Rare words in text summarization

by

Danila Morozovskii

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Applied Computer Science

University of Winnipeg

Rare words in text summarization

by

Danila Morozovskii

Supervisory Committee

Dr. S. Ramanna, Supervisor
(Department of Applied Computer Science)

Dr. C. J. Henry, Member
(Department of Applied Computer Science)

Dr. A. M. Kumar, External Member
(Department of Information Technology, National Institute of Technlogy Karnataka - Surathkal)

ABSTRACT

Automatic text summarization is a difficult task, which involves a good understanding of an input text to produce fluent, brief and vast summary. The usage of text summarization models can vary from legal document summarization to news summarization. The model should be able to understand where important information is located to produce a good summary. However, infrequently used or rare words might limit model's understanding of an input text, as the model might ignore such words or put less attention on them. Another issue is that the model accepts only a limited amount of tokens (words) of an input text, which might contain redundant information or not including important information as it is located further in the text. To address the problem of rare words, we have proposed a modification to the attention mechanism of the transformer model with pointer-generator layer, where attention mechanism receives frequency information for each word, which helps to boost rare words. Additionally, our proposed supervised learning model uses the hybrid approach incorporating both extractive and abstractive elements, to include more important information for the abstractive model in a news summarization task. We have designed experiments involving a combination of six different hybrid models with varying input text sizes (measured as tokens) to test our proposed model. Four well-known datasets specific to news articles were used in this work: CNN/DM, XSum, Gigaword and DUC 2004 Task 1. Our results were compared using the well-known ROUGE metric. Our best model achieved R-1 score of 38.22, R-2 score of 15.07 and R-L score of 35.79, outperforming three existing models by several ROUGE points.

Keywords: Abstractive Summarization, Extractive Summarization, Natural Language Processing, Pointer-generator, Transformer, Rare Words, Text Summarization.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to give my warmest thanks to my supervisor Dr. Sheela Ramanna for her support throughout my studies and for giving me unique opportunities for my research experience which helped me to improve my knowledge. I am also grateful to my committee members, Dr. Christopher Henry and Dr. Anand Kumar, for their valuable comments. Lastly, I would like to thank Dr. Christopher Henry for providing a GPU machine for this research and for being an amazing professor during the CUDA course, which taught me a lot of useful techniques.

The successful completion of this study could not have been possible without my wife's support, Katarzyna Morozovska, and my family, Oleksandr, Natalia and Uliana Morozovski.

Finally, I would like to thank all my friends for supporting me and bringing joy to my life.

Danila Morozovskii

Chapter 1

Introduction

The amount of textual data generated on the Internet in recent years has been enormous and is only growing [2]. Automatic text summarization helps to reduce textual information into a convenient summary, that is easier to understand. The summary, that is generated, should be as informative as possible, at the same time being fluent, brief and vast [3]. Text summarization can be used in a variety of applications, such as a search engine to provide a direct answer to a query [4], biomedical literature summarizations to provide an easier way to read evidence [5], legal document summarization [6] or a headline generation [7]. Another application is news summarization, which helps to expedite the understanding of an article for a human [8].

Automatic text summarization can be based on different aspects [8], such as input size (single-document, multi-document or multi-media (input information is gathered from several sources, such as text, image and/or video [9])), summarization algorithm (supervised, unsupervised or semi-supervised), summarization approach (extractive, abstractive or hybrid), summary type (headline, sentence-level, highlights or full summary) and others. Text summarization models can be separated into two main groups: extractive [10, 11] and abstractive [12, 13]. An extractive method generates summaries using sentences from the input text. In contrast, in an abstractive approach, the text is generated using an external vocabulary of words, which might include words that are not present in the original input. The combination of extractive and abstractive approaches is called a hybrid approach. This approach intends to overcome the weaknesses of the extractive and abstractive approach [14, 15, 16].

Early machine learning models were focused only on the extractive approach. In the past, abstractive methods used only recurrent neural networks (RNN) [17] or long short-term memory (LSTM) [18] networks to generate a summary, which

performed poorly on long text, as these networks are not good in handling long text. Encoder-decoder model [19] and attention mechanism [20, 21] helped to overcome this problem. The attention mechanism uses all the intermediate states to generate the prediction, even if words are located far from each other. Google [1] introduced a transformer model that eliminated recurrent aspects and relied only on an attention mechanism. Transformer outperforms sequence-to-sequence model [22], which uses an input sequence and output sequence with RNN or LSTM inside it. With the help of pre-trained encoders [23, 24, 25, 26] the current state-of-the-art results have outperformed other models.

1.1 Problem Definition

The critical problem for Natural Language Processing (NLP) tasks is to correctly represent words. One such representation technique is called a bag-of-words (BOW), where each sample is encoded using the vocabulary of words that are used in the text corpus, which is encoded using a binary coding of zeros and ones [27]. However, such an approach does not provide a meaningful information about a sentence, as, for example, the same word in a different context will be represented exactly the same, even if it has a different meaning (i.e., "put money in the bank" and "the bank of a river"). Word embedding solves this issue by encoding words in a fixed-length vector [28] and using surrounding words to define this vector. However, the problem occurs when the word is not present in a vocabulary, as it's usage is infrequent or it is completely new word. Such words are called *out-of-vocabulary* (OOV) words, which are encoded with the same vector.

The OOV problem is one of the most important problems in text summarization, as it affect the performance of a model [29]. The input text or the summary might contain OOV words, which leads to reduction of the amount of information that the model can process [30]. Some techniques to deal with OOV problem are pointer-generator model (where the model can "point" to the input text to copy the word) [31] or usage of dual encoder (where the first encoder encodes the text regularly, and the second encoder encodes the importance of words [32]).

A similar but somewhat different problem is a rare words problem, where the words are not new, but appear less frequently in the training dataset; therefore, it leads to a problems in training the vector representation of such words, which worsen the general performance of the model [33]. OOV words are encoded using exactly the

same vector, meaning they do not differ between each other if the word is used less or more frequently. To the best of our knowledge, there has been done a limited work regarding usage of rare words in text summarization problem [34, 35], as most of the research have been focused on the OOV problem.

Another problem is what information is provided to the model. The deep learning model has a fixed input size; therefore, only a limited amount of the input text tokens can be used. Some papers [12, 31, 23, 26] take the first N tokens as input text, which we consider as not optimal. Important information might be located further in the text if it is too long for the deep learning model. Including the first N tokens might lead to the inclusion of irrelevant information, and instead, more amount of important information could be provided.

1.2 Proposed Approach

In this thesis, we focus on the problem of rare words, meaning words that are infrequently used in the dataset. Our proposed model puts more attention on rare words and less attention on frequently used words. Our intuition is that boosting attention for rarely used words might provide additional information to the model, which might help to improve the performance and generate better summaries. We do not use pre-trained embedding encoders or knowledge transfer (like BERT [23], BART [25] or PEGASUS [26], which are used to achieve current state-of-the-art results), as knowledge transfer might be biased against rare words. The objective of this research is to experimentally test whether boosting rare words improves the summarization task by selecting and/or modifying appropriate transformed-based architecture with pointer-generator layer.

Our proposed supervised learning approach is to use the hybrid model (shown in Fig. 1.1) where the summary generated from an extractive approach will be used as *input* to the abstractive model training. The reason for applying the extractive approach prior to model training is to remove redundant information from the input. In Fig. 1.1, there are two additional steps shown (dataset preprocessing and the model testing). The model testing is done using an annotated (labelled) summary. Our proposed model uses a transformer model with a pointer-generator layer [31] as an abstractive approach. Similar model has been implemented in [36]; however, authors did not achieve high results. The pointer-generator layer decides whether to copy a word from the input text or generate a new word. We test the performance of

the this model to show that it can outperform the sequence-to-sequence model with the pointer-generator layer.

Our proposed approach can be used in situations, when the text domain is different than the one the model was trained on (i.e., testing on a different dataset, which has shorter summaries). Frequency information can help to concentrate on rare words, which might contain important information. Our hybrid model helps to extract important parts of the text, before the model generates a summary, which might be helpful, when the input text is very long and important information might be located further in the text.

We use four well-known datasets: CNN/DM [12], XSum [37], Gigaword [38] and DUC 2004 Task 1 [39]. These datasets are based on news articles, and are used to generate a summary from the article. The Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [40] score is used as a metric to evaluate the model's performance with the most commonly used variations are ROUGE-1 (R-1), ROUGE-2 (R-2) and ROUGE-L (R-L) scores.

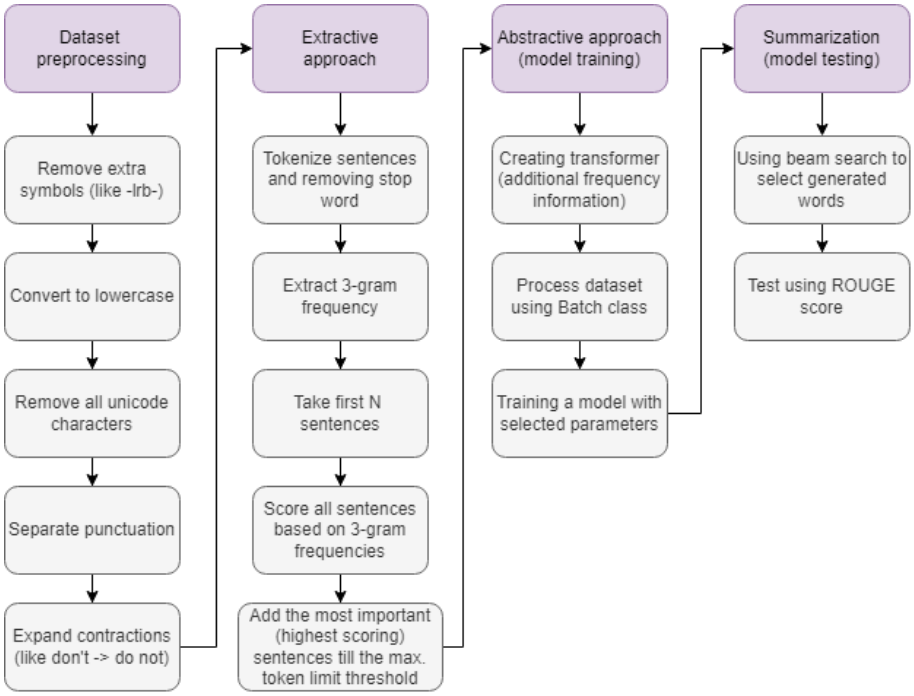


Figure 1.1: General pipeline

1.3 Contributions

The contribution of this thesis are as follows:

- We have proposed a modification to the attention mechanism of the transformer model with a pointer-generator layer. This modification adds frequency information about words in an effort to boost the attention of rarely used words.
- We have implemented a hybrid approach, which uses extractive and abstractive elements and showed that it outperforms models which use only an abstractive approach.
- We have demonstrated that our proposed model with frequency information, trained on CNN/DM and XSum datasets can perform better when tested on the DUC 2004 Task 1 dataset.
- We present case-studies of summarization experiments to test the transformer model with a pointer-generator layer as well as our proposed model with two different input text sizes (measured as tokens). Our best model achieved R-1 score of 38.22, R-2 score of 15.07 and R-L score of 35.79, outperforming other models by several ROUGE points.

1.4 Thesis Layout

The rest of this thesis is organized as follows:

Chapter 2 describes the background information for the proposed transformer-based model, as well as performance evaluation algorithms and hyper parameters.

Chapter 3 provides an overview of research related to the three main text summarization approaches: extractive, abstractive and hybrid.

Chapter 4 explains four different datasets that have been used and preprocessing steps.

Chapter 5 describes experiments that have been performed and analyzes the results.

Chapter 6 concludes the thesis and suggests improvements for future work.

Chapter 2

Preliminaries

This chapter discusses the background of machine learning, the transformer models and later proposed models and the model implemented in this thesis. Implemented model has the following novelty: transformer model with pointer-generator layer (which we refer to as M1) has been implemented; frequency information is given to the encoder output attention (which we refer to as M2); the model contains both extractive and abstractive approaches, where extractive method takes first k sentences from the input text, and then adds other sentences, depending on a score they get (M1-x or M2-x, where x is a number of tokens used).

In this thesis, we use supervised machine learning, where "*input text*" is the input and "*annotated summary*" is the label. "*Annotated summary*" refers to the summary that is taken from the dataset and generated by human (label), when "*generated summary*" refers to the summary, that is generated by a model.

2.1 Background

2.1.1 Artificial Intelligence

In the recent decades artificial intelligence (AI) became very popular, which is used in our everyday lives [41]. AI can be seen in Google Translate, our phones or laptops. AI can cover various topics and problems, and it can be summarized as a technique that tries to imitate humans behavior using special techniques. People usually think about the AI as a self-aware computer that performs different tasks, however, AI models are created to do a specific task even if it is small. The difference between an AI algorithm and a simple code, is that the AI algorithm can adapt to certain

situations and behave appropriately. But, if the task that one tries to solve requires self-correction or a learning process, the subbranch of an AI is used called "machine learning".

Machine learning (ML) is a subbranch of AI algorithms that are "trained" to find patterns in vast amounts of data in order to make predictions, enabling machines to learn and to complete tasks without any explicit programming. Statistical approach and self-learning are used to achieve high results. ML has a few terms that are used in all problems:

- **Training** is a process that tries to find a pattern, which most accurately represents or distinguish between given classes in the dataset.
- **Model** is a result of an ML training procedure, which is used for prediction. The accuracy of the model depends on how good the model is trained.
- **Testing** is a process of testing a given model. The accuracy of the model is checked using this procedure.
- **Dataset** is a set of data that is used during training and testing.

Deep learning (DL) is considered as a sub branch of ML which relies on neural network architecture. DL input information from the dataset and by feeding it through the artificial neural network structure uses pre-defined class information to generate outputs. Artificial neural networks (ANN) have been inspired by how neurons in human brain are structured. The nervous system is the main part of a human body, which consists of neurons connected with to each other with synapse. Synapse is like a pathway that the signal travels from one neuron to another. We, as humans, have 5 senses, which are smell, hearing, touch, sight and taste. They are the input for our nervous system, which are encoded into electrical signals and later fed into the neuron structure, which at the end produce our reaction (result). An ANN works in a similar way, which input the data, passes it through hidden layers (neurons between input and output) and produces output. The best property of ANN is that the user should not specify which feature or pattern the model should look for to correctly predict the result: the model does it all by itself. Each cell in the neural network is connected with weights and biases. The weight control the signal between two neurons, while biases allow the activation function to be shifted by adding a specified constant. Learning procedure is a process of an ANN model to set weights in such a way that the model produces the most accurate output.

Deep learning and ANN models work the following way: the dataset is divided into training, validation (development) and testing sets (often the training consists of 80% of original size, where validation and testing have around similar size). The training dataset is then fed to the model, where the neural network analyzes given information and depending on how accurately the model labels the data the model's weight would be tuned so that the next iteration would produce higher results. Then this process is repeated, until the whole dataset has been input to the model. This is called an "epoch". After each epoch, the validation dataset is fed to the model to see whether the model overfits. Overfitting is a model's behavior, where the model gives accurate results for the training dataset, but not the testing dataset. Overfitting leads the model to memorize the training dataset, so that when a a new data instance is input, the model performs poorly. Underfitting is a different problem, where the model cannot generalize neither training not testing dataset. Therefore, the training dataset should be big enough, so that the model wouldn't memories it. After the model is trained, it is tested on the testing dataset. It is important that the testing dataset is never used during the training procedure, as it might affect how actually accurate the model is.

Bias-variance trade-off is a process of finding a good balance between overfitting and underfitting. Bias is the difference between an average prediction of a model and the actual correct values, when the variance is a variability of model predictions. When variance is high, the model pays a lot of attention to the data (overfitting), and when the bias is high, it means that the accuracy is too low (underfitting). The ideal situation is to have both variance and bias as low as possible.

Before models can be explained, a few fundamental aspects should be described. A simple ANN structure is shown in Fig. 2.1, where all elements are described below.

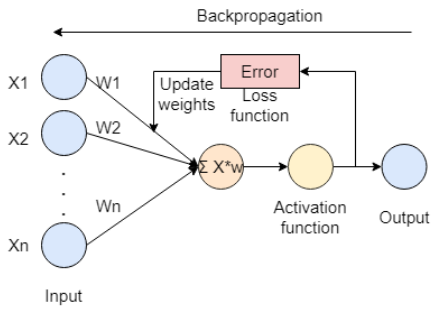


Figure 2.1: A sample of an ANN network with activation and loss functions.

2.1.2 Input data

In this thesis, we are using text as an input/output (input text/annotated summary), which cannot be fed into the model directly and, therefore, should be encoded into a numerical/vector representation. Text is combined into the dataset that is called corpus. For this corpus a vocabulary is created, which contains a set of unique words and its index number. This vocabulary is used to encode the text into the vector representation and if the word is not located in the vocabulary, it is designated as out-of-vocabulary (OOV). One of the most common approaches to encode text is to use bag-of-words (BoW) approach, which encodes a word as a zero vector of the length of vocabulary and only a single element of a vector has a value of 1, which gives the location of the word present in the vocabulary. In contrast, in this thesis, we are using an an embedding for a encoding the text.

Embedding

Word embedding is the approach of providing more useful information about the word and not only its location in the dictionary. Embedding is a process which maps words into a vector space, so that later vector representation of these words kept the relationship between these words. Word2vec [42] was one of the first and most popular embedding technique, which used neighbor words to determine the vector representation of a word. However, in our case, the embedding is trained using attention mechanism, which will be described later.

To each sentence the beginning-of-sentence <BOS> and end-of-sentence <EOS> tokens are added to signify to the model the beginning and the end of a sentence. Then, this sentence is padded to a specific length and encoded using the vocabulary.

2.1.3 Activation function

Activation function is an important part of an ANN, which is used to calculate weighted sum of its input and then decides whether the neuron should fire or not and how much. This activate function introduced non-linearity to the ANN model. There exist different activation functions and a few are described below.

Sigmoid function (often called the squashing function) is an S-shaped curve, which adds non-linearity to the model (Eqn. 2.1). It squashes the numbers between 0 and 1, even if they are either very large or very small.

$$Sigmoid = \frac{1}{1 + e^{-x}} \tag{2.1}$$

Softmax function (Eqn. 2.2) is used for computing probability distribution. Its output is between 0 and 1, and the sum of the probabilities is equal to 1.

$$Softmax = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{2.2}$$

2.1.4 Loss Function

The *loss function* is used in deep learning to determine how close the generated values are to true values and generates the error. In our case, we use Kullback-Leibler divergence loss function [43] (Eqn. 2.3) with *label smoothing*. The label smoothing [44] is used to increase the robustness of a model, as it penalizes overconfident outputs.

$$KL(P||Q) = \sum_i P(i) log(\frac{P(i)}{Q(i)}), \tag{2.3}$$

where P and Q are probabilities of each word in the vocabulary. The intuition behind it is that there will be a significant divergence if the probabilities are not close.

Backpropagation

Backpropagation is a method of tuning weights of a neural network model using the error that was obtained from the loss function. Backpropagation calculates the gradient for each weight, which are later changed using the optimization algorithm.

2.1.5 Optimization Algorithm

Optimization algorithms are used to calculates and updates values of a model's weights, where learning rate is used to signify the model whether it is moving in the right or wrong direction and to control how much the weights can be changed. The learning rate can be imagined as a step on a slope, where the lower the learning rate is, the slower the model traverse through the graph. The gradient descent optimization is used to find the local minimum on the graph, which minimizes the

general error. In our research, we use Adaptive Gradient Algorithm (AdaGrad) optimizer [45] (shown in Eqn. 2.4). There is another optimization function that we tested our models on, which is called Adaptive Moment Estimation (Adam) optimizer [46] (Eqn. 2.5-2.9). Each loss function has a learning rate, which determines how much the model should change its parameters based on the error from the loss function.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}, \tag{2.4}$$

where for each time step t for every parameter θ_i the initial step (learning rate) η is changed using $\sqrt{G_{t,ii} + \epsilon}$, where $G_{t,ii}$ is a diagonal matrix each element in a diagonal ii is a sum of the squares of the gradients of θ_i up to the time t and ϵ is a smoothing element which avoids a division by 0. $g_{t,i}$ is a gradient of objective function.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.5}$$

$$\upsilon_t = \beta_2 \upsilon_{t-1} + (1 - \beta_2) g_t \tag{2.6}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.7}$$

$$\hat{\upsilon}_t = \frac{\upsilon_t}{1 - \beta_2^t} \tag{2.8}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{\upsilon}_t} + \epsilon} \cdot \hat{m}_t, \tag{2.9}$$

where β_1 and β_2 are predefined (author proposed using $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^-8$. m_t and υ_t represent the first and second moments of the gradients (the mean and the uncentered variance, receptively). To overcome the bias towards zero, the author suggested using $\hat{m}_t$ and $\hat{\upsilon}_t$.

2.1.6 Dropout

We use a *dropout* [47], which is a technique that drops a specific number of neurons and their connections with a specified probability (in our case 20% is used). Dropout is helpful as the model learns to rely on more than just a group of specific neurons, which helps to address the problem of overfitting the model.

2.1.7 ROUGE Score

To evaluate the model's performance, the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [40] score is used as a benchmark based on the well-known recall

measure. It calculates the degree of overlap between the annotated and generated summaries using the number of N-grams (recall). N-gram takes N number of words as a sequence and uses it as a single entity. ROUGE has several scores, but only three are used the most: ROUGE-1 (R-1), ROUGE-2 (R-2) and ROUGE-L (R-L). ROUGE-N measures how many numbers of N-gram words in the annotated summary appear in the generated summary. R-1 uses 1-gram words, R-2 uses 2-gram words and R-L uses Longest Common Subsequence. So, for instance, if we have a text "*My dog jumped over the long fence*", the 1-gram (unigram), 2-gram and 3-gram will have the structures, as shown in Table 2.1. ROUGE-L is based on the Longest Common Subsequence (LCS), which calculates the longest sequence of words which two summaries share. In an example from Table 2.1, if we would have the second sentence as "Dog is over the fence", then the longest common subsequence would be "dog over the fence", as in LCS the words can be not next to each other, however, they should be in order.

	N-gram
1-gram (Unigram)	["My", "dog", "jumped", "over", "the", "fence"]
2-gram (Bigram)	["My dog", "dog jumped", "jumped over", "over the", "the fence"]
3-gram (Trigram)	["My dog jumped", "dog jumped over", "jumped over the", "over the fence"]

Table 2.1: Sample of N-grams

However, to test text summarization models, F_1 score of ROUGE-N is calculated, which includes both recall and precision measures. $ROUGE_{F_1}$ score is calculated using recall and precision, where *recall* calculates the number of N-grams found in both texts and divided by the number of N-grams in the second text (shown in Eqn. 2.10), and *precision* calculates almost the same, but the number of N-grams found in both texts is divided by the number of N-grams in the first text (shown in Eqn. 2.11). For instance, R-1 recall of the annotated sentence "My dog jumped over the long fence" and the generated summary "The dog jumped" would be $3/3 = 100\%$. R-1 precision would be $3/7 = 43\%$. To calculate $ROUGE_{F_1}$ score, we use this formula given in Eqn. 2.12.

$$recall = \frac{count(N\text{-}gram_{text_1} \cap N\text{-}gram_{text_2})}{count(N\text{-}gram_{text_2})} \tag{2.10}$$

$$precision = \frac{count(N\text{-}gram_{text_1} \cap N\text{-}gram_{text_2})}{count(N\text{-}gram_{text_1})} \tag{2.11}$$

$$ROUGE_{F_1} = 2 * \frac{precision * recall}{precision + recall} \tag{2.12}$$

2.2 Transformer

Transformer model [1] is used for text summarization and it does not contain any sequential elements like RNN or LSTM, that are used in Sequence-to-sequence (*Seq2Seq*) model [22] (see Appendix A.3). Transformer is not based on a Seq2Seq model, however, it uses an encoder-decoder structure as well. The paper that implemented the transformer model argued that even though the Seq2Seq model with attention tries to solve the problem of the usage of sequential elements, it still uses the fundamental concept of a sequential algorithm. Therefore, transformer solely relies on the attention mechanism using a simple feed forward neural network for training.

2.2.1 Transformer Model

The transformer model [1] is shown in Fig. 2.2. The encoder encodes the input text and passes this information to the decoder, which combines it with an encoded annotated summary to output the probabilities of each word from the vocabulary being generated. The decoder is auto-regressive, meaning that it generates the next word by giving the previously generated sequence. The next sections describe each item of the transformer.

2.2.2 Input/Output Information

The information about the input data has been discussed in section 2.1.2, however, a more detailed description is needed. In Table 2.2, we give an illustration of a normal and encoded text used in this thesis. In this thesis, a vocabulary of 50,000 words was created, where words are sorted in decreasing order, depending on the number of times they occur in the whole dataset. The vocabulary has information about the word and its index. Throughout this chapter, a specific example of an input text and

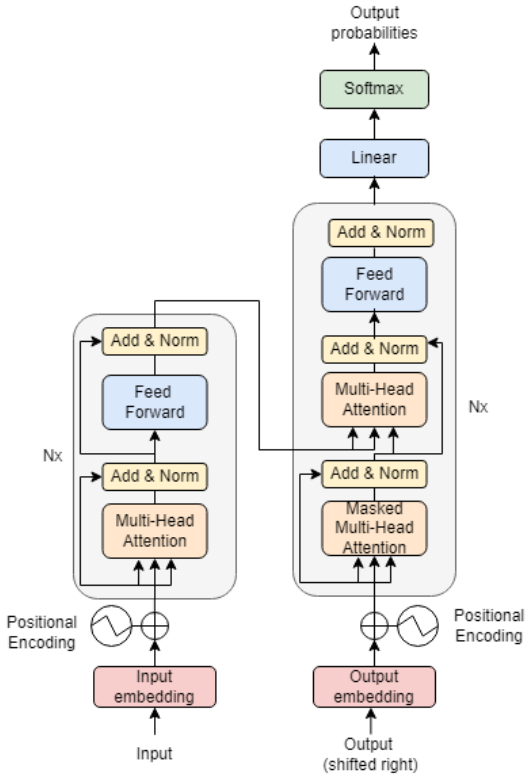


Figure 2.2: Transformer architecture. The drawing was based on a figure from [1]. "*Output (shifted right)*" is a part of annotated summary during training or a part of generated summary during testing, which is shifted right because <BOS> symbol is at the beginning. The decoder output is passed through the linear function (which is a simple feed forward neural network by connecting the input to the output) and later the softmax function generates probabilities.

an annotated summary is used, which is shown in Table 2.2. All values are created manually and not using the actual model. The idea behind it is to show the flow of data inside the transformer. Appendix A.1 shows an actual example and values for input/output and hidden layers.

From Fig. 2.2 the "Input" inside the transformer is shown as an encoded text of an "Input text" from Table 2.2, when the "Output (shifted right)" is an encoded text of an "Annotated summary" during the training process. An annotated summary is shifted right because the annotated summary uses <BOS> and <EOS> to signify the beginning and end of a sentence, so for generation of a word i only the context from *0* to *i-1* is used. A detailed explanation of how encoding is done is described in section 4.2.2.

Text version	Input text	Annotated summary
Text	Last summer I have visited such cities as New York , Philadelphia , Boston and Washington and it was great	<BOS> My trip to America was great <EOS>
Encoded text	[70, 595, 29, 28, 1641, 162, 1698, 26, 62, 256, 6, 0, 6, 1693, 9, 600, 6, 20, 14, 298]	[2, 84, 916, 7, 575 14, 298, 3]

Table 2.2: A sample of input text and annotated summary in normal and encoded format. Encoded text is a text that was created using the vocabulary, where each word is replaced with word's index in the vocabulary. 0 means OOV word. <BOS> and <EOS> are used to signify the beginning and end of a sentence.

2.2.3 Input/Output Embedding

Input/Output embedding is a process of converting encoded text to a vector representation. The embedding is a vector of floating point values, which are trainable parameters. In the beginning, those values are randomly generated, and as the model trains on data, the model makes changes to the embedding. In the thesis, the embedding of 256 is used, however, in the following example, only an embedding of 4 is used (Table 2.3).

Text version	Input text	Annotated summary
Encoded text	[70, 595, 29, 28, 1641, 162, 1698, 26, 62, 256, 6, 0, 6, 1693, 9, 600, 6, 20, 14, 298]	[2, 84, 916, 7, 575 14, 298, 3]
Embedded text	[[0.1, 1.3, -0.2, 4.3], [0.5, 0.1, 0.2, 0.8], ..., [-0.4, -1.2, 0.7, 0.2]]	[[-0.1, 0.2, 0.4, 1.3], [2.3, 1.5, 0.7, 1.1], ..., [-1.4, -1.5, 1.7, -1.2]]

Table 2.3: A sample of encoded and embedded text, where floating values are randomly generated.

2.2.4 Positional Encoding

The transformer model does not use any sequential elements, which keeps track of whether the word appears before or after each other, therefore, the information about the position of a word is lost. Positional encoding is used to encode the position of the word using *sin* and *cos* functions of different frequencies Eqn. 2.13-2.14 [1], where i is the vector dimension and *pos* is the position. The intuition behind this approach is that by adding positional encoding to the embedding vector shifts the embedding according to its location in the text and the model can distinguish whether the word is used before or after another word.

$$PE_{pos,2i} = sin(pos/10000^{2i/d_{model}}) \tag{2.13}$$

$$PE_{pos,2i+1} = cos(pos/10000^{2i/d_{model}}) \tag{2.14}$$

Text version	Input text	Annotated summary
Embedded text	[[0.1, 1.3, -0.2, 4.3], [0.5, 0.1, 0.2, 0.8], ..., [-0.4, -1.2, 0.7, 0.2]]	[[-0.1, 0.2, 0.4, 1.3], [2.3, 1.5, 0.7, 1.1], ..., [-1.4, -1.5, 1.7, -1.2]]
Embedded text with positional encoding	[[0.4, 1.5, -0.1, 4.2], [0.5, 0.2, 0.5, 0.7], ..., [-0.9, -1.4, 0.6, 0.3]]	[[0.1, 0.4, 0.4, 1.2], [2.4, 1.5, 0.6, 1.0], ..., [-1.8, -1.7, 1.5, -1.1]]

Table 2.4: A sample of embedded text and embedded text with positional encoding.

2.2.5 Attention

The main part of the transformer model is an attention mechanism, which the transformer relies on. Attention [20] was designed to overcome the problem of long dependencies. It is used for the model to identify how much attention should be placed on other words in the context given the input word and the decoder hidden layer. The attention distribution is calculated using a specific score function, which calculates the relevance of an input token to the target context. This score can be calculated using either dot-product attention or multi-layer perceptron [20], however, in this

thesis only dot-product attention is used, which calculates the dot-product between two vectors (Eqn. 2.15). Then the attention weights are calculated using the softmax function. These attention weights are later used to change the input or the embedding representation of words in the self-attention layer.

$$\text{Dot-product} = \boldsymbol{a} * \boldsymbol{b} = \sum_{i=1}^{n} \boldsymbol{a}_i \boldsymbol{b}_i \tag{2.15}$$

where a and b are two vectors.

2.2.6 Self-attention

In self-attention, tokens interact with each other to understand the context where each word is used, and later this information (or attention) is used to change the word embedding. Fig. 2.3 shows a sample of self-attention inside the encoder based on the example from Table 2.2. To calculate the weighted attention, the model uses the following equation Eqn. 2.16. The keys (K) are a group of vectors, where each key has an associate value (V). In the previous example, it would be an embedding of an input text. The query (Q) is a vector and contains information that we want to find. In the previous example, it is the embedding representation of each word. The key with the largest dot product with Q (as shown in Eqn. 2.15) will be selected, and softmax will map the key to the exponential function and divide by the sum of the exponential, which will create a distribution, that is used later to change the values V. To avoid small gradient, the paper [1] divides the dot product of Q and K by d_k, where d_k is a dimension of K.

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

From our example, this self-attention is used for an input text. Firstly the attention is calculated (Fig. 2.4) using softmax of dot-product between Q and K (the attention), where Q, K (and V) are embedded text with positional encoding from Table 2.5. Later, this information is used to multiply by V, which changes the initial embedding of each word.

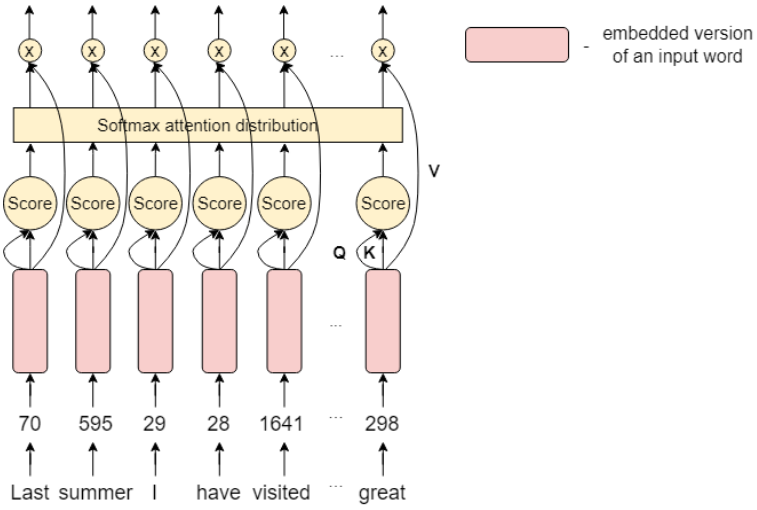


Figure 2.3: A sample of a self-attention mechanism inside the encoder. Q, K and V represents queries, keys and values respectively (word embedding as a vector).

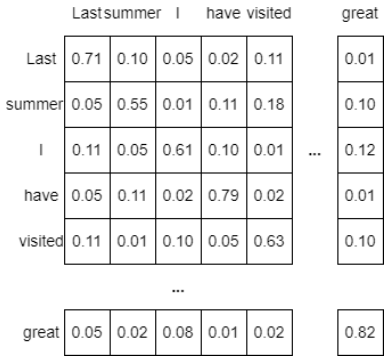


Figure 2.4: A sample of a softmax attention distribution for input text. Each value represents how much attention each word puts on another word.

Text version	Input text (V)
Embedded text with positional encoding	[[0.4, 1.5, -0.1, 4.2], [0.5, 0.2, 0.5, 0.7], ..., [-0.9, -1.4, 0.6, 0.3]]
Embedded text with positional encoding after attention	[[0.3, 1.1, -0.1, 3.5], [0.1, 0.1, 0.5, 0.3], ..., [-0.5, -0.7, 0.4, 0.1]]

Table 2.5: A sample of embedded text with positional encoding and the same embedding but after multiplied by attention.

2.2.7 Masking

The decoder contains a self-attention mechanism as well in the transformer model. However, the decoder's self-attention should not be able to focus on information that is located further in the text, therefore, masking is used. Masking ensures that the next word i is generated using the information from the annotated summary up until the position less than i. At each step i only a context from 0 to $i-1$ is used and no further context. An example in Fig. 2.5 is used to demonstrate that for the word "have" only the previous context was used, and all other information is not (white). Fig. 2.6 shows an example of masking. The mask contains "True" or "False" values, depending on whether the word should be used or not. All padded symbols are masked with "False". It can be seen that for each row, a part of an annotated summary is chosen. For instance, for position "2" the "<BOS> My trip" part is chosen.

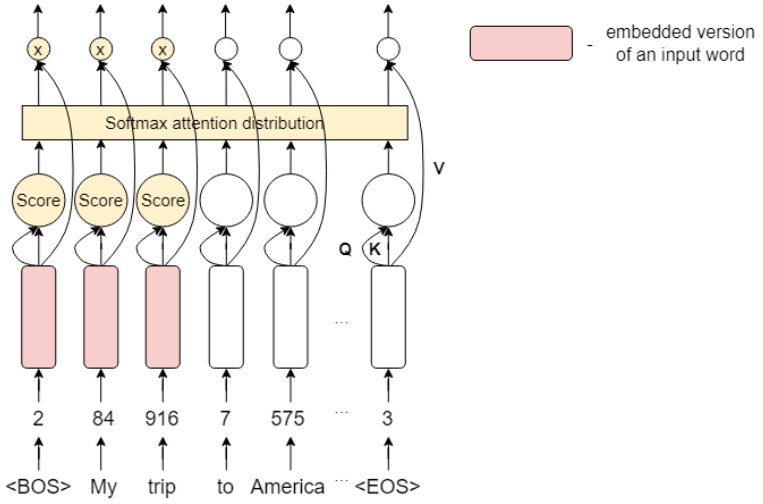


Figure 2.5: A sample of a self-attention mechanism inside the decoder with masking. Q, K and V represent queries, keys and values respectively, which are vectors and represent word embedding. The white circle and white rectangle represent that this information is not used.

From our example, this self-attention is used for an annotated summary. Firstly the attention is calculated (Fig. 2.7) using softmax of dot-product between Q and K (the attention), where Q, K (and V) are embedded text with positional encoding from Table 2.6. Later, this information is used to multiply by V, which changes the initial embedding of each word.

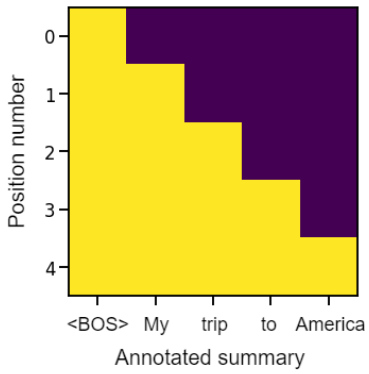


Figure 2.6: Masking of an annotated summary. The yellow colour means "True", when dark colour means "False". <BOS> is the "beginning of sentence" symbol (more is explained in section 4.2.2)

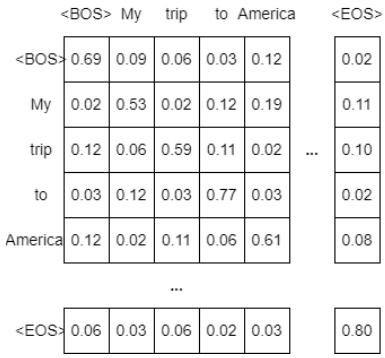


Figure 2.7: A sample of a softmax attention distribution for annotated summary. Each value represents how much attention each word puts on another word.

Text version	Input text (V)
Embedded text with positional encoding	[[0.1, 0.4, 0.4, 1.2], [2.4, 1.5, 0.6, 1.0], ..., [-1.8, -1.7, 1.5, -1.1]]
Embedded text with positional encoding after attention	[[0.1, 0.3, 0.1, 0.4], [1.2, 0.9, 0.3, 0.5], ..., [-0.4, -0.6, 1.1, -0.5]]

Table 2.6: A sample of embedded text with positional encoding and the same embedding but after multiplied by attention.

2.2.8 Multi-Head Attention

Self-attention is calculated using multi-head attention [1], which is a function that maps Q and a set of K-V pairs to output in vector representation. It allows the model to focus on different positions and gives multiple representation subspaces.

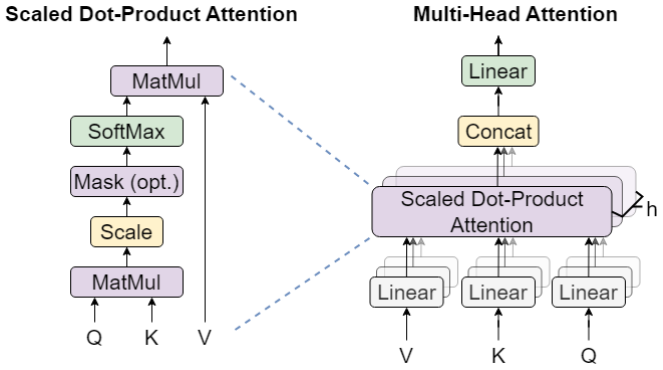


Figure 2.8: Multi-Head Attention. The drawing was based on a picture from [1]. V represents input information which might be important for generating the next word, K represents indexing of how those values can be accessed, and Q represents information which the model tries to find. "h" in Multi-Head Attention represents the number of attention layers which run simultaneously.

2.2.9 Residual Connection and Layer Normalization

Residual connection [48] is used after each multi-head attention and a feed forward neural network layer in the transformer model. A residual connection is used as a skip connection, where the input is added to the output (Fig. 2.9). The residual connection allows the gradient to flow through the network, and without it, a large part of the training signal (i.e., the original state) might be lost during backpropagation.

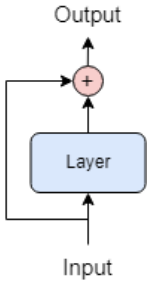


Figure 2.9: Residual connection

The layer normalization is used for convergence stability and quality. The normalization layer is placed after the residual network, and the whole equation looks as the following $LayerNorm(x + Sublayer(x))$, where x is the input, $LayerNorm$ is the normalization layer and $Sublayer$ is a specified sub-layer (like self-attention or feed forward network).

2.2.10 Feed Forward Neural Network

A feed forward neural network is an ANN where the connections do not form a loop, where all information only passes forward (Fig. 2.10).

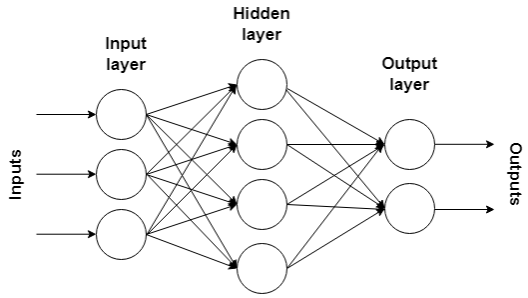


Figure 2.10: Feed Forward Neural Network

2.2.11 Encoder and Decoder

The encoder consists of a self-attention and a feed forward layer, where the residual connection is applied for each layer. The encoder and decoder are stacked in a group of Nx (paper [1] suggests $Nx = 6$), which is shown in Fig. 2.11. Stacking is used because using a single encoder/decoder architecture is not enough to capture the complexity of the language. The input of an encoder is passed through the self-attention layer, which allows it to look at all words, and then it is passed through a feed forward layer. At each step, the output is normalized and added to its sub-layer. Each sub-layer employs residual connection, which is followed by layer normalization. The embedding and the model size have the same value to boost the residual calculations. Then this information is passed to the decoder. In addition to self-attention and feed forward layers, the decoder has a third intermediate layer, which helps it focus on important input information. It performs multi-head attention over the previously generate summary (during testing) or a part of the annotated summary. The encoder changes the embedding of all words depending on the context each word is used in.

At the same time, the decoder scans the part of the annotated summary using masked multi-head attention and using the encoder's hidden layer generates the probability distribution of the next word being generated.

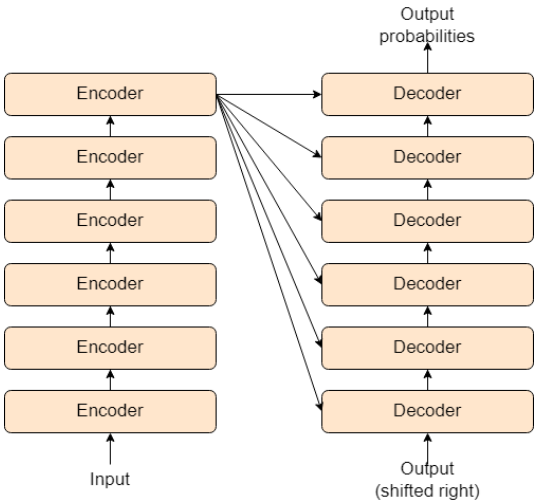


Figure 2.11: Stacked encoder

How Transformer Works

In the beginning, the vocabulary of used words is created, which contains information about a word and its index, which encodes the sentence. In each sample, an encoder receives an input text with a specific length (for instance, 400 tokens). The decoder receives as an input a part of the summary that has already been generated (or a part of an annotated summary). Then, the model process it using an attention mechanism and outputs probabilities of generating each word in the vocabulary. Then this process is repeated M number of times (where M is the maximum number of words the model can generate) or until the <EOS> symbol is generated. During training, it is possible to combine this process, as it is already known the future context. So masking is used to hide future context and an array of masked annotated summary is fed, where for each position i only the context from 0 to $i - 1$ is used.

An example of how the actual input/output and hidden layer look like are shown in Appendix A.1.

2.3 Proposed Model

In this section, the proposed model is described. It consists of several elements, such as a transformer with a pointer-generator layer, attention with frequency information and an extractive approach.

2.3.1 Transformer with Pointer-generator Layer

Proposed model as an M1 model shown in Fig. 2.12 has a transformer backbone. Additionally, we add a pointer-generator layer from [31], which decides whether to copy or generate a word. Original pointer-generator model [31] uses Seq2Seq attentional model. So we make adjustment to fit this layer to the transformer model, by using a multi-head attention and dot product attention to calculate the context vector. The pointer-generator layer does not have any sequential elements.

Additional temporary vocabulary is created to handle OOV words, where all OOV words from the input text in a given batch are present. Indexing in this vocabulary starts from the last index inside the original vocabulary. The input text and the annotated summary are encoded using both vocabularies. This vocabulary is created so that the model would know which word it should copy if the model decides to copy rather than generate a new word.

Pointer-generator layer is inserted as the last layer in a transformer model. It uses a context vector (h), which uses information about the input text and the hidden state of the decoder. To get the *context vector*, we use another multi-head attention layer, where K and V are the output of an encoder, and the Q is an output of the normalization layer inside the decoder. Later, the context vector's attention is used to decide whether the word should be copied and multiplied by $1 - p_{gen}$ and the output of the model (s) is used with the context vector to decide whether the word should be generated and it is multiplied by p_{gen}. p_{gen} contains probabilities of generating word vs copying it. p_{gen} is calculated the following way (Eqn. 2.17 [31]): the context vector, the hidden layer of the decoder and the decoder input (x) after the positional encoding (embedding of summarized text after positional encoding) are concatenated together and input into the sigmoid function.

$$p_{gen} = \sigma(w_{h*}^T h + w_s^T s + w_x^T x + b_{ptr}), \tag{2.17}$$

where $w_{\_}^T$ represents the weight matrix that is used inside hidden layers and b_{ptr} is a bias.

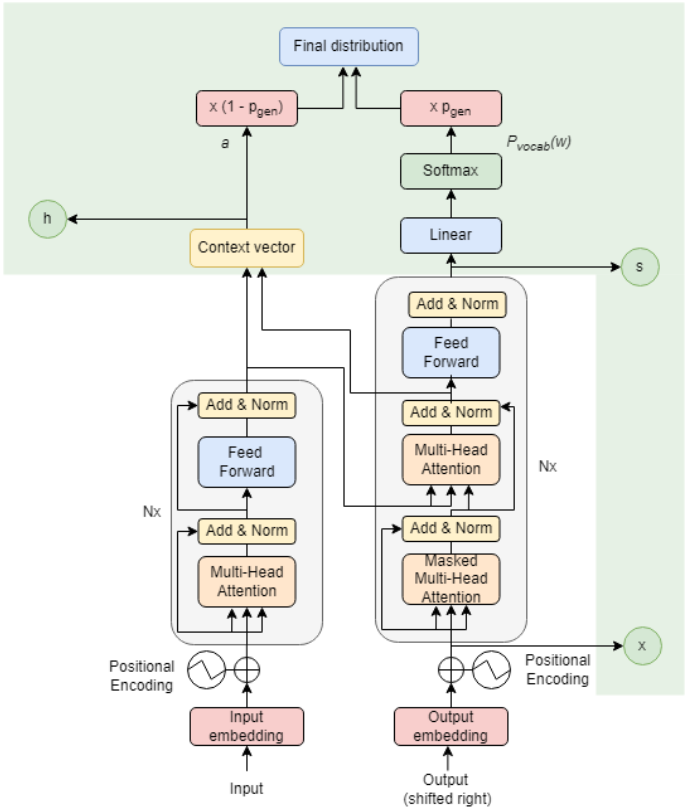


Figure 2.12: Transformer with pointer-generator layer (highlighted in green). The drawing was based on a picture from [1]. $\boldsymbol{x}$ represents the decoder input after positional encoding, $\boldsymbol{s}$ is the output of a decoder and $\boldsymbol{h}$ is a context vector. The decoder generates the probability of a word being generated, when the context vector has the attention distribution across the input text. Using the parameter $\boldsymbol{p_{gen}}$ (which is defined later) the model can regulate how much both distributions affect the final distribution. Attention from context vector $\mathbf{a}$ and probability of a word being generated $\boldsymbol{P_{vocab}(w)}$ are used to calculate the final distribution.

Later, to calculate the *final distribution* $P(w)$, the following formula is used Eqn. 2.18 [31], where $P_{vocab}(w)$ is a probability distribution over all words. Probabilities of a word being generated are multiplied by p_{gen}, and the attention distribution is multiplied by $1 - p_{gen}$ and those two distributions are added to each other to form a final distribution of the next word. The attention distribution a represents the copy mechanism which contains the attention for all words in the input text, so if the word is in OOV or belongs to OOV, the first part (which is multiplied by p_{gen}) will be closer to zero. The attention for this OOV word, which appears in the second part, will be added to the final distribution, which leads to "copying" the word with the highest attention. On the contrary, if the word does not appear in the input text,

then the $1 - p_{gen}$ will be closer to zero, and the decoder prediction of the next word being generated will be used. However, when the summary contains an unknown word (OOV token is used), which cannot be encoded with a vocabulary or being encoded, the model will learn to generate also an OOV token.

$$P(w) = p_{gen}P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} a_i \tag{2.18}$$

where a_i represents the copy mechanism which contains the attention for a word i in the input text.

2.3.2 Attention with Frequency Information

The self-attention mechanism inside the encoder focuses on important information to change the input text embedding and shows how much attention the model should put on the input words. However, the model does not have information on the *importance* of a word and learns it throughout the training phase. Therefore, we propose an approach of adding information as a frequency score about the frequency of a given word. Frequency score is passed to the multi-head attention inside the encoders self-attention mechanism through the linear neural network, multiplying it with the attention values (Eqn. 2.19). Our idea is that by boosting the attention of rarely used words (which are considered to be the most important words) and limiting the model's attention to frequently used words, it can help the model to put higher attention on uncommon words, which might contain some important information. For instance, if the word is rarely used, then the attention for this word will be much higher than for commonly used words, like "the" or "an".

$$attn_{freq} = attn * W^T_{s_{freq}} s_{freq}(k), \tag{2.19}$$

where $W^T_{s_{freq}}$ represents the weight matrix that is used inside hidden layers. s_{freq} is the frequency information about the word in the position k.

To calculate the frequency score (importance of a word) the self-attention of the encoder layer is multiplied by the score of a word, which is obtained by the inverse logarithm of how many times the word appears in the dataset (given in Eqn. 2.20). For most numbers, its inverse is lower than 1, therefore, the model will decrease attention for almost all words and only boost a few. To boost values for more words,

we shift all scores to a mean value of 1. This approach will decrease attention to the most commonly used words, while it will increase for rare words.

$$word_{freq} = \frac{1}{log(occ)} + shift, \tag{2.20}$$

where "occ" is the number of occurrences of a specific word in the training dataset and "shift" is calculated in a way so that all variables mean value is 1.

Scores of all words are predefined; therefore, it is saved into a dictionary before training (Table 2.7 shows what a predefined dictionary looks like). If the word does not appear in this dictionary (unknown word), the maximum score value will be assigned, as it might contain some important information (for instance, a name/surname or a city, which is rarely used). The final model that is used as an M2 model is shown in Fig. 2.14.

Word	Number of occurrences	Score
the	11,896,156	0.792
mid-flight	170	0.926
million-a-year-deal	1	2.17

Table 2.7: Dictionary of frequencies and scores for each word. The word "the" is used more often (11,896,156 times) in the CNN/DM dataset; therefore, its score is lower than the word "million-a-year-deal", which is used only once in the dataset.

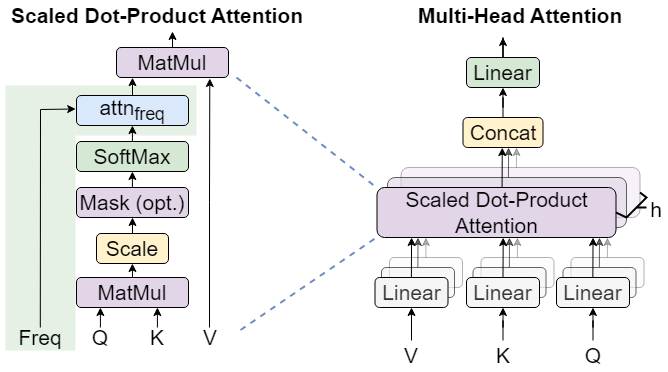


Figure 2.13: Multi-Head Attention with frequency information. The drawing was based on a picture from [1]. Added frequency information for each word (highlighted in green). Additional parameter of words frequency score is passed.

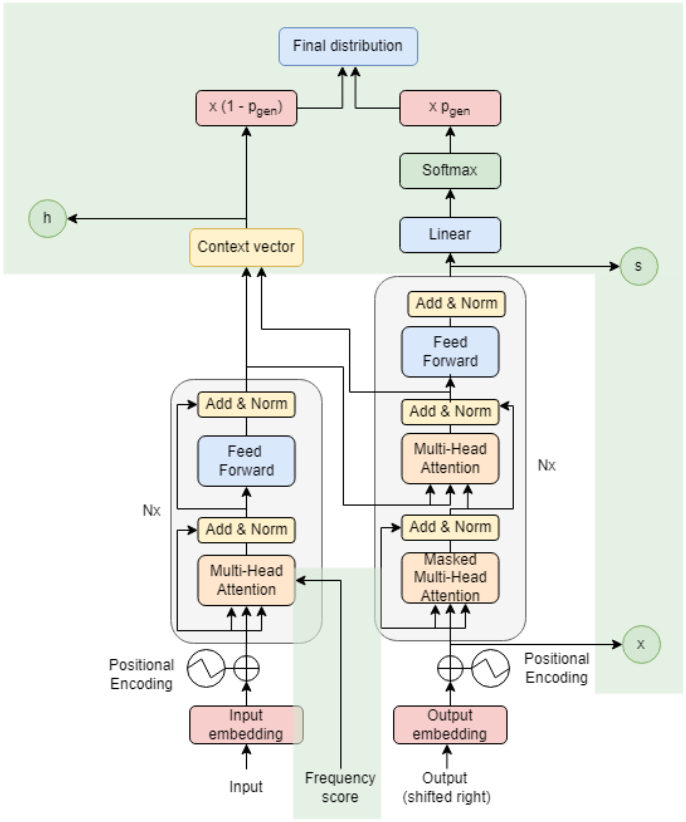


Figure 2.14: Transformer with pointer-generator layer and added frequency score inside the encoder self-attention mechanism (highlighted in green).

2.3.3 Extractive Approach for Rare Information

Extractive summarization copies parts of the text from the input text (such as sentences, phrases or words) and combines it to generate a summary. This approach does not generate any new words or information. Even though such an approach might not rephrase what is in the input text, sometimes, it might produce better and more grammatically correct summaries than an abstractive one. Extractive summarization works the following way: firstly, it creates an intermediate representation of an input text, which finds important parts in the text. Secondly, each sentence/phrase is assigned a score, representing the importance level. Finally, sentences which have the highest scores are selected and combined to generate a summary.

There exist two types of how to extract an intermediate representation of a text: *topic representation* and *indicator representation*. Topic representation focuses on topics in the text, and the importance can be found, for example, by the number of topics discussed in the text. Indicator representation transforms each sentence into

a list of features, which determines its importance, such as sentence length, sentence position, whether the sentence contains a particular word or a phrase, and others. Using these sets of features, the importance of sentences is determined using two approaches: graph methods and machine learning methods.

For the extractive approach, several techniques can be used, such as sentence tokenization, word tokenization, part-of-speech tagging, lemmatization, stemming, stop words removal (such as "a" or "the"), regular expression (regex), and others.

Proposed Approach with Indicator Representation

The algorithm for an extractive approach is described in Alg. 1, and the pipeline is shown in Fig. 2.15. To *tokenize* sentences (separate the text into individual sentences) predefined function from the *nltk* library[1] is used (line 3). Stop words are removed (line 4), as they are a distraction when selecting the most important information. Usually, the first few sentences contain the most important information, and it is useful to take the first k sentences (line 5). For instance, we take the first ten sentences when the limit of an input text is 400 tokens). We use N-gram approach, which takes N number of words as a sequence and uses it as a single entity. In our approach, we extract the level of importance by using the indicator representation of 3-gram scores of phrases in each sentence (3-gram performed better than 2 or 4-gram models), which are later used to select important sentences (line 7).

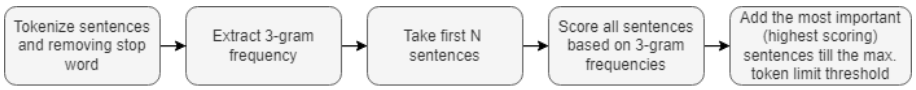


Figure 2.15: Extractive approach pipeline

N-gram frequency scoring works the following way: the predefined dictionary of N-gram words from a training dataset is extracted, where we count how many times each N-gram is used in the entire dataset. Then, the inverse logarithm of a frequency is calculated, using the same formula as frequency information of words given in Eqn. 2.20, however, the shifting is not used, as there it does not change the importance information (N-grams, which are mentioned only once, are dropped and not included, due to the problem of calculating the inverse logarithm of 1). Table 2.8 gives sample dictionary of 3-gram words with the number of times they occur and their scores. In

[1] `https://www.nltk.org/`

Algorithm 1: An algorithm for an extractive approach

Input : dataset, // Preprocessed dataset
first_k, // Take first k sentences (first_k is in 5, 10)
N // number of tokens to be extracted (N is in 200, 400)

Output: out_dataset: Dataset, which contains important information

```
for input_text in dataset do
    important_sentences ← [ ];
    tokenized_sents ← nltk.sent_tokenize(input_text);
    tokenized_sents ← remove_stopwords(input_text)
    important_sentences ← FirstKSents(first_k);
    for sent in tokenized_sents do
        scores ← ComputeScoreForSentence(3_gram);
        // Using Eqn. 2.20. Each sentence gets a mean value of a
           3-gram group
    important_sentences ← SelectImportantSents(N, scores);
    // Select important sentences in addition to the first k
       sentences
    AddProcessedText(out_dataset, important_sentences);
    // Important sentences are added to the output dataset
```

the end, each sentence is scored based on the N-gram scores, where each sentence's mean score is calculated. Unknown N-grams are assigned a maximum score, as this phrase has not been seen before; therefore, we assume that it might contain some important information. We have tried to use different values for the N-gram approach, and the 3-grams approach showed the best results.

3-gram	Number of occurrences	Score
('wales', 'british', 'irish')	45	0.605
('british', 'irish', 'lions')	504	0.370
('irish', 'lions', 'fly-half')	6	1.285

Table 2.8: Dictionary of 3-gram frequencies and scores. The score information for ('irish', 'lions', 'fly-half') is higher than for the other 3-grams because it is used less often in the given dataset. Later, for each score, the mean value of all scores is added to shift them to the right so that the mean would be close to 1.

2.3.4 Greedy Algorithm and Beam Search

The model outputs a probability distribution over each word in the vocabulary. The next step is to use an algorithm which picks the word that should be selected next. The two most popular algorithms are *greedy algorithm*[2] and *beam search* [49]. The *greedy algorithm* chooses the optimal choice at each stage, while the *beam search* explores the graph of nodes, by looking at nodes, which produce the highest probability.

The greedy algorithm takes a word with the highest probability at a given step without considering any previous steps (shown in Fig. 2.16). This algorithm is fast, but not very accurate.

On the other hand, beam search keeps track of k most likely outputs. The most common values for k are between 5 and 10. At each step, k candidates decide to choose the next output word, probabilities are multiplied, and k candidates with the highest probability are chosen for the next step. As multiplying probabilities leads to small numbers, the natural logarithm of probabilities is used to keep large numbers. An example of how a beam search works are shown in Fig. 2.17.

It is possible to modify the beam search using different approaches. For instance, the study [50] suggested using the patience factor, which controls how many finished

[2] `https://xlinux.nist.gov/dads//HTML/greedyalgo.html`

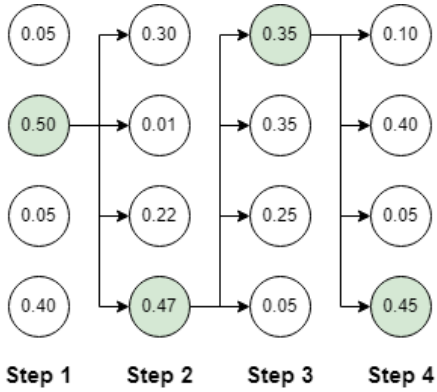


Figure 2.16: Greedy algorithm example

Remark 2.3.1. *In the case from Fig. 2.16, the vocabulary size is 4, where a probability is generated for each word at a specific step. Greedy algorithm selects the best choice at every step and does not consider other possibilities. In this case, the following probabilities would be selected: [0.50, 0.47, 0.35, 0.45].*

sentences have been found. Code modification is simple, as only one line needs to be changed. Another paper [51] introduced several modifications, such as length penalty and coverage penalty, which are implemented inside beam search instead of a transformer model. Their results show that the length penalty improves results significantly (at least 1 ROUGE point), and coverage implementation has a slight effect. In our version of a beam decoder, we use the same length control algorithm as in [52]. Some papers suggested different algorithms to control the length (for instance, the paper [53] used decoder hidden state and context vector to generate the context length vector); however, the beam search length control function is much easier to implement.

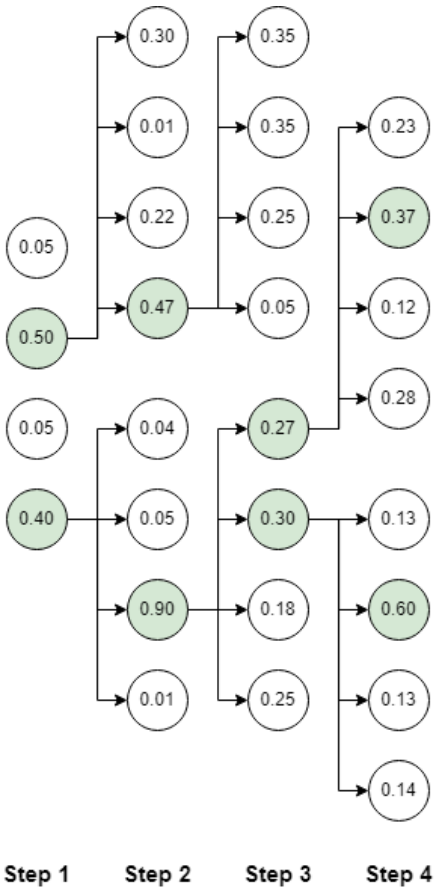


Figure 2.17: Beam search example (k=2)

Remark 2.3.2. *In the case from Fig. 2.17, the vocabulary size is the same as in Fig. 2.16. The beam search algorithm considers two candidates at every step. The higher the general probabilities each word gives, the more likely a beam search algorithm will select it. It can be seen that, unlike the greedy algorithm, beam search selected a different path, which generates a higher probability overall. It does not explore the upper branch after the step 3, because the algorithm found the best probabilities in the bottom branch. In this case, the following probabilities would be selected: [0.40, 0.90, 0.30, 0.60]. We can see that even though the first probability is smaller than the list that the greedy algorithm chose, the overall list of probabilities is higher; therefore, the sentence generated using beam search would be more similar to the annotated summary.*

Chapter 3

Related Work

There are two types of automatic text summarization approaches: *extractive* [10, 11] and *abstractive* [12, 13]. An extractive method generates summaries using sentences from the input text. In contrast, in an abstractive approach, the text is generated using an external vocabulary of words, which might include words that are not present in the original input. The combination of extractive and abstractive approaches is called a hybrid approach. The intent of this approach is to overcome the weaknesses of the extractive and abstractive approach [14, 15, 16].

3.1 Extractive Approach

Most of the early text summarization research focused on extractive approaches [54, 55, 56]. The general idea of an extractive approach is to get an intermediate representation of words/sentences and sort them by the level of importance. Some of the early text summarization techniques are commonly used in extractive approach models. For instance, TF-IDF (term frequency-inverse document frequency) [57, 58] is based on frequency, whereas TextRank (PageRank) [59] is based on graphs. A node (vertex) represents sentences and the edges represent relationships between sentences. The number of edges (connections) determine the importance of a node (vertex). In the PageRank algorithm, the nodes are also weighted so that each node can add a different level of importance. The similarity between two sentences is determined by the amount of overlap of content (e.g., number of common words). This approach computes a score of each sentence's *importance* and the top scoring sentences are combined to generate a summary. In [60], genetic algorithms (solve an optimization

problem based on a natural selection) are used to generate a summary, such as fitness function (affects whether the candidate makes it to the next generation), crossover (combining the best traits of parents into children) or mutation (to encourage further diversity). The authors used weights ranging between 0 and 1 as a genetic algorithm's population, where each weight corresponds to a word or a symbol in the vocabulary. In [61], the authors discussed various graph-based models, such as pagerank, hits, closeness, betweenness and degree measures.

A discussion of the trade-off between compression and retention, meaning including as much important information as possible without generating too much text was presented in [62]. The authors proposed using several features, such as semantic similarity, z-score, and unique tokens capturing and combining it with the TF-IDF model to get the most accurate result.

In [10], the authors proposed using feature extraction and enhancement to generate the summary using an extractive approach with deep learning. For each sentence, a score has been calculated and based on these scores, a group of sentences has been selected as a generated summary. Another paper [11] tested the use of a convolutional neural network (CNN) [63] with pre-trained word vectors to improve the performance of the model.

The main drawback of an extractive approach is that a summary is generated by combining sentences/phrases from the input text. Although it might include important information and achieve high performance, the trained model itself does not create generated summary; instead, it is copied from the input text.

3.2 Abstractive Approach

In recent years, with the help of Natural Language Processing (NLP), abstractive text summarization has become more popular. Encoders and decoders with *attention* mechanism are most commonly used. An encoder and decoder structure was first proposed in [19], and later, an attention mechanism was included [20, 21]. The attention mechanism uses all the intermediate states to generate the prediction, even if words are located far between each other. So instead of using only the final hidden vector, the model with attention can focus on the most important aspects.

Facebook [13] released its version of the abstractive summarization technique, where sentence-level summarization was used. They created a local attention-based model, which generates each word of the generated summary depending on the input

text. Unfortunately, the model has problems with analyzing previous unseen, rare words, and it might miss important information. Another paper by Facebook [64] targeted a problem of irrelevant information, and they used conditional RNN, which makes the decoder focus on appropriate input words.

Most frequently a variation of *sequence-to-sequence* and *transformer* are used, which are described below.

3.2.1 Sequence-to-sequence Models

The study [12] used a sequence-to-sequence encoder-decoder-based model, where in addition to the word, a few extra parameters have been used, such as part-of-speech tag, named-entity recognition tag, TF-IDF. They have overcome unseen words problem by using a pointer-generator approach, where the model decides whether it should copy or generate a new word.

The authors [31] created a pointer-generator model based on a sequence-to-sequence attentional model (similar to the study [12]), which can copy words from the original text (pointer) while being able to generate new words (generator). Such a model helps to improve the problem of dealing with unseen words and identifying important information. Attention is used to produce a context vector; however, the encoder-decoder backbone uses a single-layer bidirectional LSTM. Additionally, they have used a technique from the paper [65] by creating a coverage vector that tracks which information has already been included in the generated summary to avoid duplication. The model has been trained on the CNN/DM dataset and achieved a R-1 score of 39.53, a R-2 score of 37.28 and a R-L score of 36.38.

Another paper [66] used the pointer-generator model introduced in [31] to make the model generate more abstract words. They created an OOV penalty, which would control and improve the number of novel words in the generated summary. Even though they did not achieve higher results than the model from the paper [31], they stated that their model could generate more abstract text.

3.2.2 Transformer Models

Google [1] introduced a transformer that eliminated recurrent aspects of a model and relied only on an attention mechanism. Instead, they have used an encoder-decoder structure, where the encoder generates a hidden vector and inputs it to the decoder with already generated sequences.

In another paper by Google, [67] authors worked with a problem of the long input text, and they introduced a decoder-only architecture, which can adjust to very long sentences, much longer than the typical encoder-decoder model can handle. They have used Wikipedia articles as the input text, and the model tries to generate the first section of the Wikipedia article. They used a multi-documentation summarization and worked with a large dataset. Authors have mentioned that the extraction method in the first stage could be improved, as it affects the performance significantly. The study [68] addresses the issue of generating important information by creating the convolutional gated unit, which helps to focus on the primary information and filter the secondary. For instance, in the sentence "*Starbucks, which entered Chinese market early, is a brand appealing to young people of petit bourgeoisie... A Tall Americano sells about 12RMB in the United States, but 21RMB in China, which means it is 75% more expensive.*", the model could identify that the primary information is the price and secondary information is the country, which helped to generate a summary that includes price information. Additionally, they removed the duplicate words by calculating the degree of repetition, which led to generating more relevant summary. They used the Chinese[1] dataset [69] and Gigaword and achieved a R-1 score of 36.3, a R-2 score of 18.0 and a R-L score of 33.8.

The authors [36] combine transformer with the pointer-generator layer, as the original pointer-generator paper [31] used the sequence-to-sequence base model. They have used the embedded input for the decoder layer, the output of the decoder layer (as a replacement of the RNN hidden layer from pointer-generator paper [31]) and the encoder output to calculate the p_{gen}. The context vector has been calculated as an average across the source dimension from the encoder output, weighted by the source attention distribution. However, they did not achieve high results on the CNN/DM dataset, with a R-1 score of 22.10, a R-2 score of 4.03 and a R-l score of 14.66. Adding a coverage mechanism did not improve the result significantly (the results were almost the same). Subsequently, they added N-gram blocking, implemented inside a beam search which prevents a model from generating words if they appeared in previous N-gram of words. Nevertheless, even by adding this layer to a beam search, the highest result they could achieve were a R-1 score of 25.31, a R-2 score of 4.16 and a R-L score of 15.99.

Another paper [70] used the same technique of combining transformer and pointer-generator, but they used it on a task where the input text and annotated summary

[1] http://icrc.hitsz.edu.cn/Article/show/139.html

had a different vocabulary. The general architecture looks similar to one from the paper [36]. However, the context vector has been calculated as a sum of encoder output weighted by multi-head attention weights (given in Eqn. 3.1).

$$c_t = \sum_i a_i^t h_i \tag{3.1}$$

3.3 Hybrid Models

The authors in [14] proposed a hybrid of extractive-abstractive architecture, in addition to which they have used reinforcement learning (RL) to combine two architectures and to eliminate the redundancy problem in the generated summary. Reinforcement learning is a method of training where the machine (the agent) tries to make a sequence of decisions and gets either a reward for the correct ones or a penalty for the incorrect decisions. The authors suggested using two agents: one for extractive and another one for abstractive approaches. First, the extractor agent selects important sentences, and then the abstractor agent rephrases the sentences in parallel. In addition, they have implemented parallel decoding, which speeds up the inference mechanism (10x-20x) and training convergence (4x). They achieved a R-1 score of 40.88 and a R-2 score of 17.80.

Usage of Generative Adversarial Network (GAN) [15] for summarization has also been tested in [71]. The generator is an agent of RL and generates the summary, and the discriminator attempts to distinguish a generated summary from an annotated summary. The model has been trained on the CNN/DM dataset and achieved a R-1 score of 39.92.

Another study [16] used the "Condense-then-Select" model, which firstly uses an abstractive model with Sentence Bidirectional Encoder Representations from Transformers (SBERT) [72] to generate the summary and then inputs it to extractive model to get the most important sentences. The authors achieved a R-1 score of 42.71 and a R-2 score of 19.59.

3.4 Pre-trained Models

Most pre-trained models use a transformer as their base model. Pre-trained BERT (Bidirectional Encoder Representations from Transformers) [23] was used in [73] with a randomly initialized transformer for decoder. The BERT-based model achieved the

highest result on the CNN/DM dataset of a R-1 score of 43.85, a R-2 score of 20.34 and a R-L score of 39.90. Another paper [24] used a pre-trained transformer and then fine-tuned it for a summarization task. A single network has been used to encode the input text and generate the summary. The authors mentioned that only 1% of the training dataset (around 3000 examples) was needed to achieve a R-2 score of 13.10 on the CNN/DM dataset. However, their model did not achieve very high results of a model with a pre-trained encoder having a R-1 score of 39.65, a R-2 score of 17.74 and a R-L score of 36.85.

A few other techniques used masking to achieve high ROUGE score. For instance, BART (Bidirectional and Auto-Regressive Transformers) model [25] uses noise to corrupt the input text, and the model tries to reconstruct this input text. Several techniques have been used to test the model's performance, from shuffling the input text to masking the tokens. Their autoregressive decoder can be fine-tuned for summarization tasks. In subsequent paper [26], the authors created a PEGASUS (Pre-training with Extracted Gap-sentences for Abstractive SUmmarization Sequence-to-sequence) model, which masked/removed important sentences from the input text and placed them as an annotated summary. This model achieves high ROUGE score on six different datasets and generates summaries comparable to a state-of-the-art model. On the CNN/DM dataset, this model has achieved a R-1 score of 44.17, a R-2 score of 21.47 and a R-L score of 41.11. Additionally, their model could achieve high results on an unseen dataset with only 1000 examples.

More recently,the authors in [74] presented a model called ProphetNet, which predicts n tokens instead of one in a sequence-to-sequence model. In the paper, the authors discuss, that autoregressive language modelling focuses on the latest token instead of a group of tokens; therefore, it needs to grasp the general meaning of the structure. The N-gram model helps to prevent the model from overfitting on a strong local connection, which might appear in the language. They used the CNN/DM and the Gigaword datasets and achieved a R-1 score of 42.61, a R-2 score of 19.83 and a R-L score of 39.67 using a 3-gram model on a CNN/DM test set. Another paper, published by Microsoft research team [75], has presented a novel sequence-to-sequence fine-tuning toolkit (s2s-ft), which uses transformers for conditional generation tasks. They have implemented three s2s-ft models: casual fine-tuning, masked fine-tuning, and pseudo-masked fine-tuning. These models mask target tokens and learn to recover them. The paper has achieved a R-1 score of 44.79, a R-2 score of 21.98 and a R-L score of 41.93 on the CNN/DM dataset.

Another research work [76] introduced a pre-trained model for multi-document text summarization, which outperforms BART and PEGASUS. The input from multiple files is concatenated together, and a Longformer-Encoder-Decoder (LED) model [77] is used to process it, as a standard encoder-decoder model fails to generate meaningful summaries for long input texts.

To overcome a problem of rare words, authors in [78] proposed using attentive mimicking. They stated that BERT struggles to handle words which are not frequent. Therefore, they implemented a one-token approximation, which led to a higher score.

The paper [79] argued that knowledge transfer (pre-training a model on a specific dataset so that the model would have initialized connection in a network for further training) plays less of a role in pre-trained models than it was considered. They created a nonsense dataset randomly generated and pre-trained the model on this dataset, which performed as well as other models where knowledge transfer was used. They suggest that the model performs better because of optimized model initialization rather than an actual knowledge transfer.

Evaluation Metrics

The ROUGE metric is used in this thesis (which was described in section 2.1.7), however, there exist other evaluation algorithms that are used in text summarization: BLEU (Bilingual Evaluation Understudy) [80] and METEOR (Metric for Evaluation of Translation with Explicit Ordering) [81]. BLEU score is based on precision, meaning how many words in the generated summary appear in the annotated summary. The whole BLEU score equation can be seen in Eqn. 3.2-3.3, which produces a number between 0 and 1, where the higher the score, the closer generated summary to the annotated summary. BLEU, unlike ROUGE, produces a single number, as it uses the geometric average of N-gram precisions up to the length of N.

$$BP = \begin{cases} 1, & \text{if } c > r. \\ e^{1-r/c}, & \text{if } c \leq r. \end{cases} \tag{3.2}$$

$$BLEU = BP * exp(\sum_{n-1}^{N} \omega_n \log p_n), \tag{3.3}$$

where c is the length of the generated summary and r is the length of the annotated summary. A brevity penalty (BP) is used to compensate for the high-precision hypothesis. ω_n is positive weights, which sum up to 1.

As can be seen, BLEU is based only on precision and does not take recall directly into account. The paper [81] highlighted a few other problems of a BLEU algorithm, such as using geometric averaging or usage of high order N-gram. METEOR algorithm addresses these weaknesses of a BLEU score, and the equation for METEOR can be seen in Eqn. 3.4-3.6.

$$F_{mean} = \frac{10PR}{R + 9P} \tag{3.4}$$

$$Penalty = 0.5 * (\frac{\#chunks}{\#unigrams\_matched})^3 \tag{3.5}$$

$$METEOR = F_{mean} * (1 - Penalty) \tag{3.6}$$

Remark 3.4.1. *F_{mean} is calculated using a harmonic-mean [82], which places most of the weight on the recall. METEOR algorithm considers the number of the longest chunk of N-gram (#chunks) of unigrams in the generated summary that are mapped to the ones in the annotated summary (#unigrams_matched) so that having the lowest number of chunks.*

BLEU and METEOR scores are the most popular algorithm for machine translation, and the ROUGE score is the most popular algorithm to evaluate text summarization models; therefore, all performance evaluation is done using the ROUGE score in this thesis.

Chapter 4

Dataset Preparation

In this thesis, we have used the following four datasets for text summarization: CNN/DM, XSum, Gigaword and DUC 2004 Task 1 (Task 1 in DUC 2004 dataset refers to very short single-document summarizations). Most experiments have been done on the CNN/DM dataset because it contains the longest summary, and the annotated summary contains extractive and abstractive elements, where both methods could be tested. It is most commonly used as a standard text summarization dataset, so it is possible to compare our model with other models.

4.1 Dataset Overview

In Table 4.1 the number of samples from each dataset used in training, validation and testing is shown. The number of samples in each dataset are shown. These datasets have the same structure: index, input text, and annotated summary.

Dataset	**Training**	**Validation**	**Testing**
CNN/DM	286,817	13,368	11,487
XSum	204,045	11,332	11,334
Gigaword	3,803,957	189,651	1,951
DUC 2004 Task 1	None	None	500

Table 4.1: The number of samples in each dataset. The DUC 2004 Task 1 dataset has only a testing dataset; therefore, for training and validation DUC 2004 Task 1 dataset it is "*None*"

In Fig. 4.1 a few samples of the CNN/DM dataset are shown (text is already preprocessed). If we expand the first sample, we can see which phrases in an annotated summary are copied from the input text (as shown in Table 4.2). It can be observed that almost half of the text, which is present in the annotated summary, is located in the first few sentences of the input text.

Index	Input text	Annotated summary
0	editor 's note : in our behind the scenes series , cnn correspondents share their experiences in covering news and analyze the stories behind the events . here , soledad o'brien takes users inside...	mentally ill inmates in miami are housed on the forgotten floor judge steven leifman says most are there as a result of avoidable felonies while cnn tours facility , patient shouts : i am the son...
1	london , england reuters harry potter star daniel radcliffe gains access to a reported # 20 million $ 41.1 million fortune as he turns 18 on monday , but he insists the money will not cast a spell...	harry potter star daniel radcliffe gets # 20m fortune as he turns 18 monday . young actor says he has no plans to fritter his cash away . radcliffe 's earnings from first five potter films have b...
2	minneapolis , minnesota drivers who were on the minneapolis bridge when it collapsed told harrowing tales of survival . the whole bridge from one side of the mississippi to the other just complete...	new : i thought i was going to die , driver says . man says pickup truck was folded in half ; he just has cut on face . driver : i probably had a 30 - , 35-foot free fall minnesota bridge collaps...
3	baghdad , iraq dressed in a superman shirt , 5-year-old youssif held his sister 's hand friday , seemingly unaware that millions of people across the world have been touched by his story . nearby ...	parents beam with pride , cannot stop from smiling from outpouring of support . mom : i was so happy i did not know what to do burn center in u.s. has offered to provide treatment for reconstruct...
4	washington doctors removed five small polyps from president bush 's colon on saturday , and none appeared worrisome , a white house spokesman said . the polyps were removed and sent to the nationa...	five small polyps found during procedure ; none worrisome , spokesman says . president reclaims powers transferred to vice president . bush undergoes routine colonoscopy at camp david .

Figure 4.1: Sample CNN/DM text fragment samples, where column 1 is the index number, column 2 is the input text, column 3 is annotated summary for each text fragment.

Input text	**Annotated summary**
editor 's note : in our behind the scenes series , cnn correspondents share their experiences in covering news and analyze the stories behind the events . here , soledad o'brien takes users inside a jail where many of the inmates are mentally ill . an inmate housed on the forgotten floor , where many mentally ill inmates are housed in miami before trial . miami , florida the ninth floor of the miami-dade pretrial detention facility is dubbed the forgotten floor . here , inmates with the most severe mental illnesses are incarcerated until they are ready to appear in court . most often , they face drug charges or charges of assaulting an officer charges that judge steven leifman says are usually avoidable felonies . he says the arrests of...	mentally ill inmates in miami are housed on the forgotten floor judge steven leifman says most are there as a result of avoidable felonies while cnn tours facility , patient shouts : i am the son of the president leifman says the system is unjust and he is fighting for change .

Table 4.2: Expanded fragment (sample) index 0 from Fig. 4.1 from the CNN/DM dataset with highlighted phrases, which are used in input text and its summary. Highlighted text indicates common terms between input text and summary.

4.1.1 Datasets

CNN/DM

The author in [83, 31] created a new English-language dataset called the *CNN/Daily Mail* dataset (CNN/DM). It is the most popular dataset for text summarization task and is usually used as a benchmark.

There are several ways of collecting this dataset. First one is to manually collect it from CNN[1] and DM[2] websites using the published script[3]. The second one is to download the collected CNN/DM dataset[4]. In this research, we have manually collected it, combined and later compared it to the already collected dataset.

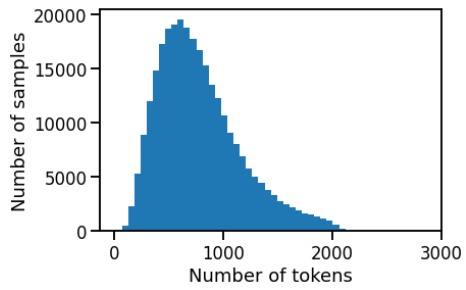
The CNN/DM dataset consists of 3 files: training, validation and testing. The training dataset contains 286,817 samples, validation contains 13,368 samples, and testing contains 11,487 samples (as given in Table 4.1). Each sample has an input article (input text) and a human-generated summary (annotated summary). Journalists at CNN and Daily Mail created the original articles.

The input text contains, on average, 30 sentences and 724 tokens, whereas the annotated summary has, on average, 4 sentences and 52 tokens (as shown in Fig. 4.2). In the study [31], the authors used 400 first tokens as input and 100 tokens as output. Therefore, on average, the model will be given 16 sentences as an input text with 382 tokens, and the annotated summary, on average, contains 4 sentences and 52 tokens (as shown in Fig. 4.3). We can see that the annotated summary does not change a lot when input text is reduced significantly, and most samples have lost half of the information in the extractive process.

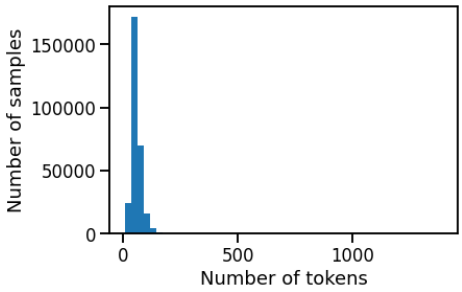
XSum

The *Extreme Summarization* (XSum) [37] is a highly abstract dataset, which was designed to answer the question "What is the article about?". It does not contain useless or redundant information and rarely has long extracted phrases from the input text. It was collected using the British Broadcasting Corporation (BBC) articles. This dataset has to be manually collected using GitHub code[5]. After collecting the

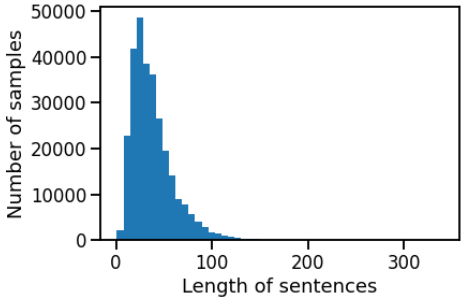
[1] `https://www.cnn.com`
[2] `https://www.dailymail.co.uk`
[3] `https://github.com/deepmind/rc-data`
[4] `https://huggingface.co/datasets/cnn_dailymail`
[5] https://github.com/EdinburghNLP/XSum



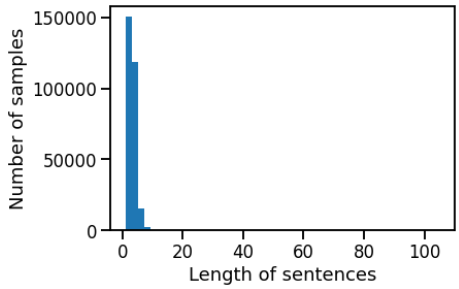
(a) № of tokens in the input text



(b) № of tokens in the annotated summary

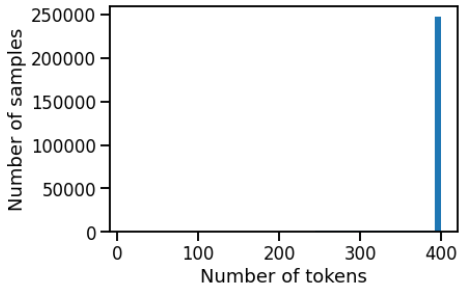


(c) № of sentences in the input text

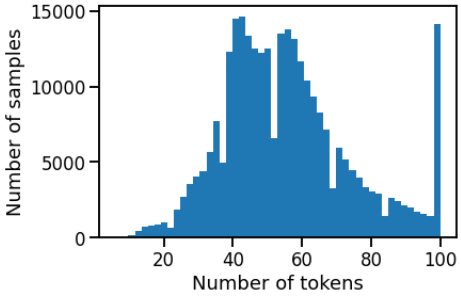


(d) № of sentences in the annotated summary

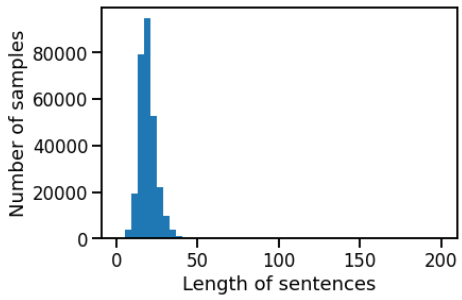
Figure 4.2: CNN/DM dataset information showing the distribution of tokens in the input text and annotated summary.



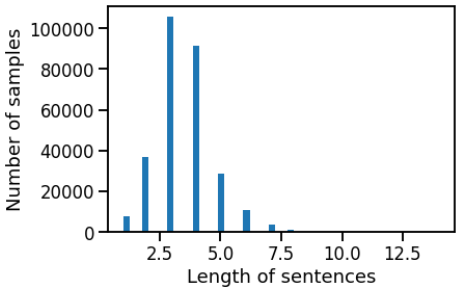
(a) № of tokens in the input text



(b) № of tokens in the annotated summary



(c) № of sentences in the input text



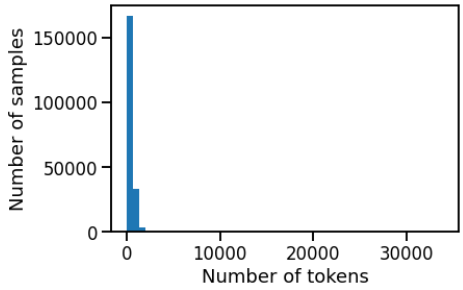
(d) № of sentences in the annotated summary

Figure 4.3: CNN/DM dataset information (max 400 tokens) showing the distribution of tokens in the input text and annotated summary.

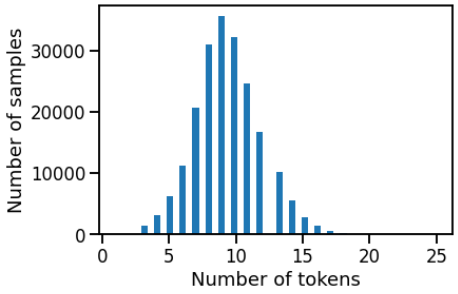
dataset, it has to be divided using a file, which contains information about which articles belong to training/validation/testing datasets using the following script [6].

The training dataset has 204,045 samples, validation has 11,332 samples, and testing has 11,334 samples, as given in Table 4.1.

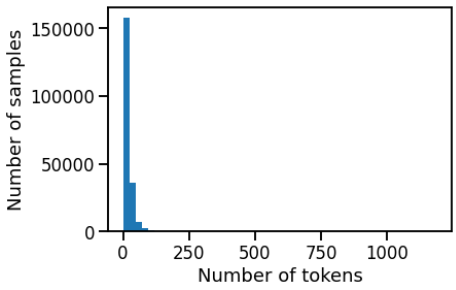
The input text, on average, contains 436 tokens; however, compared to the CNN/DM dataset, it has some samples with over 15,000 tokens. As the purpose of this dataset is to generate as condensed summary as possible and focus on the most important information, the annotated summary contains only 9 tokens on average, as shown in Fig. 4.4. The average annotated summary length is one sentence when the input text size is about 17 sentences. Only 6 samples in the dataset have an average annotated summary length of two sentences.



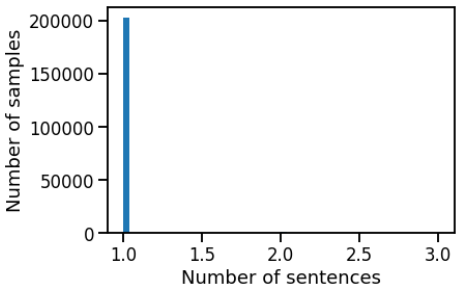
(a) № of tokens in the input text



(b) № of tokens in the annotated summary



(c) № of sentences in the input text



(d) № of sentences in the annotated summary

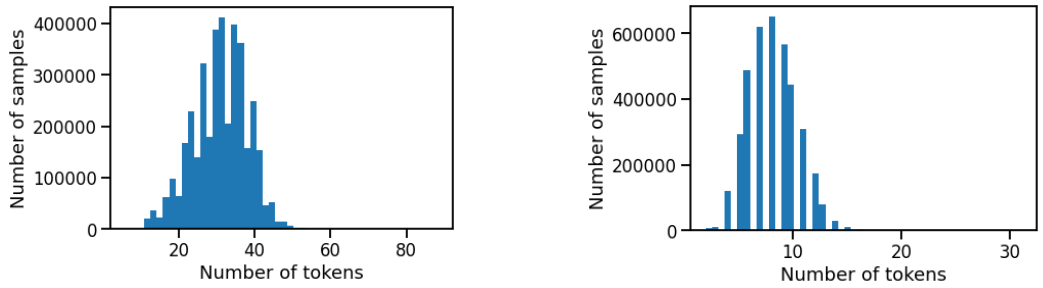
Figure 4.4: XSum dataset information showing the distribution of tokens in the input text and annotated summary.

[6] https://github.com/EdinburghNLP/XSum/blob/master/XSum-Dataset/XSum-TRAINING-DEV-TEST-SPLIT-90-5-5.json

GIGAWORD

The *Annotated English Gigaword* dataset is based on the Gigaword corpus [38]. This dataset contains the headline of each article as a summary text and the first sentence as an input text. It can be downloaded from the following website[7]. The training dataset contains around 3,803,957 samples, validation 189,651 samples and testing 1,951 samples, as shown in Table 4.1.

The average number of tokens for an input text sample is 31, and for an annotated summary is 8 (as shown in Fig. 4.5). We have not shown the distribution of the number of tokens or sentences for the Gigaword dataset, since only 0.2% of the entire dataset has more than 1 sentence in the annotated summary. In addition, only 1.3% of the input text samples use more than the first sentence as the input text.



(a) № of tokens in the input text (b) № of tokens in the annotated summary

Figure 4.5: Gigaword dataset information showing the distribution of tokens in the input text and annotated summary.

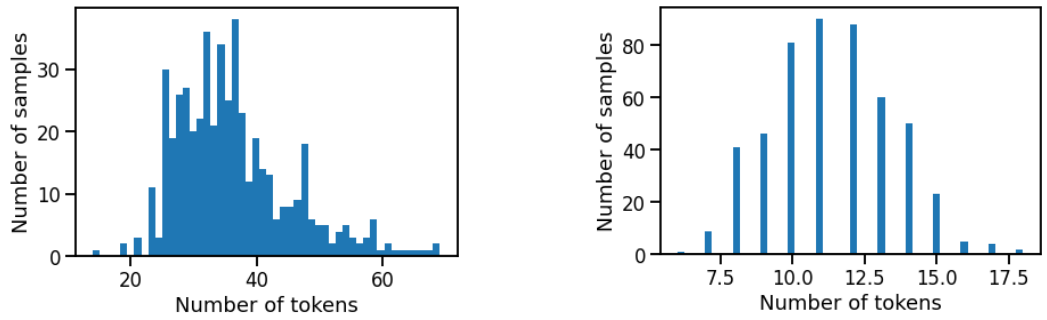
DUC 2004 Task 1

The *DUC 2004* (Document Understanding Conference) Task 1 dataset [39] is used for testing purposes and contains 500 samples. It can be obtained from the following website[8]. This dataset contains a short summary of less than or equal to 75 bytes. It has a structure similar to the Gigaword dataset, which contains an average of 36 tokens for input test and 11 tokens for annotated summary (shown in Fig. 4.6). This dataset is meant for testing purposes only for models that are trained on other benchmark datasets and comparative work.

Task 1 of the DUC 2004 dataset is used for testing, even though the DUC 2004 dataset has 1-5 tasks: task 1 refers to very short single-document summarizations,

[7]https://deepai.org/dataset/gigaword
[8]https://duc.nist.gov/duc2004/

task 2 has multi-document summarizations, task 3 includes very short cross-lingual single-document summarizations, task 4 refers to short cross-lingual multi-document summarizations, and task 5 refers to question-focused summarization.



(a) № of tokens in the input text (b) № of tokens in the annotated summary

Figure 4.6: The DUC 2004 Task 1 dataset information showing the distribution of tokens in the input test and annotated summary.

4.2 Processing for Text Summarization

In this section, we discuss three principal tasks associated with preparing raw text for the learning summaries: preprocessing, batching and marking important sentences.

4.2.1 Preprocessing

Various methods exist for text preprocessing: removing stop words, lemmatization, stemming, to name a few. However, only a limited number of preprocessing techniques could be applied for a text summarization task, as the model should generate readable text, and some techniques can make it impossible. For instance, if stop words are removed, then the model will not be able to generate meaningful summary without such words (for instance, "soon, i will go home." will become "soon, go home."). Therefore, initial research about text preprocessing techniques has been performed to identify which algorithms could be applied for a text summarization task, which are described in this section. For text preprocessing, we have used *nltk* library[9]. For all datasets, the same preprocessing techniques have been applied, and the pipeline can be seen in Fig. 4.7.

[9] `https://www.nltk.org/`

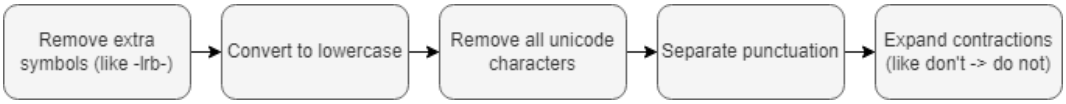


Figure 4.7: Text preprocessing pipeline

Algorithm 2 shows the steps used in the preprocessing of raw text. Firstly, *extra symbols* (which might be included in HTML text) are removed, like "-lrb-" (line 2). Secondly, the input text and annotated summary can be in *lowercase* (line 4), as it does not change the meaning of the sentence. The next step is to *separate* " 's" in the word (e.g., "amy's") and remove all *Unicode* characters (line 7). We use ASCII (American Standard Code for Information Interchange) characters only and not Unicode characters, as the used datasets contain standard English characters, and there is no need to use other characters. Then we can *separate the sentence into words* (line 5), but it cannot be done by separating words by space, as it will add punctuation to the word: "soon, i will go home." will become ["soon,", "i", "will", "go", "home."]. We also cannot simply remove the punctuation, as it is crucial to understanding the end of a sentence, since it will be important in the extractive approach. Additionally, in such words as "u.s.a." we cannot separate into three different sentences because it contains a dot. Therefore, we separate words using *regular expressions* (regex) pattern (it was taken from the online NLTK Book, Chapter 3.7[10]), which was derived from keeping words together, like "u.s.a." or "poster-print". Next, words and punctuation are *concatenated* to recreate a sentence, where each item is separated by space ("soon , i will go home .").

And finally, we expand *contractions* (line 6), for example, "don't" will be "do not" or "i'll" will be "i will". A hard-coded dictionary was used, as all contractions could be written down. This preprocessed text is used before the text is inputted to the model or before the extractive approach. We use the nltk function *sent_tokenize* to separate sentences from each other.

Remark 4.2.1. *One can use contractions library*[11]*. However, their use will result in incorrect contractions. E.g., "u.s.a" will be changed to "you.s.a".*

After preprocessing, the vocabulary is created, where each unique word is assigned a number in increasing order. Then, each sentence is encoded using the vocabulary, and the array is sent to the model.

[10]https://www.nltk.org/book/ch03.html#nltk-s-regular-expression-tokenizer
[11]`https://pypi.org/project/contractions/`

Algorithm 2: An algorithm for dataset preprocessing

Input : dataset: Dataset that should be preprocessed
Output: processed_dataset: Processed dataset

```
for input_text in dataset do
    input_text ← RemoveExtraSymbols(input_text);
    input_text ← input_text.lower();
    input_text ← RemoveUnicode(input_text);
    input_text ← SeparatePunctuation(input_text);
    processed_text ← ExpandContr(input_text);
    AddProcessedText(processed_dataset, processed_text);
    // Add processed sentences to the output dataset
```

Another idea that could have been used is to remove punctuation completely (except end-of-sentence punctuation) and later create an algorithm to insert punctuation back to the processed annotated summary in the appropriate places. The punctuation could be removed, as a significant percentage of input text and annotated summary contain commas, semicolons or other punctuation, which do not give any extra information. However, creating an algorithm to make a sentence with correct punctuation is not a focus of this thesis; therefore, we have omitted this idea.

4.2.2 Dataset Batching

There exist several ways how to input the dataset into the model, such as using libraries like torchtext[12] library from PyTorch[13] or creating a batching class. However, using a library might limit the ability to process the dataset; therefore, a separate batching class has been created. This class gets the information about the location for training/validation/testing datasets and, using the python "*yield*" command, provides a processed dataset in a batch consisting of Y samples. We are using fixed length for input text and annotated summary.

Table 4.3 gives the number of tokens for the input text, annotated summary and maximum generated summary depending on the dataset. During training, a model generates the same number of tokens as presented in "annotated summary" from Table 4.3; however, during testing, a model generates "maximum generated summary" different number of tokens, which is shown in the same table. The "maximum generated summary" has longer length, as later it will be compared to the full annotated summary (not cropped one); therefore, we generate a bit longer summary to get higher ROUGE score. However, in most cases the model will generated EOS (end of sentence) token before hitting this limit.

The vocabulary used in the dataset and the frequency information as scores for each word are saved as dictionaries. Using the torchtext library, we can generate a dictionary of used words, depending on how frequently they are used in the dataset. Later, this dictionary is used as a vocabulary in the abstractive model.

[12] https://pytorch.org/text/
[13] `https://pytorch.org`

Dataset	Input text	Annotated summary	Maximum generated summary
CNN/DM	400/200	100	120
XSum	400/200	20	30
Gigaword	50	18	25
DUC 2004 Task 1	-	18	25

Table 4.3: The number of tokens for input text, annotated summary and maximum generated summary for different datasets.

Remark 4.2.2. *For the input text (CNN/DM and the XSum datasets) we have tested 400 and 200 tokens. For the DUC 2004 Task 1 dataset, the length of the input text depends on the dataset the model has been trained on, as the DUC 2004 Task 1 dataset has only a test dataset. The CNN/DM and XSum datasets has long annotated summaries; therefore, we take a larger number of tokens. The Gigaword and the DUC 2004 Task 1 datasets have smaller length (both around 20).*

Depending on where it is used, the batching class also returns changed annotated summaries. Table 4.4 contains information about variables/symbols that are used in the batching class. For instance, "trg_inp" variable is used as an input to the model, and it contains "<*unk*>" tokens for *OOV* (out of vocabulary) words, *BOS* (beginning of sentence) and *EOS* (end of sentence) symbols. "trg_ext" variable is used in a loss function, and it contains information about all words, even if they are OOV. "trg_full" variable contains the full annotated summary as a text, which is used later to calculate a ROUGE score. Input texts and annotated summaries are padded with a special "<*pad*>" symbol, which is ignored during the loss function. The input text is encoded using *src* and *src_ext* variables.

The summarization works in a way that for a given input text and a part of a summary that has been generated so far, the next word is predicted; therefore, for each sample, we need to run the model several times, where each time we append a newly generated word. However, as the trained model receives information from the annotated summary, we can speed up the process by creating a matrix of MxM, where M is a maximum length of a annotated summary (Table 4.3) and each row adds a new word starting from BOS symbol. Using the following approach, the model can generate a matrix with a size of M, which contains information about the next generated word for a specific summary that is given.

Variable/Symbol	Definition
src	Variable containing the input text. The text is either padded with<*pad*> or cut when the length exceeds the maximum length for the input text.
src_ext	Variable containing the input text that includes extended dictionary of OOV words. The text is either padded with<*pad*> or cut when the length exceeds the maximum length for the input text. It is used inside pointer-generator layer, and the word is copied from this variable, if the model produces OOV word.
src_freq	Variable containing the frequency scores for the input text, which is used during encoder attention.
trg_inp	Variable containing the annotated summary that includes extended dictionary of OOV words. This variable is input to the model and contains such symbols as <*s*> and </*s*>. The text is either padded with <*pad*> or cut when the length exceeds the maximum length for the summary.
trg_ext	Variable containing the annotated summary that includes extended dictionary of OOV words. This variable is similar to *trg_inp*, but does not include <*s*> symbol (even though it includes </*s*> symbol to signify the end of the sentence), and is used in the loss function.
trg_full	Variable containing the full annotated summary that is not padded nor cut. This summary is used during testing.
trg_mask	Variable containing the masking information for the annotated summary.
<*unk*>	Unknown token. Used to encode words that do not appear in the dictionary.
<*pad*>	Padding token. Used to pad the text up to a specific length.
<*s*>	BOS token. Used to indicate the beginning of a sentence.
</*s*>	EOS token. Used to indicate the end of a sentence.

Table 4.4: Variables and symbols definitions that are used in the batching class.

Annotated summaries mainly consists of words, which also appear in the input text. However, some words in annotated summaries are not used in the input text vocabulary, which worsens the results and leads to the generation of "<*unk*>" token.

As an illustrative example, we use the annotated summary from Table 4.2. Each word is encoded with a specific number, which is extracted from the vocabulary. Words in the vocabulary are sorted depending on how frequently it is used. Additionally, 4 symbols are included at the beginning of the vocabulary: <*unk*>, <*pad*>, <*s*> and </*s*>. <*unk*> is used to generate unknown words, <*pad*> is used at the end of text to pad until it hits the token limit, <*s*> and </*s*>. <*unk*> are used to identify a sentence's start and end. Fig. 4.8 shows the text's encoding procedure, which is later input to the model.

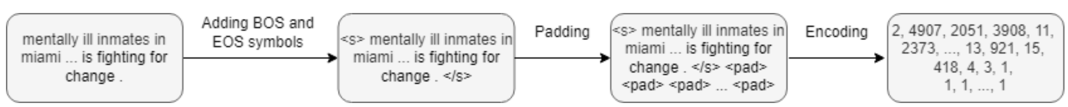


Figure 4.8: Step-by-step text's encoding procedure

We can see from Table 4.5 how words are encoded. For instance, <*s*> is encoded with "2" and "*mentally*" is encoded with 4907. Additionally, "*leifman*" was not found in the vocabulary and was encoded with "0".

Annotated summary	**Encoded annotated summary**
<*s*> mentally ill inmates in miami are housed on the forgotten floor judge steven leifman says most are there as a result of avoidable felonies while cnn tours facility , patient shouts : i am the son of the president leifman says the system is unjust and he is fighting for change . </*s*>	[2, 4907, 2051, 3908, 11, 2373, 32, 8272, 17, 5, 4863, 1205, 424, 2092, 0, 98, 116, 32, 60, 26, 8, 718, 10, 18967, 20238, 99, 199, 5416, 2150, 6, 1542, 12236, 23, 29, 127, 5, 259, 10, 5, 152, 0, 98, 5, 404, 13, 16734, 9, 19, 13, 921, 15, 418, 4, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Table 4.5: Annotated summary and its encoded version, which the model receives. The annotated summary is padded with a <*pad*> symbol up to the maximum summary length (in this case a hundred), which later is encoded with a number "1".

4.3 Important Sentences

The extractive approach is used in this research to derive the most important sentences from the input text. We are using an approach similar to TF-IDF, which uses the frequency information of different words to identify the importance of the sentence. Our extractive algorithm has been described in section 2.3.3. Some papers are taking the first N tokens as input text [12, 31, 23, 26]. We do not consider it an optimal approach, as information might be located in the middle or at the end of the text.

For example, Table 4.6 shows the input text and its annotated summary, where words used in the annotated summary are highlighted in the input text. Suppose we output the number of occurrences for each sentence of each word in the annotated summary on the input text. In that case, we can see that the most important information is present in the middle of the input text and not at the beginning (shown in Fig. 4.9). Here one can see that sentence number 6 contains the highest number of words which appear in the annotated summary, meaning that this sentence might be important for a summary generation.

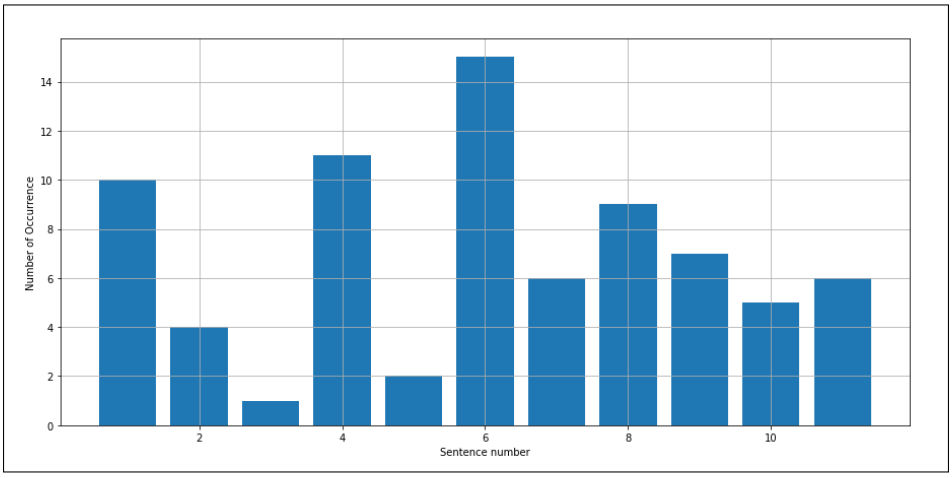


Figure 4.9: Number of occurrence of a text from the annotated summary in the input text depending on sentence's position

Another example is that sometimes important information is located at the end of the input text (Table 4.7). Word distribution graph is shown in Fig. 4.10. This figure shows the problem of other papers [12, 31, 23, 26], which uses only the first

Input text	Annotated summary
five **americans** who **were** monitored for three **weeks** at an omaha nebraska **hospital after** being **exposed to ebola in** west africa have been released a nebraska medicine spokesman said **in** an email wednesday . one **of the** five had a heart-related issue on saturday **and** has been discharged but has not left **the** area taylor wilson wrote . **the** others have already gone home . they **were exposed to ebola in sierra leone in march** but none developed **the** deadly **virus** . they are clinicians for partners **in health** a boston-based aid group . they all had contact **with** a colleague who **was diagnosed with the disease and is** being treated at **the national institutes of health in** bethesda **maryland** . as **of** monday that **health** care worker **is in fair condition** . **the** centers for **disease** control **and** prevention **in** atlanta has said **the** last **of 17** patients who **were** being monitored are expected **to** be released by thursday . more than 10 000 people have died **in** a west african epidemic **of ebola** that dates **to** december 2013 according **to the** world **health** organization . almost all **the** deaths have been **in** guinea liberia **and sierra leone** . **ebola is** spread by direct contact **with the** bodily fluids **of** an infected **person** .	americans were exposed to the ebola virus while in sierra leone in march . another person was diagnosed with the disease and taken to hospital in maryland . national institutes of health says the patient is in fair condition after weeks of treatment

Table 4.6: Highlighted words (shown in bold-face) in input text, which are also used in an annotated summary

N tokens of an input text for an abstractive model. In most cases, the first few sentences contain the most important information, which can be used to generate a summary. However, in the following example, important information is located at the very end of the article, and it would not be included in the model. The paper which implemented the pointer-generator approach [31] mentioned that they tried to use a smaller number of tokens; however, it did not lead to higher ROUGE score. It is probably due to the fact that they were cutting off important information, which was located further in the text.

Not all of our datasets could be used in the extractive approach. For instance, the Gigaword and the DUC 2004 Task 1 datasets contain only one sentence in the input text; therefore, such an approach is not used with these datasets.

Input text	Annotated summary
... bottom line : **online journalists** , operating outside restraints mainstream media , **become vulnerable** targets governments independent actors . **restrictive rule law** , **journalists vulnerable anger officialdom** . **rule law** weak , **vulnerable** attacks killers seldom , ever , answer **rule law** .	going online become path least resistance want make heard . restrictive rule law , journalists vulnerable anger officialdom . china malaysia , journalists bloggers jailed – even killed

Table 4.7: Highlighted words in the input text, which are also used in the annotated summary (sentences number 39 and 40)

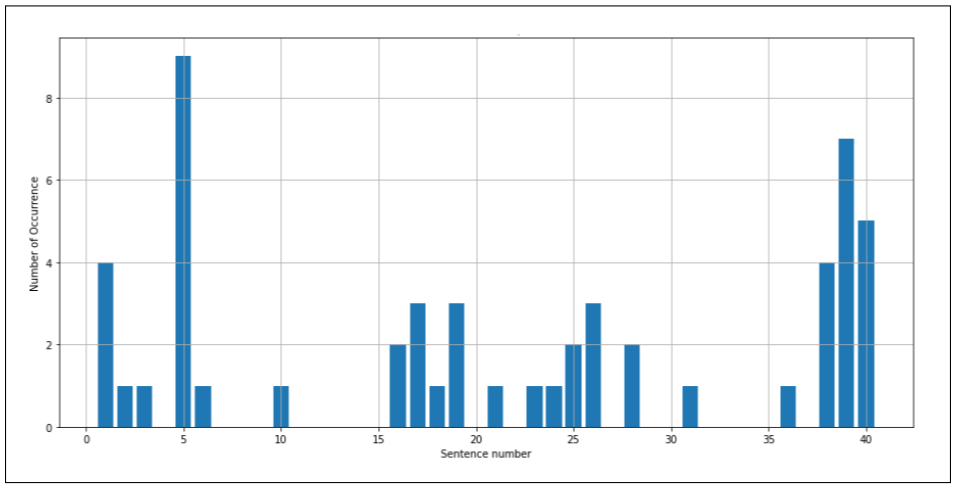


Figure 4.10: Number of occurrence of a text from the annotated summary in the input text depending on sentence's position (another example)

Chapter 5

Experiments

In this chapter, we discuss the experiments and results that have been achieved and compare models with each other.

5.1 Tracing of the Algorithm

After the input text has been processed by the extractive approach (section 2.3.3), the input text and annotated summary are used to generate a summary using transformer model (shown in Fig. 2.12 and explained in section 2.3.1). The dataset is provided with batches, which are explained in section 4.2.2.

Alg. 3 shows the training code for the transformer model (all variables are explained in Table 4.4). The model uses the encoder (line 3) which gets encoded input text (an example of an encoded text is shown in Table 4.5) and frequency information about words in the input text (each word is encoded with a score, which are explained in section 2.3.2). The encoder calculates the attention of an input text (section 2.2.8) and passes this information to the decoder. The decoder then uses this information and a part of annotated summary with mask (explained in section 2.2.7) to calculate s_t, x_t and h_t (line 4 and 5), which are explained in details in section 2.3.1. Then, the pointer-generator layer generates the probability distribution for a word from the vocabulary being generated or copied from the input text in line 6. At the very end, the loss function (Eqn. 2.3) is used to calculate the total loss for a batch in line 7. For an illustration, please see Appendix A.1.

Algorithm 3: An algorithm for a transformer model

Input : epochs, // Number of epochs
batches // Dataset batches

for *epoch in* epochs **do**
  for *batch in* batches **do**
    $encoded \leftarrow encoder(batch.src, batch.src\_freq)$;
    $s_t, x_t, mh\_attn\_decoder \leftarrow$
     $decoder(encoded, batch.trg\_inp, batch.trg\_mask)$;
    $h_t \leftarrow contextVector(encoded, mh\_attn\_decoder)$;
    $gen\_sum\_prob \leftarrow pointer\_generator(s_t, x_t, h_t, encoded)$;
    `// gen_sum_prob contains final probabilities of which word should be generated next`
    $loss \leftarrow calculateLoss(gen\_sum\_probabilities, batch.trg\_ext)$;
    `// using Kullback-Leibler divergence loss function`

5.2 Model Information

In this section, two primary models (M1 and M2) and parameters used in this thesis are discussed. Fig. 5.1 gives an overview of the combinations.

5.2.1 Models

We call the transformer model with a pointer-generator layer as M1 (described in section 2.3.1, Eqn. 2.18). Model M2 is the transformer model with a pointer-generator layer with frequency information added as an additional parameter to the attention mechanism (described in in section 2.3.2, Eqn. 2.20). Finally, to test the efficacy of the extractive approach, models M1 and M2 have been tested with an input text size consisting of 200 tokens (e.g., M1-200, M2-200) and 400 tokens (e.g., M1-400, M2-400) resulting in a total of 6 models (shown in Fig. 5.1). The goal of using different token sizes was to test if the extractive approach helps improve the performance of the models as well as observe the effects of changes to the input text size.

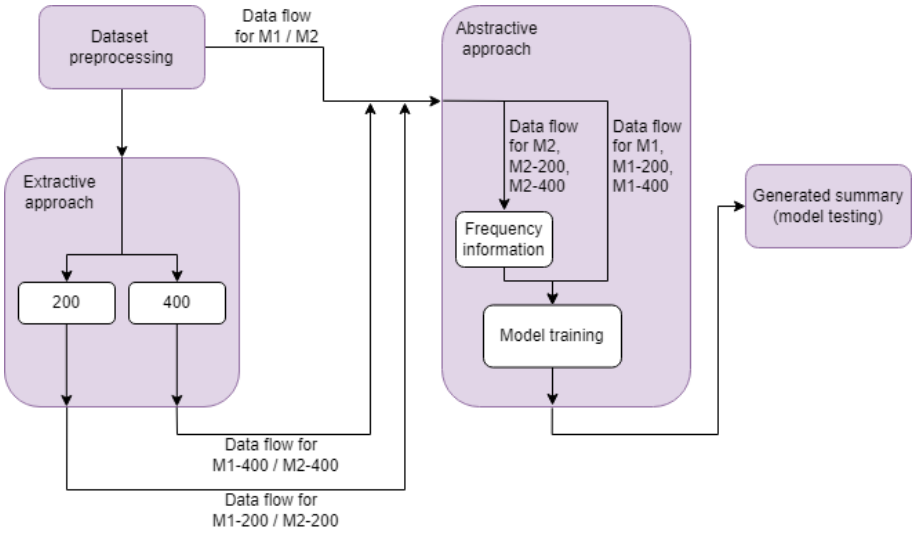


Figure 5.1: Overview of a data flow for different models. "200" and "400" in the extractive approach are the number of tokens used as input text for the abstractive model. The flow of M1/M2 and M1-x/M2-x represents the data flow that is used by specified model.

5.2.2 Model Parameters

We chose the dimension of our model (and embedding size) as 256 with an inner layer dimensionality of 1024. AdaGrad optimizer [45] is used (Eqn. 2.4). 8 layers and 8 heads for the encoder-decoder structure have been used. We have used a 3-gram model for the extractive approach. For 400 tokens, we take the first ten sentences, and for 200 tokens, we take the first five sentences, as the input text is shorter. All parameters have been chosen based on models trained on a partial dataset containing around 10,000 samples. Depending on the dataset, different number of tokens is taken for the annotated summary, which is described in section 4.2.2 Table 4.3. During testing, the model generates a summary until it hit EOS symbol or a limit (maximum generated summary from Table 4.3).

5.2.3 Performance Measure

A ROUGE score is used (Eqn. 2.12) to test the model's performance, which was described in detail in section 2.1.7. We use F_1 score of ROUGE-1 (R-1), ROUGE-2 (R-2) and ROUGE-L (R-L) scores to test the performance, as these scores are standard performance measurements and used as a benchmark. We use a beam size of 4.

5.3 Experimental Setup

Two different machine configurations were used for our experiments. One machine had one 8 GB GPU RTX 2080 and Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz, and another one four 12 GB GPUs TITAN X and Intel(R) Core(TM) i7-5930K CPU @ 3.50GHz. All models have been trained on 5 epochs, which usually takes around a day to train. The best models on the CNN/DM dataset have been trained on 30 epochs to compare results with the models from other papers [12, 31, 36].

5.4 Analysis of Results

We analyze results depending on the dataset, since each dataset has a different length of input text and annotated summary, and the annotated summary itself has been created using different methods (for instance, in the CNN/DM dataset, the annotated

summary was created by a human expert and consisted of several sentences, whereas in the Gigaword dataset the annotated summary is a one-sentence headline of an article). However, to compare models trained on a different dataset, we use the DUC 2004 Task 1 dataset to test our models.

We also include two examples from each dataset and shows the results in terms of the 6 different model configurations. Each generated summary has a heatmap matrix, which highlights the attention placed on a word inside the decoder between the input text and the generated summary.

5.4.1 Discussion of Parameters

As was mentioned in section 2.3.3, we have tested removing stop words; however, the results became much worse than without them. Removing punctuation helped; however, the generated summary becomes unreadable, as commas and other symbols are essential for text understanding.

Experiments showed that an embedding size of 512 or 128 does not change the results significantly. Optimization function that is used in [31] and transformer model [1] that is used is different: in [31] AdaGrad optimizer is used (Eqn. 2.4) with fixed learning rate of value 0.15, when in [1] Adam optimizer is used (Eqn. 2.5-2.9), where label smoothing and warm-up learning rate are used. Our experiments showed that AdaGrad performs better, even though the difference is insignificant. We have also tested the number of layers and heads in the encoder-decoder structure, and 8 layers and 8 heads showed better results, even though other values did not change results significantly. A dropout of 0.2 was used.

The extractive model has been tested with 1-gram, 2-gram, 3-gram and 4-gram, and 3-gram model showed the best result. Changing beam size did not change results significantly; therefore, we have used beam size 4.

We use 400 tokens for an input text to test models trained on the CNN/DM and the XSum datasets. Our initial experiments showed that increasing or decreasing the number of tokens worsens the results, which was also mentioned in [31]. Therefore, as an experiment, we tested 200 tokens for the input text, as having fewer tokens might speed up the training process. For 400 tokens, we use the vocabulary size of 50,000 words (similar to the size used in the paper [31]), and for 200 tokens, we use 25,000 words, as the dataset contains fewer words.

ROUGE scoring is not perfect and does not provide good information on whether

a generated summary is good. For example, suppose we take the first three sentences as a generated summary. In that case, a R-1 score is around 41 and a R-2 score is approximately 18, which is much better than, for example, in the pointer-generator model and can be compared to the models with pre-trained embedding. However, we cannot say that such summaries provide any meaningful information, as it does not generate anything new, and the generated summary does not depend on what is mentioned in the input text or what information contains at the middle of the end of the input text.

Our main experiments are trained and tested on a full dataset and have been trained either for 5 or 30 epochs. Experiments on 5 epochs are done to show how the model differ from each other, and only a few models, which are crucial for understanding the results, have been trained for 30 epochs. It is comparable to 33 epochs which were done in [31].

5.4.2 Models Trained on the CNN/DM Dataset

The results for all models trained on the CNN/DM dataset are shown in Table 5.1. We have performed some tests which can be divided into 2 phases: test the performance of a transformer + pointer-generator layer (M1) and the same model but with the addition of information about the frequency of each word (M2). As we can see, the M2 model performs better than the M1 model in R-1 and R-L scores; however, the R-2 score is larger for the M1 model. Nevertheless, the difference between R-1 and R-L of both models is small, so one can conclude that both models perform similarly on these scores.

Model	**R-1**	**R-2**	**R-L**
M1	35.56	**12.90**	33.08
M2	**35.59**	12.51	**33.22**
M1-200	**35.36**	**12.65**	**32.82**
M2-200	35.09	11.98	32.62
M1-400	**36.29**	**13.24**	**33.90**
M2-400	35.83	13.09	33.48

Table 5.1: Results of different models, which were trained and tested on the CNN/DM dataset. Best results are shown in bold-face.

But, when we test the same model with an extractive approach layer, we can see that the performance suffers significantly for the model with frequency information

with no gain in performance. Comparing models trained with 200 vs 400 tokens (M1-200, M2-200, M1-400 and M2-400), we can observe that 400 tokens performed much better, even better than models without an extractive approach. It can be seen that the extractive approach helped to improve the performance of models, where ROUGE scores shown an increase from 35.56 to 36.29 (M1-200).

Test Cases

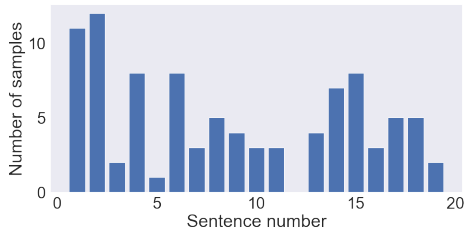
In this subsection, we provide examples of what the model generates. It is essential to mention that a ROUGE score shows average similarity of how the model performs, meaning that the model, which performed better overall, will not always generate good summaries.

We have omitted the discussion of results from models M1-200 and M2-200 since their results are lower compared to the other models. Fig. 5.2 shows the number of words from the annotated summary which appear in the input text. We can see that the extractive approach with 200 tokens has much less information. The extractive approach with 400 tokens adds more information, which improves the overall result. For instance, sentence number 12 (missing in the text without an extractive approach) contains ten words, which also appears in the annotated summary. The total number of words from the annotated summary present in the input text of a dataset without an extractive approach, with extractive (400 tokens), and with extractive (200 tokens) are 94, 102, and 51, respectively. By including more important information, the model can generate better summaries. However, too much information makes the model bigger and harder to train and worsens the overall results.

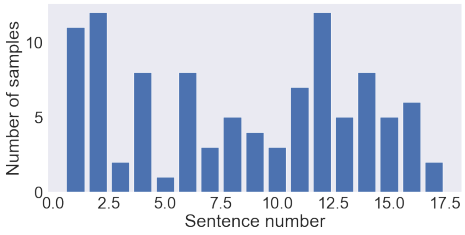
We have selected 3 best performing models on the CNN/DM dataset: M1, M1-400 and M2-400 (see Table 5.1) to illustrate the strengths and weakness of each model. Each model was executed several times with identical input text.

Case 1

The input text and the annotated summary used as the first case are shown in Table 5.2. We have chosen a short input text in order to analyse the results of three models. However, this discussion can be applied to any fragment (sample) in the dataset.



(a) Without extractive approach



(b) With extractive approach (400 tokens)

(c) With extractive approach (200 tokens)

Figure 5.2: Summarization statistics from the annotated summary, which appear in input text from the CNN/DM dataset.

Input text	Annotated summary
justin timberlake and jessica biel , welcome to parenthood . the celebrity couple announced the arrival of their son , silas randall timberlake , in statements to people . silas was the middle name of timberlake 's maternal grandfather bill bomar , who died in 2012 , while randall is the musician 's own middle name , as well as his father 's first , people reports . the couple announced the pregnancy in january , with an instagram post . it is the first baby for both .	timberlake and biel welcome son silas randall timberlake . the couple announced the pregnancy in january .

Table 5.2: Cases 1: a fragment of input text and annotated summary from the CNN/DM dataset

Generated summaries of each of three models are presented in Table 5.3. The model was trained only on 5 epochs, so most of the generated summary is not structurally similar to the annotated summary but mostly contains similar important information. We can see that the M1 model generated mostly random words as a generated summary, where the meaning of a sentence is hard to understand; however, those words are being used in the annotated summary, so the ROUGE score is relatively high. Some words are repeated.

The M1-400 model produced a much better result, and we can understand what the input text was about. In addition, it copies more sentences and understands which parts of the input text are important to include, even though it doubles the last sentence.

The M2-400 model produced worse results than the previous model, even though it still slightly understandable what the input text was about. It copied less text, and we can see it included such a word as "*maternal*", which is rarely used in the CNN/DM dataset vocabulary and was not generated by other models. In general, most of the sentences that have been tested with this model contained more specific information from the input text, especially if it is rarely used. Additionally, sometimes it uses synonyms (like, "*poland's opposition*" becomes "*polish opposition*"); however, other models can generate synonyms from time to time as well.

Generated summary (M1)	**Generated summary (M1-400)**	**Generated summary (M2-400)**
announced name , while , while . what children an died in to randall is the pregnancy in january , with son an instagram post it is with son the	justin timberlake was the middle name of timberlake 's grandfather bill . the couple announced the arrival of their son , silas in 2012 . the couple announced the pregnancy in january . the couple announced the pregnancy in january .	silas randall was the middle name of timberlake 's maternal grandfather bill bomar . randall timberlake and jessica biel welcome to parenthood . the couple 's first baby . randall bomar . the couple 's first baby for both . people reports .

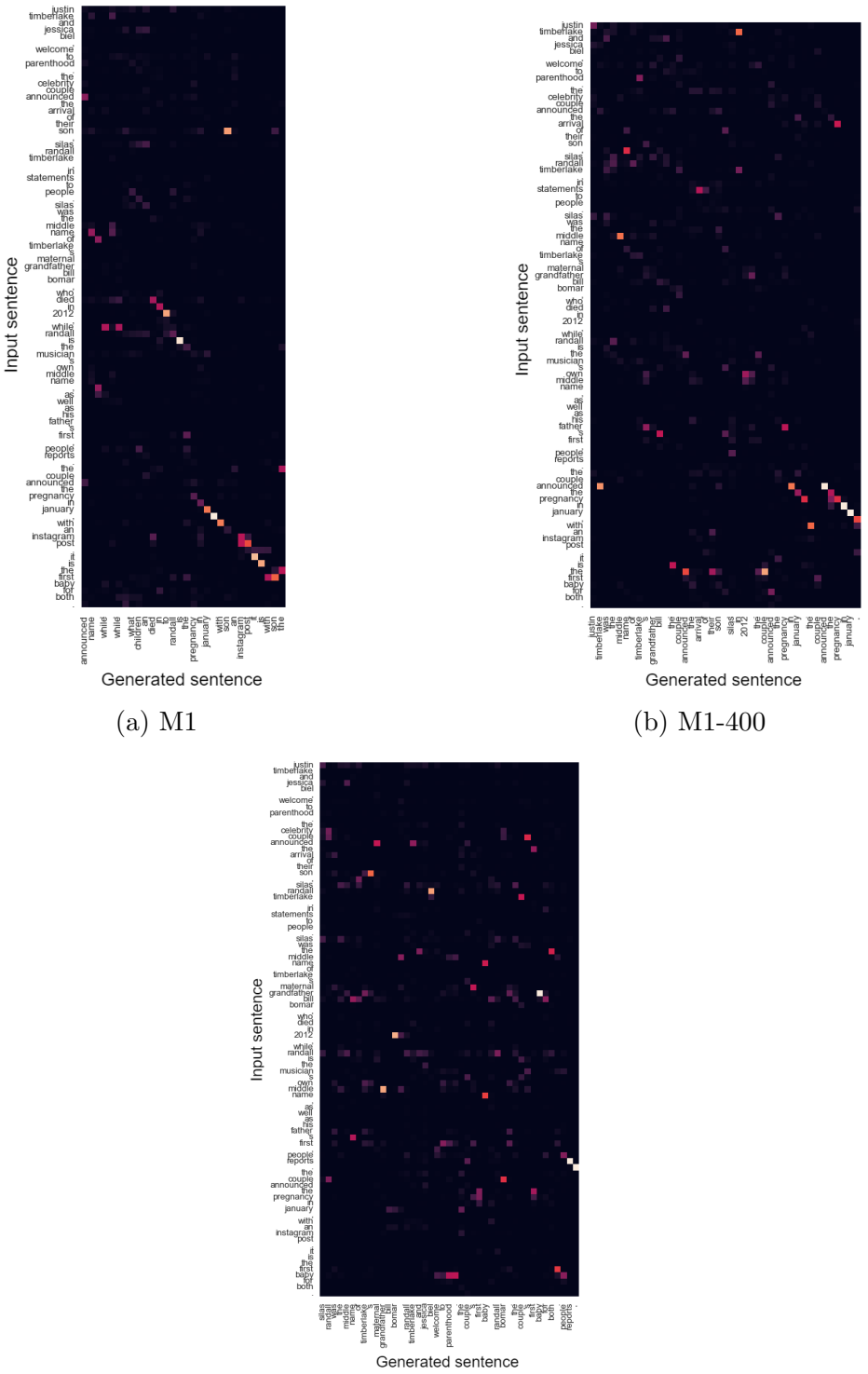
Table 5.3: Case 1: generated summary of different models

ROUGE scores for all models are shown in Table 5.4. This ROUGE score is for the generated summary from Table 5.3 that each model generated. All generated summaries achieved high ROUGE scores, with the M1-400 model being the best.

Model	**R-1**	**R-2**	**R-L**
M1	42	15	36
M1-400	53	29	53
M2-400	41	22	41

Table 5.4: Case 1: ROUGE score for different models from Table 5.3

The heatmap of attention on input text and annotated summary is shown in Fig. 5.3. If the square is lighter, the attention is higher. For very small or no attention, the square is dark. Each column represents candidate words with high attention to be used to generate a target word. The heatmap makes it possible to visualize the distribution of words with varying attention values for each model. For instance, for the M1 model the attention was often assigned only to a single word rather than a group of words. M1-400 model redistributes this attention throughout the whole input text, but sometimes, it has very strong attention when it decides to copy some words/phrases (bottom right part of Fig. 5.3b). On the M2-400 model heatmap, the attention is distributed around all words of an input text, with some high attention on specific words. The distribution occurs because the model boosts the attention of rarely used words and lowers the attention of commonly used words; therefore, we can see more attention on a group of words rather than on a single word.



(a) M1

(b) M1-400

(c) M2-400

Figure 5.3: Case 1: attention distribution between the input text and generated summary from the CNN/DM dataset for different models.

Case 2

The input text and the annotated summary of the second case are shown in Table 5.5.

Input text	Annotated summary
the build-up for the blockbuster fight between floyd mayweather and manny pacquiao in las vegas on may 2 steps up a gear on tuesday night when the american holds an open workout for the media . the session will be streamed live across the world and you can watch it here from 12am uk 7pm edt .	floyd mayweather holds an open media workout from 12am uk 7pm edt . the american takes on manny pacquiao in las vegas on may 2 . mayweather 's training is being streamed live across the world .

Table 5.5: Cases 2: a fragment of input text and annotated summary from CNN/DM dataset

Generated summaries of each of three models are presented in Table 5.6. We can see that the M1 model, as in the previous case, produced repeated words (like "*7pm*" or "*from 12 am*") and the general meaning of a generated summary is hard to understand, even though it includes information about the time of the event. M1-400 model generated a summary, which explained what was happening and where; however, a few key details are missing (like that it is "*streamed live across the world*"). M2-400 model, even though it repeated the same information several times, it gives information about what will happen and that it will be "*streamed live*". Unfortunately, this model does not include information about the time or the date the event.

ROUGE scores of these generated summaries are shown in Table 5.7. As in the previous case, the M1-400 model scored the highest ROUGE score.

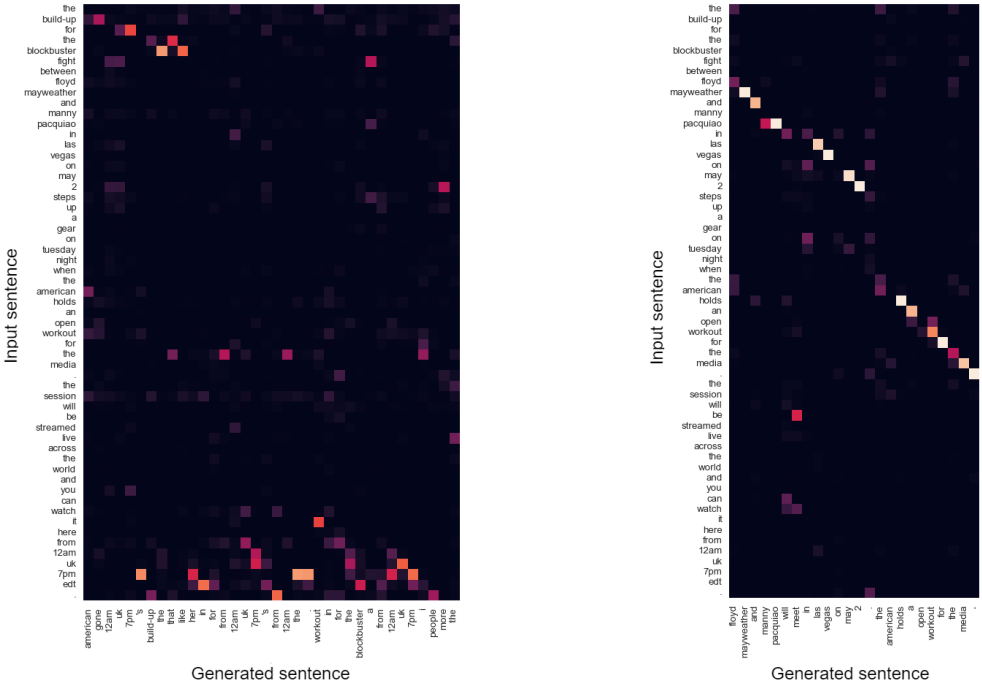
We can also plot the heatmap of attention on the input text and generated summary (Fig. 5.4). We can see that M1-400 and M2-400 models copied parts of the input text (diagonal line, where attention is the highest), and M2-400 distributed its attention more among different words.

Generated summary (M1)	**Generated summary (M1-400)**	**Generated summary (M2-400)**
american gone 12am uk 7pm 's build-up the that like her in for from 12am uk 7pm 's from 12am the . workout in for the block-buster a from 12am uk 7pm i people more the	floyd mayweather and manny pacquiao will meet in las vegas on may 2 . the american holds a open workout for the media .	the fight between floyd mayweather and manny pacquiao is set to be streamed on the media . the session will be held live across the world . the media . the session will be streamed live across the world and will be streamed on the media .

Table 5.6: Case 2: generated summary of different models

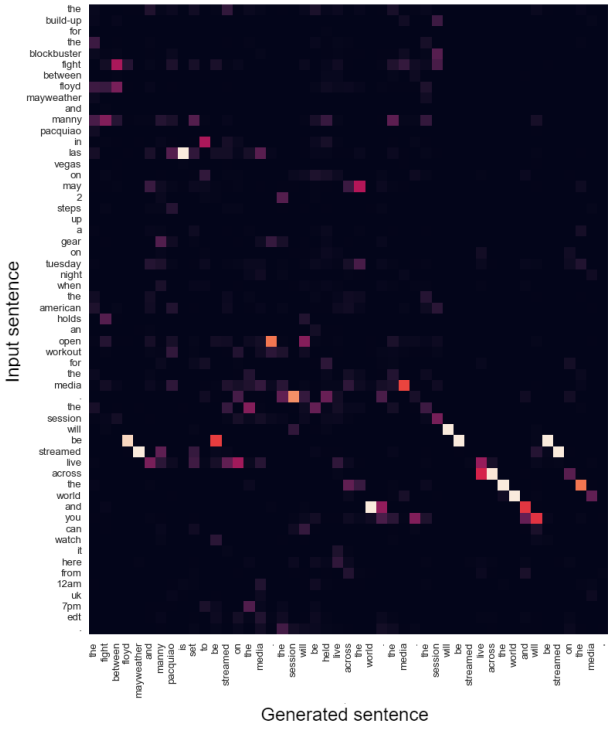
Model	**R-1**	**R-2**	**R-L**
M1	35	10	35
M1-400	62	30	58
M2-400	50	25	50

Table 5.7: Case 2: ROUGE score for different models from Table 5.6



(a) M1

(b) M1-400



(c) M2-400

Figure 5.4: Case 2: attention distribution between the input text and generated summary from the CNN/DM dataset for different models.

5.4.3 Models Trained on the XSum Dataset

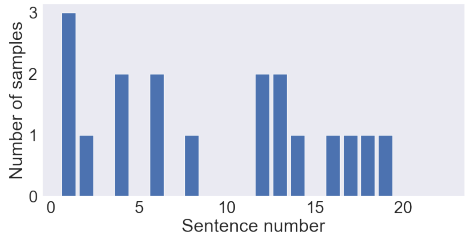
The process used to train and test the 6 models on the XSum dataset was identical to the training method used on the CNN/DM dataset since both datasets share a similar structure (i.e., long input text and short annotated summary, even though the CNN/DM dataset has longer annotated summaries). The results are shown in Table 5.8. The M1 model showed better results in R-1 and R-L, but in R-2, both models have similar results. As in the model, which was trained on the CNN/DM dataset, adding an extractive layer helped the model to achieve higher ROUGE score in both cases. However, we can see more clearly, that models without frequency information outperformed other models by almost one point score in R-1 and R-L.

Model	R-1	R-2	R-L
M1	**28.26**	8.95	**25.25**
M2	27.73	**9.13**	24.92
M1-200	**25.07**	**7.35**	**22.30**
M2-200	24.09	7.02	21.73
M1-400	**27.76**	**8.92**	**24.67**
M2-400	26.61	8.49	23.83

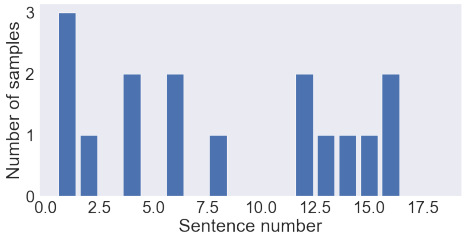
Table 5.8: Results of different models, which were trained and tested on the XSum dataset. Best results are shown in bold-face.

Test Cases

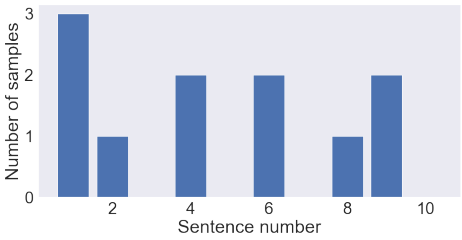
In this subsection, we provide some examples of what models generate. We will first describe how the generated text appears after applying the extractive approach. The XSum dataset uses extreme summarization, meaning that the annotated summaries will contain only essential information. The summarization tries to answer the question "What is the article about?". Because of this, the annotated summary contains fewer copied phrases or words from the input text and more rephrased sentences. Therefore, adding an extractive approach showed worse results. It is possible to show why it is the case by plotting the number of words from the annotated summary in the input text (Fig. 5.5). We can see that the number of words in the input text is much lower. This is because the default dataset contains more words from the annotated summary (in total, an unprocessed dataset, a dataset with an extractive approach with 400 tokens, and a model with an extractive approach with 200 tokens used 18, 11 and 16 words, respectively). It is not the case for all samples from the



(a) Without extractive approach



(b) With extractive approach (400 tokens)



(c) With extractive approach (200 tokens)

Figure 5.5: Summarization statistics from the annotated summary, which appear in the input text from the XSum dataset.

dataset; however, it shows that the extractive approach plays a less significant role in the XSum dataset, as the annotated summary is shorter, and the sentences are rephrased.

Case 1

The input text and the annotated summary of the first case are shown in Table 5.9.

Input text	Annotated summary
connor williams , 17 , and conor tiley , 18 , both from aberbargoed , died in a crash on new road , tir-y-berth , hengoed , on 3 january . the teenager was arrested on suspicion of causing death by dangerous driving . gwent police confirmed no further action would be taken .	no charge over connor williams and conor tiley crash deaths

Table 5.9: Cases 1: a fragment of input text and annotated summary from the XSum dataset

All models were able to generate summaries that were understandable (shown in Table 5.10), even though they did not include information that "*no one was charged*", which is present in the annotated summary. M2-400 model uses some rare words, like "*bailed*" or "*aberbargoed*". Even though all three models generated good summary, we can see that the M1 model generated the summary closest to the input text, even though the M1-400 model has the highest ROUGE score (Table 5.11).

Generated summary (M1)	**Generated summary (M1-400)**	**Generated summary (M2-400)**
teenager arrested over double fatal crash in aberbargoed	connor williams : teenager arrested over fatal bike crash	two people bailed after fatal crash in aberbargoed

Table 5.10: Case 1: generated summary of different models

Model	**R-1**	**R-2**	**R-L**
M1	22	0	22
M1-400	42	12	32
M2-400	11	0	11

Table 5.11: Case 1: ROUGE score for different models from Table 5.10

This example which shows that the ROUGE score is not a good indicator of the model performance in terms of summarizing the essence of the input text.

After plotting a heatmap (shown in Appendix A.2.1, Fig. A.1), we can see that attention in all models is distributed among all words in the input text, and models do not place high attention on specific words. This is due to the fact, as it was said previously 4.1.1, that the XSum dataset is extremely abstractive, and the words in the annotated summary might be rephrased, meaning the model cannot rely only on a single word or a group of words (to copy them).

Case 2

The input text and the annotated summary of the second case are shown in Table 5.12.

Input text	Annotated summary
the 72-year-old , formerly of eliza street close , belfast faced 12 counts of child sex abuse . he was accused of abusing a 15-year-old girl between 2009 and 2010 . he denied all the charges . the 12-member jury had spent more than six hours deliberating before being discharged on wednesday . the judge thanked the 10 women and two men for their time . the trial at belfast crown court started on 28 may .	francis mcpeake trial : jury discharged in sex abuse case

Table 5.12: Cases 2: a fragment of input text and annotated summary from the XSum dataset

Generated summary is shown in Table 5.13. None of the models generated correct summary, even though the meaning is different. M1-400 model wrongly assigned the age to the person (82 and not 72). M2-400 model wrongly used 15 years in the generated summary. Even though the ROUGE score for the M1 model is high (Table 5.14), the generated summary has a different meaning than in the input text.

Generated summary (M1)	Generated summary (M1-400)	Generated summary (M2-400)
belfast sex abuse trial : former man appears in court	belfast child sex abuser , 82 , denies child sex offences	belfast sex abuse trial adjourned for 15 years for sex .

Table 5.13: Case 2: generated summary of different models

After plotting a heatmap (shown in Appendix A.2.2, Fig. A.2), we can see that all models' attention is distributed among all words in the input text, similarly to the previous cases.

Model	R-1	R-2	R-L
M1	50	22	30
M1-400	11	0	11
M2-400	32	11	21

Table 5.14: Case 2: ROUGE score for different models from Table 5.13

5.4.4 Models Trained on the Gigaword Dataset

The Gigaword dataset is different from the previous two datasets. From the results shown in Table 5.15, we can see that there are no extractive approach results, as the input text has only a few sentences, with an average of 31 tokens used in the input text. Hence, the extractive approach is not applicable.

The results show that frequency information does not help for this dataset either. The difference between two models is much more significant than for models which were trained on other datasets, where the difference is more than 1.5 ROUGE points.

Model	R-1	R-2	R-L
M1	**33.09**	**15.64**	**31.14**
M2	32.31	14.68	30.49

Table 5.15: Results of different models, which were trained and tested on the Gigaword dataset. Best results are shown in bold-face.

Case 1

The input text and the annotated summary for the first case are shown in Table 5.16.

Input text	Annotated summary
polling stations closed at gmt local time friday on the first day of czech legislative elections shadowed by allegations surrounding social democrat prime minister jiri paroubek .	polling stations close on first day of czech legislative elections

Table 5.16: Cases 1: a fragment of input text and annotated summary from the Gigaword dataset

The summaries, that are generated by models, are shown in Table 5.17). The M1 model included more information and is more similar to the annotated summary, even though both generated summaries are similar. We can see that the generated summary is short and more understandable compared to models trained on the previous datasets. This is because the Gigaword dataset has short input text and short annotated summary, so the model can easily generate important information using

a short text. ROUGE scores are shown in Table 5.18. Even though two generated summaries are similar, M1 model has a much higher score, which shows again that a ROUGE score is not the best metric to decide whether the model is good or not.

Generated summary (M1)	Generated summary (M2)
polling stations close in czech legislative elections	polls close in czech republic

Table 5.17: Case 1: generated summary of different models

Model	R-1	R-2	R-L
M1	71	53	71
M2	27	0	27

Table 5.18: Case 1: ROUGE score for different models from Table 5.17

After plotting a heatmap (shown in Appendix A.2.3, Fig. A.3), we can see that both models put high attention on some words to generate a target word (meaning, it was mostly copying) and rarely used other words. This is because annotated summaries in the Gigaword dataset mostly contain the exact words/phrases present in the input text.

Case 2

The input text and the annotated summary of the second case are shown in Table 5.19.

Input text	Annotated summary
a young syrian woman who was arrested last year on terrorism charges at the airport here had a map of us military facilities in turkey , a canadian security official said thursday .	canada investigates syrian woman with alleged ties to pkk

Table 5.19: Cases 2: a fragment of input text and annotated summary from the Gigaword dataset

Generated summaries are presented in Table 5.20). The M2 model generated a better summary, and it is logically correct. The summary generated by the M1 model is logically wrong (the woman is arrested in Canada, not in the USA). ROUGE score is shown in Table 5.21.

Generated summary (M1)	**Generated summary (M2)**
syrian woman arrested at us airport in turkey	syrian woman arrested in canada on terror charges

Table 5.20: Case 2: generated summary of different models

Model	**R-1**	**R-2**	**R-L**
M1	24	13	24
M2	35	13	24

Table 5.21: Case 2: ROUGE score for different models from Table 5.20

The heatmap of models is shown in Appendix A.2.4, Fig. A.4. Similarly to the previous example, both models put high attention on specific words, and the M1 model copied the last few words.

5.4.5 Analysis of results

We begin with the discussion in terms of the effect of adding *information about frequency to the attention mechansim.* on the performance of the models. In general, the results are worse in most experiments with a few exceptions for the models trained on the CNN/DM and the XSum datasets. However, frequency information helps to distribute attention throughout several words rather than relying on a single word or copying words. It only helps sometimes, as, in the CNN/DM dataset, many phrases are copied from the input text. Adding frequency information also helps to generate words from the input text, which are rarely used. However, it only helps sometimes, as it forces the model to generate more information, which might worsen the result.

To compare the performance of the 6 models, we have trained them on the 3 datasets (CNN/DM, XSum, Gigaword) and tested with the DUC 2004 Task 1 dataset which we will discuss in the following section.

Model	R-1	R-2	R-L
M1 CNN/DM	19.64	4.36	18.75
M2 CNN/DM	20.86	4.50	18.82
M1-200 CNN/DM	21.13	5.03	19.18
M2-200 CNN/DM	21.19	**5.15**	19.27
M1-400 CNN/DM	20.92	4.81	18.90
M2-400 CNN/DM	**21.59**	**5.17**	**19.46**
M1 XSum	12.97	2.21	11.76
M2 XSum	12.07	2.22	11.16
M1-200 XSum	10.59	1.64	9.98
M2-200 XSum	14.45	**2.62**	13.27
M1-400 XSum	13.46	1.86	12.24
M2-400 XSum	**14.67**	**2.63**	**13.83**
M1 Gigaword	**26.35**	**9.23**	**24.50**
M2 Gigaword	26.24	8.89	24.40

Table 5.22: Results of different models, which were tested on the DUC 2004 Task 1 dataset. Best results are shown in bold-face.

Models Tested on the DUC 2004 Task 1 Dataset

In Table 5.22, we can observe that the model trained on the Gigaword dataset performed the best. It is because the DUC 2004 Task 1 dataset has short input text and short annotated summary, with only 36 and 11 tokens on average, respectively. Both the CNN/DM and the XSum datasets have, on average, around 400 tokens for an input text, meaning that for models, which were trained on these datasets, most of the tokens will have a padding symbol. These models were trained to gather information from different parts in the text, and when the text is short, we can see that models, which were trained on these datasets, performed poorly.

Models, which were trained on the CNN/DM dataset and used frequency information, always show better results than those without frequency information. Another interesting remark is that having only 200 tokens in an input text with an extractive approach shows better results in both models, compared to the default ones. The M1-400 model shows slightly worse results than the M1-200 model. The M2-400 model trained on the CNN/DM dataset showed the best ROUGE score (R-1 score of 21.59, R-2 score of 5.12 and R-L score of 19.46). For the model without frequency information, the M1-200 model is better (R-1 score of 21.13, R-2 score of 5.03 and R-L score of 19.18). It might be connected to the fact that the CNN/DM dataset has longer input text and longer annotated summary when the DUC 2004 Task 1 dataset is

shorter; therefore, models trained on a shorter input text (like the Gigaword dataset) show better results.

Models trained on the XSum dataset with frequency information are almost always better, except for the M1 model. Having 200 tokens worsens the results for the M1 model, and shows improvement for the M2-400 model. The M1-400 model shows worse results for R-2 (R-2 score of 1.86). In contrast, having frequency information boosts the results of the M2-200 and M2-400 models compared to the one without this information, where the M2-400 model has the highest overall score (R-1 score of 14.67, R-2 score of 2.63 and R-L score of 13.89).

The M1 and M2 models trained on the Gigaword dataset have a difference in ROUGE score of 0.1-0.3 points, with the M1 model having the highest score (R-1 score of 26.35, R-2 score of 9.23 and R-L score of 24.50). It is a significant improvement for the M2 model, tested on the Gigaword dataset, where the difference was 1-2 points.

The reason why the Gigaword dataset performs much better than models trained on any other datasets is that the Gigaword dataset has short input text and short annotated summary (the number of tokens for the input text and the annotated summary is similar to the DUC 2004 Task 1 dataset) whereas the XSum and the CNN/DM datasets have much longer input text. Even though the XSum dataset has a shorter annotated summary compared to the one in the CNN/DM dataset, it showed worse results, which might be connected to the fact that the XSum dataset is extremely abstractive, meaning that the annotated summary contains changed phrases, compare to the one in the input text. The model trained on the CNN/DM dataset learned to copy phrases or words; therefore, there is a higher chance of scoring a higher ROUGE score on the DUC 2004 Task 1 dataset, which might contain copied information.

Test Cases

Table 5.23 shows three samples from the DUC 2004 Task 1 dataset, which were used for testing. Table 5.24 showed generated summary from each of three models. M2-400 model (trained on the CNN/DM dataset), M2-400 model (trained on the XSum dataset) and M1 model (trained on the Gigaword dataset) were for chosen for discussion since they had the best ROUGE score.

Input text	Annotated summary
cambodian leader hun sen on friday rejected opposition parties demands for talks outside the country , accusing them of trying to internationalize the political crisis .	cambodian government rejects opposition 's call for talks abroad
cambodia 's bickering political parties broke a three-month deadlock friday and agreed to a coalition government leaving strongman hun sen as sole prime minister , king norodom sihanouk announced .	cambodian king announces coalition government with hun sen as sole premier
pope john paul ii appealed for aid wednesday for the central american countries stricken by hurricane mitch and said he feels close to the thousands who are suffering .	pope calls for aid to central america where hurricane mitch killed 9,000

Table 5.23: Three samples from the DUC 2004 Task 1 dataset

M2-400 CNN/DM	M2-400 XSum	M1 Gigaword
opposition leaders rejected talks over ' demands for talks outside of the opposition parties rejected the opposition . hun sen , on friday rejected the	nigeria elections : voting under way for cambodian opposition	cambodian leader rejects opposition demands for talks
cambodia 's strongman hun sen hun sen as sole prime minister . strongman hun sen hun sen will be sole prime minister . king norodom	cambodia profile media	cambodian parties agree to coalition government leaving strongman hun sen as sole pm
pope john paul ii says he feels close to the thousands who are suffering . pope john ii appealed for aid wednesday for the central	hurricane paul ii : aid arrives in pictures	pope urges aid for central america hurricane mitch

Table 5.24: Generation of summary from best models, which were tested on fragments of the DUC 2004 Task 1 dataset

We can see that the M2-400 CNN/DM model always generated the longest summary, as it was trained to generate longer text. The M2-400 XSum model performs poorly and cannot generate a meaningful summary containing important information. As we mentioned before, this is because models trained on the XSum dataset are learned to generate extremely abstractive summaries using long input text, and when a short input text is given, it fails to generate meaningful summaries.

The M1 Gigaword model showed the best performance. It can generate similar summaries to the annotated summaries (for instance, 3rd generated summary is very similar to the annotated summary), and it can include important information from

the input text. As discussed before, the Gigaword model was trained on a short input texts and short annotated summaries, and the DUC 2004 Task 1 dataset also has similar length for both texts.

ROUGE score for each fragment (example) is shown in Table 5.25. We can conclude that a ROUGE score of the model strongly depends on the dataset it was trained on. Moreover, models trained on separate datasets generate differently structured summaries and focus on different information. The Gigaword dataset has a similar structure to the DUC 2004 Task 1 dataset, which is why the model trained on the Gigaword dataset performed so well on the latter one. The model, which is trained on the XSum dataset, generated a summary, which is highly abstractive, and it copies less information from the input text compared to the models trained on the CNN/DM dataset.

Adding an extractive approach helps to increase a ROUGE score of the model, which was trained on the CNN/DM and even XSum dataset. In addition to that, adding frequency score inside the self-attention of an encoder helped to achieve higher results; however, if the model is tested on the same dataset, adding frequency information fails to outperform the M1 model, even though it generates more interesting summaries, which might contain additional information.

Model	**R-1**	**R-2**	**R-L**
M2-400 CNN/DM (example 1)	30	7	30
M2-400 CNN/DM (example 2)	42	24	42
M2-400 CNN/DM (example 3)	30	6	30
M2-400 XSum (example 1)	32	0	21
M2-400 XSum (example 2)	0	0	0
M2-400 XSum (example 3)	18	0	9
M1 Gigawords (example 1)	64	27	59
M1 Gigawords (example 2)	58	36	58
M1 Gigawords (example 3)	64	20	55

Table 5.25: ROUGE score for different models from Table 5.24

5.5 Comparative Analysis: Benchmark Models

The current state-of-the-art models use a pre-trained encoder and can achieve higher accuracy (more than 44 for R-1). However, since we are not using pre-trained embedding, we have selected three models as our benchmark [12, 31, 36]. The model words-lvt2k-temp-att (Nallapati et al. [12]) and pointer-generator (See et al. [31]) are based on Seq2Seq backbone with pointer-generator layer. The model from Transformer + Pointer-Generator + N-Gram Blocking (2-gram) (Deaton et al. [36]) tried to combine transformer [1] with the pointer-generator layer from [31].

The results are shown in Table 5.26. Both models (M1-400 and M2-400 models) have been trained for 30 epochs. We can see that both of our models showed better results in R-1 and R-L scores compared to other papers and slightly lower on R-2 than the model from [31]. We could achieve similar results to the model from [31] on R-1 and R-L scores while the model is being trained only for 5 epochs (Table 5.1). Even though adding frequency information lowered a ROUGE score, the results are still compatible.

Model	R-1	R-2	R-L
words-lvt2k-temp-att (Nallapati et al. [12])	35.46	13.30	32.65
pointer-generator (See et al. [31])	36.44	**15.66**	33.42
Transformer + Pointer-Generator + N-Gram Blocking (2-gram) (Deaton et al. [36])	25.31	4.16	15.99
M1-400	**38.22**	15.07	**35.79**
M2-400	37.52	14.46	35.02

Table 5.26: Comparing our models to the models from other papers

These results show, that our model outperforms the benchmarked, which use the pointer-generator model or pointer-generator with a transformer. Adding a pre-trained encoder or coverage mechanism to our models could boost the results further.

Chapter 6

Conclusion and Future Work

In this thesis, we have successfully implemented a hybrid model using extractive and abstractive methods with a transformer and pointer-generator layer, which helped to achieve a higher ROUGE score and boost rarely used words. In addition, we showed that the extractive approach is essential to preprocess the input text, as important information is located in different parts of the text, and there are better approaches than taking the first N tokens.

We have used four datasets (CNN/DM, XSum, Gigaword and DUC 2004 Task 1) for training and/or testing and compared models with each other. Adding frequency information to the model only sometimes improves results; however, the model might add extra information using less common words because the model distributes the attention to different words rather than to a single word. M1-400 achieved a high ROUGE score of R-1 score of 38.22, R-2 score of 15.07 and R-L score of 35.79 and outperformed [31] model in R-1 and R-L score by at least 2 points, even though R-2 score is slightly lower. M2-400 model showed better results in almost all tests on the DUC 2004 Task 1 dataset compared to either the M1 or the M1-400 models, which is because the M2 model has additional information about the rarity of each word which seemed helpful when tested on a different dataset that the model was trained on. A heatmap matrix of attention for each case study has been shown so that the distribution of attention can be visualized easily.

In future work, coverage mechanism [65] can be implemented, which tracks what information has been included and eliminates repetitive words or phrases. In some cases that have been shown (for instance, examples of a generated summary of models that have been trained on the CNN/DM dataset) important information has been repeated several times. In addition, a more robust evaluation technique should be

tested. $ROUGE_{F_1}$ score is not the best evaluation technique. For instance, if a synonym of a word is used, which does not change the meaning of a sentence, the ROUGE score will be lowered, as in this case, it will not be counted towards the N-gram metric.

Even though adding frequency information to the encoder's attention improved ROUGE score of models, which were tested on the DUC 2004 Task 1 dataset, in most of cases that have been described in section 5.4 adding frequency information worsen ROUGE score and the generated summary sometimes was including more information then is needed. Changing how the frequency scoring is passed to the encoder could be tested, as well as adding frequency scoring to a decoder part, which can help to focus on important information that has been generated so far. This might help to eliminate repetition of important information in the generated text, as the model would be able to "notice" this information using the attention, which might lead to lowering probability of generating particular words.

In addition, the method to compute the frequency can be changed, so that it is based not only on how frequently the word appears in the input text.

Encoder-decoder network could be trained in parallel to speed up the process (as it is done in [14]); however, it is not clear whether the speed time might increase significantly in the model that has been described in this research.

Appendix A

Appendix

A.1 Tracing of the Algorithm

In this section, some variables are shown with their values that are used during training.

A.1.1 Annotated summary

Variables and a sample of its value that are used for the annotated text. All padded symbols are masked with "False"; therefore, last list of arrays in *trg_mask* has only "False" at the end. *trg_inp* and *trg_ext* are almost identical. The difference is that *trg_ext* contains OOV words (highlighted in blue).

A.1.2 Input text

Variables and a sample of its value that are used for the input text. *src* and *src_ext* are almost identical. The difference is that *src_ext* contains OOV words (highlighted in blue).

Variable	Value
annotated summary	legendary football coach and broadcaster john madden announces he is retiring . the nfl has been my life for more than 40 years , it has been my passion , he says . madden is best known to millions as an ebullient football commentator .
trg_inp	[2, 5217, 467, 998, 9, 5710, 347, 16059, 9634, 19, 13, 7605, 4, 5, 2587, 33, 45, 84, 137, 15, 55, 72, 849, 81, 6, 20, 33, 45, 84, 3762, 6, 19, 98, 4, 16059, 13, 244, 306, 7, 1248, 26, 39, 0, 467, 8903, 4, 3, 1, 1, ..., 1]
trg_ext	[5217, 467, 998, 9, 5710, 347, 16059, 9634, 19, 13, 7605, 4, 5, 2587, 33, 45, 84, 137, 15, 55, 72, 849, 81, 6, 20, 33, 45, 84, 3762, 6, 19, 98, 4, 16059, 13, 244, 306, 7, 1248, 26, 39, 50000, 467, 8903, 4, 3, 1, 1, ..., 1]
trg_mask	[[True, False, False, ..., False, False, False], [True, True, False, ..., False, False, False], [True, True, True, ..., False, False, False], ..., [True, True, True, ..., True, False, False], [True, True, True, ..., False, False, False], [True, True, True, ..., False, False, False]]
trg_full	legendary football coach and broadcaster john madden announces he is retiring . the nfl has been my life for more than 40 years , it has been my passion , he says . madden is best known to millions as an ebullient football commentator .

Table A.1: Sample of a variable and its value for the annotated summary.

Variable	Value
input text	new york legendary football coach and broadcaster john madden is retiring , he announced thursday . john madden appears at the tv critics association press tour in beverly hills , california , in 2008 . it is been such a great ride ... the nfl has been my life for more than 40 years , it has been my passion it still is , he said in a statement released by nbc sports . madden , 73 , was a hall of fame coach for the oakland raiders , but is best known to millions as an ebullient football commentator . he won 16 emmy awards for outstanding sports analyst/personality , nbc said .
src	[62, 256, 5217, 467, 998, 9, 5710, 347, 16059, 13, 7605, 6, 19, 648, 365, 4, 347, 16059, 1136, 25, 5, 588, 1930, 1255, 658, 931, 11, 6994, 3430, 6, 646, 6, 11, 785, 4, 20, 13, 45, 162, 8, 298, 2157, 279, 5, 2587, 33, 45, 84, 137, 15, 55, 72, 849, 81, 6, 20, 33, 45, 84, 3762, 20, 145, 13, 6, 19, 22, 11, 8, 311, 381, 30, 2627, 1108, 4, 16059, 6, 6344, 6, 14, 8, 1393, 10, 3095, 998, 15, 5, 7542, 11952, 6, 36, 13, 244, 306, 7, 1248, 26, 39, 0, 467, 8903, 4, 19, 471, 694, 11538, 2266, 15, 4246, 1108, 0, 6, 2627, 22, 4, 1, 1, ..., 1]
src_ext	[62, 256, 5217, 467, 998, 9, 5710, 347, 16059, 13, 7605, 6, 19, 648, 365, 4, 347, 16059, 1136, 25, 5, 588, 1930, 1255, 658, 931, 11, 6994, 3430, 6, 646, 6, 11, 785, 4, 20, 13, 45, 162, 8, 298, 2157, 279, 5, 2587, 33, 45, 84, 137, 15, 55, 72, 849, 81, 6, 20, 33, 45, 84, 3762, 20, 145, 13, 6, 19, 22, 11, 8, 311, 381, 30, 2627, 1108, 4, 16059, 6, 6344, 6, 14, 8, 1393, 10, 3095, 998, 15, 5, 7542, 11952, 6, 36, 13, 244, 306, 7, 1248, 26, 39, 50000, 467, 8903, 4, 19, 471, 694, 11538, 2266, 15, 4246, 1108, 50001, 6, 2627, 22, 4, 1, 1, ..., 1]
src_freq	[0.8087, 0.8195, 0.8524, 0.8233, 0.8295, 0.7958, 0.8543, 0.8213, 0.8844, 0.7992, 0.8613, 0.7931, 0.8007, 0.8259, 0.8216, 0.7922, 0.8213, 0.8844, 0.8308, 0.8021, 0.7923, 0.8251, 0.8369, 0.8319, 0.8260, 0.8290, 0.7962, 0.8593, 0.8450, 0.7931, 0.8259, 0.7931, 0.7962, 0.8273, 0.7922, 0.8009, 0.7992, 0.8056, 0.8162, 0.7957, 0.8204, 0.8384, 0.8200, 0.7923, 0.8407, 0.8035, 0.8056, 0.8108, 0.8148, 0.7998, 0.8080, 0.8101, 0.8281, 0.8104, 0.7931, 0.8009, 0.8035, 0.8056, 0.8108, 0.8464, 0.8009, 0.8153, 0.7992, 0.7931, 0.8007, 0.8018, 0.7962, 0.7957, 0.8207, 0.8220, 0.8030, 0.8410, 0.8306, 0.7922, 0.8844, 0.7931, 0.8568, 0.7931, 0.7996, 0.7957, 0.8329, 0.7958, 0.8434, 0.8295, 0.7998, 0.7923, 0.8610, 0.8739, 0.7931, 0.8039, 0.7992, 0.8189, 0.8205, 0.7952, 0.8318, 0.8022, 0.8047, 2.1736, 0.8233, 0.8652, 0.7922, 0.8007, 0.8234, 0.8264, 0.8728, 0.8390, 0.7998, 0.8485, 0.8306, 2.1736, 0.7931, 0.8410, 0.8018, 0.7922, 0.0000, 0.0000, ..., 0.0000]
oov_words	['ebullient', 'analyst/personality']
oov_words (index)	[50000, 50001]

Table A.2: Sample of a variable and its value for the input text.

A.1.3 Internal variables

Values of internal variables inside the model. Values are taken from Alg. 3. When the attention is high, the model is more likely to use the word in generation of a next word. The attention of padded symbol is 0. *gen_sum_probab* contains final probabilities and has a shape of [100, 50004].

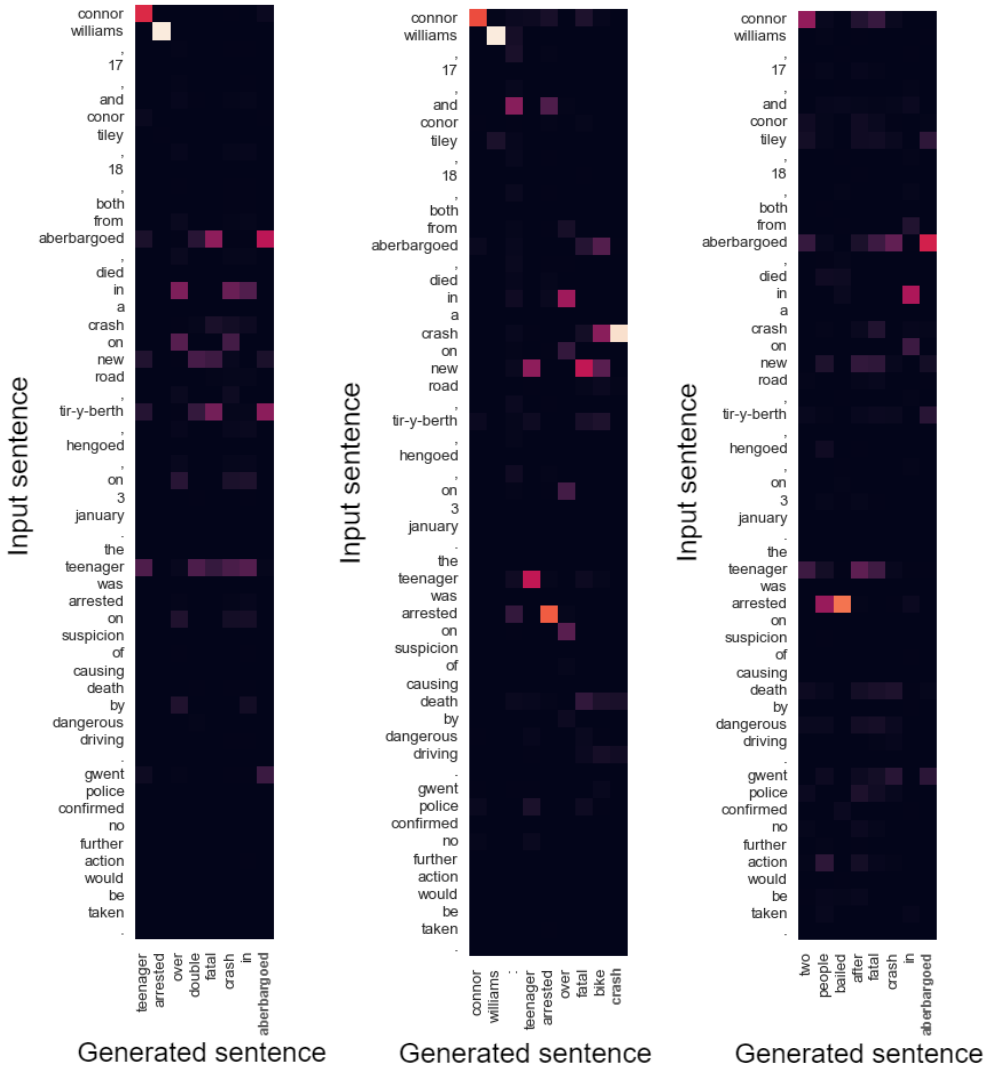
Variable	Value
x	[[-0.3389, -0.0000, -0.9329, ..., -0.1484, 0.1155, -0.0000], [0.5721, 0.7474, 0.0000, ..., 2.1475, 1.4672, 0.2143], ..., [-0.0000, -0.8321, -0.0551, ..., 1.2123, -0.1525, 1.2242], [-1.0256, 0.1225, -0.9025, ..., 1.2123, -0.1524, 1.2242]]
s	[[0.5114, -0.9785, 0.8487, ..., -0.5495, -1.4523, -0.8624], [-1.5298, 1.4677, 0.8478, ..., 0.4266, -2.7031, -0.9309], ..., [-0.5949, 1.7759, 2.0998, ..., 3.1443, 1.1263, 0.3926], [-0.0422, 1.0565, 1.8157, ..., 3.1954, 2.0054, 0.6575]]
h	[[0.4188, 0.2916, 0.5588, ..., -0.4898, 0.0350, -0.2848], [-0.3276, 0.2012, -0.0741, ..., -0.6512, 0.1389, 0.7616], ..., [-0.1768, -0.1500, 0.1356, ..., -0.3598, 0.0180, -0.2839], [-0.0890, -0.1315, 0.2067, ..., -0.3371, 0.0448, -0.3437]]
Attention	[[0.0616, 0.1194, 0.0735, ..., 0.0000, 0.0000, 0.0000], [0.0344, 0.1880, 0.0431, ..., 0.0000, 0.0000, 0.0000], ..., [0.0088, 0.0048, 0.0081, ..., 0.0000, 0.0000, 0.0000], [0.0091, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000]]
Attention $(+freq.)$	[[0.0197, 0.0334, 0.0189, ..., 0.0000, 0.0000, 0.0000], [0.0111, 0.0532, 0.0112, ..., 0.0000, 0.0000, 0.0000], ..., [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000], [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000]]
gen_sum_probab	[[1.314e-02, 5.600e-07, 2.160e-06, ..., 2.948e-03, 1.104e-02, 1.880e-03], [7.5745e-03, 6.631e-07, 2.669e-06, ..., 4.856e-04, 0.000e+00, 5.227e-05], ..., [1.093e-04, 7.920e-07, 2.704e-06, ..., 5.896e-05, 7.481e-05, 1.561e-05], [2.516e-04, 6.634e-07, 2.250e-06, ..., 1.357e-04, 1.577e-04, 3.759e-05]]

Table A.3: Values of internal variables inside the model.

A.2 Heatmap matrices

A.2.1 XSum Dataset. Case 1

The heatmap of attention on input text and annotated summary of a model trained on the XSum dataset for case 1.

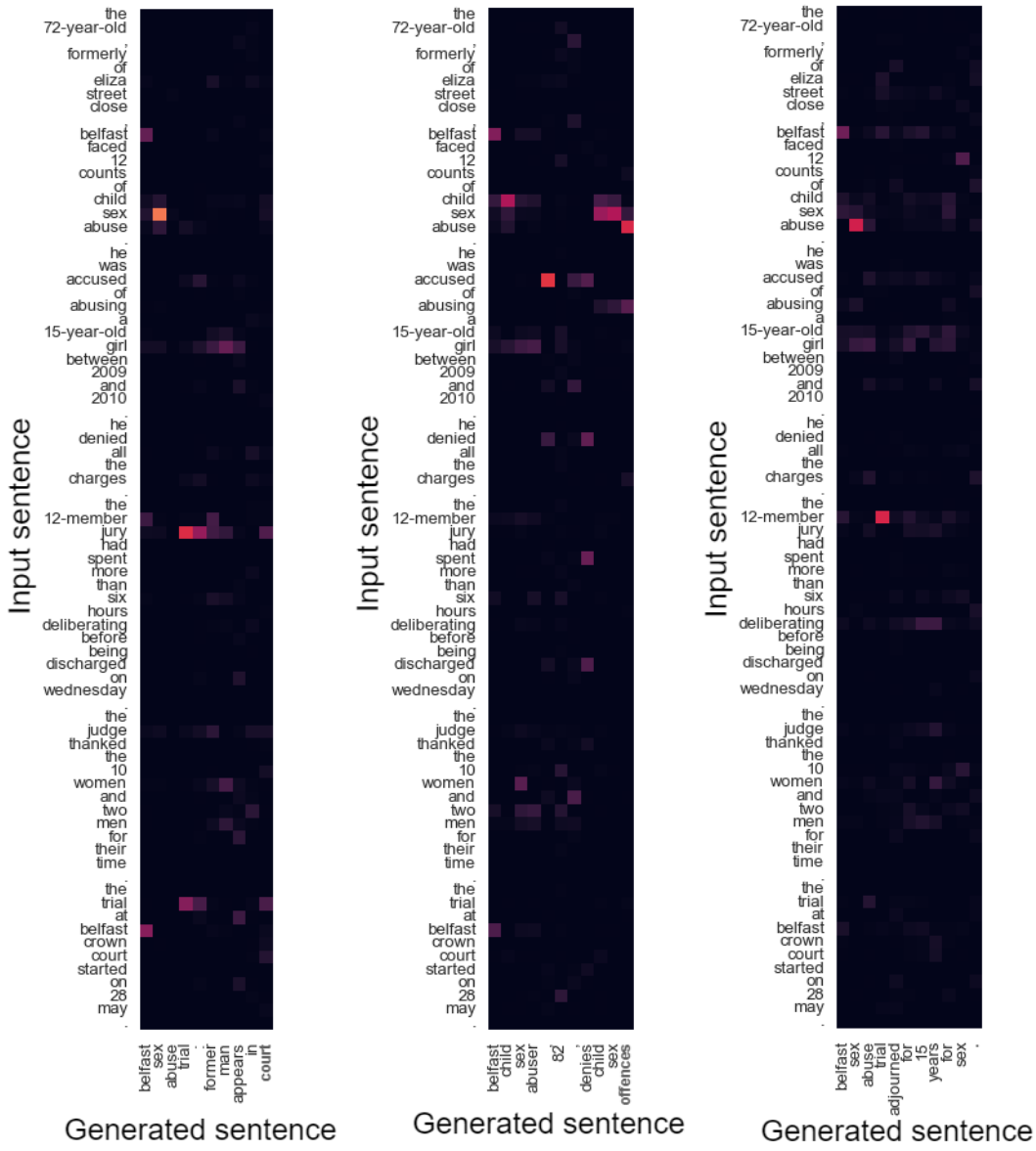


(a) M1 (b) M1-400 (c) M2-400

Figure A.1: Case 1: attention distribution between the input text and generated summary from the XSum dataset for different models.

A.2.2 XSum Dataset. Case 2

The heatmap of attention on input text and annotated summary of a model trained on the XSum dataset for case 2.

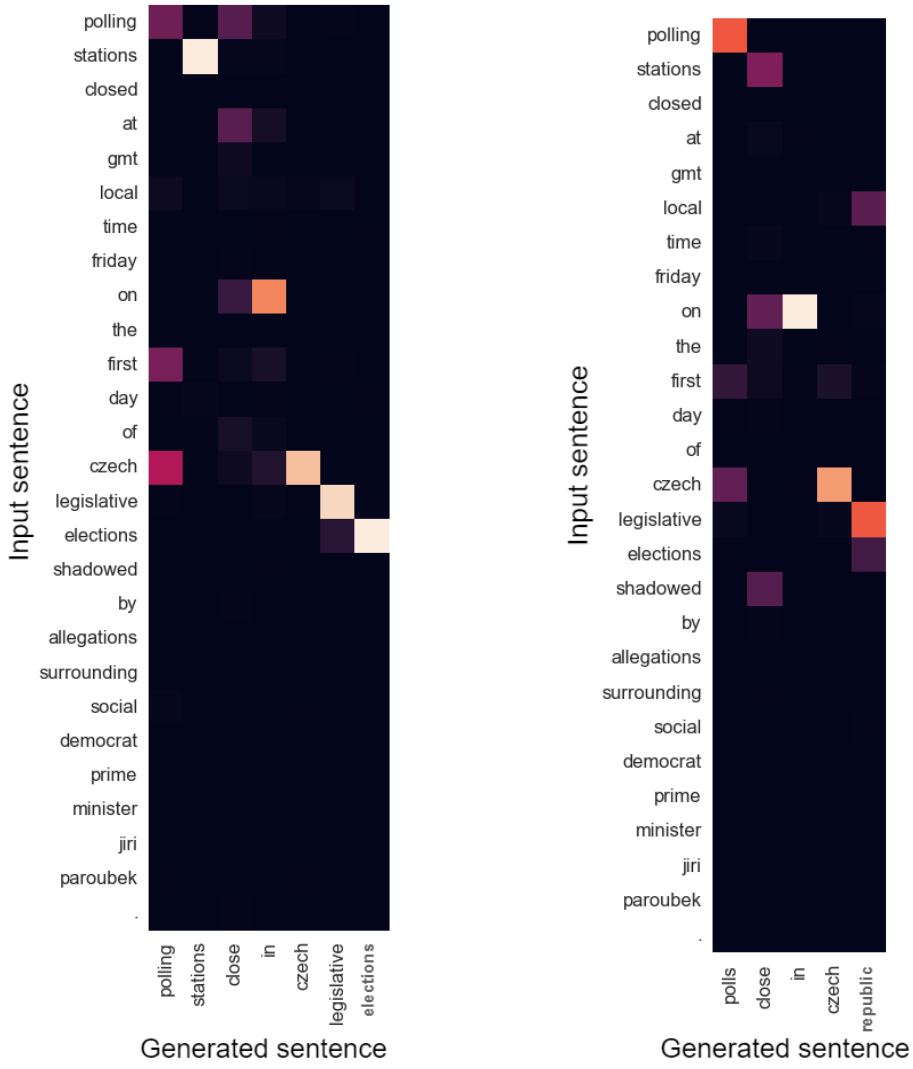


(a) M1 (b) M1-400 (c) M2-400

Figure A.2: Case 2: attention distribution between the input text and generated summary from the XSum dataset for different models.

A.2.3 Gigaword Dataset. Case 1

The heatmap of attention on input text and annotated summary of a model trained on the Gigaword dataset for case 1.

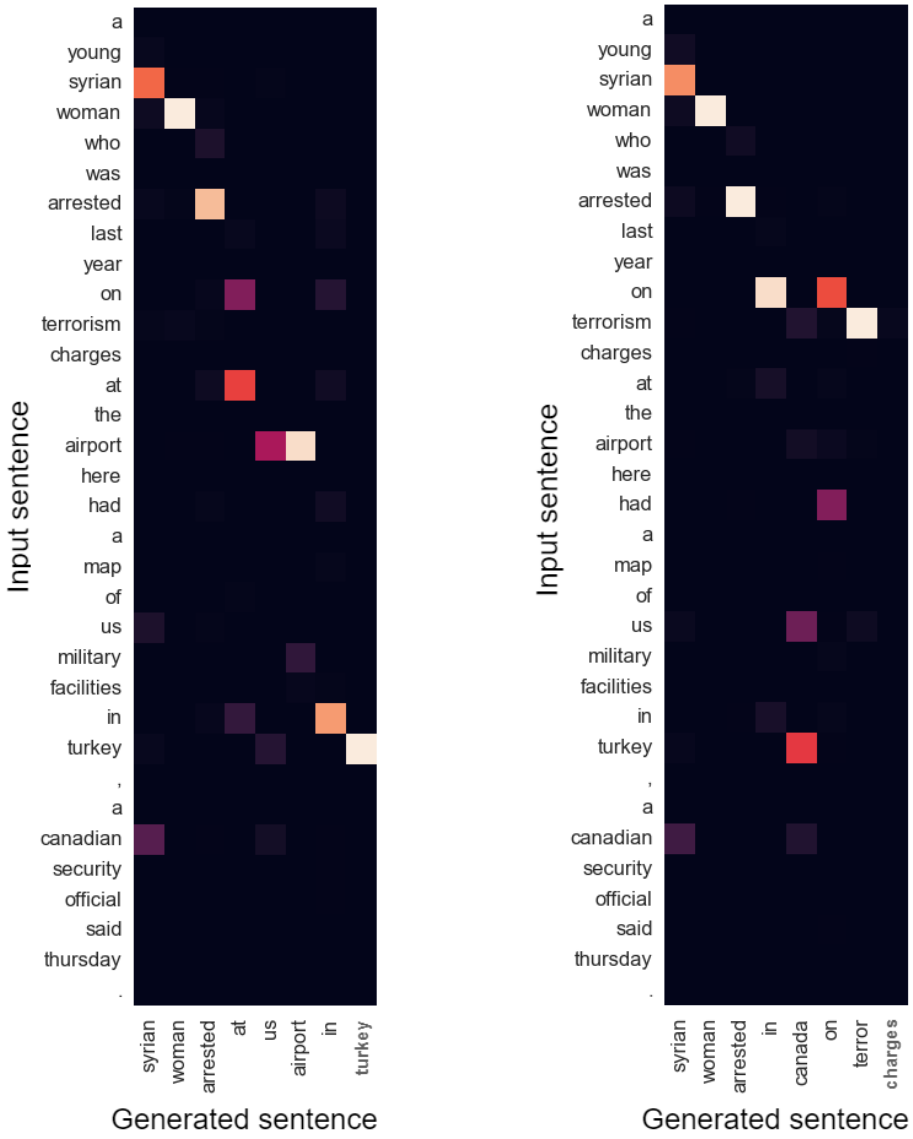


(a) M1 (b) M2

Figure A.3: Case 1: attention distribution between the input text and generated summary from the Gigaword dataset for different models.

A.2.4 Gigaword Dataset. Case 2

The heatmap of attention on input text and annotated summary of a model trained on the Gigaword dataset for case 2.



(a) M1 (b) M2

Figure A.4: Case 2: attention distribution between the input text and generated summary from the Gigaword dataset for different models.

A.3 RNN/LSTM/Seq2Seq

A.3.1 Recurrent Neural Network (RNN)

In human language, we have connections between words or phrases and without those connections, it would be impossible to understand what the text is about. When people talk, they use information that was used previously in the conversation to understand the current information. A simple neural network cannot memorize information, however, recurrent neural network (RNN) [17] can. RNN is used in a variety of tasks, such as machine translation, speech recognition or text summarization. Each RNN is connected to the other so that information from the previous context can be forwarded further.

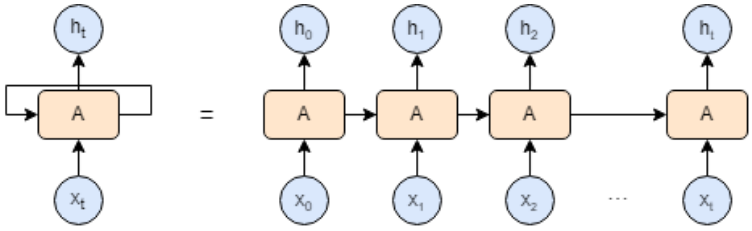


Figure A.5: Recurrent neural network (RNN), where x_t represents an input word at a given time step t and h_t represents an output word at a given time step t.

However, in longer phrases, RNN can perform poorly, as the meaning of the word should be passed through several iterations. A long short-term memory network solves some problems of an RNN network.

A.3.2 Long Short-term Memory (LSTM)

Long short-term memory (LSTM) [18] network is similar to RNN, but it can memorize longer dependencies. LSTM uses a series of gates (forget, input and output), that control which information will be saved and which one will be forgotten (Fig. A.7). The forget gate decides using a sigmoid function on the importance of each cell state based on the hidden state and input data. The input gate (memory) decides what information should be added to the network, given the input data and the previous hidden state (using tanh and sigmoid function multiplication which is later added to the hidden layer). The output gate uses information from the input data and previous hidden state to decide a new hidden state of the model (sigmoid multiplied by tanh).

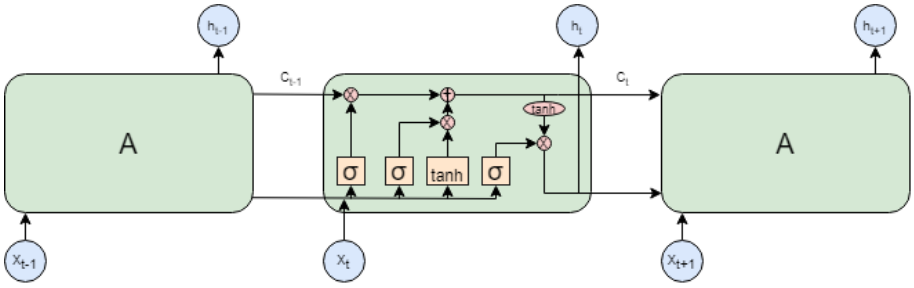


Figure A.6: Long short-term memory (LSTM) network. σ is a sigmoid function when tanh is a tanh function. x_t represents the input data at the position t, h_t represents the output data at the position t and C_t represents the previous hidden layer at the position t.

LSTM uses only the previous context of a given word. However, the meaning of the word might not only depends on what was in the context before the current word but in the future context as well.

A.3.3 Bi-LSTM

Bidirectional LSTM (Bi-LSTM) is used to attend the future context of a given word. The input goes in two ways, so at any given point it is possible to get the context information both from the past and the future information.

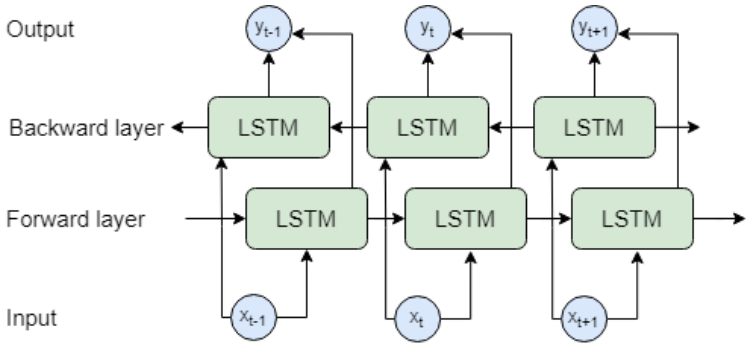


Figure A.7: Bidirectional LSTM (Bi-LSTM) network. x_t represents the input data at the position t, y_t represents the output data at the position t.

A.3.4 Sequence-to-sequence (Seq2Seq) Model

One of the models that is used for the text summarization task is called the Sequence-to-sequence model. *Sequence-to-sequence* (*Seq2Seq*) [22] is a model that takes an

input as a sequence and returns an output as a sequence. In text summarization, the Seq2Seq model consists of an encoder-decoder structure. The encoder tries to capture the context of an input text and outputs the hidden state vector, which is passed to the decoder, which uses this information to produce the output. A sample of an input text and annotated/generated summary are shown in Fig. A.8. Seq2Seq model uses either Bi-LSTM or LSTM inside the encoder and a decoder.

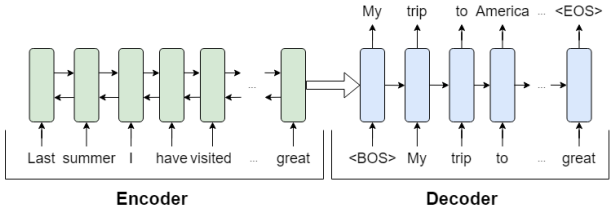


Figure A.8: A sample (taken from Table 2.2) of how Seq2Seq model works. The encoder (which consists of RNNs or LSTMs) takes the input text, while the decoder (which also consists of RNNs or LSTMs) takes the annotated summary and outputs generated summary.

The problem with such architecture is that the encoder should compress the entire sentence into a small hidden vector and the meaning of words has to traverse through the context in the encoder and then through the hidden layers of the decoder without losing its meaning, which is hard to do if the sentence is long. Therefore, the attention mechanism is used.

A.3.5 Seq2Seq with Attention

Seq2Seq with attention (Fig. A.9) during the decoding stage can look to the attention distribution of an input text to decide which word should be generated next.

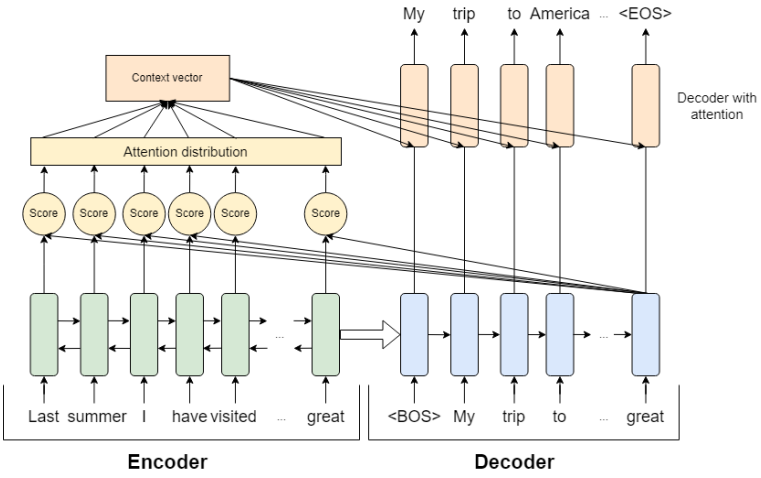


Figure A.9: A sample (taken from Table 2.2) of how Seq2Seq model works. The encoder (which consists of RNNs or LSTMs) takes the input text, while the decoder (which also consists of RNNs or LSTMs) takes the annotated summary and outputs generated summary.

Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[2] L. Clissa. Survey of big data sizes in 2021. *arXiv preprint arXiv:2202.07659*, 2022.

[3] A. A. Syed, F. L. Gaol, and T. Matsuo. A survey of the state-of-the-art models in neural abstractive text summarization. *IEEE Access*, 9:13248–13265, 2021.

[4] V. Dehru, P. K. Tiwari, G. Aggarwal, B. Joshi, and P. Kartik. Text summarization techniques and applications. *IOP Conference Series: Materials Science and Engineering*, 1099(1):012042, mar 2021.

[5] M. Wang, M. Wang, F. Yu, Y. Yang, J. Walker, and J Mostafa. A systematic review of automatic text summarization for biomedical literature and ehrs. *Journal of the American Medical Informatics Association : JAMIA*, 28(10):2287–2297, 2021.

[6] A. Kanapala, S. Pal, and R. Pamula. Text summarization from legal documents: A survey. *Artif. Intell. Rev.*, 51(3):371–402, mar 2019.

[7] S. Panthaplackel, A. Benton, and M. Dredze. Updated headline generation: Creating updated summaries for evolving news stories. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics Volume 1: Long Papers*, pages 6438–6461. Association for Computational Linguistics, May 2022.

[8] W. S. El-Kassas, C. R. Salama, A. A. Rafea, and H. K. Mohamed. Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*, 165:113679, 2021.

[9] A. Jangra, A. Jatowt, S. Saha, and M. Hasanuzzaman. A survey on multi-modal summarization. *arXiv preprint arXiv:2109.05199*, 2021.

[10] S. Verma and V. Nidhi. Extractive summarization using deep learning. *Research in Computing Science*, 147:107–117, 2018.

[11] Y. Zhang, J. E. Meng, and M. Pratama. Extractive document summarization based on convolutional neural networks. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, page 918–922, 2016.

[12] Ramesh Nallapati, B. Zhou, C. dos Santos, Ç. Gülçehre, and B. Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany, August 2016. Association for Computational Linguistics.

[13] A. M. Rush, S. Chopra, and J. Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[14] Y.-C. Chen and M. Bansal. Fast abstractive summarization with reinforce-selected sentence rewriting. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 675–686, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, page 2672–2680, 2014.

[16] H. P. Chanb and I. Kinga. A condense-then-select strategy for text summarization. *Knowledge-Based Systems*, 227:107235, 2021.

[17] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

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

[19] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[20] D. Bahdanau, K. Cho, , and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2015.

[21] Y. Kim, C. Denton, L. Hoang, and A. M. Rush. Structured attention networks. *arXiv preprint arXiv:1702.00887*, 2017.

[22] I. Sutskever, O. Vinyals, and Q. V Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[24] U. Khandelwal, K. Clark, D. Jurafsky, and L. Kaiser. Sample efficient text summarization using a single pre-trained transformer. *arXiv preprint arXiv:1905.08836*, 2019.

[25] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880. Association for Computational Linguistics, July 2020.

[26] J. Zhang, Y. Zhao, M. Saleh, and P. J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20, pages 11328–11339. JMLR.org, July 2020.

[27] W. A. Qader, M. M. Ameen, and B. I. Ahmed. An overview of bag of words;importance, implementation, applications, and challenges. In *2019 International Engineering Conference (IEC)*, pages 200–204, 2019.

[28] F. Almeida and G. Xexéo. Word embeddings: A survey. *arXiv preprint arXiv:1901.09069*, 2019.

[29] J. V. Lochter, Renato M. Silva, and Tiago A. Almeida. Deep learning models for representing out-of-vocabulary words. In Ricardo Cerri and Ronaldo C. Prati, editors, *Intelligent Systems*, pages 418–434, Cham, 2020. Springer International Publishing.

[30] M. Zhang, G. Zhou, W. Yu, N. Huang, and W Liu. A comprehensive survey of abstractive text summarization based on deep learning. *Computational intelligence and neuroscience*, page 7132226, 2022.

[31] A. See, P. J. Liu, and C. D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[32] K. Yao, L. Zhang, D. Du, T. Luo, L. Tao, and Y. Wu. Dual encoding for abstractive text summarization. *IEEE Transactions on Cybernetics*, 50(3):985–996, 2020.

[33] C. Gulcehre, S. Ahn, R. Nallapati, B. Zhou, and Y. Bengio. Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 140–149, Berlin, Germany, August 2016. Association for Computational Linguistics.

[34] S. Song, H. Huang, and T. Ruan. Abstractive text summarization using lstm-cnn based deep learning. *Multimedia Tools Appl.*, 78(1):857–875, jan 2019.

[35] T. Schick and H. Schütze. Rare words: A major problem for contextualized embeddings and howto fix it by attentive mimicking. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8766–8774, Apr. 2020.

[36] J. Deaton, A. Jacobs, K. Kenealy, and A. See. Transformers and pointer-generator networks for abstractive summarization. 2019. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/custom/15784595.pdf.

[37] S. Narayan, S. B. Cohen, and M. Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[38] D. Graff, J. Kong, K. Chen, and K. Maeda. English gigaword. *Linguistic Data Consortium, Philadelphia*, 4(1):34, 2003.

[39] P. Over, H. Dang, and D. Harman. Duc in context. *Information Processing and Management*, 43(6):1506–1520, 2007. Text Summarization.

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

[41] A. Toosi, A. Bottino, B. Saboury, E. L. Siegel, and A. Rahmim. A brief history of ai: how to prevent another winter (a critical review). *PET clinics*, 16 4:449–469, 2021.

[42] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*, 2013.

[43] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951.

[44] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[45] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. 12(null):2121–2159, jul 2011.

[46] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way toprevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[48] K. He, X. Zhang, and S. Ren et al. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 770–778, 2016.

[49] M.F. Medress, F.S. Cooper, J.W. Forgie, C.C. Green, D.H. Klatt, M.H. O'Malley, E.P. Neuburg, A. Newell, D.R. Reddy, B. Ritea, J.E. Shoup-Hummel, D.E. Walker, and W.A. Woods. Speech understanding systems: Report of a steering committee. *Artificial Intelligence*, 9(3):307–316, 1977.

[50] J. Kasai, K. Sakaguchi, R. Le Bras, D. Radev, Y. Cho, and N. A. Smith. Beam decoding with controlled patience. *arXiv preprint arXiv:2204.05424*, 2022.

[51] S. Gehrmann, Y. Deng, and A. Rush. Bottom-up abstractive summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4098–4109, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[52] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. R. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. S. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[53] Z. Yu, Z. Wu, H. Zheng, Z. XuanYuan, J. Fong, and W. Su. Lenatten: An effective length controlling unit for text summarization. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 363–370. Association for Computational Linguistics, August 2021.

[54] Y. Gong and X. Liu. Generic text summarization using relevance measure and latent semantic analysis. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, page 19–25, New York, NY, USA, 2001. Association for Computing Machinery.

[55] G. Erkan and D. R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22(1):457–479, December 2004.

[56] F. Kyoomarsi, H. Khosravi, E. Eslami, P. K. Dehkordy, and A. Tajoddin. Optimizing text summarization based on fuzzy logic. In *Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008)*, pages 347–352, 2008.

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

[58] K. Sparck Jones. *A Statistical Interpretation of Term Specificity and Its Application in Retrieval*, page 132–142. Taylor Graham Publishing, GBR, 1988.

[59] R. Mihalcea and P. Tarau. TextRank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 404–411, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[60] W. Chen, K. Ramos, K. N. Mullaguri, and A. S. Wu. Genetic algorithms for extractive summarization. *arXiv preprint arXiv:2105.02365*, 2021.

[61] K. Kadriua and M. Obradovic. Extractive approach for text summarization using graphs. *arXiv preprint arXiv:2106.10955*, 2021.

[62] Waseemullah, Z. Fatima, S. Zardari, M. Fahim, M. A. Siddiqui, Ag. A. Ag. Ibrahim, K. Nisar, and L. F. Naz. A novel approach for semantic extractive text summarization. *Applied Science*, 12(9):4479–4514, 2022.

[63] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradientbased learning applied to document recognition. *IEEE*, 86(11):2278–2324, 1998.

[64] S. Chopra, M. Auli, and A. M. Rush. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98, San Diego, California, June 2016. Association for Computational Linguistics.

[65] Z. Tu, Z. Lu, Y. Liu, X. Liu, and H. Li. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 76–85, Berlin, Germany, August 2016. Association for Computational Linguistics.

[66] X. Li S. Chakraborty and and S. Chakraborty. A more abstractive summarization model. *arXiv preprint arXiv:2002.10959*, 2020.

[67] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

[68] J. Lin, X. Sun, S. Ma, and Q. Su. Global encoding for abstractive summarization. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 163–169, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[69] B. Hu, Q. Chen, and F. Zhu. LCSTS: A large scale Chinese short text summarization dataset. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1967–1972, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[70] N. Prabhu and K. Kann. Making a point: Pointer-generator transformers for disjoint vocabularies. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 85–92, Online, December 2020. Association for Computational Linguistics.

[71] L. Liu, Y. Lu, M. Yang, Q. Qu, J. Zhu, and H. Li. Generative adversarial network for abstractive text summarization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018.

[72] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddingsusing siamese bert-networks. In *EMNLP/IJCNLP (1)*, pages 3980–3990, Hong Kong, China, November 2019. Association for Computational Linguistics.

[73] Y. Liu and M. Lapata. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*, 2019.

[74] W. Qi, Y. Yan, Y. Gong, D. Liu, N. Duan, J. Chen, R. Zhang, and M. Zhou. ProphetNet: Predicting future n-gram for sequence-to-SequencePre-training. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2401–2410, Online, November 2020. Association for Computational Linguistics.

[75] H. Bao, L. Dong, W. Wang, N. Yang, and F. Wei. s2s-ft: Fine-tuning pretrained transformer encoders for sequence-to-sequence learning. *arXiv preprint arXiv:2110.13640*, 2021.

[76] W. Xiao, I. Beltagy, G. Carenini, and A. Cohan. PRIMERA: Pyramid-based masked sentence pre-training for multi-document summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5245–5263, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[77] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[78] T. Schick and H. Schütze. Rare words: A major problem for contextualized embeddings and howtofix it by attentive mimicking. volume 34, pages 8766–8774, April 2020.

[79] K. Krishna, J. Bigham, and Z. C. Lipton. Does pretraining for summarization require knowledge transfer? In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3178–3189, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[80] K. Papineni, S. Roukos, T. Ward, , and W.-J. Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics.

[81] S. Banerjee and A. Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[82] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 2nd edition, 1979.

[83] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. In *NIPS*, pages 1693–1701, 2015.