expected to generate summaries as the output by receiving the original text as the input. The input text can have one or more sentences. It is frequently called a *document* or an *article* if the input contains several sentences. The process can be called *sentence summarization* if the input contains only one but a long sentence. On the other hand, the output is typically called *summary*. There is no restriction about the length of it. It can be as short as one or two words, or as long as two or three sentences. However, it should naturally be shorter than the input. The process can also be called *headline generation* if the output contains two or three words and is suitable for it to be used as a headline.

Summarization can be divided into many different types since the desired summaries can have many different intentions or aspects. For example, it can be single-document or multi-document summarization according to the number of documents to be summarized. It can be generic summarization or query-focused summarization in which the summary is specifically expected to involve the information regarding a query. It can also be divided into different types according to a specific domain or genre to which the articles or documents are related.

Summarization can typically be done with two different methods, namely *extractive summarization* and *abstractive summarization*. The approach can be called extractive if the actual words, phrases or sentences are selected from the input and used as the summary. This approach is relatively simple for automated systems. As a result, there are many studies about this type of summarization technique. Prior to the deep learning techniques, there were many different approaches mostly based on rule-based techniques, graph-based approaches, statistical approaches, etc.

1.1. Abstractive Summarization

Abstractive summarization is another method of summarization, which aims to grasp the semantic information of a text and tries to construct a natural, reformed and novel summary that might consist of new words, phrases or sentences. An abstractive summarization system should understand and internalize the content of articles properly. The necessary concepts such as actions, actors, objects, locations, etc. should be accurate, and the cause-effect relationship should be preserved in the understanding. According to this understanding, the system should be able to generate acceptable words, phrases or even sentences that contain these important concepts.

The research on abstractive summarization followed a similar path to extractive summarization. Early extractive models include statistical approaches in which some pre-determined statistics are obtained from articles, such as the frequency of words, the similarity of sentences, the similarity of words with the title, the length of the sentences, the existence of named entities or numbers, etc. There were also graph-based studies in which the sentences or words were represented as nodes and the relation between them as edges. With several pre-determined rules and suitable statistics, some associations could be found and used to construct summaries. Machine learning approaches were used as well, such as supervised approaches like support vector machines, naive Bayes classifiers, decision trees; or unsupervised approaches like clustering, hidden Markov models, genetic algorithms, etc. Early works about abstractive summarization used these approaches along with external tools that helped to generate new words such as dictionaries, thesauri, semantic relation databases, etc. Currently, like extractive summarization, deep learning models are frequently used. Natural language generation becomes relatively easier with deep learning, which helps abstractive summarization a lot. Alongside generation, such models have more effective natural language understanding abilities too. Technical details of such abstractive models can be found in Chapter 5.

1.2. Motivation

With recent advances in technology, more and more information flow occurs on the Internet. Too many materials are unprecedentedly shared, such as news articles, blogs, scientific papers, reports, documentations, online books, and so on. In this digital age of overloaded information, the need for automatic summarization systems has naturally arisen. Inherently, people begin to need fast ways of retrieving and reading the main ideas of any textual material. We need to find and learn the related key information quickly when we desire. Retrieving and processing short summaries that preserve the important information and meaning can be done easily, effectively and efficiently compared to long texts.

Because it is relatively easier than abstractive summarization, researchers have proposed many ways for automatic extractive summarization. They have even achieved successful results over some widely-used metrics by preserving salient information and constructing grammatically correct sentences. However, extractive summarization systems do not produce verbally innovative summaries. It is not a humane way of summarizing, which means that people do not summarize in this way. Abstractive summarization, on the other hand, can produce novel sentences, which is how we all summarize. The idea of generating something from scratch with some deep learning and machine learning approaches can be exciting. Moreover, because of its innovative potential and the use of external knowledge, abstractive summarization has the capability of producing high-quality summaries.

The metrics for evaluating the effectiveness of summarization systems generally consider some forms of n-gram matches between the predicted and reference summaries. This helps extractive systems to achieve high results. Although this is the case, abstractive systems begin to achieve comparable results with the adaptations of deep learning techniques. Recent successes in deep learning have made abstractive summarization viable and attract many researchers' attention. For example, sequence-to-sequence models in which recurrent neural networks both read and generate novel text can be

adapted for abstractive summarization and such models can achieve state-of-the-art results.

Semantic information of words is very important in natural language understanding. The better the semantic representations are, the more effective understanding becomes. In order to make the representations better, researchers combine some additional features to them. One possible addition can be the syntactic information of words in a sentence. Using dependency parsing techniques to obtain and integrating the resulting word dependency relations to the models can be used for this purpose. We see that very few models were proposed that integrates word dependency relations in abstractive summarization. Their effectiveness on such models is not particularly analyzed. In this thesis, we integrated word dependency relations and analyze their effects on abstractive summarization models as a contribution. We have found that using dependency features indeed increases the performance of abstractive summarization tasks.

Recent deep learning models about language modeling use subword entities as tokens instead of whole words. These models have achieved state-of-the-art results and have quickly begun to be used widely. The popular subword tokenization algorithms separate words into meaningful subwords. They can successfully capture the morphological structures of words such as their base forms, prefixes, suffixes, etc. As a result, using subwords helps natural language understanding and generation of the models. However, there are not many studies using subwords in recurrent neural network based models in abstractive summarization. It is interesting to analyze their effectiveness in common abstractive summarization models. In this thesis, we used various subword models and analyzed their influences on these models as another contribution.

1.3. Thesis Outline

The organization of the thesis is as follows. The information that needs to be known as a background to understand the subsequent chapters is described in Chap-

ter 2. Some important studies related to abstractive summarization are mentioned in Chapter 3. The descriptions of frequently-used datasets can be found in Chapter 4. Chapter 5 contains the descriptions of the models and the additional mechanisms regarding abstractive summarization. It also includes the additions required for the analysis of word dependency relations and subword models. All of the related experiments, their results and our deductions are mentioned in Chapter 6. Lastly, we give a brief summary of the thesis and offer potential future works in Chapter 7. We also give some example summaries generated from our models in Appendix A.

2. BACKGROUND INFORMATION

In this chapter, we summarize some topics required for the rest of the thesis as background information. There are explanations related to deep learning, recurrent neural networks, sequence-to-sequence models, attention mechanism, dependency parsing, subword models, extractive and abstractive summarization.

2.1. Deep Learning and Neural Networks

With recent advances in deep learning, many researchers try to solve various problems in computer science using some varieties of deep neural networks. These problems can range from computer vision through bioinformatics, from robotics through finance, and so on. As another important subfield in computer science, NLP is no different. Morphological analysis tasks such as part-of-speech tagging or stemming, syntactic analysis tasks such as dependency or constituency parsing, other main topics such as named entity recognition, semantic role labeling, machine translation, speech recognition, sentiment analysis, word sense disambiguation, textual entailment, story generation, question answering, etc. are all suitable areas for the use of deep neural networks. New models related to each of these problems are being proposed frequently in recent years. Summarization is another important subfield that is considered among the ones described above.

Neural networks are basically collections of artificial neurons that compute the desired transformations based on their inputs and send signals to other neurons as outputs. The connection between them is typically called an edge. The edges have a weight that is used for a weighted summation for every input signal, and a non-linear function is applied. The whole network is trained for obtaining the optimum weight values for each edge based on some loss or objective function.

Neural networks can be trained with the use of large datasets. The examples in a dataset are processed by repeatedly trying to predict the actual (target) output. The difference between the predicted output and the target output is considered as the error. Typically, the loss functions are defined such that large errors give large values of loss, which means the lower loss values give better accuracy. By processing a dataset repeatedly, the model tries to minimize the loss with the use of some optimization techniques. In general, the gradient of the loss function over every weight is computed with the help of the chain rule. This provides how much error each weight compensates for the overall loss value. With these gradients, gradient descent algorithms (or some other methods) can be used to update each weight to decrease the loss. Overall, this method is called backpropagation and is at the heart of many deep learning model trainings.

2.2. Recurrent Neural Networks

The inputs to be used by the neural networks in NLP have a sequential nature. This means that letters, words or sentences exist consecutively in input texts. Even though the standard feed-forward neural networks are good for many application areas, they are not natively suitable for sequential tasks. For this purpose, Recurrent Neural Networks (RNNs) are used. These networks have an internal state that holds the information of the previously processed sequential elements, which are generally words (or subwords) in the context of many NLP subfields. This internal state can intuitively be regarded as a memory. Generally, the word embeddings are sequentially fed into an RNN step by step as inputs. At each step, RNN uses the internal state, among the other necessary vectors such as the word embedding vectors of the current step, to depend on the previous words. This effectively provides the ability to hold temporal information.

The parameters of RNNs are optimized using the backpropagation algorithm like the other neural networks. However, basic RNNs have the problem of vanishing gradient. Note that the backpropagation algorithm must be used for the flow of gradient

over the memory state as well. Since the input sequences can be very long, the gradients tend to get smaller and smaller as the RNN rolls up. At some point, because of the precision issues of floating numbers in computers, the gradient becomes zero. As a result, this causes no update for the model parameters. Even if the precision were not a concern, gradients would become so small that the update is very little, and, as a result, the training would take too long.

2.2.1. Long Short-Term Memory

There are several RNNs that can overcome the vanishing gradient problem. *Long Short-Term Memory (LSTM)* is one of the widely used networks for an RNN. The computations done by an LSTM cell can be seen in Equation 2.1 [1].

$$\begin{aligned} i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ g_t &= \tanh(W_g x_t + U_g h_{t-1} + b_g) \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\ c_t &= f_t * c_{t-1} + i_t * g_t \\ h_t &= o_t * \tanh(c_t) \end{aligned} \tag{2.1}$$

where $*$ is the element-wise multiplication operation (Hadamard product), $x_t \in \mathbb{R}^d$ is the input vector, $c_t \in \mathbb{R}^d$ is the cell state vector, $h_t \in \mathbb{R}^d$ is the hidden state vector (or the output vector of the LSTM), $i_t \in \mathbb{R}^d$, $f_t \in \mathbb{R}^d$, $g_t \in \mathbb{R}^d$ and $o_t \in \mathbb{R}^d$ are the input, forget, cell and output gate vectors respectively. Note that d is the dimension of these vectors and is often altered as a hyper-parameter in training. The sizes of the weight matrices W_i, U_i, W_f, U_f, W_g, U_g, W_o, U_o and the bias terms b_i, b_f, b_g, b_o are altered according to the hyper-parameter configuration of the value d and the size of the input vector. A diagram that represents the computation carried by an LSTM cell can be seen in Figure 2.1.

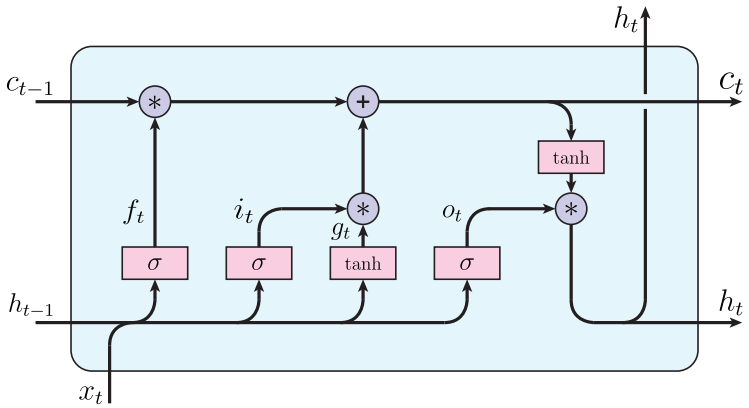

Figure 2.1. Long Short-Term Memory Cell.

2.2.2. Gated Recurrent Unit

Gated Recurrent Unit (GRU) is another widely used example of RNNs. The computations done by a GRU cell can be seen in Equation 2.2 [2]. Note that it is very similar to an LSTM cell. Because it lacks an output gate, it has fewer parameters to be trained.

$$\begin{aligned} r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\ z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\ n_t &= \tanh(W_n x_t + U_n(r_t * h_{t-1}) + b_n) \\ h_t &= (1 - z_t) * h_{t-1} + z_t * n_t \end{aligned} \tag{2.2}$$

where $h_t \in \mathbb{R}^d$ is the hidden state vector (or the output vector of the GRU), $r_t \in \mathbb{R}^d$, $z_t \in \mathbb{R}^d$ and $n_t \in \mathbb{R}^d$ are the reset, update and new gate vectors respectively. Same discussions regarding the value d, the weight matrices W_r, U_r, W_z, U_z, W_n, U_n and the bias terms b_r, b_z, b_n also holds for GRU. Figure 2.2 is the diagram that represents the computation carried by a GRU cell.

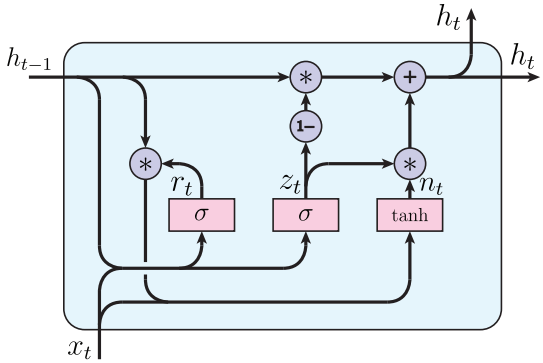

Figure 2.2. Gated Recurrent Unit Cell.

2.3. Influential Deep Learning Models in NLP

With the introduction of deep learning techniques, the models proposed for machine translation specifically shone out compared to the models of other NLP subfields. The problems related to other subfields are typically solved using models that are very similar to the state-of-the-art ones for machine translation. These models are adapted for the corresponding problems with small suitable changes. For example, researchers simply change the final computational layers or units so that the model becomes a classifier for sentiment analysis (instead of a word generator for machine translation). It turns out that even some simple adaptations give state-of-the-art results for their corresponding problems as well. Summarization is no different. The state-of-the-art models up to now are very similar to popular machine translation models with suitable adaptations for the summarization tasks and datasets. For this reason, it is important to observe the popular machine translation models.

2.3.1. Sequence-to-Sequence Encoder-Decoder Models

Most of the machine translation models that use neural networks consist of two sub-components, namely the *encoder* and the *decoder* [3]. Encoder networks first read the input text and encode it into some suitable intermediate representation, which is

commonly called the *context*.

$$\begin{aligned} h_t &= f(x_t, h_{t-1}) \\ c &= q(h_1, h_2, ..., h_n) \end{aligned} \tag{2.3}$$

where $h_t \in \mathbb{R}^d$ is the hidden state of the encoder at the time (or step) t and c is the context vector. Typically, the last hidden state of the encoder (i.e. h_n), or average-pooling over all of the hidden states of the encoder is used as the function q to compute the context vector.

Decoder networks, on the other hand, read this representation and generate the output words. It computes the probability of the output words (translation) by

$$p(\mathbf{y} \mid \mathbf{x}) = \prod_{t=1}^{m} p(y_t \mid y_1, y_2, ..., y_{t-1}, \mathbf{x}) \tag{2.4}$$

where $\mathbf{y} = (y_1, y_2, ..., y_m)$ and $\mathbf{x} = (x_1, x_2, ..., x_n)$. Each conditional probability term is modeled by an RNN as

$$p(y_t \mid y_1, y_2, ..., y_{t-1}, \mathbf{x}) = q'(y_{t-1}, s_t, c) \tag{2.5}$$

where $s_t \in \mathbb{R}^d$ is the hidden state of the decoder at the decoding step t. Note that the function q' is typically a (possibly multi-layered) feed-forward neural network.

The decoder constructs its hidden state in a similar way to the encoder. It is computed by

$$s_t = f(y_{t-1}, s_{t-1}, c) \tag{2.6}$$

which uses the previously generated word y_{t-1} instead of the input words and the context vector c as an addition.

An *encoder-decoder network* is basically defined as the combination of these two sub-components, which can be jointly trained. In the context of machine translation; the encoder reads the input words of a sentence in language A ($\mathbf{x}$) and converts it into a suitable representation (c); the decoder reads this representation and generates the resulting words as the potential translation of that sentence in language B ($\mathbf{y}$). Both of these two sub-components are typically RNNs. The encoder RNN reads the input word by word and the computations are conditioned over the previous words (or also following words if it has bidirectional nature). This makes the encoder handle sequential data. On the other hand, the decoder RNN generates a new word by conditioning over the previously generated words in addition to the encoder representations. This enables the decoder to handle sequential data as well. Overall, these kinds of models are typically called *sequence-to-sequence* models since the model reads sequences of words as the input and generates sequences of words as the output. A diagram representing a sequence-to-sequence model can be seen in Figure 2.3.

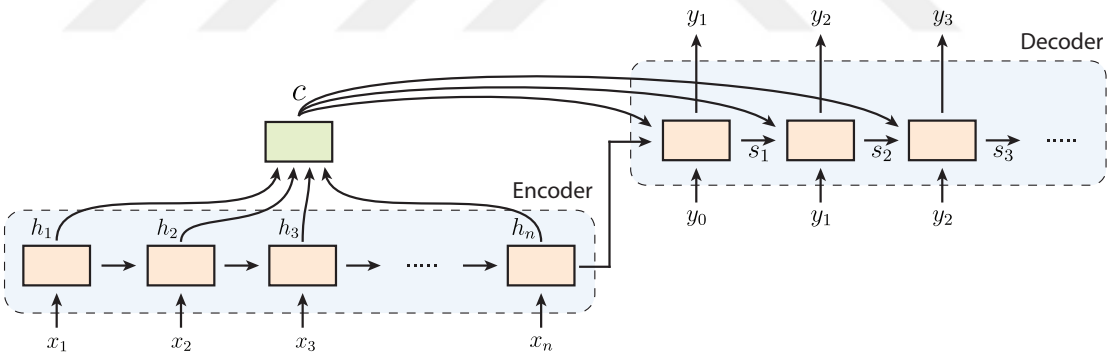

Figure 2.3. Sequence-to-Sequence Encoder-Decoder Network.

2.3.2. Attention Mechanism

The approach explained in Section 2.3.1 has a huge bottleneck. The representation constructed by the encoder is generally a fixed-sized vector. All necessary information needs to be compressed into this fixed vector. Since it has a fixed length, the longer the input sentences are, the more information loss there will be. The decoder cannot generate suitable translation when it cannot effectively condition over the source sentences to be translated. On the other hand, the decoder uses the same con-

text vector when generating different parts of the predicted translated sentences. This also brings another bottleneck for the decoder as it would be easier to condition over only the relevant parts of the source sentences while the generation of the translations as the irrelevant parts most likely cause errors.

In order to overcome these problems, researchers have proposed the idea of *attention*. The context vector does not need to contain all of the information. Instead, it should contain only the relevant parts of the source sentence. The attention mechanism can be regarded as a soft-searching over the words in a source sentence that has the most relevance [3]. Therefore, at each decoding step, the context vector gets dynamically changed using all of the previously computed hidden vectors of the encoder h_i. It is generally computed by

$$c_t = \sum_{i=1}^{n} \alpha_{ti} h_i \tag{2.7}$$

which is basically a weighted sum of the hidden states. The attention weights α_{ti} are computed by

$$\begin{aligned} e_{ti} &= a(s_{t-1}, h_i) \\ \alpha_{ti} &= \frac{\exp(e_{ti})}{\sum_{j=1}^{n} \exp(e_{tj})} \end{aligned} \tag{2.8}$$

where the function a is generally modeled as a simple feed-forward neural network. Note that this network can be jointly trained with the encoder and the decoder, which means that the backpropagation of the gradients flows through this model towards the encoder as well. In the context of machine translation, the function a can intuitively be regarded as the alignment model. By visualizing α_{ti} values corresponding to all of the source words x_i, we can have an intuitive way of thinking about what the attention mechanism actually does. It mimics how much focus over the parts of the source sentence there needs to be while generating a particular word for a potential translation.

2.4. Dependency Parsing

Dependency parsing is the process of obtaining *word dependency relations* in a sentence, which can be very important in speech and language processing systems. The relations are often represented as dependency parse trees. Dependency parsing focuses on the syntactic features of a sentence. Basically, it finds the dependencies between words of a sentence and analyzes the grammatical structure.

A dependency parse tree is a collection of dependency relations that consist of three elements, namely head, dependent (or modifier), and dependency type [4]. Head and dependent are the words that are linked with each other. The dependency type indicates the grammatical relation between the head-dependent pair. Typically, it is represented with three or four letter abbreviations with capital letters. For example, "NSUBJ" is used for a dependent word being a nominal subject of a head word. Similarly, "DOBJ" is used for a direct object, "NMOD" is for nominal modifier, "AMOD" is for adjectival modifier, "DET" is for determiner, etc. Researchers have constructed various taxonomies of different dependency types. With the introduction of Universal Dependencies formalism, many studies follow this same inventory of categories and guidelines for consistent annotations and structures.

By focusing on the syntactic structure of sentences, systems might perform better on their specific tasks. Therefore, the features obtained from the outputs of dependency parsing are used in some studies as an external aid. Note that there are a lot of software packages available that do automatic dependency parsing. Many of them capture not only dependency relations but also the part-of-speech tags of each word.

2.5. Subword Models

Tokenization is the process of splitting text into smaller chunks, which are generally the words and/or the punctuation. It is an important step that is almost always done as preprocessing. After tokenization, a vocabulary is constructed by combining

the unique words in the corpus. However, the size of the vocabulary becomes very large for big corpora as they simply include too many unique words. This causes very big embedding matrices. Considering the memory and computation time complexities, the size of the vocabulary should be trimmed down to a fixed value, and the least frequent tokens should be represented with an out-of-vocabulary token.

Instead of tokenizing the text into words, it can be tokenized into characters. The number of all unique characters in a corpus is much smaller. However, the models will have a harder time learning the context of each character. This means that the model will be too busy to capture the meaning of the words let alone the sentences. It is simply too fine-grained. Researchers thought that a hybrid approach can be used. Each unique word can further be divided into *subwords*. Several algorithms were proposed for this purpose, which try to set a balance between more vocabulary entries and further subword tokenization. It can be observed that the prefixes and suffixes, such as -ing, -s, -tion, -ly, etc., are captured as subwords with these algorithms. Note that all of these algorithms eliminate the possibility of out-of-vocabulary words.

2.5.1. Byte-Pair Encoding

Byte-Pair Encoding (BPE) [5] is an example of a subword tokenization algorithm. It starts by setting the vocabulary with entries that have only one character observed in the corpus. First, the corpus gets tokenized into words in a standard fashion, and the frequencies of each unique word are computed. Then, the algorithm repeatedly learns the merge rules. It counts the frequencies of all possible combinations of two characters in the vocabulary being consecutively seen in the corpus. The most frequent character pair is selected and the resulting subword is added to the vocabulary as a new entry. The algorithm iteratively does this until the size of the vocabulary reaches the desired number. With the learned merge rules, it is possible to tokenize into subwords when a new text is seen.

2.5.2. WordPiece

WordPiece [6] is another example of a subword tokenization algorithm, famously used for the BERT model [7]. It is very similar to BPE. Instead of choosing the most frequent character pairs, the algorithm chooses the pair that maximizes the likelihood of the training data if the corresponding pair is added to the vocabulary. The character pair with the greatest value of "the probability of two characters divided by the probability of the first character followed by the second character" gets selected. Similarly, this process is iteratively done until the size of the vocabulary gets the desired value.

2.5.3. Unigram Language Model

Unigram language model [8] differs from both of the previous algorithms. Instead of merging smaller chunks, it trims down bigger chunks into smaller ones to obtain smaller vocabulary. At each training step, the algorithm computes loss values over the whole training data given the current vocabulary and a unigram language model. It tries to find out how much the overall loss would increase if the symbol was to be removed from the vocabulary for each character in the vocabulary. After computing the loss values, typically 10-20% of the characters whose corresponding loss values are the lowest are removed. This means that the characters with the lowest effect on the overall loss get trimmed out. This process is done until the vocabulary is reduced to the desired size. There are no merge rules in this algorithm. In fact, there is more than one way to tokenize a text after training. The probability of each token in the training corpus is saved along with the vocabulary. Therefore, the probabilities of each possible tokenization can be computed.

2.6. Summarization

Summarization can be defined as the following [9]. Let the input consists of a sequence of n words, $\mathbf{x} = (x_1, x_2, ..., x_n)$. A summarizer takes $\mathbf{x}$ as an input and outputs another sequence of m words, $\mathbf{y} = (y_1, y_2, ..., y_m)$. Notice that n should be larger than

m. Given a scoring function s, a model is *extractive* if it tries to find

$$\underset{\mathbf{y} \in S}{\operatorname{argmax}}\, s(\mathbf{x}, \mathbf{y}) \tag{2.9}$$

where S is the set of all possible combinations of words, phrases or sentences obtained from the input. In contrast to that, a model is *abstractive* if it tries to find

$$\underset{\mathbf{y} \in Y}{\operatorname{argmax}}\, s(\mathbf{x}, \mathbf{y}) \tag{2.10}$$

where Y is the set of all possible sentences as summaries.

2.6.1. Extractive Summarization

Like the other subfields of NLP, text summarization using deep learning techniques is achieved by the adaptations of popular machine translation models. Instead of generating words of the target language for potential translation in decoding, an extractive summarization model is expected to output the relevant sentences in the input document. For a sentence-based extractive summarization, instead of Equation 2.9, the objective becomes finding

$$\underset{y \in \{0,1\}^{m'}}{\operatorname{argmax}}\, s(x, y) \tag{2.11}$$

where m' is the number of sentences required for the resulting summaries. Notice that this is basically a binary decision over every input sentence, whether to include that sentence in the summary or not.

Researchers have proposed many hierarchical encoders. For words in every sentence, standard word-level encoders are used. On top of the word-level encoder, there is also a sentence-level encoder that expects sentence representations obtained from the previous level as input. Generally, the concatenation of the first and last hidden states of the bidirectional word-level encoders is used as the sentence representations.

The decoders are not necessary for these kinds of models, instead, the main work is typically done by a classification layer. On top of the sentence-level encoder, a logistic layer makes binary decisions about whether a particular input sentence should be included in the resulting summary or not [10].

$$\begin{aligned} p(y_t = 1 \mid h_t, s_t, d) = \sigma(&W_1 h_t \\ &+ h_t^T W_2 d \\ &- h_t^T W_3 \tanh(s_t) \\ &+ W_4 p_t^a \\ &+ W_5 p_t^r \\ &+ b) \end{aligned} \tag{2.12}$$

where y_t is a binary variable indicating the tth sentence is in the resulting summary. h_t is the hidden state of the sentence-level encoder which corresponds to the tth sentence in the document, W_1, W_2, W_3, W_4 and W_5 are the weight matrices and the vector b is the bias term. The vector d is used for the overall document representation. It is generally computed as the average pooling of all hidden states. The vector s_t is the dynamic representation of the summary up to the sentence t. It can be computed by $s_t = \sum_{i=1}^{t-1} h_i p(y_i = 1 \mid h_i, s_i, d)$ [10].

Each term in Equation 2.12 has different contributions to the resulting probability estimation. The first term represents the content of the tth sentence in the input document. The second term captures the salience of that sentence with respect to the document. The third term models the novelty of that sentence regarding the current state of the summary (notice the minus sign before the term). The fourth term represents the importance of the absolute position of that sentence in the document and p_t^a is the encoding of the absolute position. Similarly, the fifth term represents the importance of the relative position of that sentence and p_t^r is the encoding of the relative position. Generally, the relative position values of the sentences are categorized into 10-15 classes, which are decided priorly.

2.6.2. Abstractive Summarization

An important difference between abstractive and extractive summarization is the inclusion of natural language generation techniques. They both share similar principles of natural language understanding with their use of encoders. However, extractive summarization tries to come up with a simple binary decision with this understanding whereas abstractive summarization simply generates new words from scratch. In this sense, abstractive summarization is very similar to machine translation since they both share these principles. Indeed, many abstractive summarization models are very similar to machine translation ones including their similarity of decoders or decoding techniques (e.g. beam search). Instead of generating words of the target language for potential translation in decoding, an abstractive summarization model is expected to generate words of the same language. Note that the resulting summaries should be substantially short and have the main information, of course.

3. RELATED WORK

In this chapter, we briefly explain some important studies related to abstractive summarization. Prior to deep learning, abstractive summarization was a big challenge for the researchers. Because of the absence of the current natural language understanding and natural language generation techniques, researchers tried to come up with statistical summarization models or graph-based approaches. After the advances of deep learning techniques, abstractive summarization has gained more focus as it is an interesting challenge to generate words and summaries from scratch.

Ganesan et al. proposed an unsupervised method to obtain "micropinion" (2-3 words-long opinions) from input texts [11]. They designed an optimization problem that forms higher-order n-grams step-by-step by scoring them in terms of their representativeness over the original input and readability. They achieved rather good results with their works. However, it was about opinions, not summaries, and the outputs were too short.

In order to construct new phrases or sentences, researchers often used the idea of word graphs. For example, Ganesan et al. [12] proposed Opinosis which constructs an associated word graph that represents all of the input sentences. By using the properties of the edges and the nodes, they constructed and ranked all valid paths with respect to their scores. The resulting paths were regarded as potential summaries. Lloret et al. [13] also used word graphs to generate new sentences by traversing the graph. Moawad et al. [14] constructed rich semantic graphs of the original documents instead of word graphs and repeatedly reduced the graph to have abstracted graphs. They used various NLP tools to obtain parse trees, dependencies, word senses, etc. With some heuristics they proposed, the graphs were reduced to obtain the resulting summaries.

Word graphs provide an effective way to produce new sentences. However, researchers often added additional methods to the path traversing to make it even more

effective. For example, Banerjee et al. [15] aligned similar sentences of different documents to form clusters and generated summaries from each cluster to obtain final summaries. Each sentence from the most important document, which was calculated using LexRank [16] and pairwise cosine similarity, was initialized as separate clusters. Other sentences from other documents were assigned to respective clusters based on their similarities. Then, word graphs were formed for each cluster. In the end, the best paths were obtained regarding the informativeness and the linguistic quality with the help of integer linear programming.

Another way to generate new sentences is the use of natural language generation tools. Khan et al. [17] defined a framework that uses predicate-argument structures by employing semantic role labeling. Each predicate-argument structure was compared and clustered using its features. At each cluster, the structures were ranked and selected. The resulting summaries are obtained by the SimpleNLG tool using the selected structures. Similarly, Genest et al. [18] proposed the concept of "Information Items", which is the smallest element of coherent information for a text. In their case study, they also used SimpleNLG to generate new sentences, which provided abstractiveness to their summarization model.

3.1. Studies Using Deep Learning

With the eyebrow-raising uses and successes of deep learning models, the researchers tend towards data-driven models with neural networks, which can be trained end-to-end. Indeed, Rush et al. [9] designed a fully data-driven model for abstractive summarization. They used a convolutional network to encode the source sentences and an attentional feed-forward neural network to generate the resulting summaries. Both networks were trained jointly. Their proposed model, which is called the Attention-Based Summarization (ABS) model, can also be used for any dataset consisting of document-summary pairs. Many subsequent works usually regard this model as a baseline model for any comparative evaluation over the Gigaword or the DUC datasets. This work was a very important landmark for abstractive summarization because of the

introduction of deep learning techniques over the rule-based extractive or abstractive summarization techniques used previously.

Nallapati et al. [19] proposed an attentional encoder-decoder RNN that is very much similar to the standard machine translation model by Bahdanau et al. [3]. They expressed that capturing the unknown words in the source sentences is one of the major problems in abstractive summarization. In addition to their model adaptation for abstractive summarization, they focused on this problem by adding a new mechanism called "Switching Pointer-Generator". Before generating a new word, the model decides whether to generate a new word ordinarily over the whole vocabulary, or it will choose a word that is in the source sentences but not in the vocabulary. This has the chance of generating a word that is not in the vocabulary and the writers thought that it mimics how a person produces summaries. Lastly, they also focused on the problem of a source document being too long, which might cause generating irrelevant summaries. They introduced a two-level attention mechanism for that. This hierarchical attention re-weights the word-level attention probabilities by the sentence-level attention values. By also attending over the sentences, the model focuses on the important parts of the document better.

Chopra et al. [20] used almost the same model as ABS. However, instead of using a feed-forward neural network for generation, they used RNNs for generation, specifically LSTM and Elman RNN architecture. They have achieved notable increases over the Gigaword dataset, mainly with the model using Elman structure, which is shortly mentioned in the literature as Recurrent Attentive Summarizer with Elman RNN (RAS-Elman).

Xu et al. [21] proposed a new method by decoupling the encoder and the decoder with the use of the doc2vec method. The doc2vec method, which is an extension of word2vec by Mikolov et al. [22], is a successful unsupervised document encoding method [23]. Feeding the doc2vec vectors of the input document to the decoder using several ways (e.g. concatenation, element-wise summation, via multi-layer perceptron,

etc.) leads to the summary sentences that are coherent and sensible considering the original documents. They noted that, while their method works well in-domain data, it does not work that well for out-of-domain data. They showed very similar scores compare to ABS for the Gigaword dataset.

Zhou et al. [24] implemented a selection mechanism on the encoder in addition to the standard sequence-to-sequence networks, which they shortly named SEASS. This mechanism produces second-level sentence representations between the encoder and the decoder and is achieved by a selective gate network. Their encoder consists of bidirectional GRUs. The last forward and the backward states are concatenated to construct the overall sentence representation. In the selective gate network, a multi-layer perceptron with a sigmoid activation function logically chooses whether a particular word should be selected or not using the hidden states of the encoder and the overall sentence representation. This intermediate network actually computes element-wise multiplication of the encoder hidden states to generate new ones. Lastly, they used an attention mechanism over these new hidden states and another GRU for the decoder to generate the resulting summary words. They showed significant increases in the ROUGE scores compared to ABS in both Gigaword and DUC datasets.

Tan et al. [25] pointed out the problem of too long documents being hard to train in deep learning techniques. They introduced a "coarse-to-fine" approach in which they first select the important sentences by using several well-known document summarization techniques and then using a hierarchical attention mechanism to generate the summary words. They used some popular extractive summarization methods, namely Lead, Luhn, LSA, LexRank, TextRank, SumBasic and KL-Sum, to identify the important source sentences. For each output of these methods, the model generates an overall representation vector with its LSTM encoders. A control layer, which is also an LSTM, works on top of these representations. In the decoder, they implemented a hierarchical attention mechanism, which considers the importance of these overall representations and the actual words of a particular extractive summarization technique mentioned above. Their scores are very similar to the RAS-Elman structure.

Li et al. [26] used both discriminative deterministic states and generative latent variables by introducing variational autoencoders in abstractive summarization. They stated that human-generated summaries have common structures of latent information and believed that this can improve the quality of the resulting abstractive summaries for any system. Traditional models compute the internal decoding states completely deterministically. The researchers used GRUs for the encoder and the decoder in a standard fashion. However, before generating the summary words, the hidden states were fed to variational autoencoders. The output is then fed as an additional feature in the generation step of summary words, which mimics the representation of the latent information. The whole model is jointly trained by using backpropagation. The scores are even higher than RAS-Elman, let alone ABS for both Gigaword and DUC.

Pasunuru et al. [27] considered using entailment generation techniques. They used a standard sequence-to-sequence encoder-decoder model consisting of LSTMs for both summary generation and entailment generation. It encodes the premise and decodes the entailed hypothesis via bilinear attention between them. In the end, multitask learning approach is used to share the knowledge generated by both of the networks with the help of the summary/entailment decoder. Two loss functions (for summarization and entailment generation) are optimized in training. The researchers showed that sharing the parameters of both networks improves the quality of the results and maintains salient information. The ROUGE scores are slightly lower than the RAS-Elman structure for the Gigaword dataset.

Paulus et al. [28] introduced a key attention mechanism for both encoder and decoder and integrated reinforcement learning methods. They used a simple bilinear function to temporally incorporate the hidden states of the encoder and the decoder. However, they normalized these attention weights with their previous ones while decoding, which results in penalizing the source words that were already attended to previously. In addition, they used a new intra-decoder attention mechanism, which also provides more information flow about the previously generated summary words. This is mainly because of the prevention of repeated phrases. In their model, they

also used the pointer/generator mechanism described by Nallapati et al. [19]. The overall model has a mixed training objective function that combines both the classical maximum-likelihood cross-entropy loss and policy learning. The policy learning maximizes the ROUGE scores of the sampled words since ROUGE favors more natural summaries [29]. Their results over the CNN/Daily Mail dataset are noticeably higher regarding the other abstractive models at that time.

See et al. [30] proposed a hybrid pointer/generator network along with a coverage mechanism. On top of the classical sequence-to-sequence encoder-decoder models, they implemented a switch that effectively decides whether to generate a new word from the vocabulary or point a word in the input to copy that word to the summary at each decoding step. This procedure is a very good way to solve out-of-vocabulary token problems in many abstractive summarization systems since out-of-vocabulary words are generally very important for resulting summaries. It differs from the switch proposed by Nallapati et al. [19] since the pointer/generator network from See et al. can also copy a token from the input that is already in the vocabulary. In addition, they implemented a coverage mechanism, which is basically the summation of the attention distributions computed in the previous decoding steps. Their results showed that using the coverage mechanism after the model is trained for some time is quite useful for not attending the same parts of the input sentences, which means avoiding repetition over the input parts. Both of their mechanisms are explained in Sections 5.2 and 5.3. The scores with the coverage mechanism are significantly higher than Nallapati et al. [19] for the CNN/Daily Mail dataset. Their works are widely used on the Internet, especially the specific splits of the training-validation-test set of the non-anonymized CNN/Daily Mail dataset.

Chen et al. [31] proposed a hybrid extractive-abstractive system. Their extractor agent first selects the salient sentences. It consists of LSTM, for global information, on top of a Convolutional Neural Network (CNN), for temporal information of the source sentences. Sentence selection is done by a pointer network, which is also an LSTM with attention. Their abstractor agent is similar to the Bahdanau network [3]

along with the pointer/generator mechanism. They separately trained the extractor and the abstractor. The whole model is optimized using policy gradient techniques by computing the ROUGE scores of the generated summary from the abstractor. If the extractor chooses a good sentence and the abstractor produces a good summary with a higher ROUGE score using that sentence, the action is favored by reinforcement learning. They showed a slight score improvement over See et al. [30] and very similar results with Paulus et al. [28]. They also expressed that their methods, as the name of their paper implies ("Fast Abstractive Summarization with Reinforce-Selected Sentence Rewriting"), are very fast in terms of training and testing time compared to the other similar works.

Cao et al. [32] underlined the problem of irrelevant or wrong facts generated as the resulting summaries by many models. To fix this, they integrated information extraction and dependency parsing techniques. By using OpenIE [33], they extracted facts of the source sentences as relation triples containing subject, predicate and object. These facts are used as an input to one encoder, similar to the input words, which are used for another encoder (both are biGRUs). With a dual attention mechanism, the decoder computes two context vectors for relations and words and combines them with the use of a gate (switch) network. The generation of the words is done according to this vector. Over the Gigaword dataset, the ROUGE scores are significantly higher than the RAS-Elman structure.

Amplayo, Lim et al. [34] used an off-the-shelf entity linking system to extract a sequential list of linked entities from the source sentences because they showed that some important entities are not on the resulting summaries by many systems. Some entities need to be selectively disambiguated by an RNN or a CNN using all other neighboring entities, and the hidden states of the networks can then be used for corresponding entities. The resulting vectors are used by a pooling sub-module which uses a firm attention mechanism that considers only top k entities. As a result, this sub-module constructs a topic vector. The topic vector is then concatenated with all of the decoder hidden states in each decoding step while generating summaries. They

achieved slightly better results compared to SEASS for the Gigaword dataset.

Liu et al. [35] proposed a generative adversarial network for abstractive summarization. The generative model consists of an LSTM encoder and decoder with an attention mechanism. The discriminative model is basically a binary classifier that tries to guess whether its input sequences, which are encoded by CNNs, are generated by a machine or not. They showed very similar results with Paulus et al. [28], except for ROUGE-2 scores which were almost 2 points better.

Lin et al. [36] considered the problems of repetition and semantic irrelevance and tried to solve these problems with some global encoding. They used a convolutional gated unit on top of an LSTM encoder, which makes the source context representation better by improving the connection of word representations with the global context. They noted that this unit finds some useful n-gram relations as well. In this unit, the researchers also implemented a self-attention mechanism that considers all of the encoder outputs. This encourages the model to learn long-term dependencies. The output of this model is concatenated with the hidden states of the RNN encoder with a gate (switch), and the decoder works using these states with a standard attention mechanism. Their scores are almost the same as SEASS with very few improvements.

Hsu et al. [37] tried to unify extractive and abstractive approaches. They showed that extractive approaches produce higher ROUGE scores compared to abstractive approaches, but lower readability. They used the then-state-of-the-art extractive model by Nallapati et al. [10] and the abstractive model by See et al. [30]. The extractive model, which is a classifier over the source sentences in its core, uses sentence-level attention whereas the abstractive model has a decoder with word-level attention. Their novel inconsistency loss function encourages the consistency between these two attentions. Both of their models were pre-trained separately beforehand. Then, they either trained the whole model in an end-to-end fashion or by using a two-staged approach. They showed slightly better results compared to See et al. [30], Paulus et al. [28] and Liu et al. [35] over the CNN/Daily Mail dataset.

All of the studies described above are generally for academic purposes, each tries to introduce their novel approaches. Fan et al. [38], on the other hand, proposed an extensible controllable model that is suitable for any live system. Their model consists of deep convolutional networks with word embedding layers and alternated convolutions with gated linear units. The decoder uses Bahdanau attention [3]. They achieved the length constraint specified by the user by adding additional tokens into vocabulary and prepending these tokens. Similarly, they also achieved entity and source constraints, which are used to focus only on these entities or sources in the resulting summaries. Lastly, they also explained a remainder summarization technique, which can be used to generate the summary of the remainder after reading some sentences in the document.

Song et al. [39] pointed that the models are not successful enough to keep the original meaning of the documents and do not include important words or relations. They added some syntactic constructions in order to overcome these problems. Their model is very similar to See et al. [30]. Additionally, they used the dependency parse tree of the sentences for the pointer/generator mechanism to consider the syntactic constructions better. After obtaining some features from the dependency tree, they proposed using two parallel attention for both words (semantic) and dependency features (structural). They achieved between 1.5-2 increase of ROUGE scores over the Gigaword dataset compared to the RAS-Elman structure and slightly lower results than Li et al. [26].

Wang et al. [40] incorporated topic information into their model which is a convolutional sequence-to-sequence model. They used latent Dirichlet allocation [41] to obtain topic embeddings. They fed the words and the topic to two parallel convolutional encoders. Then, a joint attention mechanism is used to combine them for decoding. Lastly, they utilized self-critical sequence training which is a policy gradient algorithm for reinforcement learning to maximize the ROUGE metric (similar to Paulus et al. [28]). Their results are slightly better compared to SEASS regarding both Gigaword and DUC datasets.

Çelikyılmaz et al. [42] proposed a different approach, which is called deep communicating agents. The encoding task is divided into subsections and handled by multiple agents communicating with each other. These encoder agents have two parts. The first part acts as a local encoder and consists of biLSTMs. The hidden states are passed to the second part which acts as a contextual encoder, which is also a biLSTM with several layers. The contextual encoder also takes the message sent by the other agents at each layer. These messages are then average-pooled. This makes each encoder agent to condition over them as well. Their decoder uses a hierarchical attention mechanism over the agents and the source words. They also proposed a pointer/generator mechanism working hierarchically over the agents using these attention weights as well. In addition to the classical negative log-likelihood, they added semantic cohesion loss to penalize the similarity between the hidden states of the end-of-sentence while decoding. Their three-agent model is slightly better than Paulus et al. [28] over the CNN/Daily Mail dataset, which they report as the optimal number of agents.

Cohan et al. [43] tried to deal with long documents by adding discourse-awareness to the system, which can specifically be used for research papers. They used a hierarchical encoder. The word embeddings for each section are fed to the "section RNN". The first and last hidden states are concatenated to get the section vectors. These vectors are then fed to the "document RNN". With the same principle, a document vector is obtained. In each decoding step, the model also attends to the relevant discourse section in addition to the actual words (the discourse-level attention weights of each section are multiplied with the corresponding word-level attention weights). The model also has the pointer/generator and coverage mechanism as explained in See et al. [30].

Li et al. [44] complained about the resulting summaries lacking key information. They thought that, without external guidance, the pointer/generator mechanism does not work optimally. They offered a combination of extractive and abstractive methods to solve this problem. First, they used the unsupervised TextRank algorithm [45] to extract keywords. These keywords are fed to "key information guide network", which

is an RNN, and the first and last states are concatenated to get the overall keyword vector. While calculating the attention weights of Bahdanau attention, they added this vector with a learnable weight parameter as an additional term. This vector is also used as another term in decoding and pointer/generator switch. Lastly, at test time, they used cosine similarity with the overall keyword vector to predict the extent of the key information covered up to that point in their beam search algorithm. They showed that the results are 1.5-2 points better than their baseline model with pointer/ generator over the CNN/Daily Mail dataset.

Xie et al. [46] also offered to include external approaches for improving semantic relevance by adding a WordNet-based sentence ranking algorithm. Their model has two LSTM encoders, which are for source words and extracted sentences. They combined these with a dual attention mechanism, which is basically a gate (switch) network running over the context vectors generated by both. The extracted sentences are obtained by a ranking algorithm that uses WordNet. For each sentence, they filtered the stop words and unambiguous tokens out, sorted the words regarding their number of senses, and kept the first n words. Then, they counted the common number of senses for each word as weights, and these weights were summed to get the weights for each sentence. The highest-rated n sentences are extracted and used as described above. Their model has the pointer/generator and coverage mechanisms as described in See et al. [30]. Their ROUGE scores are comparable to Paulus et al. [28] and some extractive models regarding the CNN/Daily Mail dataset.

4. DATASETS

There are two main datasets that are widely used for abstractive summarization, namely *Gigaword* and *CNN/Daily Mail*. In addition to these two, there are also minor datasets that are used at DUC shared tasks. They both are pretty basic, contain approximately 500 document-summary example pairs obtained from New York Times and Associated Press. The summaries are human-generated and there are four of them for each document. Because of the concerns of the computational resources at that time, the size of the datasets and the length of the summaries are limited. Therefore, they are not suitable for the use of training deep learning models. Instead, they can be used for evaluation purposes as Rush et al. also propose [9].

DUC shared tasks play an important role by setting a standard in the evaluation of abstractive summarization models. Their evaluation workflow uses ROUGE [29], which is a set of metrics for evaluating machine translation and summarization systems. It can be used to compare reference summaries with the summaries generated by systems. Typically, ROUGE-1, ROUGE-2 and ROUGE-L metrics are used. ROUGE-1 and ROUGE-2 consider the overlap of unigrams and bigrams respectively. ROUGE-L measures the longest matching sequences of words.

4.1. Gigaword

Rush et al. formed a new dataset using the annotated Gigaword dataset [9]. Originally, Gigaword has around 9.5 million news articles from various sources. Rush et al. used the headline of each article as the target (reference) summary, and the first sentence in the document as the input sentence, both of which form input-summary example pairs for the dataset. In this sense, it can be regarded as a sentence summarization dataset rather than a document or an article summarization dataset. The examples were then filtered using some heuristics. For example, they eliminated the examples in which the summary contains a question mark, a colon, byline or editing

marks, etc. This reduced the number of examples to approximately 4 million pairs.

Normally, the original annotated Gigaword dataset can only be obtained from Linguistic Data Consortium. However, Rush et al. shared the dataset on which their original preprocessing scripts applied [47]. Almost every paper using Gigaword uses either the same scripts as Rush et al. applied to construct the same dataset or uses the shared dataset directly for a fair evaluation. The preprocessing was done using Stanford CoreNLP tools. PTB tokenization and sentence separation utilities were used along with lower-casing all tokens. Additionally, they replaced all numeric characters with the character "`#`" and replaced the word types that are not seen more than five times with the "`UNK`" token.

The Gigaword dataset has around 3.8 million pairs in its training set, 190,000 pairs in its validation set and 2,000 pairs in its test set. The sentences in the dataset have an average size of 31.3 words whereas the summaries have 8.3 words. The vocabulary for the sentences in each pair consists of 119 million tokens and 110,000 unique word types. The vocabulary for the summaries (headlines) consists of 31 million tokens and 69,000 unique word types. It is a huge dataset and that makes it suitable for training deep learning models.

We think that the test set of Gigaword is not of high quality. The examples contain very odd sentence-summary pairs. For example, one summary contains only one token which is an out-of-vocabulary word considering the original vocabulary obtained by Rush et al. Moreover, the size is also very small. Therefore, we construct three more test sets and show the evaluation results of those as well. The first test set is obtained from the original training set by randomly picking 100,000 example pairs, which we called *Gigaword I*. The second one *Gigaword II* is similarly obtained from the original validation set by picking 100,000 pairs as well. The third one *Gigaword III* is constructed using both sets. Considering the ratio of the sizes of training and validation sets, we randomly picked 95,238 pairs from the training set and 4,762 pairs from the validation set ($95,238 : 4,762 \cong 20 : 1 \cong 3,800,000 : 190,000$). Note that,

for the sake of fair and proper validation and evaluation of the model, we removed the randomly picked examples from their corresponding sets. Constructing new test sets also makes the original validation set smaller, which was unnecessarily big. We also show the results of the original test set for better comparison of other studies.

4.2. CNN/Daily Mail

Nallapati et al. formed the CNN/Daily Mail dataset by modifying an existing dataset by Hermann et al. [19, 48]. The original dataset is for the task of passage-based abstractive question answering. It originally has human-generated summary bullets from articles published by CNN and Daily Mail websites. These summaries were transformed into questions by hiding one of the entities. In a question answering task, a system is expected to find the proper answer from the article for the generated fill-in-the-blank questions.

Nallapati et al. used the same web-crawling scripts to extract the original articles. With some simple modifications, they achieved article-summary pairs with no entities hidden. All summary bullets were used as multi-sentence summaries corresponding to an article. The CNN/Daily Mail dataset is a document summarization dataset rather than a sentence summarization dataset such as Gigaword.

The CNN/Daily Mail dataset has around 280,000 training pairs in its training set, 13,000 pairs in its validation set, 11,000 pairs in its test set. Even though it looks a lot smaller than Gigaword, the articles are very long so that it is also suitable for the training of deep learning models. There are 781 tokens and 29.74 sentences in the articles of the training set on average, whereas there are 56 words and 3.75 sentences as summaries. Note that there are actually two versions of CNN/Daily Mail. One of them is the non-anonymized version which consists of the actual entity names. The other one is the anonymized version in which the entities are replaced with some additional document-specific tokens with integer-ids to distinguish them. The non-anonymized version is used more than the anonymized one in summarization studies.

5. MODEL

In this chapter, we explain the models that we built for the experiments. We also describe some additional mechanisms suited for abstractive summarization, namely *pointer/generator* and *coverage*. In addition, we explain the uses of *word dependency features* and different *subword models* in the baseline models.

5.1. Baseline Model

The baseline model is basically an encoder-decoder sequence-to-sequence model. The encoder reads the input word embedding vectors $(e_1, e_2, ..., e_n)$ and transforms them into the hidden states $(h_1, h_2, ..., h_n)$. The encoder can be any type of RNN, but typically LSTMs or GRUs are used.

$$h_i = \text{LSTM}(e_i, h_{i-1}) \tag{5.1}$$

for all $i = \{1, 2, ..., n\}$. Note that h_0 is typically set as a zero vector.

The RNNs have a sequential nature, as described in Section 2.2, and the sequential nature is directional. This means that the input tokens are processed in one way, most of the time it is from left to right. Researchers often use bidirectional LSTMs or GRUs hoping that the hidden states capture the overall meaning of the whole sentence better. Bidirectional RNNs produce two different hidden states corresponding to each input token. One of them represents the pass from left to right $(\overrightarrow{h_1}, \overrightarrow{h_2}, ..., \overrightarrow{h_n})$. The other one represents the pass from right to left $(\overleftarrow{h_1}, \overleftarrow{h_2}, ..., \overleftarrow{h_n})$. These two different hidden states are generally concatenated with each other in order to unify the output of the encoder

as a whole.

$$\begin{aligned} \overrightarrow{h_i} &= \text{LSTM}(e_i, \overrightarrow{h_{i-1}}) \\ \overleftarrow{h_i} &= \text{LSTM}(e_i, \overleftarrow{h_{i+1}}) \\ h_i &= \overrightarrow{h_i} \oplus \overleftarrow{h_i} \end{aligned} \tag{5.2}$$

where $\oplus$ is the concatenation operation. Considering the hyper-parameter of hidden state size as d, $\overrightarrow{h_i} \in \mathbb{R}^d$, $\overleftarrow{h_i} \in \mathbb{R}^d$ and consequently $h_i \in \mathbb{R}^{2d}$.

The decoder predicts an output token step by step. It is a unidirectional LSTM or GRU network, which means that the words are generated one by one from left to right. At each step, the token generation is done with the help of a softmax layer. This layer makes the model estimate the token generation probability distribution over all of the tokens in the vocabulary.

$$p_{\text{vocab}}(w_t) = p(w_t \mid w_1, w_2, ..., w_{t-1}, c_t) = \text{softmax}(W_{\text{v}}(s_t \oplus c_t) + b_{\text{v}}) \tag{5.3}$$

where $c_t \in \mathbb{R}^{2d}$ is the context vector and s_t is the hidden state of the decoder RNN at step t. This expression is actually a single-layer feed-forward neural network with softmax as the activation function. $W_{\text{v}} \in \mathbb{R}^{v \times 3d}$ is the learnable parameter and $b_{\text{v}} \in \mathbb{R}^V$ is the bias term. Note that V is the vocabulary size.

The decoder updates its hidden state using the embedding vector of the previously generated summary token y_{t-1} and the context vector c_t.

$$s_t = \text{LSTM}(y_{t-1} \oplus c_t, s_{t-1}) \tag{5.4}$$

for all $t = \{1, 2, ..., m\}$. For s_0, a combination of $\overrightarrow{h_n}$ and $\overleftarrow{h_0}$ is used, hoping that they represent the whole input sufficiently. For this purpose, we can use

$$s_0 = \tanh(W_{s_0}(\overrightarrow{h_n} \oplus \overleftarrow{h_0}) + b_{s_0}) \tag{5.5}$$

which is basically a single-layer feed-forward neural network. The hidden state sizes of the encoder and the decoder can be different. However, if they both are set as d as a choice of hyper-parameter, the learnable parameter $W_{s_0} \in \mathbb{R}^{d \times 2d}$ and the bias term $b_{s_0} \in \mathbb{R}^d$.

The context vector c_t is computed with the help of the attention mechanism. Simply, the same equations in Section 2.3.2 can be used.

$$\begin{aligned} \alpha_{ti} &= \frac{\exp(e_{ti})}{\sum_{j=1}^{n} \exp(e_{tj})} \\ c_t &= \sum_{i=1}^{n} \alpha_{ti} h_i \end{aligned} \tag{5.6}$$

and the values e_{ti} are computed by

$$e_{ti} = V_a^T \tanh(W_a(s_{t-1} \oplus h_i) + b_a) \tag{5.7}$$

where $W_a \in \mathbb{R}^{2d \times 2d}$ and $V_a \in \mathbb{R}^{2d}$ are the learnable parameters and $b_a \in \mathbb{R}^{2d}$ is the bias term.

As usual, negative log-likelihood is used as the loss function in training.

$$\begin{aligned} \text{loss}_t &= -\log p_{\text{vocab}}(w_t^*) \\ \text{loss} &= \frac{1}{m} \sum_{t=0}^{m} \text{loss}_t \end{aligned} \tag{5.8}$$

where w_t^* is the predicted word at the decoding step t.

5.2. Pointer/Generator Mechanism

The model described in Section 5.1 is a basic summarizer with attention. Although it works better than most abstractive models, it has several important prob-

lems. Notice that the softmax function of Equation 5.3 converts the scores of each token given by the model into a probability distribution over every token in the vocabulary. This might seem convenient as the out-of-vocabulary tokens are represented by the vocabulary as a special "OOV" token. This approach is used for many sequence-to-sequence models. However, this is not viable in the context of summarization.

Out-of-vocabulary tokens, as their names imply, are the tokens that are very rarely seen if not never in the whole corpus. If the corpus is big enough, it is logical to assume that many verbs, adjectives, adverbs, or the tokens that belong to other parts of speech are not represented as "OOV", except nouns. Common nouns are generic names for items in classes or groups. Because of the generalization, it is also logical to assume that they have a high frequency in a corpus, which is not the concern in this case. Proper nouns, on the other hand, are specific names for particular people, places, or things. Therefore, they particularly have a high chance of low frequency in a corpus. As a result, they often get represented as out-of-vocabulary.

Summarization is generally applied to articles, and articles often describe events. The main constituents of an event are the performers, receivers and/or location of the action. These typically are proper nouns unless they are taken the place by pronouns. For a summary to have "OOV" tokens are useless because it lacks this important information. Summaries should contain as many informative tokens and as few non-informative tokens as possible, and proper nouns are often part of the informative ones. For a potential summary to be a qualitative one, the proper nouns in the source document should be present.

Pointer/generator mechanism is a way of generating out-of-vocabulary tokens with their original forms. To do that, we introduce a binary switch into the model. The probability of the switch is estimated by the value p_{gen} at each decoding step t [30].

$$p_{\text{gen}} = \sigma(W_g(s_t \oplus c_t \oplus y_{t-1}) + b_g) \tag{5.9}$$

where $W_g \in \mathbb{R}^{1\times(3d+E)}$ is the learnable parameter and $b_g \in \mathbb{R}$ is the bias term. E is the word embedding vector dimension and can be set as a hyper-parameter. Note that this is actually a single-layer feed-forward network with sigmoid as the activation function. With the concatenation operation, we unify the input of this network so that the probability of the switch being on or off depends on the hidden state of the decoder s_t, the context vector c_t and the embedding of the previously generated token, y_{t-1}.

The model generates a token w_t from the vocabulary if the value p_{gen} becomes 1, which means the switch is on. Conversely, it points to a particular token in the original input if the value becomes 0, which means the switch is off. When the model opts to point, it actually copies the token that is being pointed to into the potential summary. This means that the model now needs a mechanism for pointing over the input tokens. Indeed, the original attention mechanism can be used for this purpose. It produces a probability distribution over all of the input tokens regardless of the out-of-vocabulary tokens. If the switch becomes off, the model points to the token according to this same probability distribution.

Notice that the probability to generate a particular token is p_{vocab} and can be computed using Equation 5.3 previously. Now, in order to formulate the whole pointer/ generator mechanism, the estimation of the probability of generating a particular token is changed to

$$p(w_t) = p_{\text{gen}} p_{\text{vocab}}(w_t) + (1 - p_{\text{gen}}) \sum_{i:w_i=w_t} \alpha_{ti} \tag{5.10}$$

which still uses p_{vocab} in Equation 5.3 [30]. The term $\sum_{i:w_i=w_t} \alpha_{ti}$ collects all of the attention distribution values of the word w_t if it exists in the input document. With this term, the whole model gains the ability to generating the out-of-vocabulary tokens.

With the addition of the pointer/generator mechanism, the whole model graphically becomes as in Figure 5.1.

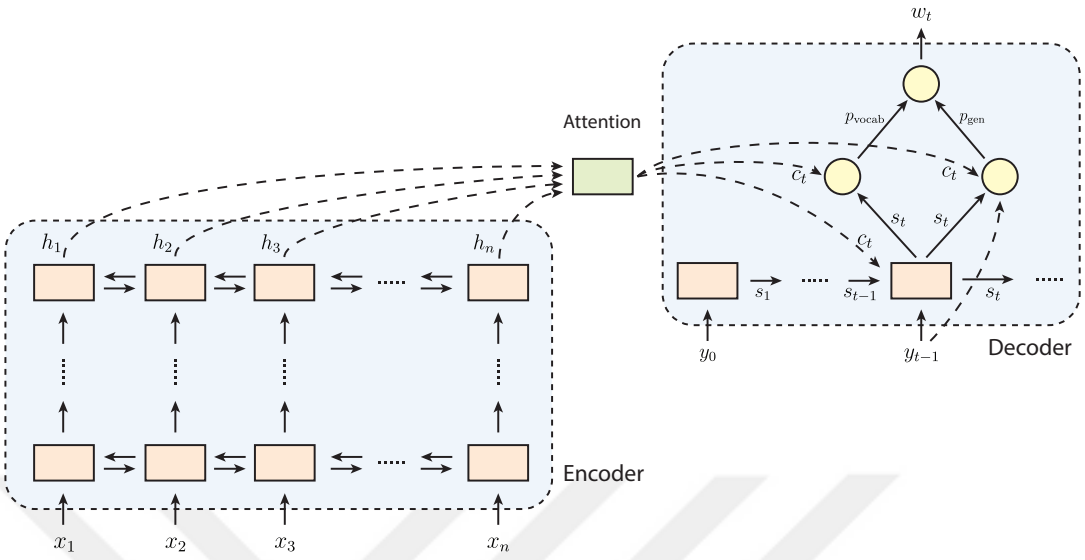

Figure 5.1. Sequence-to-Sequence Encoder-Decoder Model with Pointer/Generator Mechanism.

5.3. Coverage Mechanism

Another important problem with the baseline model is that the summary does not contain the important facts of the original input article. Instead, it contains repetitions of some words or phrases. Most of the encoder-decoder sequence-to-sequence models are designed for machine translation, as described in Section 2.3.1. For each token in the source sentence, machine translation models generally try to generate translated tokens. The lengths of the source and translated sentences usually have a ratio close to 1:1. However, summarization is not like this by definition. Using the same models often produces repetitive words or phrases, and causes to miss the important facts. The larger the input document is, the more likely it will be for the summaries to contain repetitions. This is even a problem for the Gigaword dataset, in which the inputs consist of only one sentence, let alone the CNN/Daily Mail dataset, in which the inputs are long articles.

Oftentimes, the model outputs the same tokens or phrases subsequently. In this case, It is usually deduced that the model is not trained properly and the weights

might not have optimal values. However, even if the model was forced to overfit over the dataset purposefully, these kinds of outputs could still be seen. Generating the same phrases subsequently makes the summaries contain the same facts more than once. For the inputs that are longer articles, it can even generate the same sentence more than once. This obviously is not suitable for summarization.

The problem occurs because the attention mechanism ignores the past distributions. In theory, only the hidden state of the decoder might help for this purpose. However, repetition still occurs regardless of the hidden state. Attention mechanism shows how much focus there needs to be on specific parts of the input. There is no need the focus on the same parts. For this purpose, a new coverage vector is defined as

$$\text{cov}_t = \sum_{t'=0}^{t-1} \alpha_{t'i} \tag{5.11}$$

which is basically the sum of attention distributions of the previous decoding steps [30]. We integrate this coverage vector into the attention mechanism, specifically in Equation 5.7, such that

$$e_{ti} = V_a^T \tanh(W_a(s_{t-1} \oplus h_i \oplus \text{cov}_t) + b_a) \tag{5.12}$$

which uses the coverage vector cov_t as an addition to the previous hidden state of the decoder s_{t-1} and the hidden states of the encoder h_i.

As See et al. stated, this modification alone does not produce successful results, as the model might opt to ignore this term while training [30]. Therefore, a new loss term is defined as

$$\text{loss}_{\text{cov}_t} = \sum_{i=1}^{m} \min(\alpha_{ti}, \text{cov}_{ti}) \tag{5.13}$$

which is upper bounded by 1 since α_{ti} is a proper probability distribution. In addition

to the modification of the attention computations, the original loss computation in Equation 5.8 is also modified.

$$\text{loss}_t = -\log p(w_t^*) + \lambda \text{loss}_{\text{cov}_t} \tag{5.14}$$

where λ is a hyper-parameter. See et al. suggested that the best value for it is 1 [30]. Higher values do not work as desired since the loss becomes too much to handle.

The coverage loss is actually used like a regularization term. It contributes to the overall loss more if both the coverage vector and the attention distribution for a particularly generated word at any decoding step becomes high. Two becoming high means that the model tries to attend over the same parts of the input. This forces the model to decrease the loss more in training and, as a result, the model will try to attend to different parts. The overall coverage mechanism pushes the model to cover all of the tokens in the input article by forbidding it to attend the same parts. This effectively makes the resulting summaries consider the overall semantic of input text better, and therefore, not miss the important facts and avoid repetitions.

5.4. Word Dependency Features

Word embeddings are widely used in deep learning. Almost every NLP-related networks have word embedding layers which every token is fed as inputs. A properly trained word embedding matrix captures the semantic relations successfully. Nevertheless, it does not capture the syntactic features. When we summarize documents, we might not analyze these features for every sentence we face in particular. However, we actually do it instinctively when we try to understand sentences. RNNs, because of their sequential nature, might capture these relations by themselves. Still, we can help further the models by integrating syntactic features of words in sentences.

We used dependency parsing to obtain a dependency parse tree for each sentence in the datasets. By using the properties of the tree and the results of the parsing, we

obtain five syntactic features:

- Part-of-speech tag,
- Label of the incoming edge,
- Token position in the sentence,
- Relative token position in the sentence,
- Depth in the parse tree.

The number of different part-of-speech tags and the label of the incoming edges are fixed, which means that it is easy to categorize them. We set different non-negative integers for each category and use an embedding layer in the model for each of these features separately. The relative position feature is a real number between 0 and 1. To categorize this continuous feature, we discretize this interval into 10 classes. For example, the interval $(0, 0.1]$ is the category `0`, $(0.1, 0.2]$ is `1`, and so on. Then, a different embedding is applied just like the previous features as usual. The remaining features are simply non-negative integers. Considering the biggest values of each feature in the datasets, we can decide on a safe maximum value to make these features bounded and easy to discretize. Similarly, different embeddings are applied for these features as well.

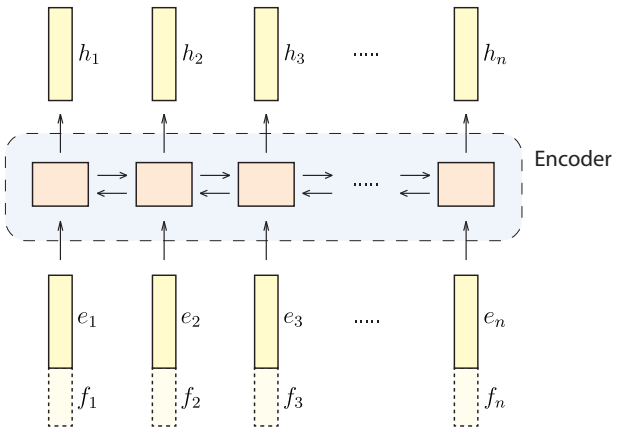

Figure 5.2. Incorporating Word Dependency Features with Inputs of Encoders.

After embedding these features for each token in the input, we concatenate the resulting embedding vectors to get an overall structural embedding vector f_i. This vec-

tor can be integrated into the model in two different places. First, the word embedding vector e_i and the structural embedding vector f_i can be concatenated and used as the input of the encoder

$$e_i = e_i \oplus f_i \tag{5.15}$$

which is visualized in Figure 5.2. Another possibility to integrate is the output of the encoder. This means that the hidden states of the encoder h_i and the structural embedding vector f_i can be concatenated

$$h_i = h_i \oplus f_i \tag{5.16}$$

which is visualized in Figure 5.3. The attention mechanism and the decoder can use these new states as if they are the original outputs of the encoder. By combining the structural features with one of these ways, we hope that the model depends on them as well as it depends on the semantic features.

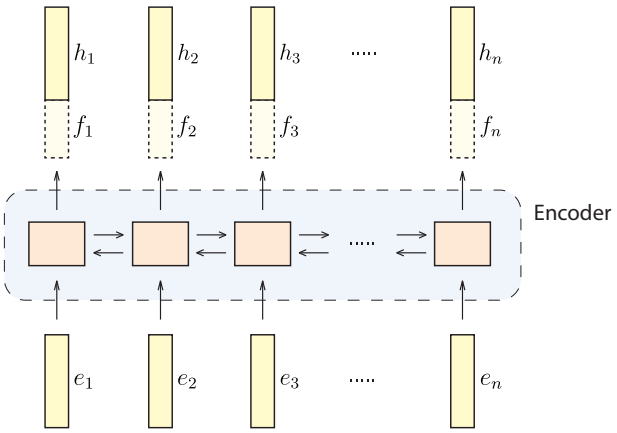

Figure 5.3. Incorporating Word Dependency Features with Hidden States of Encoder.

5.5. Subword Usage

Recently, subwords are very widely used in NLP-related deep learning models because of their contributions and successes. Many new models frequently use sub-

word models and achieve state-of-the-art results regarding their specific problems. For example, the famous BERT model uses a variation of WordPiece [7]. We find that not many studies in abstractive summarization use subwords in their models let alone analyzing their effect.

We used the `Tokenizers` library from Hugging Face [49], which provides implementations of many popular tokenizers. It provides functionalities to train new tokenizers and vocabularies from scratch. The library is designed for research and production purposes and is pretty fast. It also has several widely-used preprocessing tools.

We used byte-pair encoding, WordPiece and unigram language model as the subword models. We trained each of them separately with different vocabulary sizes and integrated them into the models. For all article-summary pairs in the datasets, we first used a Unicode normalization algorithm, namely NFD. Then, we applied lowercasing and stripped accents. After these normalization procedures, we replaced some of the special tokens in the datasets with suitable ones. For example, both of the datasets contain special tokens for parentheses, braces and brackets. Since the datasets are already tokenized, we used a simple tokenizer that uses white-space as the separator.

The resulting tokenizers and vocabularies can easily be used on any input text. They can encode the input text into a sequence of integer values based on the resulting vocabulary in order to be used as an input for the embedding layer. We used these values as the input to the encoder as if they were normal tokens. This means that e_i represents the embeddings of the subwords in Equations 5.1 and 5.2. The decoder, in the end, generates integer values corresponding to specific subwords in the vocabulary. We can easily decode them into their textual representations to obtain the whole resulting text.

6. EXPERIMENTS AND DISCUSSIONS

In this chapter, we show the results of the experiments we conducted for the effectiveness of different additions explained in Chapter 5. We also compare their successes and try to give our reasonings behind the results.

6.1. Preliminaries

Datasets. We used the same Gigaword dataset provided by Rush et al. [9]. The preprocessing workflow can be found in Section 4.1. On top of it, we clipped its input sentences to make them contain 100 tokens maximum, and the output summaries to 50 tokens. In addition, we used the non-anonymized version of the CNN/Daily Mail dataset through the use of the scripts provided by See et al. [30]. Similar to Rush et al. and See et al., we used PTB tokenization and lower-casing as preprocessing. We clipped the input articles to 400 and the output summaries to 100 tokens since it contains larger texts. For both of the datasets, we limited the vocabulary size to 50,000 most frequent unique tokens for the baseline models, and the rest of the tokens are set as out-of-vocabulary.

Word Embeddings. We used pre-trained GloVe vectors as the word embeddings. They were trained using the Wikipedia 2014 and the Gigaword 5 datasets, in which there are a total of 6 billion tokens and have a vocabulary size of 400,000 [50]. The word embedding vector dimension is 200. We chose to freeze the parameters of the word embedding layers in the baseline models while training, and use the same parameters for embedding layers of both encoder and decoder.

Model Architecture. The baseline models consist of 2-layer bidirectional GRU as the encoder and 1-layer unidirectional GRU as the decoder. Both of them have a hidden vector dimension of 256. The weights and biases for feed-forward layers are initialized by sampling from a uniform distribution with bounds $\pm\sqrt{\frac{1}{\text{input vector dimension}}}$.

The weights on RNNs are similarly initialized from a uniform distribution with bounds $\pm\sqrt{\frac{1}{\text{hidden vector dimension}}}$.

Training. We used a batch size of 16. The optimization algorithm that we used was Adagrad [51]. We set the learning rate to 0.15 and the initial accumulator value to 0.1 as its hyper-parameters. In order to overcome the problem of exploding gradient, we used gradient clipping with the value 2.0 as the threshold for each gradient with respect to every parameter. We used beam search for decoding the output summaries in evaluation, and the beam size was set to 4.

We trained the models on the NVIDIA DGX-1 server with Tesla V100 devices. Generally, the training of the models took approximately 2-3 days on average. The models with the pointer/generator converge earlier than the others. The models have around 30 million parameters, but it depends on various configurations. For constructing and training the models, we mainly used `PyTorch` which is an optimized tensor library for deep learning and can utilize GPUs for efficient training.

Word Dependency Relations. For the models using dependency features, we used `spaCy`, which is a library for NLP in Python [52]. It uses some of the latest studies for its functionalities. It was aimed to be used in real products and live environments. We used its part-of-speech tagging and dependency parsing models to get the corresponding predictions for each token in input sentences. Note that it provides the annotations that follow Universal Dependencies formalism.

Abbreviations Used in Tables. We used abbreviations to refer to various models we constructed for the sake of simplicity in the subsequent tables. *"Baseline"* model uses the same hyper-parameters and design choices explained up to this point. *"PG"* model uses the pointer/generator mechanism on top of *"Baseline"* model, as described in Section 5.2. *"COV"* model uses only the coverage mechanism on top of *"Baseline"* model, as described in Section 5.3. As its name also implies, *"PG+COV"* uses both of these mechanisms. Note that all of the other parts of these three models are the same

as *"Baseline"* model. We specifically compare their effectiveness in Section 6.2.

We also used different names for the models that use different hyper-parameters. *"Large Vocab"* model uses a vocabulary of size 100,000 instead of 50,000. In *"No Freeze"* model, the word embedding matrices can get updated in training. *"No Pretrain"* model does not use any pre-trained word embedding matrix. The embedding layers start to learn from scratch. For the parameters in the lookup table for word embeddings, they are initialized from a Gaussian distribution with 0 mean and 1 standard deviation. Lastly, *"LSTM"* model uses LSTMs instead of GRUs as the RNNs in both encoder and decoder. Similarly, all of the other parts of these models are the same as *"Baseline"* model. The names of the test sets are explained in Section 4.1.

Additionally, we show the results of the models that use the word dependency features in Sections 6.2, 6.3, and 6.4. All of the models have the corresponding ones that do not use dependency relations. In order to distinguish them, we used *"Dep"* prefix for the names of the models that use dependency features and kept the rest the same. For example, *"Dep Baseline"* model corresponds to *"Baseline"* model with the only addition of integration of dependency features. Similarly, *"Dep+PG+COV"* model is *"PG+COV"* model with dependency usage. *"Dep w/LSTM"* stands for dependency features with LSTMs as the RNNs instead of GRUs, and its corresponding model that does not use dependency features is *"LSTM"* model.

Lastly, we present the results of the models that use subwords in Section 6.5. Each model that uses a different subword model can be distinguished with a prefix. The prefixes are *"BPE"*, which stands for byte-pair encoding, *"WordPiece"* for WordPiece, and *"Unigram"* for unigram language model. *"BPE No PG+COV"* model is similar to *"Baseline"* model. It does not use any additional mechanisms. The only difference is that it uses BPE subwords instead of words as the input. *"BPE"* model uses the pointer/generator and coverage mechanisms on top of *"BPE No PG+COV"* model. The only difference between *"BPE w/LSTM"* and *"BPE"* models is that the former uses LSTMs as RNNs instead of GRUs. Note that for all these models, the subword

algorithm is used to generate a vocabulary size of 16,000 unique subwords. *"BPE 30k"* model, on the other hand, uses 30,000 as the vocabulary size on top of *"BPE"* model. The models using other subword algorithms can be distinguished with corresponding prefixes instead of *"BPE"*.

Note that for each table, R-1, R-2 and R-L stand for the F_1 scores of ROUGE-1, ROUGE-2 and ROUGE-L scores respectively. We used the `pyrouge` library, which is a Python wrapper for the original ROUGE summarization evaluation package, to obtain these scores [53].

6.2. Dependency Features on Abstractive Summarization Models

6.2.1. Gigaword Experiments

Table 6.1 contains the ROUGE scores of the models with different mechanisms evaluated by using all of the Gigaword test sets explained in Section 4.1. First of all, the table shows that there are big differences in ROUGE scores between *"Gigaword Original"* and the other custom test sets. The models evaluated by the custom ones generally have consistent results with each other. We randomly selected the examples as explained in Section 4.1. This provides that the test sets resemble the training (and validation) set well. As a result, these three custom test sets actually have very similar characteristics, and these consistent results are expected. The original test set, however, is very different. It might not resemble the whole dataset like the others do since it only contains a little less than 2,000 examples.

We first give our observations of the models that do not use dependency features (whose names do not start with *"Dep"*) in Table 6.1. For all models, ROUGE-1 scores are between around 39-41.3, ROUGE-2 scores are between 16.8-18.5, and ROUGE-L scores are between 36.7-38.9 for all of the custom Gigaword test sets, except *"COV"* model. It can only achieve around 28, 9 and 26.5 for each ROUGE score respectively, which are significantly lower scores than the other models for all test sets. This ten-

Table 6.1. The ROUGE scores of the models with different additional mechanisms over the Gigaword test sets.

Gigaword Original							
Model	**R-1**	**R-2**	**R-L**	**Model**	**R-1**	**R-2**	**R-L**
Baseline	29.889	12.402	27.889	Dep Baseline	30.172	12.580	28.019
PG	**31.169**	**13.333**	**29.142**	Dep+PG	**31.711**	**13.403**	**29.571**
COV	21.645	6.718	20.211	Dep+COV	20.468	6.112	19.782
PG+COV	30.494	13.310	28.758	Dep+PG+COV	30.986	13.348	28.998
Gigaword I							
Model	**R-1**	**R-2**	**R-L**	**Model**	**R-1**	**R-2**	**R-L**
Baseline	39.019	17.003	36.708	Dep Baseline	39.274	17.513	37.207
PG	**41.007**	**18.264**	**38.511**	Dep+PG	**41.747**	**19.941**	**39.004**
COV	27.962	9.022	26.214	Dep+COV	26.325	9.007	26.179
PG+COV	39.322	17.792	37.217	Dep+PG+COV	39.937	18.480	37.717
Gigaword II							
Model	**R-1**	**R-2**	**R-L**	**Model**	**R-1**	**R-2**	**R-L**
Baseline	39.574	17.383	37.209	Dep Baseline	39.814	17.917	37.617
PG	**41.331**	**18.569**	**38.932**	Dep+PG	**41.926**	**19.264**	**39.479**
COV	28.654	8.966	26.847	Dep+COV	28.614	8.938	26.848
PG+COV	39.713	18.159	37.501	Dep+PG+COV	40.516	18.730	38.167
Gigaword III							
Model	**R-1**	**R-2**	**R-L**	**Model**	**R-1**	**R-2**	**R-L**
Baseline	38.975	16.865	36.732	Dep Baseline	39.157	17.422	37.067
PG	**41.265**	**18.563**	**38.809**	Dep+PG	**41.123**	**18.490**	**38.946**
COV	28.324	8.899	26.557	Dep+COV	28.394	8.878	26.547
PG+COV	39.055	16.852	36.754	Dep+PG+COV	39.617	17.655	37.236

dency can also be observed in the other results shared below. All models except *"COV"* can achieve scores between 29.9-31.1, 12.4-13.3, 27.9-29.1 respectively when evaluated by using *"Gigaword Original"*. Even though the models achieve lower ROUGE scores for the original Gigaword test set, the differences between the scores of different models are similar.

Compared to *"Baseline"* model, we can see that *"PG+COV"* model increases all of the ROUGE scores by around 0.1-0.9. This is a rather good improvement since even the slightest increase in any ROUGE score is considered noteworthy in the context of abstractive summarization. However, the performance expectation might rightfully be higher than this since it contains all of the new mechanisms we explained in Chapter 5. It can also be observed that *"PG+COV"* model has lower ROUGE scores than *"PG"* model which performs the best for all of the test sets. It is surprising since *"PG+COV"* model has one more additional mechanism that we hoped to make the performance better. Moreover, it is obvious that *"COV"* model produces significantly worse results compared to the others. Because of these situations, we can conclude that the coverage mechanism decreases the performance of the models. Further explanations about the situation can be found in Section 6.2.3.

The models that use dependency features (whose names start with *"Dep"*) in Table 6.1 have a similar tendency in terms of the ROUGE scores. Excluding *"Dep+COV"* model, each model can achieve between 39.2-41.9 as ROUGE-1 scores, 17.4-19.9 as ROUGE-2, 37-39.5 as ROUGE-L for the custom test sets. It is clear that *"Dep+COV"* model still gives very bad results (approximately 28, 9, 26.5), which is similar to *"COV"* model. The scores are in between around 30.2-31.7, 12.5-13.4, 28-29.5 respectively when the models are evaluated with *"Gigaword Original"*.

Each model performs very similarly to its counterparts that do not use dependency features. Apparently, *"Dep+PG"* model produces the best results for every ROUGE score in every test set. *"Dep+PG+COV"* model achieves worse results than *"Dep+PG"*, but it surpasses *"Dep Baseline"* model despite the coverage mechanism.

"Dep+COV" clearly performs the worst, even worse than *"Dep Baseline"* model.

The models that use dependency features achieve better results than their corresponding models that do not use the features. This is true with an exception of the models with the coverage mechanism. *"Dep+COV"* model generally produces even lower results than *"COV"* model. However, we can consistently see notable increases in the other scores for all of the test sets. Generally, the addition of dependency features helps the models increase the ROUGE-1 score by around 0.3-0.8, ROUGE-2 by around 0.1-0.7, ROUGE-L by 0.2-0.5.

Table 6.2. The ROUGE scores of the models with different additional mechanisms over the CNN/Daily Mail test set.

CNN/Daily Mail							
Model	**R-1**	**R-2**	**R-L**	**Model**	**R-1**	**R-2**	**R-L**
Baseline	31.183	11.769	28.429	Dep Baseline	32.047	11.990	28.718
PG	**34.914**	**14.934**	**32.025**	Dep+PG	**34.807**	**14.072**	**31.198**
COV	25.406	9.923	24.101	Dep+COV	25.278	6.942	22.135
PG+COV	33.023	13.886	30.353	Dep+PG+COV	34.038	14.022	30.989

6.2.2. CNN/Daily Mail Experiments

We first give our observations of the models that do not use dependency features in Table 6.2. *"COV"* model gives significantly lower results for all ROUGE scores in the CNN/Daily Mail test set as well. It achieves 5.7, 1.8 and 3.7 less than *"Baseline"* model for each ROUGE score respectively. The conclusion about the coverage mechanism decreasing the performance can also be observed in Table 6.2. *"Baseline"* model for CNN/Daily Mail gives very similar results to its corresponding model for *"Gigaword Original"*. It can achieve better ROUGE-1 and ROUGE-L scores, but worse ROUGE-2 score. The remaining models can achieve scores higher than 33, 13.8 and 30.3 for each score respectively. *"PG"* model performs the best for CNN/Daily Mail as well

as Gigaword. It produces 3.8 higher score than *"Baseline"* model for ROUGE-1, 3.2 higher for ROUGE-2, 3.6 higher for ROUGE-L. Although *"PG+COV"* also performs much better than *"Baseline"* model, it cannot surpass *"PG"* model. This behavior was also observed in Table 6.1. Overall, The differences between each score are very similar to their counterparts using Gigaword.

Looking at Table 6.2, the tendency of the scores achieved by the models using dependency features is similar to the ones that do not use dependency features. *"Dep Baseline"* model achieves 32, 12 and 28.7 as ROUGE scores respectively. *"Dep+PG"* and *"Dep+PG+COV"* models perform similarly to each other. Still, they can achieve scores around 34, 14 and 31. *"Dep+PG"* model performs the best, and *"Dep+COV"* model performs the worst. *"Dep+COV"* model can only produce 25.2, 6.9 and 22.1, which is the worst performance.

The addition of dependency features seems to help the models for CNN/Daily Mail as well. We can even see an improvement of ROUGE-1 score as high as 1. In general, these models achieve 0.3-0.8 higher scores than their counterparts with no dependency features. The most obvious effect of adding dependency features can be observed by comparing the scores of *"Baseline"*-*"Dep Baseline"* and *"PG+COV"*-*"Dep+PG+COV"* models. The bad effect of coverage can also be seen from both *"COV"* and *"Dep+COV"* models, which can also be observed in Table 6.1.

6.2.3. The Effects of Additional Mechanisms

With the observations we made, it is clear that integration of the pointer/ generator mechanism increases the performance. Every model that has this mechanism clearly achieves better results compared to the baseline models. This shows its usefulness in an abstractive summarization task. With its addition to any model, the ability to generate out-of-vocabulary words is gained. For a summarization task, it is vital to include them in the resulting summaries as explained in Section 5.2. As it can be observed from the results, the models can achieve much better ROUGE scores with

this ability.

On the other hand, the coverage mechanism does not seem to be useful since it decreases the scores drastically. This is actually a true statement but a misleading one. The actual reason for the low scores is due to the nature of coverage. By definition, it is actually a regularization term that penalizes the model if it tries to attend the same parts of the input repeatedly. At the starting time of the training from scratch, the model knows absolutely nothing about the nature of the dataset and abstractive summarization task. We penalize the model by introducing the coverage term to the loss. Therefore, it becomes hard to train the model to get lower loss values. This is why the models with coverage mechanism produce inconsistent results in Tables 6.1 and 6.2. Note that despite the coverage mechanism, *"PG+COV"* and *"Dep+PG+COV"* models can still surpass the baselines, which also shows the improvement provided by the pointer/generator mechanism alone.

The coverage mechanism specifically shines by getting activated after the training reaches a decent point without using it. When the parameters are adequately optimized, enabling the coverage mechanism and continuing the training for a little while increases the performance. It forces the models to avoid generating repeated phrases and cover all of the input sentences, which helps the summaries not to miss the important facts and avoid repetition, as described in Section 5.3. This obviously ensures us to achieve better scores since the predicted summaries will have more similarities to the reference summaries. Indeed, the model we trained considering the proper usage of coverage mechanism, which we called *"Dep+PG→COV"* model, has the highest ROUGE scores compared to the other models that we constructed, as can be seen in Section 6.6.

6.2.4. The Effects of Dependency Features

Generally, the integration of dependency features to each model increases almost every ROUGE score. The only exception seems to be the models only with the

additional coverage mechanism, namely *"Dep+COV"*. It produces worse results for *"Gigaword Original"* and *"Gigaword I"*, and similar or slightly better results for *"Gigaword II"* and *"Gigaword III"* in Table 6.1. The other models generally have increased ROUGE scores by around 0.5-1, specifically for *"Gigaword I"* and *"Gigaword II"*. Even though the scores seem to be slightly worse for *"Dep+PG"* model in *"Gigaword III"*, the ROUGE-L is slightly better than *"PG"* model. These behaviors can similarly be observed for CNN/Daily Mail in Table 6.2.

The improvements of the scores show the benefit of integrating dependency features into abstractive summarization models. When we try to read and understand sentences, we instinctively observe the syntactic structures because of our knowledge of that particular language. This helps our understanding and makes us grasp the main ideas easier. As a result, we can construct novel words, phrases and sentences to form a whole summary. Actually, a similar process is followed for these models. Along with the semantic features, the encoder can learn the syntactic structure of words in sentences. It can understand the important parts of the sentences easier with the integration of dependency features and generate its own hidden states more efficiently. The decoder can then generate words better along with the attention mechanism which uses these better encoder hidden states.

Another case to note is the difference between *"Dep+PG"* model and *"Dep Baseline"* model for both Gigaword and CNN/Daily Mail test sets. Regarding the CNN/ Daily Mail test set, the addition of the pointer/generator mechanism increases the scores of *"Dep Baseline"* model by around 2.8, 2.2 and 2.5 respectively for each ROUGE score. However, for the Gigaword test sets, even though it is still impressive, the increases of the scores of corresponding models are at most around 2 for each score. It is clear that the pointer/generator mechanism works better for the CNN/Daily Mail dataset. This is due to the fact that the CNN/Daily Mail dataset contains much longer input texts compared to the Gigaword dataset as explained in Section 4.2. It is difficult for *"Dep+Baseline"* model to generate summaries for the CNN/Daily Mail test set. After all, *"Dep+Baseline"* model shows many resemblances to standard machine trans-

lation models. It is designed for the input and output length ratio to be 1:1. However, CNN/Daily Mail expects a much asymmetrical ratio. On the other hand, the Gigaword dataset also has an asymmetrical ratio, but the input texts are much shorter. The models can learn the important phrases or facts of shorter inputs compared to the long ones. With the pointer/generator mechanism, *"Dep+PG"* and *"Dep+PG+COV"* models have better scores for both CNN/Daily Mail and Gigaword test sets. However, this mechanism helps *"Dep+PG"* and *"Dep+PG+COV"* models more for CNN/Daily Mail than Gigaword.

6.3. Dependency Features with Different Hyper-Parameter Configurations

6.3.1. Gigaword Experiments

In addition to the effectiveness of the mechanisms, we also compare the influences of some important hyper-parameter configurations. In Table 6.3, the related results can be found for each Gigaword test set. First, we mention the models that do not use dependency features. Each ROUGE score is close to the scores of *"PG+COV"* model in Table 6.1, which is explained in detail previously. We wanted to use this model as a baseline for Table 6.3 since the others are also based on this model. Note that the scores for custom test sets are significantly higher than the original test set, as observed in Table 6.1 as well.

In general, *"No Freeze"* model achieves the best results compared to the others, including *"PG+COV"* model. Its scores are around 1.2-1.5, 0.5-1.1 and 0.5-1.2 more than *"PG+COV"* model for custom test sets for each ROUGE score respectively. It also increases the ROUGE-1 and ROUGE-L scores for *"Gigaword Original"*. *"Larger Vocab"* model can also achieve slightly better results than *"PG+COV"* model. However, *"No Pretrain"* model seems to show rather inconsistent behavior. Most of the time, the scores are lower than *"PG+COV"* model when evaluated using the test sets, including the original one. Lastly, *"LSTM"* model also achieves good results compared to *"PG+COV"* model. It can perform almost as high as *"No Freeze"* model.

Table 6.3. The ROUGE scores of the models with different hyper-parameter configurations over the Gigaword test sets.

Gigaword Original							
Model	**R-1**	**R-2**	**R-L**	**Model**	**R-1**	**R-2**	**R-L**
PG+COV	30.494	13.310	28.758	Dep+PG+COV	30.986	13.348	28.998
Large Vocab	30.650	13.118	28.599	Dep+Large Vocab	31.381	13.622	29.312
No Freeze	**31.125**	13.152	29.070	Dep+No Freeze	**31.606**	13.343	29.325
No Pretrain	30.119	12.092	28.048	Dep+No Pretrain	30.767	12.175	28.317
LSTM	30.993	**13.760**	**29.412**	Dep w/LSTM	31.562	**13.628**	**29.509**
Gigaword I							
Model	**R-1**	**R-2**	**R-L**	**Model**	**R-1**	**R-2**	**R-L**
PG+COV	39.322	17.792	37.217	Dep+PG+COV	39.937	18.480	37.717
Large Vocab	39.882	18.387	37.799	Dep+Large Vocab	40.517	18.918	38.693
No Freeze	**40.540**	**18.886**	**38.362**	Dep+No Freeze	40.690	**19.581**	**38.817**
No Pretrain	39.494	16.904	37.073	Dep+No Pretrain	39.897	17.721	37.602
LSTM	40.284	18.190	37.879	Dep w/LSTM	**40.796**	18.957	38.486
Gigaword II							
Model	**R-1**	**R-2**	**R-L**	**Model**	**R-1**	**R-2**	**R-L**
PG+COV	39.713	18.159	37.501	Dep+PG+COV	40.516	18.730	38.167
Large Vocab	40.366	18.354	38.030	Dep+Large Vocab	41.174	19.004	39.103
No Freeze	**41.282**	**18.637**	**38.707**	Dep+No Freeze	**41.986**	**19.323**	**39.447**
No Pretrain	40.273	17.400	37.712	Dep+No Pretrain	40.989	18.136	38.258
LSTM	40.410	18.357	38.234	Dep w/LSTM	41.180	18.906	38.831
Gigaword III							
Model	**R-1**	**R-2**	**R-L**	**Model**	**R-1**	**R-2**	**R-L**
PG+COV	39.055	16.852	36.754	Dep+PG+COV	39.617	17.655	37.236
Large Vocab	39.433	17.466	37.222	Dep+Large Vocab	40.275	17.993	38.271
No Freeze	**40.530**	**17.660**	**38.121**	Dep+No Freeze	**41.097**	18.475	**38.646**
No Pretrain	39.572	16.795	37.118	Dep+No Pretrain	40.228	17.657	37.434
LSTM	39.684	17.447	37.919	Dep w/LSTM	40.847	**18.621**	38.408

Integrating dependency features to the models increases the scores as expected. *"Dep+No Freeze"* model can achieve 40.7-41.9, 18.5-19.6 and 38.6-39.5 ROUGE-1, 2 and L scores over the custom test sets. Overall, we can say that it performs the best among the other models with some exceptions in Table 6.3. *"Dep+Large Vocab"* and *"Dep w/LSTM"* model have similar results, which are slightly less than *"Dep+No Freeze"* model. For *"Gigaword Original"* and *"Gigaword I"*, *"Dep w/LSTM"* model can even surpass *"Dep+No Freeze"* model in several types of ROUGE scores. *"Dep+No Pretrain"* model performs slightly worse than *"Dep+PG+COV"* model for the original test set. Generally, it again produces inconsistent results when evaluated using the custom test sets.

Similar to the tendency observed in Table 6.1, integration of dependency features makes the models achieve better results. Literally, all of the ROUGE scores of the models that use dependency features for every test set are better than the corresponding models that do not use those features. The difference between *"Large Vocab"* and *"Dep+Large Vocab"* models is noticeably higher than the other differences. The dependency features help *"Dep+Large Vocab"* model more than the others so that it can perform as well as *"Dep+No Freeze"* model. ROUGE-L scores of *"Dep+Large Vocab"* can even increase as much as 1. Even *"Dep+No Pretrain"* model can perform noticeably better than *"No Pretrain"* model. On average, it seems that dependency feature usage increases the scores by around 0.2-0.8, 0.5-1.2 and 0.3-1 for each ROUGE score respectively over the custom datasets. The increases of ROUGE-1 and ROUGE-L scores are also significant for the original test set.

6.3.2. CNN/Daily Mail Experiments

We also show the results of the same experiments in Table 6.3 for CNN/Daily Mail, which can be seen in Table 6.4. Considering the models that do not use dependency features, the tendency is very similar. *"No Freeze"* model performs the best compared to the others. It can achieve 1.9, 1 and 1.6 more than *"PG+COV"* model for each ROUGE score respectively, which is a quite significant improvement. *"Large*

Vocab" model also performs better than *"PG+COV"* model in terms of ROUGE-1 and ROUGE-L scores, but slightly worse for ROUGE-2. *"LSTM"* model achieves ROUGE scores between *"Dep+Large Vocab"* and *"Dep+Large No Freeze"* models. The same situation about *"No Pretrain"* model performing worse than *"PG+COV"* model can also be seen in Table 6.4.

Table 6.4. The ROUGE scores of the models with different hyper-parameter configurations over the CNN/Daily Mail test set.

CNN/Daily Mail							
Model	**R-1**	**R-2**	**R-L**	**Model**	**R-1**	**R-2**	**R-L**
PG+COV	33.023	13.886	30.353	Dep+PG+COV	34.038	14.022	30.989
Large Vocab	33.324	13.841	30.350	Dep+Large Vocab	34.718	14.024	31.010
No Freeze	**34.966**	**14.884**	**31.997**	Dep+No Freeze	**35.155**	**14.912**	**32.244**
No Pretrain	32.871	13.806	30.210	Dep+No Pretrain	33.994	13.350	30.321
LSTM	33.986	14.214	31.264	Dep w/LSTM	34.943	14.151	31.592

By looking at the models that use dependency features in Table 6.4, we can see that *"Dep w/LSTM"* model gets closer to *"Dep+No Freeze"* model. *"Dep+No Freeze"* model still performs the best among the other models with dependency. The scores of *"Dep+Large Vocab"* model also increase so that the difference between *"Dep+No Freeze"* model decreases. Even if the increase is not as high as the other models, *"Dep+No Freeze"* model achieves significantly better scores than *"Dep+PG+COV"* model. *"Dep+No Pretrain"* model still produces worse results than *"Dep+PG+COV"* model.

Incorporating the dependency features into the models clearly increases the scores as expected. Even *"Dep+No Pretrain"* model produces much better results than the worst performer *"No Pretrain"* model. *"Dep+PG+COV"* model can achieve 1, 0.2 and 0.6 better results than *"PG+COV"* model for each ROUGE score respectively, which is a worthy improvement. *"Large Vocab"* model seems to react much better than this. ROUGE-2 scores of *"Dep+Large Vocab"*, *"Dep+No Pretrain"* and *"Dep w/LSTM"*

models are slightly worse than their counterparts. However, ROUGE-1 and L scores are better. In general, it is correct to say that the integration of dependency features increases performance.

6.3.3. The Effects of Hyper-Parameters

Both in Tables 6.3 and 6.4, *"No Freeze"* and *"Dep+No Freeze"* models perform the best considering almost all of the ROUGE scores and test sets. This is due to the word embeddings getting properly adjusted to each dataset by not freezing the parameters. As a result, this makes the job of the encoder and decoder easier since they use better representations of the word embedding vectors. Even though *"No Pretrain"* and *"Dep+No Pretrain"* models produce slightly worse results compared to the other test sets, they can achieve better results for *"Gigaword II"* and *"Gigaword III"*. Considering that they do not use GloVe embeddings, which are meticulously trained, these worse results could have been even much worse. After all, the models have a harder time learning the suitable word embeddings. However, note that the parameters of the word look-up table in *"No Pretrain"* and *"Dep+No Pretrain"* models are not frozen. Therefore, a similar discussion about the non-frozen parameters of the word embeddings for *"No Freeze"* model can apply in *"No Pretrain"* model as well.

"Larger Vocab" and *"Dep+Larger Vocab"* models also achieve higher scores than *"PG+COV"* and *"Dep+PG+COV"* models respectively. This is expected since there are fewer out-of-vocabulary words. For summarization tasks, it is very important for the output to contain some important non-frequent tokens. However, it is not vital to have a very large vocabulary because the model also has the pointer/generator mechanism, which can handle out-of-vocabulary tokens. After all, the increases in ROUGE scores are not as big as the increase observed between *"No Freeze"* and *"Dep+No Freeze"* models. Lastly, *"LSTM"* model performs noticeably better. For abstractive summarization, we can conclude that LSTM works better than GRU, at least for these datasets.

6.4. Integration Position of Word Dependency Features

We also examine the effect of the placement position of dependency features in the models. There are two possible approaches, and we named the models corresponding to each approach as *"Input"* and *"Hidden"*. *"Input"* model uses Equation 5.15, which concatenates the embeddings of the dependency features to input word embeddings just before the computation of encoder RNN. *"Hidden"* model uses Equation 5.16, which concatenates the embeddings of the dependency features to the hidden states of the encoder. The other parts of these models are the same as *"PG+COV"* model.

Table 6.5. The ROUGE scores of *"Input"* and *"Hidden"* models over the Gigaword test sets.

	Gigaword Original			**Gigaword I**		
Model	**R-1**	**R-2**	**R-L**	**R-1**	**R-2**	**R-L**
Input	**30.986**	**13.348**	**28.998**	**39.937**	**18.480**	**37.717**
Hidden	30.190	12.653	28.345	39.015	17.987	37.061
	Gigaword II			**Gigaword III**		
Model	**R-1**	**R-2**	**R-L**	**R-1**	**R-2**	**R-L**
Input	**40.516**	**18.730**	**38.167**	**39.617**	**17.655**	**37.236**
Hidden	39.480	18.090	37.411	38.367	16.958	36.173

Table 6.6. The ROUGE scores of *"Input"* and *"Hidden"* models over the CNN/Daily Mail test set.

	CNN/Daily Mail		
Model	**R-1**	**R-2**	**R-L**
Input	**34.038**	**14.022**	**30.989**
Hidden	33.886	13.350	30.334

By looking at Tables 6.5 and 6.6, we can see that *"Input"* model achieves better results than *"Hidden"* model. This is true for literally every ROUGE score for all datasets. It is due to the fact that the encoder uses them along with the word embeddings to generate better hidden states. These hidden states depend on the dependency features in addition to the word embeddings. Because each hidden state depends on all of the previous (and subsequent) ones in RNNs, the resulting hidden states become more efficient than the ones in *"Hidden"* model. Note that we used the same approach as in *"Input"* model as the baseline for the models including the use of dependency features in all of the tables above.

Table 6.7. The ROUGE scores of the models that use byte-pair encoding over the Gigaword test sets.

	Gigaword Original			**Gigaword I**		
Model	**R-1**	**R-2**	**R-L**	**R-1**	**R-2**	**R-L**
BPE No PG+COV	30.219	13.080	28.647	39.299	17.212	36.918
BPE	31.492	13.580	29.511	41.080	18.090	39.177
BPE 30k	31.477	13.514	29.430	41.028	18.040	39.101
BPE w/LSTM	**32.186**	**14.003**	**30.344**	**41.973**	**18.579**	**40.107**
	Gigaword II			**Gigaword III**		
Model	**R-1**	**R-2**	**R-L**	**R-1**	**R-2**	**R-L**
BPE No PG+COV	39.923	17.490	37.566	39.417	17.002	37.027
BPE	41.797	18.318	40.029	41.146	17.780	39.302
BPE 30k	41.791	18.310	40.022	41.138	17.760	39.290
BPE w/LSTM	**42.620**	**18.812**	**40.893**	**41.972**	**18.428**	**40.734**

6.5. Subword Models on Abstractive Summarization Models

Table 6.7 shows the scores of the models that use BPE as the subword model evaluated by using the Gigaword test sets. We can see that the best performer is *"BPE w/LSTM"* model, with producing around 42 as ROUGE-1, 18.5 as ROUGE-2

and 40.5 as ROUGE-L score for the custom test sets. It can achieve 32.2, 14 and 30.3 as ROUGE scores respectively for the original test set as well. *"BPE"* and *"BPE 30k"* models both have lower scores than *"BPE w/LSTM"* model but significantly surpass *"BPE No PG+COV"* model. They achieve around 1.8, 0.8 and 2.4 more scores than *"BPE No PG+COV"* model for each ROUGE score respectively over the custom test sets. Similar differences can be observed for the original test set as well. Moreover, both *"BPE"* and *"BPE 30k"* models have almost identical results for every score, with *"BPE"* model having marginally higher scores. The worst performer is *"BPE No PG+COV"* model. It can achieve ROUGE-1 scores between 39.3-39.9, ROUGE-2 between 17-17.5 and ROUGE-L between 36.9-37.5 for the custom test sets.

Table 6.8. The ROUGE scores of the models that use WordPiece over the Gigaword test sets.

	Gigaword Original			**Gigaword I**		
Model	**R-1**	**R-2**	**R-L**	**R-1**	**R-2**	**R-L**
WordPiece No PG+COV	30.240	13.017	28.698	39.401	17.211	37.027
WordPiece	31.510	13.502	29.571	41.057	18.180	39.351
WordPiece 30k	31.500	13.499	29.560	41.031	18.160	39.304
WordPiece w/LSTM	**32.201**	**14.014**	**30.367**	**41.997**	**18.591**	**40.230**
	Gigaword II			**Gigaword III**		
Model	**R-1**	**R-2**	**R-L**	**R-1**	**R-2**	**R-L**
WordPiece No PG+COV	39.989	17.560	37.600	39.409	16.816	36.988
WordPiece	41.864	18.297	40.131	41.170	17.128	39.327
WordPiece 30k	41.853	18.291	40.126	41.167	17.120	39.311
WordPiece w/LSTM	**42.605**	**18.813**	**40.890**	**41.997**	**18.431**	**40.754**

In Table 6.8, we can see the results of the models that use WordPiece as the subword model. The tendency of the performances of each model is very similar to Table 6.7. In fact, almost all of the scores of the corresponding models are nearly identical. BPE-based models seem to perform marginally worse than WordPiece models,

but considering that researchers generally round ROUGE scores towards the nearest hundredth, the difference is negligible. Therefore, the same observations we mentioned above apply to WordPiece-based models as well.

Table 6.9. The ROUGE scores of the models that use unigram language model over the Gigaword test sets.

	Gigaword Original			**Gigaword I**		
Model	**R-1**	**R-2**	**R-L**	**R-1**	**R-2**	**R-L**
Unigram No PG+COV	30.014	13.010	28.498	38.871	17.894	39.241
Unigram	31.502	13.216	29.506	40.830	18.001	40.513
Unigram 30k	31.417	13.175	29.501	40.816	17.983	40.510
Unigram w/LSTM	**32.584**	**13.502**	**30.681**	**41.698**	**18.237**	**41.149**
	Gigaword II			**Gigaword III**		
Model	**R-1**	**R-2**	**R-L**	**R-1**	**R-2**	**R-L**
Unigram No PG+COV	40.040	17.580	37.689	39.581	17.137	37.186
Unigram	41.840	18.094	40.079	41.163	17.136	39.331
Unigram 30k	41.816	17.993	40.033	41.109	17.131	39.286
Unigram w/LSTM	**42.577**	**18.308**	**40.676**	**41.640**	**17.735**	**40.105**

The results of the models that use unigram language model as the subword model can be seen in Table 6.9. The scores are again very close to the ones in Tables 6.7 and 6.8. *"Unigram"* and *"Unigram 30k"* models achieve very close ROUGE scores. *"Unigram w/LSTM"* model performs the best among all. All three models surpass *"Unigram No PG+COV"* model significantly for each test set.

Even though the scores are very close to the ones in Tables 6.7 and 6.8, the improvements between the corresponding models are not similar. For example, *"Unigram w/LSTM"* model surpasses its counterparts over the original Gigaword test set. For *"Gigaword I"*, Unigram models also seem to achieve higher ROUGE-L scores than other subword models. In all of the test sets, *"Unigram No PG+COV"* performs better

than its counterparts with different subword models. The other results, however, are generally worse so it is not accurate to compare unigram language model with others.

Table 6.10. The ROUGE scores of the models that use byte-pair encoding over the CNN/Daily Mail test set.

	CNN/Daily Mail		
Model	**R-1**	**R-2**	**R-L**
BPE No PG+COV	33.176	13.890	30.483
BPE	35.304	14.107	32.745
BPE 30k	35.291	14.100	32.739
BPE w/LSTM	**36.196**	**14.371**	**33.554**

Table 6.11. The ROUGE scores of the models that use WordPiece over the CNN/Daily Mail test set.

	CNN/Daily Mail		
Model	**R-1**	**R-2**	**R-L**
WordPiece No PG+COV	33.204	13.895	30.451
WordPiece	35.345	14.113	32.781
WordPiece 30k	35.318	14.116	32.777
WordPiece w/LSTM	**36.196**	**14.372**	**33.556**

We also present the results of the same models trained with the CNN/Daily Mail dataset. Table 6.10 contains the ROUGE scores for BPE-based models. The scores of *"BPE"* and *"BPE 30k"* models are again very close, each surpassing *"BPE No PG+COV"* model by around 2.2, 0.2 and 2.3 for each ROUGE score respectively. *"BPE w/LSTM"* model achieves the best results. It improves the scores of these two models by 1.1, 0.2 and 0.8 respectively. A similar tendency can also be observed in Table 6.7. WordPiece-based models achieves very similar results to BPE-based models. WordPiece-based models can produce only marginally higher scores than BPE-based

models as can be observed in Table 6.11. The models that use unigram language model as the subword model perform worse than BPE and WordPiece, as can be observed in Table 6.12. The only exception seems to be the ROUGE-2 scores, as unigram models achieve slightly better scores than their counterparts.

Table 6.12. The ROUGE scores of the models that use unigram language model over the CNN/Daily Mail test set.

	CNN/Daily Mail		
Model	**R-1**	**R-2**	**R-L**
Unigram No PG+COV	33.006	13.925	30.203
Unigram	34.053	14.245	32.248
Unigram 30k	34.051	14.233	32.240
Unigram w/LSTM	**34.963**	**14.506**	**32.972**

For both datasets, the scores of *"BPE No PG+COV"*, *"WordPiece No PG+COV"* and *"Unigram No PG+COV"* models are very close to the scores of *"PG+COV"* models. Some of the ROUGE scores are higher, some of them are lower. Considering that these subword-based models do not use any additional mechanism suitable for abstractive summarization, these results are pretty decent. There are two reasons for that. First, these subword-based models do not use the coverage mechanism. As we concluded above, enabling the coverage mechanism from the start of the training decreases the performance, which is how coverage is used in *"PG+COV"* model. This causes a decrease in the scores of *"PG+COV"* model, and they get closer to the scores of *"BPE No PG+COV"*, *"WordPiece No PG+COV"* and *"Unigram No PG+COV"* models. The second reason is that the subword models naturally handles the out-of-vocabulary tokens. It might seem that *"PG+COV"* model should perform better because the pointer/generator mechanism is enabled, and it handles the out-of-vocabulary tokens. The same situation also holds for all subword-based models, which causes the corresponding scores to be similar to the scores of *"PG+COV"* model.

The most obvious observation that can be seen in Tables 6.7 to 6.12 is that the addition of the pointer/generator mechanism significantly improves the performances. The pointer/generator mechanism is used because of its ability to generate out-of-vocabulary tokens. However, the same ability is inherently gained by using subword models. It could be confusing to understand which factor plays a role in this improvement when they both solve the problem of out-of-vocabulary tokens. The pointer/generator mechanism does not only point to an out-of-vocabulary token seen in input sentences. It can also decide to point to a token that is already in the vocabulary. This effectively introduces extractiveness to the models. Indeed, *"BPE"*, *"WordPiece"* and *"Unigram"* models have gained extractiveness and this, as a result, helps to increase the scores.

As mentioned in Section 5.5, all of the subword models that we use have an adjustable vocabulary size as a hyper-parameter. We also wanted to see the effect of it. There is hardly any change in the scores if the vocabulary size gets increased for both datasets. In fact, it even causes a very small decrease. Note that obviously, this decrease is negligible. In the end, *"BPE 30k"*, *"WordPiece 30k"* and *"Unigram 30k"* models can achieve between 1 or 2 more than the scores of their counterparts that do not use any additional mechanism. However, the model complexity gets higher if the vocabulary size becomes large. Because of this, the sensible approach is to use a smaller vocabulary size as we did for *"BPE"*, *"WordPiece"* and *"Unigram"* models.

Using subword models alone increases the effectiveness of models as they solve the problem of out-of-vocabulary tokens. With the addition of the pointer/generator mechanism, these models improve their performances even more. We conclude that integrating subwords into abstractive summarization models is beneficial as the related scores increase in our experiments.

6.6. Proper Usage of Additional Mechanisms

We also built a model by combining the approaches used in the best results shared above. Because *"LSTM"* and *"No Freeze"* models perform well, we used LSTMs instead of GRUs and did not freeze the parameters of word embeddings while training. This model uses the pointer/generator mechanism. We trained the model without using the coverage mechanism until it converged to a point (approximately 2 days). Then, we enabled coverage and trained the model further for a while (approximately 3-4 hours) as it shows its benefits in this way. The model also uses dependency features. We call this model *"Dep+PG→COV"*.

Table 6.13. The ROUGE scores of various models including *"Dep+PG→COV"* over the Gigaword test set.

	Gigaword Original		
Model	**R-1**	**R-2**	**R-L**
ABS (Rush et al., 2015)	29.55	11.32	26.42
ABS+ (Rush et al., 2015)	29.76	11.88	26.96
lvt2k-1sent (Nallapati et al., 2016)	32.67	15.59	30.64
RAS-LSTM (Chopra et al., 2016)	32.55	14.70	30.03
RAS-Elman (Chopra et al., 2016)	33.78	15.97	31.15
Multi-Task (Pasunuru et al., 2017)	32.75	15.35	30.82
UniLM (Dong, Yang, Wang, Wei et al., 2019)	38.90	20.05	36.00
PEGASUS (Zhang, Zhao et al., 2020)	39.12	19.86	36.24
ProphetNet (Qi, Yan, Gong, Liu et al., 2020)	**39.51**	**20.42**	**36.69**
Dep+PG→COV	32.95	15.30	30.97

With the proper usage of the coverage mechanism, we can see that *"Dep+PG→COV"* model produces very good results compared to *"Dep+PG+COV"* and *"Dep+PG"* in Tables 6.1 and 6.2 for both Gigaword and CNN/Daily Mail datasets. It surpasses them by more than 2 points in each ROUGE score for the Gigaword

dataset. This shows the benefit of the coverage mechanism in abstractive summarization tasks. On the other hand, the improvement is even higher for the CNN/Daily Mail dataset. It achieves around 4, 3 and 4.5 more for each ROUGE score respectively. The reason for the higher contribution over the CNN/Daily Mail dataset comes from the nature of these two datasets. As explained in Section 6.2.4, the input texts of Gigaword are a lot shorter than CNN/Daily Mail. The longer texts cause the model to generate repeated words or phrases more frequently. Therefore, using the coverage mechanism for the Gigaword dataset is not as vital as using it for the CNN/Daily Mail dataset.

Table 6.14. The ROUGE scores of various models including *"Dep+PG→COV"* over the CNN/Daily Mail test set.

	CNN/Daily Mail		
Model	**R-1**	**R-2**	**R-L**
words-lvt2k-temp-att (Nallapati et al., 2016)	35.46	13.30	32.65
seq-to-seq + attn baseline (See et al., 2017)	30.49	11.17	28.08
pointer-generator (See et al., 2017)	36.44	15.66	33.42
pointer-generator + coverage (See et al., 2017)	39.53	17.28	36.38
UniLM (Dong, Yang, Wang, Wei et al., 2019)	44.17	21.47	41.11
PEGASUS (Zhang, Zhao et al., 2020)	43.33	20.21	40.51
ProphetNet (Qi, Yan, Gong, Liu et al., 2020)	**44.20**	**21.17**	**41.30**
Dep+PG→COV	38.17	16.93	35.88

Table 6.13 contains the results of some popular models evaluated using the Gigaword dataset. We can see that our model can achieve better scores compared to the models by Rush et al. [9], Nallapati et al. [19], Chopra et al. (except their RAS-Elman model) [20] and Pasunuru et al. [27]. In Table 6.14, the scores of various models can be seen for the CNN/Daily Mail dataset. Our model can perform better than Nallapati et al. [19]. It also surpasses the baseline and the pointer/generator model from See et al. [30]. However, it fails to achieve better scores than "pointer-generator + coverage"

model designed by See et al. Note that we failed to create a baseline model performing close to their baseline model. We think that integrating dependency features on top of their baseline model would achieve better scores than their best model.

UniLM [54], PEGASUS [55] and ProphetNet [56] are relatively new models. They differ from the others in Tables 6.13 and 6.14. The other models use RNN-based networks whereas ProphetNet, UniLM and PEGASUS are based on transformers. It can clearly be seen that these transformer-based models achieve significantly higher results. However, we showed that integration of dependency features increases the performance. Utilizing transformer-based approaches with dependency usage might surpass some baseline transformer models if not these models.

6.7. Discussions Regarding the Evaluation Metrics

In all of the tables above, we shared the F_1 scores of ROUGE-1, ROUGE-2 and ROUGE-L for each model evaluated by using each test set. These scores are typically shared in order to show how well the proposed models perform and to compare various models fairly in many abstractive summarization studies. However, ROUGE-N metrics are basically based on n-gram matches. They favor the models that can generate the summaries in the exact way that the target summaries are constructed. This means that even the words should be the same as the words in the target summary, and the order of words should be very close to. We think that summarization systems should not be evaluated in this way. They can generate summaries with different wordings compared to the target summaries. This should not mean that they are of bad quality. In fact, summaries can have many different ways to describe facts. Many similar summaries can be constructed by extensive paraphrasing, and all of them can keep the condensed information intact.

In addition to the ROUGE scores, we wanted to show the METEOR scores [57] for some of our important models in Table 6.15. Note that "CNN/DM" stands for the CNN/Daily Mail test set, and "Org." stands for the original test set for each

dataset in the table. In addition to the exact matching, METEOR can also utilize functionalities for aligning based on stems, synonyms and paraphrase matches between words and phrases. We used its full mode when computing the scores since these approaches provide better evaluation for the system summaries that use very similar words instead of the exact words in the target summaries. It can fairly evaluate the paraphrased sentences too. Therefore, METEOR might actually be a better way to evaluate models, specifically for abstractive summarization systems. Unfortunately, almost none of the studies in abstractive summarization evaluate their models using the METEOR metric. The researchers typically do not report these results. Therefore, it is hard to compare the models based on the METEOR scores.

Table 6.15. The METEOR scores of the different models over both the Gigaword test sets and the CNN/Daily Mail test set.

	Gigaword				**CNN/DM**
Model	**Org.**	**I**	**II**	**III**	**Org.**
Baseline	20.77	28.94	28.51	29.09	13.76
PG+COV	21.97	29.52	29.12	29.61	14.29
Dep+PG+COV	22.45	30.51	30.43	30.98	16.91
Dep+PG→COV	**25.81**	**34.62**	**34.72**	**34.90**	**18.83**

The score improvements gained by each model for the ROUGE metric in the previous tables can also be observed in Table 6.15 for the METEOR metric. For example, *"PG+COV"* model performs better than *"Baseline"* model, as can also be seen in Tables 6.1 and 6.2. This means that the additional mechanisms help the models improve their performances. *"Dep+PG+COV"* model can also produce better scores than *"PG+COV"* model. This shows that the addition of dependency features increases the METEOR scores as well. Lastly, *"Dep+PG→COV"* model performs the best for each test set in Table 6.15.

7. CONCLUSION

In our experiments, we analyzed the effects of the additional mechanisms used in abstractive summarization models. We have found that the pointer/generator mechanism increases the scores of models significantly. Most of the out-of-vocabulary words in the input texts are very important to be included in summaries in any summarization task. We observed that the pointer/generator mechanism handles the issue of generating out-of-vocabulary words. It also brought extractiveness to the models and balanced between these two summarization approaches, which was another reason for the improvements that we saw in our results. We also observed the effects of the coverage mechanism. We have found that using it after the parameters of the models converge in training increases the performance of the models too. It helped the models avoid any repeated words or phrases, and was very useful for summarizing long input texts.

We further improved these models with the addition of word dependency relations. We used dependency parsing on every input sentence and obtained some of the features for each word. These features were then embedded and integrated into the model. We have found that this inclusion indeed helped the models achieve better results. The knowledge of dependency structure made natural language understanding more efficient and natural language generation better. We also analyzed the use of subwords in our models since it is widely used in recent state-of-the-art NLP models. We trained three different subword models and used their resulting subwords in our models. The results showed that the use of subwords is a good approach. They naturally handle the out-of-vocabulary word issues in summarization tasks. With the additional mechanisms, the performances of the models got even better.

For future work, we think that other training approaches can be adapted for the models that we used. For example, policy gradient techniques in reinforcement learning to directly maximize the ROUGE scores are used, and they produce worthy

results [28]. Along with the additions of word dependency relations or subword models, such techniques can help the models perform even better. There are also some other approaches that can be adapted. For example, deep communicating agents can be used for abstractive summarization tasks as well [42]. Integrating the additions we have shown into such models might be useful. Similarly, generative adversarial networks can also be used with these additions [35].

Recently, transformers are used very frequently, and they achieve state-of-the-art results in many NLP problems, including abstractive summarization [54–56]. Even though we could not achieve the scores of recent studies, using the same models as a baseline and integrate our additions into them might produce better scores. Many popular models use subwords internally. However, finding a suitable way to incorporate word dependency features into these models might provide better results.

REFERENCES

1. Hochreiter, S. and J. Schmidhuber, "Long Short-Term Memory", *Neural Computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

2. Cho, K., B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, "Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation", *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP*, pp. 1724–1734, ACL, 2014.

3. Bahdanau, D., K. Cho and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate", *3rd International Conference on Learning Representations, ICLR*, 2015.

4. Jurafsky, D. and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 2nd Edition*, Prentice Hall, Pearson Education International, 2009.

5. Sennrich, R., B. Haddow and A. Birch, "Neural Machine Translation of Rare Words with Subword Units", *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, ACL, 2016.

6. Schuster, M. and K. Nakajima, "Japanese and Korean Voice Search", *International Conference on Acoustics, Speech and Signal Processing, ICASSP*, IEEE, 2012.

7. Devlin, J., M. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, pp. 4171–4186, ACL, 2019.

8. Kudo, T., "Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates", *Proceedings of the 56th Annual Meeting*

of the Association for Computational Linguistics, pp. 66–75, ACL, 2018.

9. Rush, A. M., S. Chopra and J. Weston, "A Neural Attention Model for Abstractive Sentence Summarization", *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP*, pp. 379–389, ACL, 2015.

10. Nallapati, R., F. Zhai and B. Zhou, "SummaRuNNer: A Recurrent Neural Network Based Sequence Model for Extractive Summarization of Documents", *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pp. 3075–3081, AAAI Press, 2017.

11. Ganesan, K., C. Zhai and E. Viegas, "Micropinion Generation: An Unsupervised Approach to Generating Ultra-Concise Summaries of Opinions", *Proceedings of the 21st World Wide Web Conference, WWW*, pp. 869–878, ACM, 2012.

12. Ganesan, K., C. Zhai and J. Han, "Opinosis: A Graph Based Approach to Abstractive Summarization of Highly Redundant Opinions", *COLING, 23rd International Conference on Computational Linguistics, Proceedings of the Conference*, pp. 340–348, Tsinghua University Press, 2010.

13. Lloret, E. and M. Palomar, "Analyzing the Use of Word Graphs for Abstractive Text Summarization", *Proceedings of IMMM, The First International Conference on Advances in Information Mining and Management*, pp. 61–66, 2011.

14. Moawad, I. F. and M. Aref, "Semantic Graph Reduction Approach for Abstractive Text Summarization", *Seventh International Conference on Computer Engineering Systems (ICCES)*, pp. 132–138, 2012.

15. Banerjee, S., P. Mitra and K. Sugiyama, "Multi-Document Abstractive Summarization Using ILP Based Multi-Sentence Compression", *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pp. 1208–1214, AAAI Press, 2015.

16. Erkan, G. and D. R. Radev, "LexRank: Graph-based Lexical Centrality as Salience in Text Summarization", *Journal of Machine Learning Research*, Vol. 22, pp. 457–479, 2004.

17. Khan, A., N. Salim and Y. J. Kumar, "A Framework for Multi-Document Abstractive Summarization Based on Semantic Role Labelling", *Applied Soft Computing*, Vol. 30, pp. 737–747, 2015.

18. Genest, P. and G. Lapalme, "Framework for Abstractive Summarization Using Text-to-Text Generation", *Proceedings of the Workshop on Monolingual Text-To-Text Generation*, pp. 64–73, ACL, 2011.

19. Nallapati, R., B. Zhou, C. N. dos Santos, Ç. Gülçehre and B. Xiang, "Abstractive Text Summarization Using Sequence-to-sequence RNNs and Beyond", *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL*, pp. 280–290, ACL, 2016.

20. Chopra, S., M. Auli and A. M. Rush, "Abstractive Sentence Summarization with Attentive Recurrent Neural Networks", *NAACL HLT, The Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 93–98, ACL, 2016.

21. Xu, Y., J. H. Lau, T. Baldwin and T. Cohn, "Decoupling Encoder and Decoder Networks for Abstractive Document Summarization", *Proceedings of the Workshop on Summarization and Summary Evaluation Across Source Types and Genres, MultiLing@EACL*, pp. 7–11, ACL, 2017.

22. Mikolov, T., K. Chen, G. Corrado and J. Dean, "Efficient Estimation of Word Representations in Vector Space", *1st International Conference on Learning Representations, ICLR*, 2013.

23. Le, Q. V. and T. Mikolov, "Distributed Representations of Sentences and Doc-

uments", *Proceedings of the 31st International Conference on Machine Learning, ICML*, Vol. 32, pp. 1188–1196, JMLR, 2014.

24. Zhou, Q., N. Yang, F. Wei and M. Zhou, "Selective Encoding for Abstractive Sentence Summarization", *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 1095–1104, ACL, 2017.

25. Tan, J., X. Wan and J. Xiao, "From Neural Sentence Summarization to Headline Generation: A Coarse-to-Fine Approach", *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, pp. 4109–4115, IJCAI, 2017.

26. Li, P., W. Lam, L. Bing and Z. Wang, "Deep Recurrent Generative Decoder for Abstractive Text Summarization", *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP*, pp. 2091–2100, ACL, 2017.

27. Pasunuru, R., H. Guo and M. Bansal, "Towards Improving Abstractive Summarization via Entailment Generation", *Proceedings of the Workshop on New Frontiers in Summarization, NFiS@EMNLP*, pp. 27–32, ACL, 2017.

28. Paulus, R., C. Xiong and R. Socher, "A Deep Reinforced Model for Abstractive Summarization", *Computing Research Repository (CoRR)*, Vol. abs/1705.04304, 2017.

29. Lin, C.-Y., "ROUGE: A Package for Automatic Evaluation of Summaries", *Text Summarization Branches Out*, pp. 74–81, ACL, 2004.

30. See, A., P. J. Liu and C. D. Manning, "Get To The Point: Summarization with Pointer-Generator Networks", *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 1073–1083, ACL, 2017.

31. Chen, Y. and M. Bansal, "Fast Abstractive Summarization with Reinforce-Selected Sentence Rewriting", *Proceedings of the 56th Annual Meeting of the Association*

for Computational Linguistics, pp. 675–686, ACL, 2018.

32. Cao, Z., F. Wei, W. Li and S. Li, "Faithful to the Original: Fact Aware Neural Abstractive Summarization", *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI), the 30th Innovative Applications of Artificial Intelligence (IAAI), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI)*, pp. 4784–4791, AAAI Press, 2018.

33. Banko, M., M. J. Cafarella, S. Soderland, M. Broadhead and O. Etzioni, "Open Information Extraction from the Web", *IJCAI, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 2670–2676, 2007.

34. Amplayo, R. K., S. Lim and S. Hwang, "Entity Commonsense Representation for Neural Abstractive Summarization", *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, pp. 697–707, ACL, 2018.

35. Liu, L., Y. Lu, M. Yang, Q. Qu, J. Zhu and H. Li, "Generative Adversarial Network for Abstractive Text Summarization", *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI), the 30th Innovative Applications of Artificial Intelligence (IAAI), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI)*, pp. 8109–8110, AAAI Press, 2018.

36. Lin, J., X. Sun, S. Ma and Q. Su, "Global Encoding for Abstractive Summarization", *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 163–169, ACL, 2018.

37. Hsu, W. T., C. Lin, M. Lee, K. Min, J. Tang and M. Sun, "A Unified Model for Extractive and Abstractive Summarization Using Inconsistency Loss", *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 132–141, ACL, 2018.

38. Fan, A., D. Grangier and M. Auli, "Controllable Abstractive Summarization", *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation, NMT@ACL*, pp. 45–54, ACL, 2018.

39. Song, K., L. Zhao and F. Liu, "Structure-Infused Copy Mechanisms for Abstractive Summarization", *Proceedings of the 27th International Conference on Computational Linguistics, COLING*, pp. 1717–1729, ACL, 2018.

40. Wang, L., J. Yao, Y. Tao, L. Zhong, W. Liu and Q. Du, "A Reinforced Topic-Aware Convolutional Sequence-to-Sequence Model for Abstractive Text Summarization", *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pp. 4453–4460, IJCAI, 2018.

41. Blei, D. M., A. Y. Ng and M. I. Jordan, "Latent Dirichlet Allocation", *Advances in Neural Information Processing Systems 14, NIPS*, pp. 601–608, MIT Press, 2001.

42. Çelikyılmaz, A., A. Bosselut, X. He and Y. Choi, "Deep Communicating Agents for Abstractive Summarization", *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, pp. 1662–1675, ACL, 2018.

43. Cohan, A., F. Dernoncourt, D. S. Kim, T. Bui, S. Kim, W. Chang and N. Goharian, "A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents", *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, pp. 615–621, ACL, 2018.

44. Li, C., W. Xu, S. Li and S. Gao, "Guiding Generation for Abstractive Text Summarization Based on Key Information Guide Network", *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, pp. 55–60, ACL, 2018.

45. Mihalcea, R. and P. Tarau, "TextRank: Bringing Order into Text", *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP*, pp. 404–411, ACL, 2004.

46. Xie, N., S. Li, H. Ren and Q. Zhai, "Abstractive Summarization Improved by WordNet-Based Extractive Sentences", *Natural Language Processing and Chinese Computing – 7th CCF International Conference, NLPCC*, Vol. 11108 of *Lecture Notes in Computer Science*, pp. 404–415, Springer, 2018.

47. "GitHub - harvardnlp/sent-summary", `https://github.com/harvardnlp/sent-summary`, accessed in June 2021.

48. Hermann, K. M., T. Kociský, E. Grefenstette, L. Espeholt, W. Kay, M. Süleyman and P. Blunsom, "Teaching Machines to Read and Comprehend", *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems*, pp. 1693–1701, 2015.

49. "GitHub - huggingface/tokenizers: Fast State-of-the-Art Tokenizers Optimized for Research and Production", `https://github.com/huggingface/tokenizers`, accessed in June 2021.

50. Pennington, J., R. Socher and C. D. Manning, "GloVe: Global Vectors for Word Representation", *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP*, pp. 1532–1543, ACL, 2014.

51. Duchi, J. C., E. Hazan and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *Journal of Machine Learning Research*, Vol. 12, pp. 2121–2159, 2011.

52. Honnibal, M., I. Montani, S. Van Landeghem and A. Boyd, "spaCy: Industrial-Strength Natural Language Processing in Python", `https://spacy.io/`, accessed in June 2021.

53. "GitHub - bheinzerling/pyrouge: A Python Wrapper for the ROUGE Summarization Evaluation Package", `https://github.com/bheinzerling/pyrouge`, accessed in June 2021.

54. Dong, L., N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou and H. Hon, "Unified Language Model Pre-training for Natural Language Understanding and Generation", *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS*, pp. 13042–13054, 2019.

55. Zhang, J., Y. Zhao, M. Saleh and P. J. Liu, "PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization", *Proceedings of the 37th International Conference on Machine Learning, ICML*, Vol. 119, pp. 11328–11339, PMLR, 2020.

56. Qi, W., Y. Yan, Y. Gong, D. Liu, N. Duan, J. Chen, R. Zhang and M. Zhou, "ProphetNet: Predicting Future N-gram for Sequence-to-Sequence Pre-training", *Proceedings of the Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP*, pp. 2401–2410, ACL, 2020.

57. Denkowski, M. J. and A. Lavie, "Meteor Universal: Language Specific Translation Evaluation for Any Target Language", *Proceedings of the Ninth Workshop on Statistical Machine Translation, WMT@ACL*, pp. 376–380, ACL, 2014.

APPENDIX A: A CLOSER LOOK ON GENERATED SUMMARIES

Some example summaries generated by both *"PG+COV"* and *"Dep+PG→COV"* models can be observed below. The details of these models can be found in Chapter 6. We present the input text, the reference summary and the summaries generated by these models consecutively. Note that these examples are selected from the custom Gigaword test sets.

Original Text: british foreign office minister mike o'brien on tuesday criticized the latest violence in israel, saying the attack by israeli troops on gaza would jeopardize the peace process in the middle east region.
Reference Summary: britain slams latest israeli attack on gaza
System Summary: britain condemns israeli violence in israel
System Summary (w/Dependency): british fm criticizes latest mideast violence

The system summary looks successful, but it gives wrong information. The location is not Israel, instead, it should be Gaza. The addition of dependency features seems to fix this issue. Even if the location is now more general, it is correct. In addition, the person that criticizes the situation becomes more informative. It even uses the abbreviation "fm" instead of "foreign minister", which fits the idea of summarization. However, the perpetrator is missing in the last summary.

Original Text: belgian maritime police arrested an iranian man after discovering a compatriot in the trunk of his car on board a passenger ferry between zeebrugge and britain, belgian media reported monday.
Reference Summary: iranian man arrested trying to smuggle compatriot into britain in car trunk
System Summary: belgian man arrested in belgian ferry
System Summary (w/Dependency): belgian police arrest iranian man

The system summary contains an unnecessary word. The word "belgian" in "belgian ferry" is unnecessary. Besides, the arrested person is not Belgian. These issues seem to be handled by the model with dependency usage. The knowledge of the structural information of the sentence helps the model generate that summary.

Original Text: britain's new security chief has warned that the country's battle against terrorism could take up to ## years, while prime minister gordon brown says he wants an expanded european system to share information on potential threats.
Reference Summary: britain's new terror chief says ## years needed to fight islamic terrorism
System Summary: british security chief warns against terror threat
System Summary (w/Dependency): british security chief warns against terror threat

Both of the models generate the same results, and both seem to be successful.

Original Text: eight men who used donkeys to smuggle anti-tank missiles and rpgs across a desert border received prison terms of five to ## years with hard labor on tuesday.
Reference Summary: court sentences eight men to prison terms of #-## years for arms smuggling
System Summary: eight men sentenced for anti-tank smuggling
System Summary (w/Dependency): eight jailed in prison for smuggling missiles

Both of the system summaries are similar to each other, but the first one has wrong information. The smuggled object should "anti-tank missiles", and it is correctly generated by the second model.

Original Text: prime minister viktor chernomyrdin settled his differences thursday with polish officials over the mistreatment of russian travelers by police in warsaw, the itar-tass news agency reported.
Reference Summary: dispute over beaten tourists settled
System Summary: chernomyrdin settles differences with russia
System Summary (w/Dependency): chernomyrdin settles differences with polish

The summaries are very different from the reference. As a result, this summary's contribution over the ROUGE score is very low. However, both of the resulting summaries are not particularly bad. This shows that the ROUGE metric is not totally adequate for abstractive summarization tasks. On the other hand, the name "chernomyrdin" is an out-of-vocabulary word. Both summaries can successfully generate it, which shows the benefit of the pointer/generator mechanism. In addition, the first summary contains wrong information. Instead of "russia", it should be "polish", which is correctly generated in the second summary with the help of the word dependency features.