CHALMERS
UNIVERSITY OF TECHNOLOGY

Using pre-trained language models for extractive text summarisation of academic papers

Master's thesis in Computer Science and Engineering

ERIK HERMANSSON
CHARLOTTE BODDIEN

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Using pre-trained language models for extractive text summarisation of academic papers

ERIK HERMANSSON
CHARLOTTE BODDIEN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Safety
Chalmers University of Technology
Gothenburg, Sweden 2020

Using pre-trained language models for extractive text summarisation of academic papers
ERIK HERMANSSON
CHARLOTTE BODDIEN

© ERIK HERMANSSON, CHARLOTTE BODDIEN, 2020.

Supervisor: Selpi, Department of Mechanics and Maritime Sciences
Examiner: Selpi, Department of Mechanics and Maritime Sciences

Master's Thesis 2020:01
Department of Mechanics and Maritime Sciences
Division of Vehicle Safety
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Printed by Department of Mechanics and Maritime Science
Gothenburg, Sweden 2020

Using pre-trained language models for extractive text summarisation of academic papers
ERIK HERMANSSON
CHARLOTTE BODDIEN
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

Given the overwhelming amount of textual information on the internet and elsewhere, the automatic creation of high-quality summaries is becoming increasingly important. With the development of neural networks and pre-trained models within natural language processing in recent years, such as BERT and its derivatives, the field of automatic text summarisation has seen a lot of progress. These models have been pre-trained on large amounts of data for general knowledge and are then fine-tuned for specific tasks. Datasets are a limiting factor for training summarisation models, as they require a large amount of manual summaries to be created. Most of the current summarisation models have been trained and evaluated using textual data mostly from the news domain. However, pre-trained models, fine-tuned on data from the news domain, could potentially also be able to generalize and perform well on other data as well.

The main objective of this thesis is to investigate the suitability of several pre-trained language models for automatic text summarisation. The chosen models were fine-tuned on readily available news data, and evaluated on a very different dataset of academic texts to determine their ability to generalise.

There were only slight differences between the models on the news data. But more interestingly, the results on the academic texts showed significant differences between the models. The results indicate that the more robustly pre-trained models are able to generalise better and according to the metrics perform quite well. However, human evaluation puts this into question, showing that even the high-scoring summaries did not necessarily read well. This highlights the need for better evaluation methods and metrics.

Keywords: natural language processing, nlp, machine learning, deep learning, automatic text summarisation, extractive summarisation, transformer, bert, roberta, xlnet

Acknowledgements

We would like to thank our supervisor Selpi for her ongoing support and guidance during the project.

For providing computational resources enabling the work in this thesis and support we would like to thank Chalmers Centre for Computational Science and Engineering (C3SE) provided by the Swedish National Infrastructure for Computing (SNIC).

We thank SAFER for providing access to their facilities.

Erik Hermansson, Charlotte Boddien, Gothenburg, February 2020

Contents

List of Figures

List of Tables

1
Introduction

When trying to investigate a topic, researchers nowadays have access to huge amounts of data - news articles, papers, blog posts, etc. However, due to the sheer volume of data, surveying it for relevant information can be a difficult task. Even when summaries or abstracts are provided, as is often the case with papers, they do not necessarily constitute a good summary of the actual document. Instead, many of them are written with the intention of enticing the reader to read the rest, not with the intention of providing them with all the most interesting information up front. Reading the full documents, however, can be an impossible task, and reading only parts of them selectively risks missing out on important pieces of information.

Ideally, what a researcher in this position would want is for someone to sort through the material and provide them with summaries comprised of all the most important bits of the documents. These summaries can then help them to get a good overview over the topic(s) and to decide which documents to read in full. Since producing such summaries is a very time-intensive task when done by humans, many attempts have been made over the past decades to automate this process, especially in the news domain. In this thesis, we aim to investigate the current state-of-the-art methods and apply them to the summarisation of academic papers from the field of traffic safety.

Natural Language Processing (NLP) in general and automatic text summarisation in particular are still very active fields of research with new papers and models being released all the time. Most of the current state-of-the-art models for various NLP tasks are still very new and have not been tested very extensively on the task of text summarisation or on data sets comprised of anything else than relatively short news articles. This is what we wish to investigate in this thesis.

1.1 Objective and Scope

The main objective of this thesis is to investigate the suitability of several pre-trained language models for automatic text summarisation. For this purpose we will investigate several sub-tasks:

- As many different models exist, we will compare how the chosen models perform against each other for the purpose of extractive text summarisation.
- Since few datasets for summarisation exist outside the news domain, and creating one is out of scope of this thesis, we will instead investigate how well a

model fine-tuned on a dataset consisting of news articles is able to generalize and perform on the very different academic texts.
- To properly evaluate the models, adequate evaluation methods are required. Currently used evaluation methods have limitations, for example requiring exact word matching. Therefore, we will investigate a method based on sentence similarity. We will employ this and the current standard metrics to evaluate the generated summaries and compare results against human judgement of the summaries' quality.

The scope of this thesis project is limited in the following ways:
1. Developing a new method for text summarisation or creating a new model for text representation is out of scope for this project.
2. Development of a complete method to automatically pre-process scientific documents (e.g., from PDF to clean text) for summarisation is outside the scope of this project.
3. No graphical interface for any end-user will be created. This thesis focuses solely on the scientific investigation of methods for automatic text summarisation, not on the development of a finished software product.
4. The aim is to produce summaries of individual scientific papers from the field of traffic safety. Multi-document summaries (producing a single summary combining the information from several source documents) are out of scope for this project.

1.2 Outline

The rest of the thesis is structured as follows: In Chapter 2 we will provide an overview over the current state of the art in the fields of NLP and automatic text summarisation. We will offer definitions and descriptions of the most important concepts and methods in NLP and will in particular introduce the most promising approaches to the automatic creation of text summaries and to their evaluation. Chapter 3 will detail how we performed our experiments. We will describe the implementations of the models and methods we used, how we obtained our training, test and evaluation data, and what kind of experiments we performed. The results of those experiments will be presented and discussed in Chapter 4. In Chapter 5 we will look at some ethical considerations regarding automatic text summarisation in general, review our most important findings with respect to the limitations of our project, and suggest possible directions for future work.

2
Theory

Automatic text summarisation is part of a research area called natural language processing (NLP). In the following, different approaches to text summarisation and evaluation of text summarisation will be presented in Sections 2.1 and 2.2, respectively. In Sections 2.3 and 2.4, the two important concepts of text embedding and language modeling will be explained. Section 2.5 follows the development of increasingly powerful machine learning models that have been developed for various NLP tasks and are useful for automatic text summarisation.

2.1 Automatic Text Summarisation

Automatic text summarisation techniques can be divided into two different approaches: extractive and abstractive. As described by See in [1], extractive summarisation techniques produce summaries by directly picking a subset of relevant sentences/phrases/words from the source document(s). Abstractive methods generate summaries using a separate vocabulary, rebuilding each sentence from scratch thus allowing the summary to contain words that don't necessarily appear in the source documents. This has the potential to result in a more cohesive text, but can also distort facts. Abstractive methods require encoding a much deeper level of understanding of natural language to be successful. Most research so far has been focused on extractive methods, as they don't require the machine to "understand" the text semantically and are easier to implement. For the above reasons, and to be able to draw on this wealth of available research, this thesis will be focusing on extractive summarisation.

2.1.1 Extractive Summarisation Methods

The summaries produced by the extractive method are a subset of the sentences of the source document(s). In the following subsections, we will look at two main approaches to this task: Score and Select, and Sequence Labeling.

2.1.1.1 Score and Select

For the score and select method summarisation is treated as a problem of assigning each sentence a score which captures how important it is to include in the summary. Then, the n highest-ranking sentences are selected to form the summary. Alternatively, the sentence selection can be approached as an optimisation problem: Importance and coherence are to be maximised, redundancy minimised.

2.1.1.2 Sequence Labeling

Another way to model the extractive summarisation task is to treat it as a binary classification problem: Each sentence needs to be labeled either as a summary sentence (which will be included in the summary) or a non-summary sentence (which will not be). Usually, a neural network (see Section 2.5.1) is trained for this task, using training data of texts and their sentence-labels. Many current state-of-the-art methods are using this model, like BertSum as introduced by Liu [2], which will be described in Section 2.5.4.1.

2.2 Summarisation Evaluation

Automatically creating summaries is a difficult in itself, and evaluating the quality of these summaries is another non-trivial task.
Evaluation of summaries can be done manually: People may read at least a sizable part of the documents as well as the produced summary and subjectively judge how well it summarises the documents. Another often used human-judgement metric is how well the given summary can be used to answers certain queries. However, there can be a significant amount of variation in how different people judge the quality of the same summary. Formulating objective quality measures and automating the evaluation process can help with this problem. Being able to obtain an exactly defined score for each summary, makes it possible to better compare the results of different summarisation approaches with one another. According to Allahyari et al. [3], the most widely used metric for automatic evaluation is ROUGE (Recall-Oriented Understudy for Gisting Evaluation). This metric can be used to compare the produced summaries against a set of typically manually created reference summaries according to certain criteria.

2.2.1 ROUGE

ROUGE, introduced in 2004 by Lin [4], is a set of measures that can be used for evaluation of text summaries by comparing them to other *reference summaries* assumed to be ideal. The comparison is performed by evaluating the overlap of text units, such as single words, word pairs (bi-grams) or word sequences, between the summary to be evaluated (also called the *candidate summary*) and the reference summary. Using ROUGE, the evaluation of summaries can be completely automated, which both saves time and allows for a more objective measure to compare different summarisation methods against each other.
In [4], Lin introduces five different ROUGE measures: ROUGE-N, ROUGE-L, ROUGE-W, ROUGE-S and ROUGE-SU. Each of them will be briefly described in the next few paragraphs.

ROUGE-N is defined by the overlap of *n-grams* between the candidate summary and the reference summary. An n-gram is a sequence of n adjacent words. In particular, the two commonly used measure ROUGE-1 and ROUGE-2 refer to the overlap

of single words and the overlap of bi-grams respectively.

ROUGE-L refers to the Longest Common Sequence (LCS, see [4]). Its score is based on the longest matching sequence of words that can be found between the candidate summary sentences and the reference summary sentences. The total ROUGE-L-score of the summaries is computed from the LCS-scores of the individual candidate-reference sentence pairs.

ROUGE-W is a weighted variant of ROUGE-L that favours consecutive LCSs.

ROUGE-S is similar to ROUGE-N, but measures the co-occurence of *skip-bigrams* rather than n-grams. Skip-bigrams allow for arbitrary gaps between the two words of the bigram. For example, the sentence “I had lunch today.” contains the following skip-bigrams: “I had”, “I lunch”, “I today”, “had lunch”, “had today” and “lunch today”. Note that the order in which the words appear matters.

The last one is **ROUGE-SU**. This is an extension of ROUGE-S, and additionally takes unigrams into account. Using ROUGE-S, there would be no skip bigram match between the two sentences “I had lunch today” and “today lunch had I”, as the second sentence is the exact reverse of the first. With ROUGE-SU, however, we get four unigram matches. ROUGE-SU can be obtained from ROUGE-S by adding a begin-of-sentence token at the beginning of each candidate and reference sentence. For the example above, this would give us the two sentences: "[START] I had lunch today" and "[START] today lunch had I", which would give us the following ROUGE-S matches between them: "[START] I", "[START] had", "[START] lunch" and "[START] today".

Lin [4] concludes that the ROUGE-scores correlate well with human judgement for single-document summarisations, but less well for multi-document summarisation. ROUGE-1, ROUGE-2, ROUGE-S4, ROUGE-S9, ROUGE-SU4 and ROUGE-SU9, however, performed “reasonably well when stopwords were excluded from matching” [4]. Correlations with human judgement can be further increased by using multiple reference summaries per document.

ROUGE, in particular ROUGE-1, ROUGE-2 and ROUGE-L, has been used widely in recent papers on automatic summarisation techniques such as [5], [2] and [6] to evaluate their performance. These evaluations are very commonly done on the CNN/Daily Mail dataset [7]. This is because it has long been one of the biggest available datasets of texts and reference summaries, and because evaluating different methods on the same dataset facilitates easy comparison between them.

2.2.2 Other evaluation metrics

2.2.2.1 BLEU

Another once very commonly used metric, originally developed by Papineni et al. to evaluate machine translations, is BLEU (bilingual evaluation understudy, [8]). It was used to evaluate machine-generated text against human-generated text. Subsequent papers, like [9] and [10], however, have called the usefulness of BLEU for anything other than the evaluation of machine translations into question. In our research, we have not come across BLEU being used for evaluation of text summarisation today, which is why we will not discuss it here any further.

Steinberger and Ježek [11] give a good overview over various approaches to text summarisation evaluation. In the following, we want to mention in particular: precision, recall and F-score, and cosine similarity.

2.2.2.2 Precision, Recall and F-Score

As extractive summarisation is essentially a binary classification problem, we can use precision and recall to evaluate generated summaries against extractive reference summaries in the following way:

Precision (P) is the number of sentences in the generated summary that are also in the reference summary divided by the number of all the sentences in the generated summary.

Recall (R) is the number of sentences in the generated summary that are also in the reference summary divided by the number of all the sentences in the reference summary.
It is worth noting that a very high precision score can be achieved by a summary that includes only a single sentence, as long as that sentence is also in the reference summary. High recall, on the other hand, can be achieved by a generated summary that contains a multitude of irrelevant sentences, as long as it also includes many of the sentences in the reference summary.

In the case of extractive summarisation, the optimal summary is likely to lie somewhere in between those two extremes. A more useful metric than precision or recall alone is therefore the **F-score**, which combines the two:

$$F = \frac{(\beta^2 + 1) \cdot P \cdot R}{\beta^2 \cdot P + R} \tag{2.1}$$

where β is a weighting parameter that favours precision when chosen greater than 1 and recall when smaller than 1.

2.2.2.3 Cosine Similarity

Cosine similarity is a measure for the similarity of two vectors. If the candidate and reference summaries have been converted into vector space, where similar summaries

are closer to each other (more on how this is done can be read in Section 2.3), then cosine similarity can be used to determine how similar two summaries are by performing the following measure to compare them:

$$cos(X, Y) = \frac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i {x_i}^2} \cdot \sqrt{\sum_i {y_i}^2}} \tag{2.2}$$

2.3 Text Embedding

2.3.1 Word Embeddings

For a computer to be able to work with text, we need to first convert that text into numerical input, usually into vectors. This process is called *text embedding* and can be done on word, sentence or document level. The input text itself is treated as a sequence of tokens, which usually are either words or sub-word entities (like "play" and "#ing" for the word "playing").
Ideally, these embeddings are not just arbitrary, but capture some syntactic and semantic information. The goal is to embed the words in such a way that words with similar meanings are embedded similarly, meaning they are close together in the vector space. The linguist Zellig Harris noted already in 1956 [12] that words that appear in similar contexts tend to have similar meanings, which is known today as the *distributional hypothesis*. This hypothesis implies that, with a large enough corpus of text to train on, word embeddings can be learned unsupervised by neural networks, simply by observing the contexts (other words) they often appear in.

A very commonly used word embedding method applying the distributional hypothesis with very good results is word2vec [13]. The embeddings the algorithm produces clearly capture semantic properties of words, as the following example, taken from [13], illustrates:

> "vector(”King”) - vector(”Man”) + vector(”Woman”) results in a vector that is closest to the vector representation of the word Queen"

However, one of the drawbacks of word2vec and similar embedding methods like Glove [14] is that they embed each word only as a single, fixed vector, regardless of the specific context it appears in. For example, in the sentence “I lost my cell phone in the prison cell”. The word vector for “cell” would be the same in both occurrences - capturing some mixture of all the different meanings that “cell” can take on. Recently, new approaches for text embedding that utilize deep learning and attention have improved on this, like BERT (see 2.5.3.1 for details) and XLNet (see 2.5.3.4 for details). The concept of attention will be described in Section 2.5.2.7.
Rather than returning fixed vectors for each word, these models, after training, can be used to obtain context-specific word vectors. In our previous example, the two occurences of “cells” would be represented as two different vectors, one most likely being much closer to the vectors of words like “telephone” and “conversation” and the other being closer to words like “crime” and “punishment”.

2.3.2 Sentence Embedding

Sentence embeddings can be computed from the vectors of the words they are made of (the simplest approach being to take their average). Language models like BERT and XLNet can also be trained to produce sentence embeddings for specific tasks.

2.3.3 Document Embedding

Just like sentence embeddings can be obtained from aggregating word embeddings in some way, document embeddings can be obtained from the embeddings of the sentences they contain. Again, the simplest approach would be to simply average the sentence vectors.

Another method is Doc2Vec [15]. Doc2Vec, also called *Paragraph Vector*, is an unsupervised algorithm that extends the basic concept of Word2Vec to variable-length pieces of text. These may be single sentences or long documents. Doc2Vec learns fixed-length vector representations of these text pieces by trying to predict words in it. This method of document encoding is able to capture semantic information about the text unit it is given, much like Word2Vec is able to do that for words, and some researchers like Campr and Ježek [16] found it useful for the evaluation of automatic text summarisations. Dai et al. [17], too, investigated Doc2Vec's general usefulness for measuring the similarity of two texts and found that it performed better or on par with other methods of document embedding. They also found that vector operations can be performed on the vectors, much like with word2vec.

2.4 Language Modeling

Another important concept for the field of NLP is that of *Language Modeling*, which means representing a language as a probability distribution over sequences of words. Jozefowicz et al. [18] give a good overview over the developments in language modeling up to 2016. Ideally, a language model is able to capture both grammatical and semantic information, assigning high probabilities to sentences that are both grammatically correct and likely to appear in the context of the corpus, which is often limited to texts belonging to a certain topic, and low probabilities otherwise. Language models are used for many NLP tasks like speech recognition, machine translation and text summarisation.
In the past, RNNs (see section 2.5.2.1) were very commonly used to train such models. However, as of 2020, when this thesis was written, two of the most promising models for language modeling are BERT [19] and XLNet [20]; both employ the Transformer architecture (as described in section 2.5.2.8). In the following sections, these models will be described in more detail.

2.5 Machine Learning Models for NLP

In more recent years, machine learning, in particular neural networks, have enabled great progress in NLP in general and automatic text summarisation in particular.

In the following, we will trace the developments of these techniques. Section 2.5.1 gives an introduction to Artifical Neural Networks. In Section 2.5.2 networks for handling sequential data, such as text, are introduced from the early Reccurent Neural Networks 2.5.2.1 to the more recent Transformer 2.5.2.8. Section 2.5.3 introduces several language models. Section 2.5.4 introduces two task specific models using pre-trained models.

2.5.1 Artificial Neural Networks

Artificial Neural Networks (ANN, often also just referred to as neural networks, see [21] for a more detailed overview) are computing systems inspired by the biological neural network found in the brain. As a biological network consists of neurons, an artificial neural network is made up of artificial neurons (from here on just referred to as neurons), which are essentially functions. Each such neuron receives input and performs some computation on it, before passing on the result of that computation, multiplied by some weight, to one or more neurons of the next layer it is connected to, until the ones in the last layer produce the output of the network.

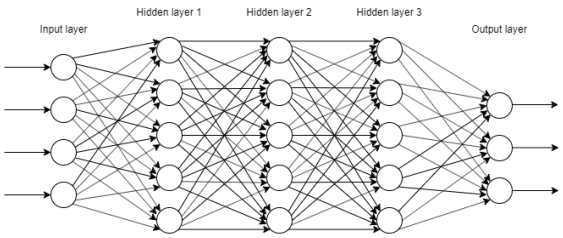


Figure 2.1: A simplified illustration of an ANN.

In order to perform a specific task, a neural network, once set up, needs to be trained. To do this, a set of training data is required, meaning a set of inputs with the corresponding outputs we want the network to produce. If the network does not produce the desired outputs, the weights of the neurons are adjusted through a process called *backpropagation*. For details on this process, the interested reader is referred to [21].

Each ANN model has an *objective function*, which captures the desired outcome. For example, the objective function of an ANN trying to guess a number correctly might be to minimize $|number_{guess} - number_{true}|$. A so-called *loss function* expresses how well the model fits the training data. The loss function depends on the parameters of the model (the weights). The aim of training is to find the parameters/weights that will minimize the loss function. This is done via a process called *Gradient Descent*. A gradient is the multi-dimensional equivalent of a function's derivative, which measures the slope of a function. This slope will be 0 for parameter values for which the function has a maximum/minimum. The aim of training is to find this minimum of the loss function, and therefore to find parameters for which the gradient will be 0. We "descend" the gradient until we hit its low-point of zero. Computing the gradient of a function produces a vector that points in the "uphill" direction of the gradient, which is why gradient descent happens in two steps:

1. Calculate the gradient.
2. Take a small "step" in the direction opposite to the gradient. (by adjusting the weights)

This is repeated until the gradient is close enough to zero. The step size is important: If it is too large, we risk "stepping over" the optimum in our adjustment step, never hitting it. If the step size is too small, however, reaching the optimum might take more time than we have available. We will encounter these problems in Section 2.5.2.4 in the form of the *exploding/vanishing gradient problem.*

In order to train an ANN robustly, multiple runs through the training data may be necessary. Such a run through all the training data is called an *epoch*. Due to the massive size that modern ANNs can reach and the memory requirements of training, many times it is not possible to train the ANN in entire epochs at a time. In this case, the training data is split into so-called *batches*, which are processed one by one. This also affects the gradient calculation since we only have access to a random subset of the data. In this case, a stochastic approximation is used, *stochastic gradient descent (SGD)*. After each batch has been processed, backpropagation is applied.

Neural Networks form the base for all the models that will be described in the following sections.

2.5.2 Sequential Models

For NLP tasks, input data often takes the form of sequences: A sentence, for example, is a sequence of words and a document is a sequence of sentences. Such sequences will vary in length, which poses a problem for the neural networks. By encoding the input, using Continuous Bag Of Word representations like [13], it is possible for neural networks to process such data. But this process is limited, as it does not take the order of words into consideration. The word order, however, can be very important for the meaning of a sentence: For example, “bad, not good” and “not bad, good” have very different meanings, even though they contain the exact same words. Convolutional Neural Network models, [22], would be able to represent such relations and dependencies, but are limited to only local ordering and have trouble with relations over large distances, such as a long sentence. This is because of the convolution process, which generally only covers a short range. For an example of why this is important, imagine a text describing somebody’s biography. The first sentence of this text might be something like: "XYZ was born in France." Then many other sentences may follow, which don’t refer to XYZ’s country of birth, until the last sentence: "XYZ returned to his country of birth and died there." In order to know what "his country of birth" refers to, it is important to remember the information from the beginning of the text. CNN models would struggle with this and potentially not be able to resolve that "his country of birth" and "France" refer to the same country. To model sequence dependencies over large distances, other models are required. One such model that is specifically designed to model dependencies between sequential inputs, is the so-called *Recurrent Neural Network*, described in the next section.

2.5.2.1 Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNN) can process data of varying length, while maintaining structured relations and dependencies. A RNN takes as input a sequence of vectors, each of which is processed in a step-by-step fashion, outputting a state vector which is used to pass on information to the next step. As more of the input is processed, the state vector gathers more information, better representing the sequence. Figure 2.2 illustrates the basic RNN architecture. For the first step an initial randomized state vector is used, for each subsequent step the previous state vector is used as input.

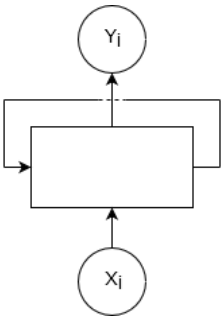


Figure 2.2: Single RNN cell.

If the length of the input sequence is known in advance, the network can be unrolled to display the full network, as illustrated in Figure 2.3 .

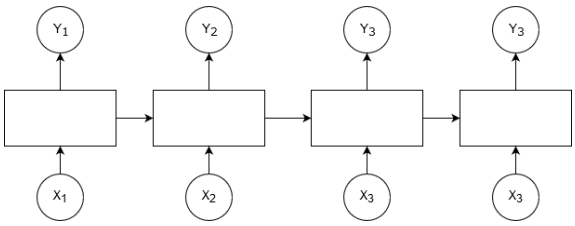


Figure 2.3: RNN unrolled over 4 steps.

When unrolled, it can be seen that the RNN is a deep neural network and can thus be trained like a feed-forward NN by backpropagation through time.

2.5.2.2 Stacked RNN

RNNs can be stacked [23], such that the output from one layer is used as the input to the next layer. This creates hierarchical structures, often called Deep RNNs. As Goldberg writes in [24], stacked RNNs often perform better on various NLP tasks but it is not theoretically clear as to why.

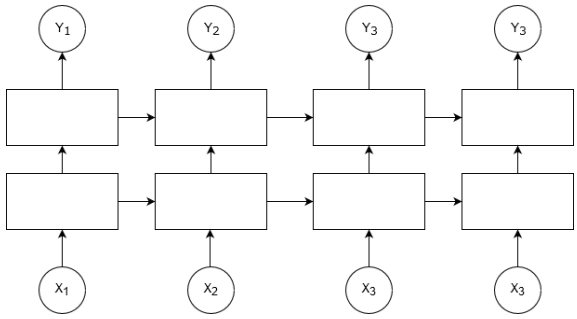


Figure 2.4: Stacked RNN.

2.5.2.3 Bidirectional RNN

An issue with RNNs is that they can only use past states for predictions, future states, however, might also contain useful information. Additionally when processing a sequence, later states will contain more information than earlier states, thus accuracy improves as more of the sequence is processed. Bidirectional RNNs attempt to solve this by utilizing two layers. Each layer processes the same inputs, but in opposite directions, i.e., one does so from front-to-back as can be seen in Figure 2.5, the other from back-to-front. The output of each step is a combination of that steps layers. This allows the network to use past and future states with more accumulated information.

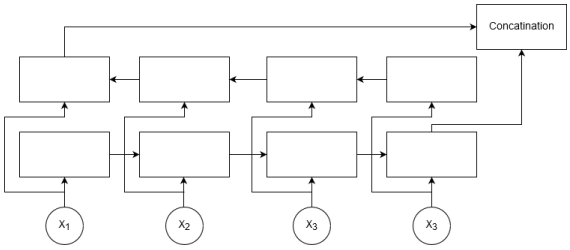


Figure 2.5: Bidirectional RNN.

2.5.2.4 Simple RNN

A simple version of RNN was proposed by Elman [25]. In this version, the state vector is simply the linear combination of the previous state and of the current input passed through a non-linear activation function. This simple architecture suffers from the *vanishing/exploding gradient problem (EVGP)* Hanin [26] explains how and under which circumstances the EVGP occurs. It means that when the weights of the network are updated, the increment in which this is done is either too big and therefore too imprecise, or too small and therefore effectively meaningless. For the simple RNN, this happens especially when handling long sequences. Over long sequences information is lost and thus the ability to represent dependency is compromised.

2.5.2.5 Long Short Term Memory (LSTM)

The LSTM architecture was developed by Hochreiter et al. [27] to solve the vanishing gradient problem among others. The main addition is the use of a memory cell in combination with a number of "gates" that control it. The gates are values computed using the previous and current steps of the sequence. As each segment of a sequence is processed, the gates influence what should be added to the memory, what should be forgotten, and what the new output should be. The gating components allow for gradients to be passed through the memory cell over longer ranges.

2.5.2.6 RNN Modes

There are different modes for handling the outputs produced by a RNN:

Acceptor: The output is based on the information contained in the final state. For the example of sentence classification, a classification would be produced after all words in the sentence have been processed.

Encoder: The output is the final state vector, an encoding of the sequence into a single vector. Often used in combination with a decoder. For the sentence example, after all words have been processed, an encoder outputs an encoding of the sentence.

Transducer: The output is based on combined information of each step's state vector. For the sentence example, there would be a single output after each word has been processed.

Encoder/Decoder An Encoder-Decoder architecture is often used for sequence-to-sequence NLP tasks. RNN Encoder-Decoder was first introduced by Cho et al. [28]. The *encoder* encodes an input sequence into an intermediate vector representation, as described above. This vector is then used as the initial state for the *decoder*. A decoder is often autoregressive, meaning that it consumes its own output, using so far produced output as input in the next step. Using the example of translation, a sentence is given as input for the encoder, producing the intermediate vector. The decoder, with this vector as the initial state, works step-by-step producing and consuming the translated sentence word-by-word until it finds itself generating an end-token. This architecture is illustrated in Figure 2.6.

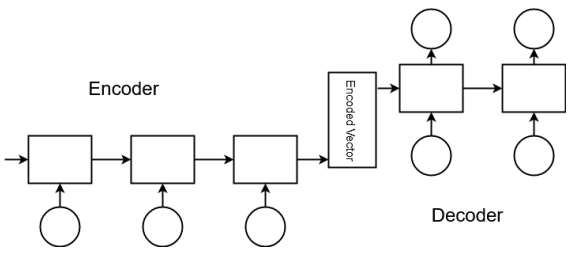


Figure 2.6: Encoder/Decoder model.

2.5.2.7 Attention

The problem with the Encoder/Decoder sequence-to-sequence model described in Section 2.5.2.6 is that it encodes the entire input sequence into a single, fixed-length context vector, which the decoder then uses to generate the output. In order to produce this vector, the input sequence is processed sequentially from beginning to end, and at each step only some of the information from the previous step is passed on. This means, that the final output of the encoder is much more influenced by the last couple of tokens than it is by the first. For very long sequences, this can lead to important information from the beginning of the sequence simply being "forgotten". Even LSTM can not fully solve this problem.

In order to alleviate this problem, Bahdanau et al. [29] suggest a new way of processing sequential data: Instead of using an encoder to produce a single context vector while discarding all the intermediary hidden states of the encoder, the authors propose to utilize all the encoder states. The goal of training such a model is then no longer to produce the one context vector that perfectly encodes the input sequence, but rather to learn which parts of the input sequence to pay attention to in order to generate each part of the output sequence. For illustration purposes, Figure 2.7 shows a machine translation example from [29]. It shows the attention that was paid to each French word of the input sequence to produce each word of the English output sequence. Note, for example, that in order to generate the English word "Syria", full attention was paid to both the French words "la" and "Syrie" and little to no attention to any of the other words in the sentence.

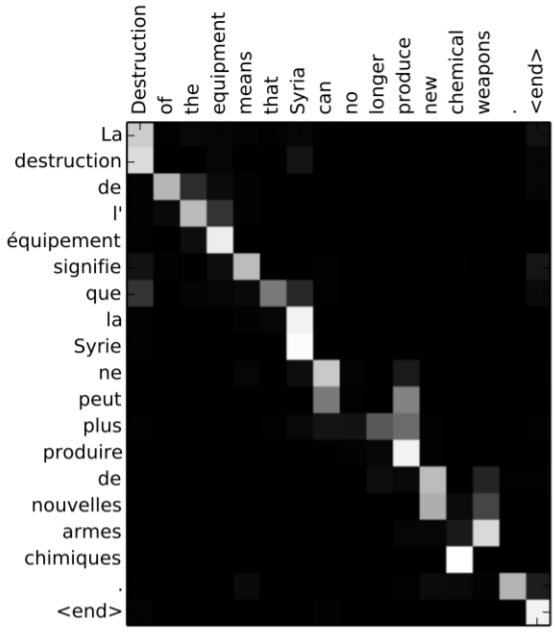


Figure 2.7: Translation of a French sentence into an English one. Figure taken from [29] with permission from the authors.

2.5.2.8 Transformer

Another architecture for sequence modeling is the *Transformer*, introduced by Vaswani et al. in the paper "Attention Is All You Need" [30]. This model follows the encoder/decoder structure introduced in 2.5.2.6, but as the title of the paper suggests, it relies primarily on attention.
It was developed to solve some of the issues with existing models, which were largely based on RNN (see section 2.5.2.1) and CNN [22]: Most prominently the problem of retaining information over many steps when encoding long input sequences, and the limited possibility of parallelization, since every step of the encoding and decoding requires the output of the previous step. Even when RNN models were enhanced by the addition of attention to alleviate the former problem, the problems with parallelization remained. CNN models, on the other hand, can be parallelized but suffer from an increased path length in the network as sequence length increases, which increases the amount of information that is potentially lost. The Transformer architecture discards the recurrent approach of sequence modelling and utilizes attention instead, as described in Section 2.5.2.7. Due to its non-sequential approach, this method is highly parallelizable while only requiring a constant, $O(1)$, number of operations and path lengths. This allows for faster training and better retention of information.

The basic architecture of the Transformer model can be seen in Figure 2.8 and will be described in the following paragraphs.

Encoder
The encoder of the transformer consists of 12 stacked encoder layers. Each such layer consists of two sub-layers, an attention layer and a feedforward network layer. The input to the first of these layers is a sequence of embeddings, very commonly word embeddings. Before these embeddings are passed to the model, they are supplemented with positional encoding to be able to retain the information of word order, despite no longer processing the input in a sequential manner. With the positional encoding, the same token at different positions is encoded differently, and their final embeddings will have some meaningful distances in vector space.

The attention sub-layers allow the system to focus on the most relevant parts of a sequence. During the encoding process this is used to determine how much a word relates to all other words in the sequence.
The attention mechanism used by the Transformer is called Multi-Head Self-attention: "Self", because it encodes each word of the sequence in relation to the other words in the sequence, and "multi-head" because each attention layer utilizes not just one, but several different attention weights ("attention heads") [30]. This means that multiple attention processes are performed in parallel. The idea is for each to focus on different aspects of the sequence.

The second sub-layer is a fully connected feed-forward neural network. This network is applied to each element of the sequence separately but identically.

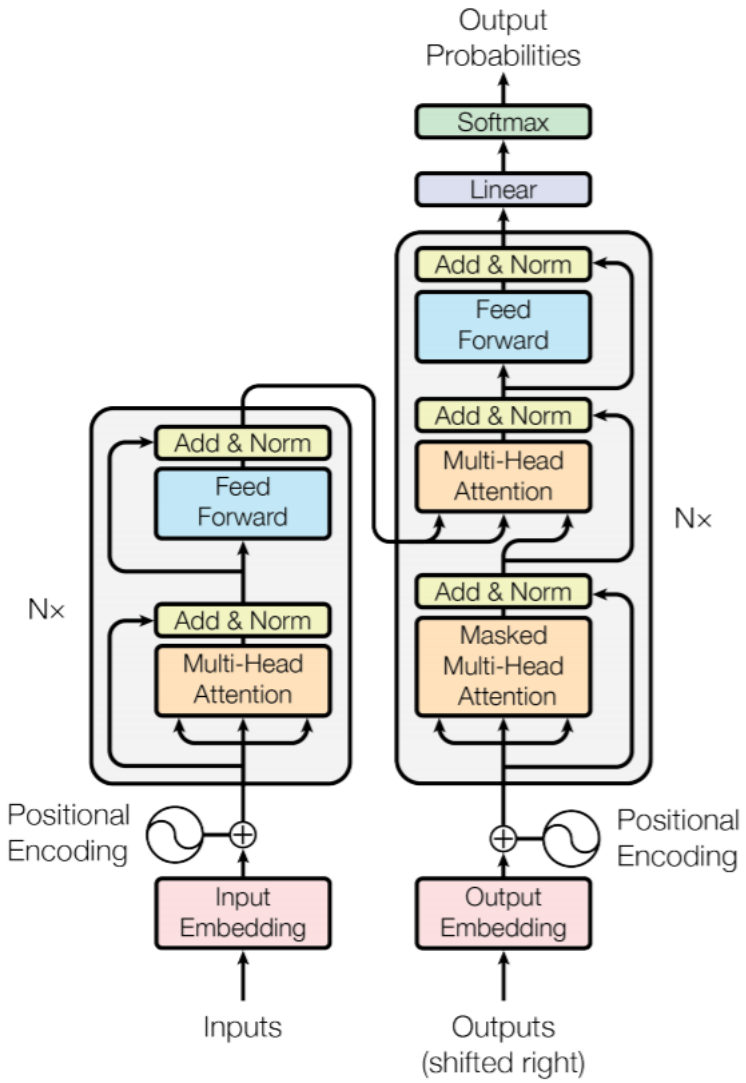


Figure 2.8: Transformer architecture. Taken from [30] with the authors' permission.

All sub-layers of the encoder also contain residual connections, their purpose is to combine the input, which has not been affected by the layers, with the output produced by the layers. In the Transformer architecture, these residual connections are used to restore positional encoding after processing the word embeddings. Without these connections the performance suffers greatly, as positional information gets lost.

Decoder
The decoder differs only slightly from the encoder. It contains the same two sub-layers, but has an additional sub-layer, an encoder-decoder attention layer, between them. The decoders role is to produce output, using the information produced by the encoder. In the Transformer it does this by performing multi-head attention over the output of the encoder and the so far produced output by the decoder. This differs from the other forms of attention in the model as it is no longer self-attention, instead it uses multiple sequences.

2.5.2.9 Transformer-XL

The Transformer model described above, as proposed by Vaswani et al. [30] handles the whole input at once, as such there must be some limit on the length of the input, due to computational and resource limitations. The default implementation uses a token limit of 512. This means the transformer can only consider any token's context in 512 token blocks. Solutions for longer texts have been suggested in later works, such as [31], where the longer corpus is split into multiple 512 length blocks. But this has two problems: Firstly, no contextual information is shared between blocks, and secondly, the splitting of the corpus is often done without any respect for sentence or semantic structure, leading to context fragmentation.

Transformer-XL [32] is an architecture proposed to solve the problem of fixed length contexts. Its main contribution is the re-introduction of recurrence to the Transformer, which allows context to flow between blocks of a split corpus. To achieve context flowing over the boundaries of blocks, the previous block's attention vectors are saved and can be used to "look back" on for context, resulting in better long-term dependency and avoiding the fragmentation problem. Applying this method to every two consecutive blocks creates a combined context that can represent context over much longer than just two blocks. The method could also be extended to allow for further connections, beyond just two blocks.

For this method to work, another type of positional encoding is required. Since the model looks back at previous blocks, the absolute positional encoding employed by the Transformer no longer works - since each token will be in multiple positions and tokens in different segments would be assigned the same positions. Instead a relative positional encoding based on the distance between tokens is used. The attention score, too, is calculated slightly differently from the Transformer.
The Transformer-XL model is able to generalize from training on short sequences to much longer sequences quite well. For example, Dai et al. detail in [32] how the model was trained with an attention length of 784 tokens and evaluated on corpus of 3,800 tokens and achieved a new state of the art result.

2.5.3 Pre-Trained Language Models

In the following, we will introduce several pre-trained models for NLP tasks that have been built using the Transformer/TransformerXL architectures: *BERT* (2.5.3.1) and *XLNet* (2.5.3.4) are built upon, respectively. Additionaly number of variations of BERT will be presented as well: We will elaborate on the models *RoBERTa* (Section 2.5.3.2), *DistilBert* (Section 2.5.3.3).

2.5.3.1 BERT

BERT is a Transformer-based model for language encoding, introduced in [19] by Devlin et al. The authors identify the fact that models could only be trained unidirectionally as one of the big limitations of previous approaches to language modeling. This meant that a token could only be encoded using the information of either the tokens to its left *or* the tokens to its right, but never using information from both combined at the same time. The objective in creating BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) was to create a model that could take the full context of a token into account, left and right.

In order to use BERT for some down-stream task (like machine translation or text summarisation), two steps are necessary:

1. The model needs to be pre-trained. This means the model is not yet trained in any task-specific way, but instead is taught to encode language itself in a sensible way. This is done so the same model can be used for several different down-stream tasks without needing to be trained from scratch. Task-specific training (fine-tuning) is done in the next step. Pre-training results in a general-purpose Transformer that can encode input tokens. This pre-training is done on unlabeled training data over two different training tasks, described later in this section.
2. The model can then, once it is initialized with the parameters obtained through pre-training, be fine-tuned to be used for a particular down-stream task. In order to do this, importantly, the architecture of the model itself does not need to be changed. Instead, the same pre-trained model can be applied to several different tasks, by layering task-specific layers on top and training the model on labeled training data pertaining to the desired downstream task.

Since the authors made their pre-trained BERT models (a larger and a smaller one) available for download and free to use, this means that with relatively little effort, these already pre-trained models can be applied to a wide variety of text-based tasks.

The architecture of the model itself is almost identical to the Transformer architecture described in section 2.5.2.8. Perhaps more interesting is how textual input is processed and how the model is (pre-)trained:

As input, BERT accepts textual sequences that may each be composed of either a single sentence or a pair of sentences, where a "sentence" means any arbitrary span of contiguous text, not necessarily sentences in the grammatical sense. Each

such sequence is preceded by a classification token ([CLS]). The final hidden state for this token can be trained to obtain an aggregated representation of the entire sequence. This is useful for some classification tasks, like summary/non-summary sentence classification for extractive summarisation.
If the sequence consists of two sentences, then they are separated by a [SEP] token. Additionally, BERT adds a so-called segment embedding to each token, which indicates whether it belongs to Sentence A or Sentence B. The input representation of each token is obtained by adding together the token's WordPiece embedding (see [33] for details), segment embedding and positional embedding. The latter encodes where in the sequence the token is located. This is necessary, as BERT, being a Transformer model, is not going through the tokens sequentially, and therefore does not "know" the order of the input tokens.
Figure 2.9 illustrates the BERT input representation.

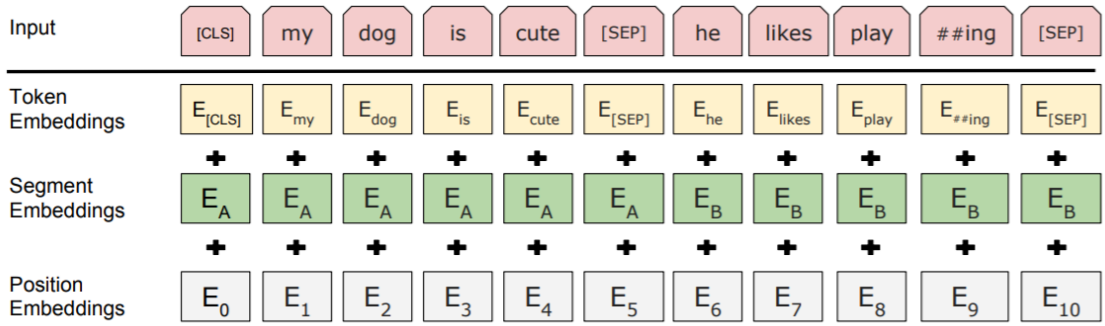


Figure 2.9: BERT input representation. Reproduced from [19] with the authors' permission.

Once the input representation is obtained, BERT is pre-trained on it by trying to solve two tasks, as mentioned above. These two tasks are the following:

1. **Masked Language Model (MLM)** Some percentage of the input tokens (in the paper the authors chose 15%) is masked and BERT is tasked with predicting them by using the entire context - left and right. Notably, only these masked tokens are predicted by the model, and no attempt is made to reconstruct the entire input.
2. **Next Sentence Prediction (NSP)** This task is meant to help the model learn the relationships between sentences: To create the pre-training dataset, for each training instance two sentences A and B are picked from the training corpus. With 50% probability, sentence B will be sentence A's successor, with 50% probability it will be a random sentence from anywhere else in the corpus. BERT is tasked with predicting (binary) if sentence B is indeed sentence A's successor.

For pre-training, the authors used BooksCorpus ([34]) and English Wikipedia texts. The pre-trained models $BERT_{LARGE}$ and $BERT_{BASE}$ are publically available at: `https://github.com/google-research/bert`

In the next two subsections, we will look at variants of BERT that aim to improve on the original model.

2.5.3.2 RoBERTa

In [35], the authors claim that BERT, as introduced in the original paper [19] is actually undertrained and show that with some modifications, significantly better results can be achieved, which are competitive with the performance of every model published after BERT. They name their modified BERT-version RoBERTa. (a **r**obustly **o**ptimized **BERT** **a**pproach)

Apart from changing some of BERT's hyperparameters, the main differences from BERT pertain to how the model is pretrained. The main changes from the training suggested in [19] are the following:

Dynamic Masking
In the original BERT, the masking of the input sequences is done in a static way: Only once in pre-processing. To ensure that BERT will encounter the same sequences with different masking patterns, the training data instances are multiplied by 10 before masking.
RoBERTa, however, applies dynamic masking instead. A new masking pattern is generated every time a sequence is fed to the model. This means that the model will encounter many more different masking patterns of the same instance. This in turn removes the need to drastically increase the number of training instances.

Full sentences
As opposed to BERT, RoBERTa uses exclusively full sentences as input to the model. Such sentences are sampled contiguously from the documents, such that the total length does not exceed 512 tokens. If document boundaries are crossed while sampling, a special inter-document separator token is inserted.

Training in large mini-batches
BERT was trained in 1 million training steps with a batch size of 256 sequences. RoBERTa, on the other hand, was trained in only 125.000 steps, using a much larger batch size of 2.000 sequences. The authors express uncertainty over whether they have already found the ideal batch size with this, but it produces better results than the original BERT while taking less time to train.

Larger byte-level BPE
BPE (byte-pair encoding) is a hybrid between character- and word-level text encoding. It bases its encodings on subword units, for example: Instead of encoding the word "playing" or each of its letters separately, BPE might encode "play" and "#ing", building blocks which can be re-used to form other words as well. This allows for a much larger vocabulary as would otherwise be possible. However, a lot of the time in BPE, a large portion of the encodings are encodings of single uni-code characters, which limits the total number of words that can be captured.
RoBERTa instead makes use of a variation of BPE introduced in [36], which is

based on bytes instead of characters. This means that less subword units are needed to encode a larger vocabulary. While BERT used character-level BPE with a size of 30.000 subword units and requires the input to be tokenized in preprocessing, RoBERTa requires no such preprocessing and uses byte-level BPE to encode 50.000 subwords.

Longer pretraining on larger data sets
RoBERTA was trained over 500.000 steps, while BERT over 100.000, and trained on 160GB of textual training data, resulting in much better end-task performance.

The pretrained RoBERTa model is publically available at: `https://github.com/pytorch/fairseq`

2.5.3.3 DistilBert

Pre-trained language models such as the ones described in the previous sections, can easily have several hundred million parameters. This means that they require great amounts of memory and computational power to be trained and run.

Sanh et al. **Distilbert** therefore set out to create a much smaller language model, which is less resource demanding. They did so using the method of knowledge distillation ([37], [38]), creating a much smaller Transformer model, which they called DistilBERT.
DistilBERT has the same general architecture as BERT (see section 2.5.3.1), but less layers. The authors also pre-trained a BERT model according to the best-practise suggestions of [35] (see the section on RoBERTa: 2.5.3.2). DistilBERT was then trained, using the same corpus of training data, to produce the same outputs as this BERT model. This method is also called Teacher-Student Knowledge Distillation, as the BERT model functions as a teacher, whose behaviour the DistilBERT model, the student, tries to replicate.
The resulting pre-trained DistilBERT model is only 40% the size of the original BERT and, according to the authors, 60% faster. Through experiments, Sanh et al. showed that despite being so much smaller, DistilBERT retains 97% of BERT's language understanding capabilities, as measured by its performance on various language understanding tasks.
DistilBERT is openly available in the Transformers library from HuggingFace. [1]

2.5.3.4 XLNet

XLNet takes a slightly different approach to language modeling than BERT and its variants. It is an autoregressive model for capturing bidirectional dependencies, which the authors developed to solve some perceived problems of BERT. Firstly, the method of masking words and then predicting them corrupts the input, creating texts filled with [MASK] tokens that are not seen in regular texts. This causes a discrepancy between pre-training and fine-tuning, since the latter does not contain any

[1] https://github.com/huggingface/transformers

[MASK] tokens. Secondly, BERT assumes that masked tokens are independent, but for a sentence containing multiple masked tokens, these may in fact be dependent on each other. BERT also has a fixed length context of 512 tokens, while XLNet builds on the Transformer-XL architecture to be more suitable for longer texts.

Despite its problems, BERT is still very good at capturing bidirectional context which is what lead to gains against previous models. XLNet captures bidirectional dependencies slightly differently, by utilizing language-permutation modeling, which works by making predictions in a random pattern. Given a sentence of 5 words, for instance, the model could be asked to predict the word in the random order $[word_5, word_1, word_2, word_4, word_3]$. This allows the model to learn bidirectional dependencies. For example: When predicting $word_2$, the context will contain words that, in the original context, were both earlier ($word_1$) and later ($word_5$) than it. XLNet takes this approach with all possible permutations of the factorization order. It does so by utilizing masking in the *Transformer*, so as to not change the order of the actual input as this would create unrealistic text combinations creating discrepancies between pre-training and fine-tuning.

XLNet Architecture
XLNet is a Transformer-XL based architecture with some modifications. When pre-training a Transformer based model, the embedding for the token being predicted is masked, this includes its positional embedding. This, however, is potentially useful information. When predicting a token, only the position should be known, not the content. The solution is a two-stream self-attention architecture consisting of a content stream and a query stream. The content stream is a standard self-attention model without masking, which allows access to the full context and token content. The query stream has limited access to only the context of the previous steps and the current token's position.

The query stream is only used for pre-training and can be dropped during the fine-tuning process, turning the model into a normal Transformer based model.

To handle multiple segments, XLNet utilizes a relative positional encoding scheme similar to the one proposed by Transformer-XL (see section 2.5.2.9). Each word has a segment encoding, which indicates whether any two words belong to the same segment or not. This means that it does not encode which specific segment a word belongs to or where exactly its position in that segment is, only whether or not two words are from different segments. This has the additional benefit of allowing encoding of more than two segments, which BERT does not.

2.5.4 Task Specific Models

In this section we introduce two task specific uses of pre-trained models *BertSum* (Section 2.5.4.1) for the task of summarisation and *SBert* (Section 2.5.4.2) for producing sentence embeddings.

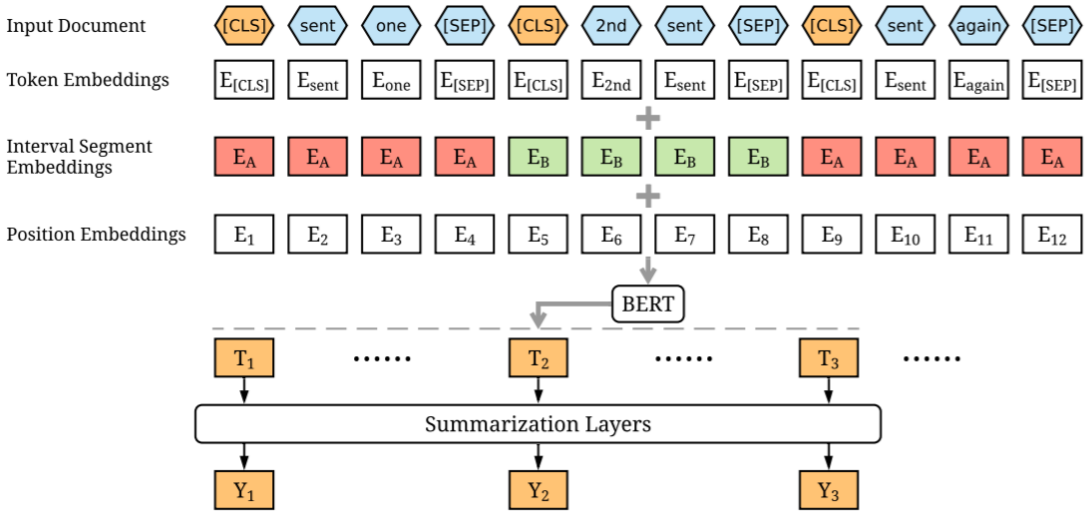


Figure 2.10: BERTSum architecture. Reproduced from [2] with the authors' permission.

2.5.4.1 BERTSum

BERTSum is publically available at: `https://github.com/nlpyang/PreSumm`
BERTSum [2] is a variant of BERT fine-tuned for extractive single-document summarisation. For the purpose of extractive summarisation, two problems need to be overcome:

1. Each sentence needs to be labeled as either a summary sentence or a non-summary sentence. By default, however, BERT outputs token representations, not sentence representations, and no classifications either.
2. BERT accepts inputs of either a single sentence or a pair of sentences. For summarisation purposes, however, the model should be able to process documents containing multiple sentences.

In order to overcome these problems, the author of [2] modified both the input sequence and the embedding slightly, extending them to sequences of multiple sentences:

1. In the input sequence, each sentence is preceded by a [CLS] token and succeeded by a [SEP] token.
2. In the original BERT, two sentences A and B are distinguished by segment embedding. Each token of sentence A is embedded using E_A and each token of sentence B is embedded using E_B. For BERTSum, this is extended to: The tokens of the i-th sentence are embedded using E_A if i is odd and E_B if i is even.

Thus, the output of the top BERT-layer for each [CLS] token is treated as the sentence representation of the sentence following that token. The architecture of the BERTSum model and especially the input embedding is shown in Figure 2.10
Having obtained sentence representations for multiple sentences, there are several ways the author suggests to fine-tune BERT for extractive summarisation:

1. Adding a single sigmoid classification layer on top of the BERT ouputs.
2. Adding more Transformer layers on top of the BERT output. (Inter-sentence Transformer) On top of that, a sigmoid classification layer.
3. Adding an LSTM on top of the BERT outputs. On top of that, a sigmoid classification layer.

Liu's experiments in [2] showed that the option of adding a two-layer Transformer and a single sigmoid classification layer on top of BERT produced the best results.

Like BERT, BERTSum has an input limit of 512 tokens.

2.5.4.2 Sentence-BERT (SBERT)

Reimers et al. [39] propose a modified BERT model for semantic textual similarity (STS) tasks using BERT. The default BERT implementation can be used for STS tasks by utilizing the input of sentence pairs. But this is computationally expensive, as each sentence pair would need to be compared against each other. The authors name as an example, that finding the most similar pair in a collection of 10.000 sentences using conventional BERT requires $n(n-1)/2 = 49995000$ operations, which would take take roughly 65 hours.

A common solution to these types of problems is to map the inputs to some vector space which can then be compared via for example clustering. Sentence embeddings can also be performed using BERT. This is typically done by feeding the model a sentence and either averaging the output layer or using the [CLS] token. However, without additional fine-tuning these are not very useful for semantic textual similarity (STS) tasks. In order to mitigate these problems, the authors developed a modification of the BERT model, which they called *Sentence-BERT* or *SBERT*. SBERT is then fine-tuned to produce semantically meaningful fixed-length sentence embeddings (i.e., so that semantically similiar sentences are close together in the vector space) which can be easily compared with the cosine similarity score (see section 2.2.2.3). This makes finding the most similiar pair of sentences much quicker. Using the same example as before, finding the most similiar pair of sentences in a collection of 10.000 sentences takes, according to the authors, only a few seconds with SBERT, as opposed to 65 hours with BERT.

The authors did not find that using RoBERTa instead of BERT resulted in any improvements for their purposes. They also found that XLNet performed even worse than BERT on STS tasks, which is why they used BERT as the basis of their work.

The authors also tried alternative similarity measures, like the Manhattan and negative Euclidean measure, but found that they had no advantages compared to cosine similarity.

Reimers et al. developed several possible architectures for SBERT, depending on the kind of training data available. SBERT can be built with a siamese or triplet network structure [40], which means that the same weights are used to process two

or three input sentences at the same time. SBERT also adds a pooling operation to the output of BERT to derive fixed-size sentence embeddings from it. The authors tried different pooling strategies, but found that taking the mean of all the output vectors produced the best results. There are different objective functions available, depending on the task to be trained for:

1. **Classification Objective Function**. The sentence embeddings are concatenated with the element-wise difference and a softmax-function[2] is applied to obtain the classification label. This structure is depicted in Figure 2.11.

2. **Regression Objective Function**. The sentence embeddings are used to compute the cosine similarity score. This structure is depicted in Figure 2.12

3. **Triplet Objective Function**. The network is fed three sentences, one of which is the so-called anchor sentence, while the other two are the so-called positive and negative sentences. The training objective is to make sure that the distance between the positive sentence and the anchor sentence is always smaller than the distance between the negative sentence and the anchor sentence.

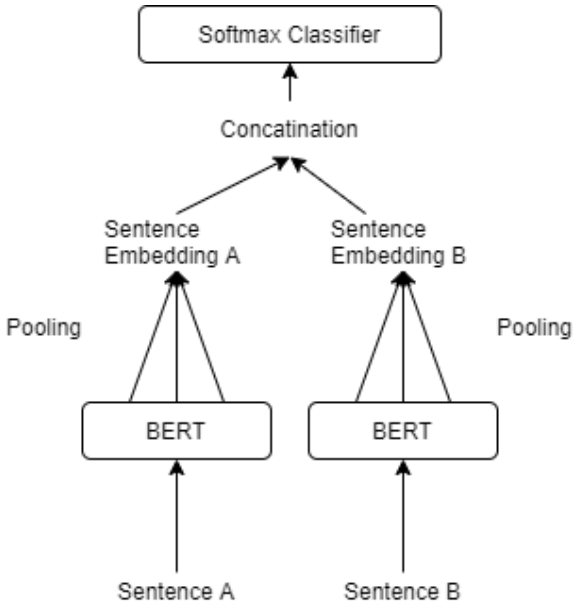


Figure 2.11: The SBERT architecture for a classification objective function.

[2]A softmax function takes a vector of dimensionality k and normalizes it into a probability distribution of k probabilities, proportional to the original inputs, which add up to 1.

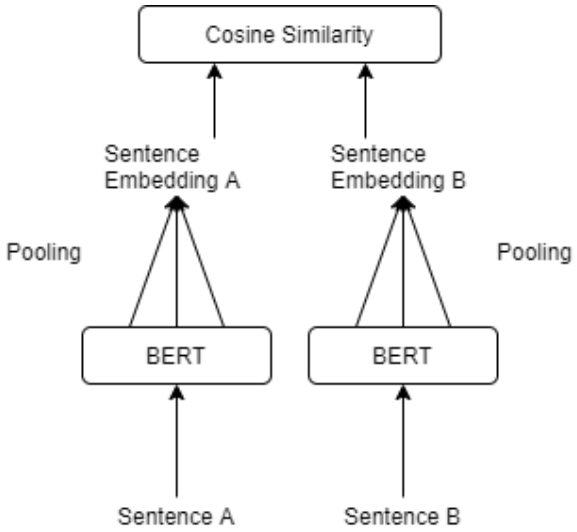


Figure 2.12: The SBERT architecture for a regressive objective function. This can be used to compute similarity scores.

SBERT raised the state of the art for sentence embedding and several STS tasks.

3
Methods

In this chapter the methods used to achieve the goals of the thesis are described. Section 3.1 gives an overview over the changes that this project went through during the course of its conception and execution. In Section 3.2 we describe the datasets we used during training and evaluation, their properties and how we obtained our training data from them. In Section 3.3 we describe some of the challenges of using BertSum for summary generation and how we solved them. In the Section 3.3.2 we describe our implementation and training related details. Section 3.5 describes our evaluation metrics. Finally we describe the experiments we performed using different pre-trained models in Section 3.6.

3.1 Changes in the direction of the project

The initial plan for this thesis project was to use a dataset of academic papers to fine-tune a BertSum model for summarisation. But it became clear early on that creating a large enough dataset would not be possible within the scope of the project. We identified a few bottlenecks for such a project, which will be described in the section on future work 5.4.
Instead, we decided to investigate how well a BertSum model fine-tuned on the widely used CNN/DM news dataset would transfer and perform on our academic papers dataset. We will also investigate how different pre-trained models compare against each other. In the next section we will go into more detail on the datasets we used and how we obtained them.

3.2 Datasets

In the following, we will describe the datasets we used for our experiments: the CNN/DM dataset, which we used to fine-tune the models, and the "Academic Papers" dataset, which is a very small dataset we created ourselves for the purpose of evaluation.

A dataset to train for the extractive summarisation task requires a text (sequence of sentences) and reference labels: For sequence-labeling, each sentence needs to be labeled as 1 (summary) or 0 (non-summary). This means that the summary length must be known before label selection. For score-and-select, the label for each sentence is a score indicating its importance to the summary. This does not require summary length to be known in advance.

3.2.1 CNN/DM

Few datasets exist for the task of text summarisation, especially extractive text summarisation. The CNN and DM datasets[1], which contain news articles, gathered from CNN and Daily Mail, each accompanied by a short abstractive summary, are commonly used for training and evaluating models for summarisation tasks. This dataset exists in an anonymised and a non-anonymised version. The anonymised version replaces identifiers with non-identifiables. For better comparisons with previous works, which largely used the non-anonymised version, we chose that one for our experiments, too.

The dataset was split for training, validation and testing as suggested by Hermann et al. in [41], statistics can be seen in Table 3.1.

Table 3.1: Statistics for CNN/DM dataset

	Train	**Validation**	**Test**
No. of Samples	287083	13367	11489
Avg. Sent. Length	35.56	32.24	32.62
Avg. Number of Tokens	927.96	910.17	921.96
Summary Avg. Sent. Length	3.73	4.11	3.88

3.2.1.1 Label Generation

The summaries included in the CNN/DM dataset are so-called "highlights", a few sentences for each news article, which aims to summarise it. These summaries are abstractive and can therefore not be used directly for training an extractive summariser. Instead labels were generated using these abstractive summary as guidance. We generated three sets of label data.

1. Binary Labels: As in the BertSum paper [2], we generated binary labels for the sequence-labeling problem definition. Up to three sentences(the pre-determined length of summaries) are selected from each news article, by maximize the ROUGE scores against the abstractive summaries. For this purpose, Liu [2] proposes two different algorithms. The first is a greedy algorithm, which is fast but does not consider all combinations of sentences. The second algorithm does consider all combinations, but is slower. Liu opted for the faster but inaccurate algorithm and so did we because of the insignificant differences between them in terms of score.

In addition to this, we propose two additional label selection schemes for the score-and-select problem definition:

2. Score Labels: We assigned each sentence a score, based on its ROUGE score against the abstractive summary. We hope that this method will allow models to

[1]Both are available for download here: https://cs.nyu.edu/ kcho/DMQA/ which is where we obtained our data from. (Last accessed 14.02.2020.)

generalise better for a wider range of summary lengths.

3. Sentence BERT Score(SBS) Labels: Similar to the previous score labels, but instead each sentence is assigned a score based on the cosine similarity between its sentence embedding and that of the summary, as produced by SBERT (see Section 2.5.4.2).

3.2.2 Academic Paper Dataset

This dataset consists of a small number of papers on driving styles in the domain of traffic safety. This dataset was too small to perform meaningful fine-tuning on, but we did use it to evaluate our models' ability to transfer from the news data they were fine-tuned on to this different type of texts. The papers were provided to us in PDF-form which was a limiting factor for the number of documents we were able to include in the dataset, because of the additional work required to extract and pre-process the texts. To obtain labeled data, we created extractive reference summaries from scratch.

In the following sections, we will describe how we extracted the text from the PDFs, how we pre-processed these texts and how we obtained the reference summaries.

3.2.2.1 Text extraction

For our experiments, we collected 31 PDF-documents. These are scientific publications on the topic of traffic safety between 5 and 33 pages of length, with the average number of pages being 14. The first step to use these documents was to extract the text from each of the PDFs.

Since developing a dedicated tool for text extraction from PDF was out of scope for this thesis project, we utilised an existing one, pdftotext[2] from the open source toolset Xpdf. We configured the pdftotext tool to cut out all images from the PDFs and then had it produce a TXT-document for each original PDF-document containing only its extracted text, which we then cleaned up manually. This manual clean-up was necessary, because even the best text extraction tool we were able to find had various issues, which will be described in the next section.

3.2.2.2 Pre-processing

The automatically extracted TXT-files, while giving us a good base to work from, had various problems, which would have limited the quality of potential summaries:

1. Tables, headers, footers and page numbers were not recognised as not being part of the text. Therefore, they appeared in the extracted TXT-files, breaking up the actual text in unfortunate ways. Often, these artifacts would be inserted mid-sentence.

[2]http://www.xpdfreader.com/pdftotext-man.html

2. Occasionally letters, words or phrases would be printed repeatedly. In rare cases, even nonsensical streams of letters were produced for no apparent reason.
3. Sometimes the original texts would contain periods in headlines or mid-sentence, which pdftotext would preserve, impeding our ability to automatically recognise the beginning and end of sentences.

Because of the relatively small number of documents, we came to the conclusion that trying to develop a piece of software to solve these problems automatically would take more time than simply cleaning up the TXT-files by hand. The following manual changes were made to the automatically extracted texts:

1. Headers, footers, page numbers, tables etc. were removed from the texts wherever we found them.
2. Corrupted words (such as "trraaaaffffiiiiiiic", "tra?c", "traf-fic") were corrected ("traffic"), and where whole sentences were corrupted, we manually copied over the correct text from the source PDF.
3. Periods that were used in any other way than to end a sentence were removed from the sentence. Periods were added after headlines so as to prevent them from being interpreted as the beginning of the following sentence.
4. Anything before the "Introduction" section and after the "Conclusion" was removed from the document. In particular, we removed the abstracts from the text as we intended to use them during evaluation.
5. Formulas, which pdftotext was rarely able to extract in a readable format, were either cleaned up or deleted from the text, where we deemed their exact content irrelevant to the surrounding text.
6. The headlines were translated into a machine-readable format in the following fashion:
   1. Headline -> # Headline.
   1.1. Second headline -> ## Second headline.
   etc.

We thought that the information on sections and subsections of the text might be interesting to preserve, though we ended up not using it.

3.2.2.3 Obtaining reference summaries

Next, we had to create reference summaries for each of the documents, to be used for evaluation. This, too, was done manually. Due to time constraints, we only did this for ten of the documents. The summaries were created from the cleaned up TXT-files in the following way: First, we read carefully through the entire document to familiarise ourselves with what they were about. We then went through the document again from top to bottom and removed all sentences that did not seem essential to convey the most important information of the document. We repeated this step until we felt that no more sentences could be removed without leaving out important information.

We did this individually, each of us producing a separate summary for each of the ten documents. We thus ended up with two reference summaries per document.

This was done to have a measure for how much we can expect two summaries of the same text to differ from one another, even under the assumption that both of them are optimal. Knowing this will help us better interpret the quality of our generated summaries.

All the summaries except for one were roughly 10-20% of the length of the original documents, measured in word count. (The single outlier was almost 50% of the document length, but with 1676 words total, it was a short document to begin with.)

3.3 Summary Generation

In this section we will describe the problems we had to overcome to be able to generate summaries for the Academic Paper dataset, and how we implemented the summarisation models.

3.3.1 Adressing BERTSum's Token Limit

As mentioned in Section 2.5.4.1, the BERTSum model has a limit of 512 input tokens. For short news data like the CNN/DM dataset, texts are usually truncated to fit these limitations. This, however, affects summary generation, as it only allows the model to select sentences contained within these first 512 tokens. For the Academic Paper dataset, which consists of texts much longer than this limit, this is unreasonable, as it would disregard the majority of the texts. A possible solution, suggested by the BERTSum [2] authors, is to simply extend the token limit of the model, but as the pre-trained models it uses to generate token embeddings have the same limit, this would not affect pre-training. Additionally, the academic papers are of such a length that the token limit would have to be increased 20-fold to fit some of the texts, and we would run into computational and memory limitations. Another solution was needed. As suggested by Al-Rfou et al. in [31], we instead split the input texts into multiple blocks below the limit and feed each block to the model individually, later combining its output, this allows the model to generate summaries for longer texts.

This method, however, is not without problems: Since no information is shared between the blocks, contextual information is lost, which will have an effect on the generated summaries. Positions will be repeated within each block and thus potential positional bias will also be repeated.

To allow summary generation for texts of any length, the following steps were taken:

1. Input texts are split into blocks of a maximum length of 512 tokens, with a maximum sentence length of 200 tokens. Longer sentences are truncated to avoid single sentence blocks.
2. Blocks are created sentence by sentence: If adding another sentence would exceed the limit, a new block is created. Thus, sentence integrity is maintained.

3. Each block is put through the model separately, which outputs a score for each sentence in the block.
4. The scores of each block are combined and the top scoring sentences are selected to form the summary.

3.3.2 Implementation

We used the BERTSum model implementation proposed by Liu[3] as presented in [2] as the base for our implementations. This BERTSum implementation is built on top of "Open Source Neural Machine Translation in PyTorch"[4], which is an open source framework for sequence models. To better facilitate our goals of using multiple pre-trained models, we made some changes to the original implementation, which will be described in the next sections.

We used the Transformers library, maintained by Hugging Face[5], which contains standardised PyTorch implementations of many of the newest Transformer-based models. This is the latest version of the commonly used pytorch-transformers library (which BERTSum utilises). Our own implementation is based on their examples. In the following sections we will explain some key alterations we made.

BERTSum model alterations: The BERTSum model, which originally builts on just BERT, was altered to support multiple pre-trained models for generating sentence embeddings. Since some of the pre-trained models we intend to use, like RoBERTa, do not use segment embeddings for pre-training and since the BERTSum paper [2] showed only small differences, we also removed segment embedding from our altered BERTSum model. Support for the XLNet-specific feature of context sharing between blocks was implemented.

Data-loading: We utilise a different dataloading process than BERTSum, which uses a dataset already containing pre-computed word token representations specifically for BERT. As some of the other pre-trained models use different tokens, this method would require us to create a new dataset for each model, containing such pre-computed tokens. Instead, we used a PyTorch-style data-loader that performs model-specific tokenisation *during* data-loading. This causes some computational overhead, which can be mitigated by utilising several worker processes.

Hyper-Parameters: Trying to find the optimal hyper-parameters was out of scope for this project. Instead we refer to previous work and use the hyper-parameters suggested by the authors in each pre-trained models' paper. The authors of the original paper on BERT [19] suggest that hyper-parameter tuning is of less importance when the fine-tuning data set is large (as is the case for the CNN/DM dataset), which also matched our initial findings when trying different parameters. Therefore, we simply stick to the suggested hyper-parameters for each model.

[3]https://github.com/nlpyang/BertSum
[4]https://opennmt.net/
[5]https://github.com/huggingface/transformers

Optimisers and Schedulers: In addition to the BERTSum optimiser/scheduler we also implemented support for the PyTorch implementation of AdamW[6] with linear schedule decay. We used AdamW as the optimiser and scheduler for our experiments to better match how the models were originally pre-trained.

Loss Functions: BERTSum uses the summed Binary Cross Entropy as its loss function, which is suitable for the binary classification task of sequence-labeling. To also support the score-and-select training objective, we implemented support for a mean squared error (MSE) loss function as well.

Selection Layer: The BERTSum paper [2] explores several different selection layers and come to the conclusion that a Transformer layer produced the best results. This is therefore what we used as well. The authors also used a tri-gram blocking scheme, which blocks the addition of a sentence to the summary if it contain an overlapping tri-gram with the summary. This ensures a more diverse text, and lead to improved scores for the authors.

Checkpoint Averaging: During training, checkpoints of the model are saved at regular intervals. Checkpoint Averaging is a method where a number of checkpoints are combined and averaged into a single, supposedly more robust, model. The authors of the BERTSum paper used multiple checkpoints of the model saved during training and combined the weights of the top 3 performing checkpoints (on the validation set) into the final model. We also employed this method.

3.4 Hardware

Most of the training was done on the high-performance clusters of C3SE[7] which is the centre for scientific and technical computing at Chalmers University of Technology in Gothenburg, Sweden. C3SE is part of the Swedish National Infrastructure for Computing, SNIC[8].

The training was performed on the GPU-nodes of the Vera Cluster, which are outfitted with Tesla V100 32GB model GPUs. These GPUs support half-precision float format (FP16, 16-bit floats), which allows for mixed-precision training. Utilising this can lead to a significantly faster training speed. Mixed-precision training[9] uses FP16 for operations, while important network information is stored in single-precision (FP32). This reduces memory requirements and allows for larger models and batches. FP16 structures are also faster to access and transfer than FP32. The loss of precision, which can lead to small numbers being interpreted as 0, is combated by a technique called loss scaling, which helps preserve small gradients.

[6] `https://www.tensorflow.org/addons/api_docs/python/tfa/optimisers/AdamW`
[7] Chalmers Centre for Computational Science and Engineering: https://www.c3se.chalmers.se/
[8] https://www.snic.se/
[9] https://docs.nvidia.com/deeplearning/sdk/mixed-precision-training/index.html

3.5 Evaluation

In this section we will describe the metrics used to evaluate the models' performance.

3.5.1 ROUGE Evaluation

The three commonly used ROUGE metrics for evaluating summaries, ROUGE-1, ROUGE-2 and ROUGE-L, are used for evaluation. For each of these we use the F-1 score which is a combination of the precision and recall. For a more detailed description of these metrics, see Section 2.2.

Several suites exist for performing ROUGE evaluation. We opted for a Python implementation of ROUGE[10]. This implementation produces slightly different scores compared to the implementation used in the BERTSum paper. This hurts comparison against previous works, but it is faster, and as our goal required evaluating many models against each other we deemed it the better option.

3.5.2 Sentence Similarity Evaluation

We also explored another evaluation metric based on sentence similarity. We hoped that this metric would be more accurate when for example comparing against abstractive candidate summaries, since it does not require exact word matchings as ROUGE does. We use SBERT (see 2.5.4.2) to produce an embedding for each sentence. These are then averaged into a single combined document embedding. This is a naive approach based on the same methodology that SBERT employs, in taking the mean of word embeddings to create the sentence embedding. The score for each summary is the cosine distance between the reference summary and the generated candidate summary. To bring it into a similiar range and thus make it more comparable with the other metrics we took $(1 - dist_{cosine}) \cdot 100$ as the final score.

3.5.3 Evaluation on the CNN/DM dataset

The held-out test data of the CNN/DM dataset (see Section 3.2) was used to evaluate the models' performance. The fine-tuned models were used to generate three-sentence summaries, and these were evaluated against the abstractive summaries included in the CNN/DM dataset, using the methods outlined above. Having no explicitly extractive reference summaries likely limits the ROUGE score that can be reached, as there might not always be exact word matched between the provided abstractive summary and the generated extractive one. The sentence similarity metric, however, should not be as dependent on word matches.

Evaluation was performed on both a truncated and a full version of the CNN/DM dataset. For the truncated version, the BERTSum token limit is enforced by simply truncating the texts. This will bias the results, as the model can only select from

[10]https://github.com/pltrdy/ROUGE

sentences that appear before this limit is reached. The full dataset evaluation used the block splitting method introduced in Section 3.3.1 to allow the model to select from the full range of sentences.

3.5.4 Evaluation on the Academic Paper dataset

For this dataset the evaluation of the generated summaries was performed against the manually created reference summaries. Since these reference summaries are extractive, we also measured *sentence overlap*, in addition to the above evaluation methods. Sentence overlap is measured as follows:
$S_c = set\ of\ sentences\ in\ candidate\ summary$
$S_r = set\ of\ sentences\ in\ reference\ summary$

$$Score = \frac{S_c \cap Sr}{S_c \cup S_r}$$

3.5.5 Human Evaluation

Extensive human evaluation was not performed, due to constraints in time and resources. We did, however, want to have some sort of measure of how well the different models performed by human standards. We obtained this in the following way: A random text was selected from the Academic Paper dataset for which we evaluated and ranked the generated summaries of all models. Summaries were ranked on relevant sentence selection, cohesion and readability. More formally the ranking was performed as follows: The origin of the generated summaries were obscured so as to not influence our rankings. We assigned each sentence a score of 1 if it was a "good" sentence (according to our subjective judgement), a sequence of good sentences was assigned an increasing score to capture a notion of cohesion. Neutral sentences were assigned a score of 0. Bad sentences were assigned a score of -1. Finally we summed the scores to give a final score for the sentence. When performing the final rankings, tie breakers or close scores were determined by subjectively assessing the whole summary on the above criteria.

3.6 Experiments

We performed several experiments to evaluate and compare the pre-trained models against each other on the task of extractive text summarisation. The pre-trained models we decided to investigate are:

1. BERT
2. DistilBERT
3. RoBERTa
4. XLNet

Most of the models are also available in larger versions with a deeper network, we opted to only train the smaller "base" versions of the models. Previous work has shown gains for the large models, but for our purpose of comparing several models

we decided that the base versions were a better choice because of the lower resource and training time requirements.

Experiment 1: BERTSum Reference
For the first experiment, no fine-tuning was performed. We only evaluated the BertSum model published by Liu and Lapata [2] as is on the Academic Paper dataset to obtain a baseline.

Their model was trained for 50 000 iterations using 3 Nvidia 1080-ti GPUs with a gradient accumulation of 2, resulting in an approximately combined batch-size of 36. Training for 50 000 iterations with this batch size resulted in approximately 6 epochs.

Experiment 2-7 For these experiments, we used some different parameters than for Experiment 1. The main differences are the warmup and weight-decay: We opted to run for 4 epochs using 10% of total training steps as warm-up steps and a linear learning rate decay, motivated by suggestions in each pre-trained model's paper, time to train and resource availability.
We decided to use the same batch size as BERTSum, 36. With the available hardware and using mixed-precision training we were able to fit the entire batch size onto one GPU without using gradient accumulation for all models except one. The when fine-tuning the XLNet model we could not fit an entire batch onto a single GPU, instead it was trained using two GPUs.
For these experiments, the pre-trained models were fine-tuned as sequence-labeling models using the binary label data for the CNN/DM dataset, as described in Section 3.2.1. We evaluated their performance on the held-out portion of the dataset. Additionally, we evaluated their performance on our Academic Paper dataset, to measure how well the models would transfer to the new task. The models used in our experiments were the following:

- **Experiment 2:** The BERT base pre-trained model, which has 12 encoding layers, 12 attention heads and a total of 110M parameters. This model will in the following be referred to as "BERT".
- **Experiment 3:** The RoBERTa base model which has 12 encoding layers, 12 attention heads and 125M parameters. This model will in the following be referred to as "RoBERTa".
- **Experiment 4:** The DistilBERT base model which has 6 encoding layers, 12-heads and a total of 66M parameters. This model will in the following be referred to as "DistilBERT".
- **Experiment 5:** The XLNet base model which has 12 encoding layers, 12 attention heads and a total of, 110M parameters. This model will in the following be referred to as "XLNet". Where applicable the memory functionality will be utilzed and referd to as "XLNet Mem".

Based on early results we also decided to train two additional models, using the score-and-select training objective. Due to time constraints, we could not do this for all the models, and so we chose only Roberta for these additional experiments, as this model showed the most promising performance. These models were trained

using mean squared error (MSE) loss and the Score and SBS label data.

- **Experiment 6:** RoBERTa-base model using label data based on ROUGE score. In the following chapters, this will be referred to as the "RoBERTa S" model. (S: Score)
- **Experiment 7:** RoBERTa-base model using label data based on sentence similarity. This model will in the following be referred to as "RoBERTa SBS".

These experiments are illustrated in Figure 3.1.

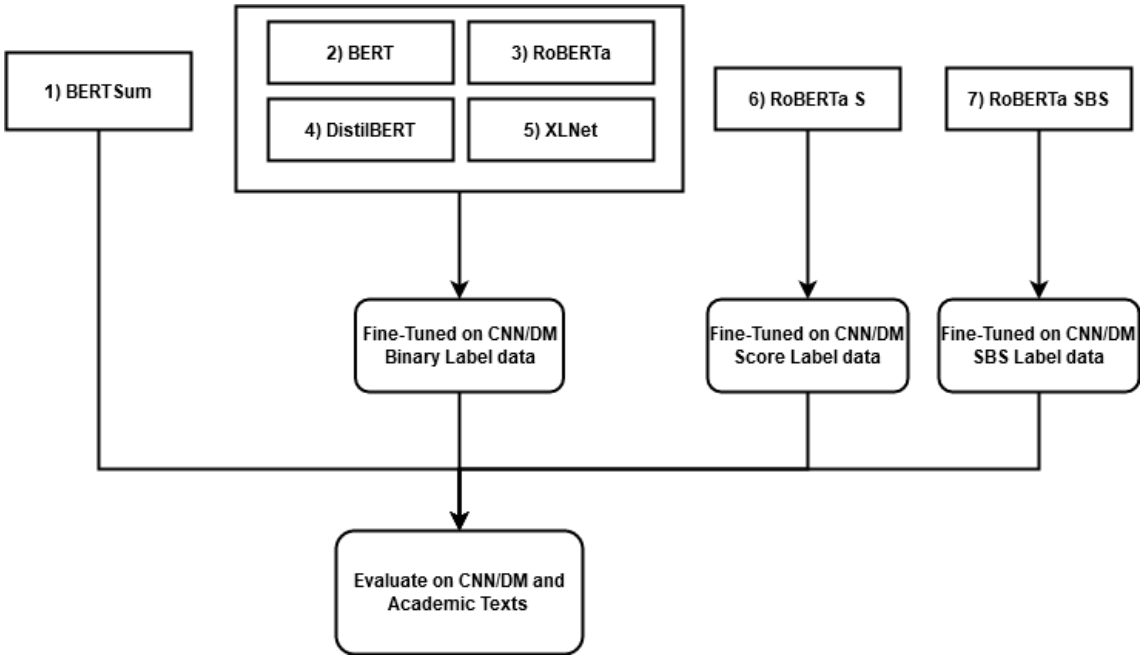


Figure 3.1: An overview of the experiments we performed.

4
Results and Discussion

In this chapter we will present the results of our experiments and discuss the most interesting findings. First, in Section 4.1, we will look into the training and validation of the models. In Section 4.2 we will present the results of evaluating on the CNN/DM dataset. In doing this, Section 4.2.1 takes a look at the truncated CNN/DM dataset, while Section 4.2.2 deals with the full dataset. In Section 4.2.3, these findings will be discussed. In Section 4.3 we will detail how the models performed when applied to the Academic Paper dataset. First, we will look at the general results on this dataset in Section 4.3.1. Then, in Section 4.3.2 we will take a look at one randomly selected sample of this dataset for closer inspection, comparing the metrics against human evaluation. These findings will be discussed in Section 4.3.3. In Section 4.5 and Section 4.6 we will look into the inherent positional bias in the CNN/DM dataset and how it affected the models fine-tuned on it, respectively. Finally in Section 4.7 we will discuss the model confidence.

4.1 Training and Validation

In this section we will take a look at the results related to the training and validation of the models.

Table 4.1 shows the time it took to train each model together with its size. With only 2.5 hours, DistilBERT was by far the fastest of the models to train, and with a size of 317 MB, it was also the smallest. The other models are roughly 60% larger and took 5 hours to train, except for XLNet, which took very much longer: 11.5 hours.

The training loss plot in Figure 4.1 shows that the BERT-based models all have similar loss curves. The curves also correlate to the models' performance on the

Table 4.1: Size in MB and required training time for all the models used in our experiments.

Exp. No.	Model	Train Time	Size(MB)
2	BERT	5h	486
3	DistilBERT	2h30m	317
4	RoBERTa	5h	545
5	XLNet	11h30m	514

validation set as can be seen in Figure 4.2. XLNet is an outlier, which is not unexpected, because of the differences in pre-training and architecture of this model. Because of this, its plot cannot be directly compared to the other three in a meaningful way.

The loss curve starts leveling before the first epoch has been completed. A possible reason for this could be that the pre-trained model has learned to produce adequate sentence embeddings by this point, and is in the following only adapting to the specifics of the training set. As stated by [42] and [39], the [CLS]-token is not a good sentence embedding without fine-tuning. At some point during the fine-tuning process the model will have adapted to produce good sentence embeddings and we suspect that his might require less time compared to other parts of the model. Another possible explanation is that all the entries in the CNN/DM dataset are fairly similar to one another in their properties, and that further training therefore does not result in a whole lot of new knowledge gains.

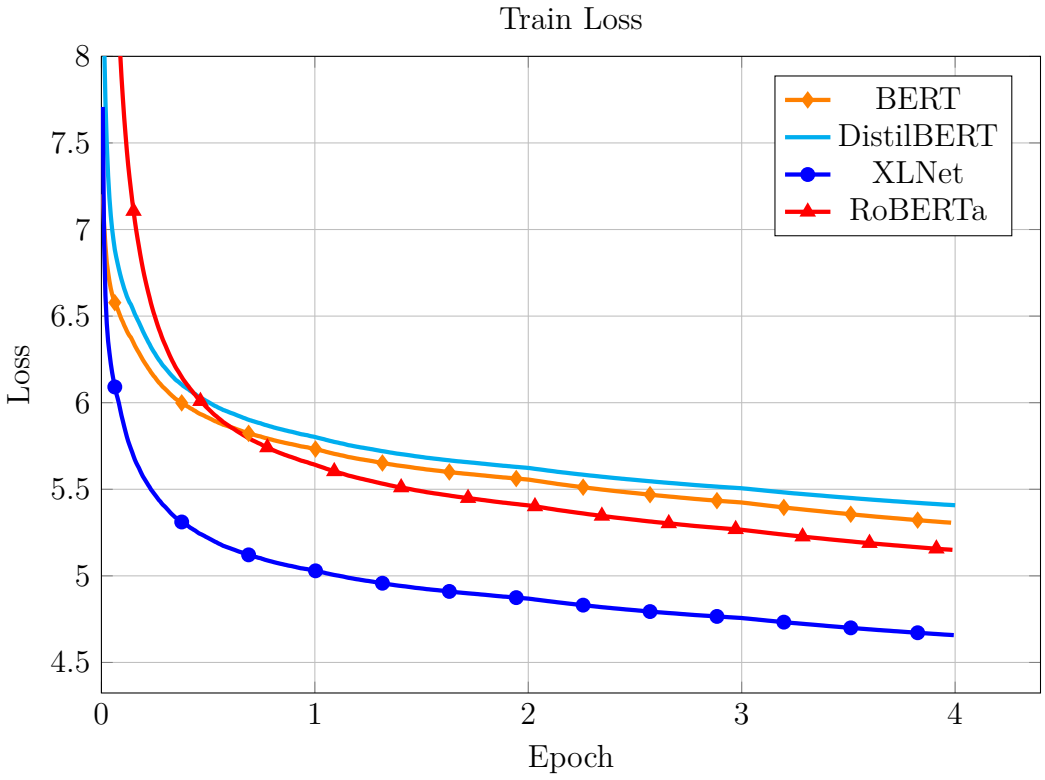
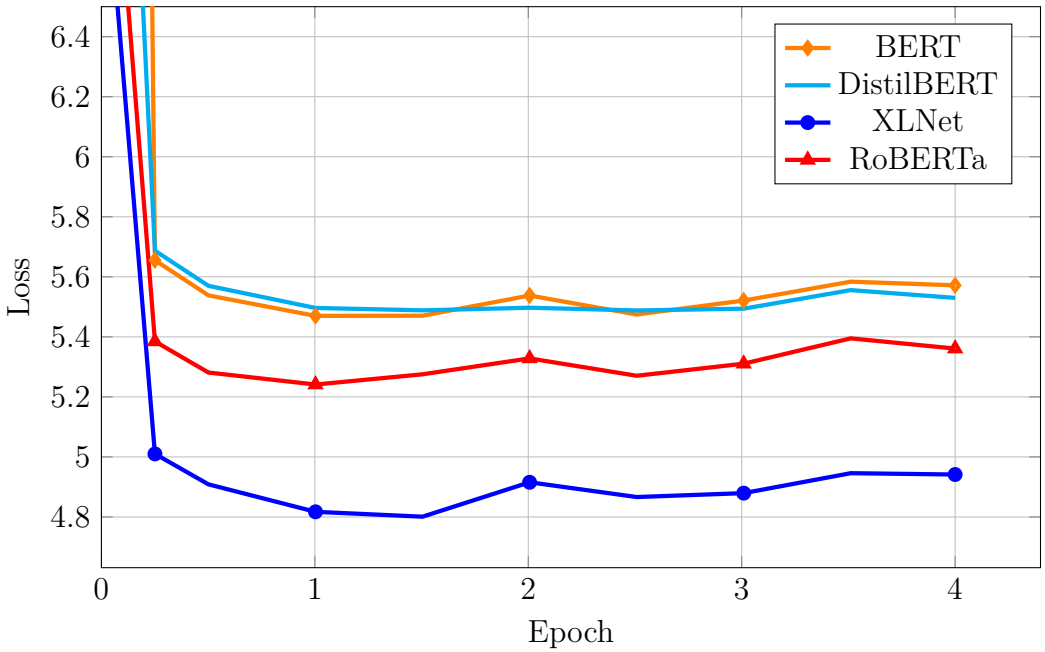


Figure 4.1: The plot of training loss with BERT,DistilBERT, RoBERTa and XLNet

The loss and scores of the validation set for the different models can be seen in Figure 4.2. The scores match the loss plot up until a point. Most of the models do not see much improvement after 1.5 epochs and even start performing slightly worse again, despite continuously decreasing training loss, which could be an indicator of overfitting. This might be helped by utilizing the full dataset for training, something we did not investigate.

In Figure 4.2 (b) we can compare the scores of XLNet against the others. We now see that the lower loss scores we saw previously were not necessarily an indicator of the model's performance. Another observation is that BERT and DistilBERT behave roughly the same over the epochs. This is expected, as DistilBERT was developed and pre-trained specifically to imitate BERT's behaviour.

(a) Validation Loss



(b) Validation Score

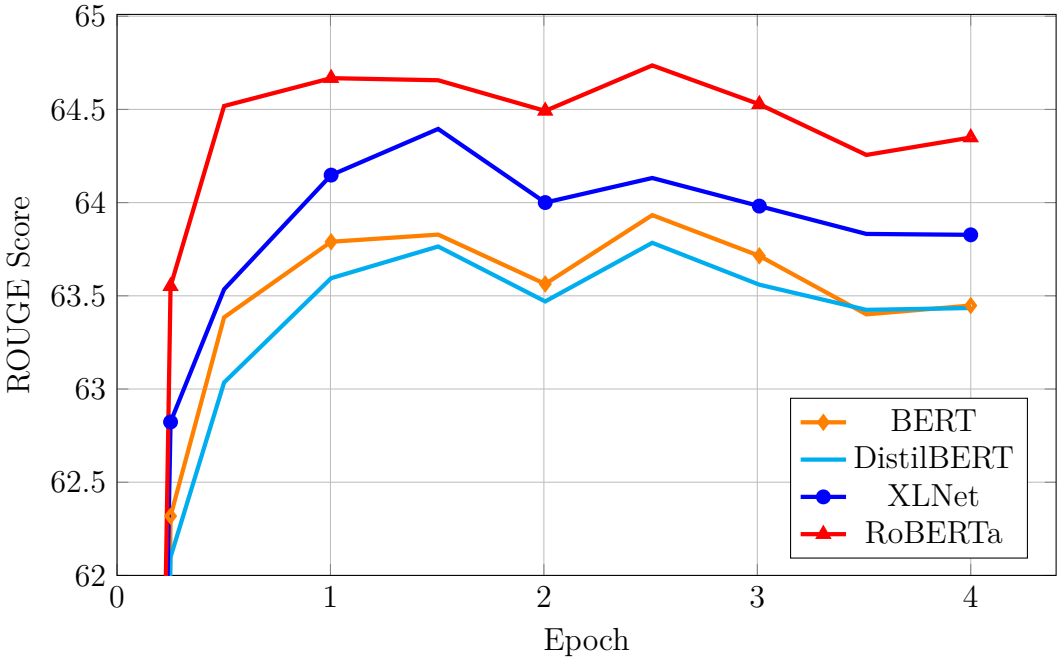


Figure 4.2: Plots of validation loss and score of BERT, DistilBERT, RoBERTa and XLNet

4.2 CNN/DM Dataset

After the fine-tuning of the models, which was performed using only the truncated dataset, they were evaluated on both the truncated and the full CNN/DM dataset. We will first look at the truncated results followed by the full results.

4.2.1 Truncated Results

The results presented in this section were obtained by evaluating on the truncated CNN/DM dataset. The texts in this dataset were truncated to fit the 512-token limit, as such the models can only select from the first couple of sentences contained within this limit.

The scores that the different models achieved are displayed in Table 4.2. A commonly used baseline is the Lead-3 method, which simply picks the first three sentences of each text as the summary. Another baseline we utilized, more useful for longer texts and summaries, is Every-7 which picks every seventh sentence. There is quite a discrepancy between the Lead-3 and Every-7 scores. This is an indication of bias towards early sentences in the dataset, which we will further explore in Section 4.5.

The bars labaled "Binary Labels", "Score Labels" and "SBS Labels" show the score of the corresponding label data. They can be seen as a maximum value for the scores, depending on which scores were used to obtain the label data for fine-tuning. (See Section 3.6 for an explanation of which model uses which scores.)

The results are visualised in Figure 4.3.

Almost all the models, for all metrics, surpass the Lead-3 baseline, though not by a large margin. The only exception to this is RoBERTa SBS, whose score is slightly lower on the ROUGE-2 and ROUGE-L metrics. The models far exceed the Every-7 baseline.

We could not observe a lot of variation between the different models. All of them perform closer to the Lead-3 baseline than to the upper bound of the Binary Labels/Score/SBS Score. This tells us that there is room for improvement.

RoBERTa achieved the overall highest scores, by a small margin. At the time of writing this result constitutes a new state-of-the-art score for a 'base' (smaller) model. XLNet produced disappointingly low scores, considering the additional time and resources required to train the model. The RoBERTa model trained with a score-and-select objective (RoBERTa S) achieved lower scores compared to sequence-label trained (RoBERTa). This was not unexpected, as the "Score Labels" themselves also produce a lower score than the "Binary Labels". The RoBERTa SBS model produced the lowest ROUGE scores, but it did achieve slightly higher sentence similarity score, even if not by a lot.

Table 4.2: Scores on the truncated CNN/DM dataset. (S: Score, SBS: Sentence BERT Score)

No.	Model	ROUGE-1	ROUGE-2	ROUGE-L	Similarity	Mean
-	Lead-3	40.977	17.038	30.081	15.773	103.869
-	Every-7	30.850	9.150	20.880	10.767	71.647
-	Binary Labels	58.021	32.996	46.251	25.318	162.586
1	BERTSum	43.733	19.516	32.146	18.751	114.146
2	BERT	43.669	19.624	32.008	18.652	113.953
3	DistilBERT	43.608	19.514	32.143	18.633	113.898
4	RoBERTa	44.199	20.022	32.432	19.079	115.732
5	XLNet	43.887	19.733	32.284	18.754	114.658
-	Score Labels	54.310	29.314	40.819	25.477	149.920
6	RoBERTa S	43.403	19.087	31.394	19.269	113.153
-	SBS Labels	46.866	22.227	34.383	28.483	131.959
7	RoBERTa SBS	41.456	17.255	29.547	19.481	107.739

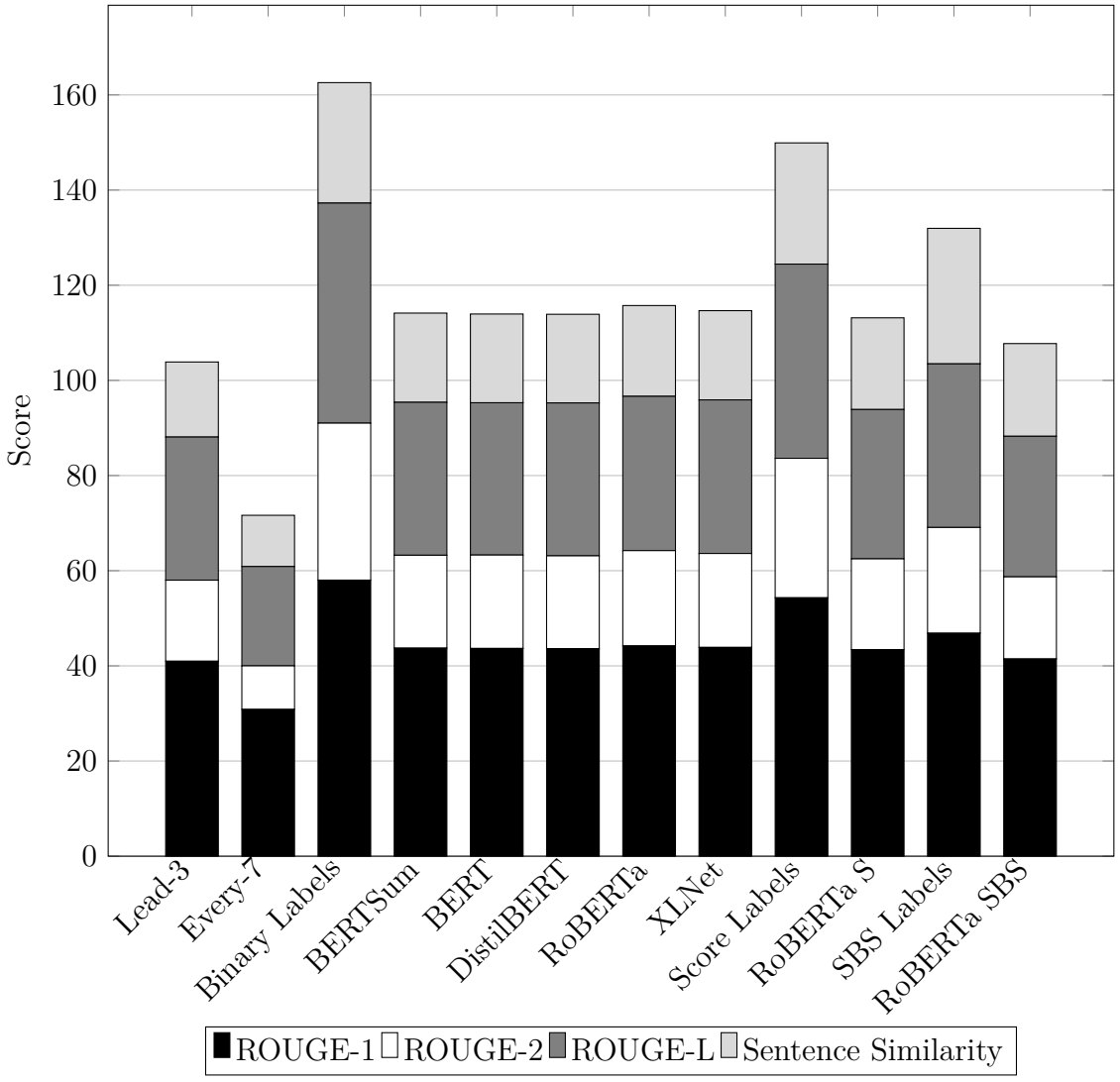


Figure 4.3: The combined scores on the truncated CNN/DM dataset.

4.2.2 Full Results

In the following, we investigate the models' performance on the full, un-truncated, CNN/DM dataset. The block splitting method described in Section 3.3.1 was used to get around the models' token limit. It is worth noting that this differs from how the models were trained, which will likely affect the results.

Evaluation on the full dataset resulted in a lower score for every model compared to the truncated results. Several models now perform worse than the Lead-3 baseline, but they still outperform Every-7. The BERTSum model saw a bigger drop than the BERT model, which could indicate that the segment embeddings cause issues when the block splitting method is utilized. The RoBERTa-based models still perform best, but this time the Score model (RoBERTa S) did overall best. The XLNet model saw the biggest drop in scores compared to the truncated results. Since there are now multiple blocks, the memory functionality sharing context across blocks can be utilised. This resulted in an increased score but it is still underwhelming when compared to the other models. The scores achieved by the different models on the full CNN/DM dataset are displayed in Table 4.3 and visualised in Figure 4.4.

Table 4.3: Scores on the full CNN/DM dataset. (S: Score, SBS: Sentence BERT Score)

No.	Model	ROUGE-1	ROUGE-2	ROUGE-L	Similarity	Mean
-	Lead-3	40.977	17.038	30.081	15.773	103.869
-	Every-7	30.850	9.150	20.880	10.767	71.647
-	Binary Labels	58.021	32.996	46.251	25.318	162.586
1	BERTSum	38.671	15.660	27.888	14.166	96.385
2	BERT	40.284	16.999	29.003	15.687	101.973
3	DistilBERT	40.354	17.008	29.253	15.766	102.381
4	RoBERTa	41.174	17.712	29.784	16.287	104.957
5	XLNet	35.140	12.815	25.084	11.123	84.162
5*	XLNet Mem.	37.569	14.591	26.945	13.420	92.525
-	Score Labels	54.310	29.314	40.819	25.477	149.920
6	RoBERTa S	41.539	17.601	29.550	17.785	106.475
-	SBS Labels	46.866	22.227	34.383	28.483	131.959
7	RoBERTa SBS	39.706	15.835	27.976	18.336	101.853

4.2.3 Discussion

In summary, using different pre-trained models to generate sentence embeddings had an affect on the metrics, although only a small one. We were able to achieve higher scores than BERTSum as presented in [2]. Using RoBERTa as a sentence embedder resulted in the highest scores. The current overall state-of-the-art is BERTSum with the 'large' BERT model, which we have not evaluated in our experiments. Using the 'large' RoBERTa model would likely lead to an even higher score and a new

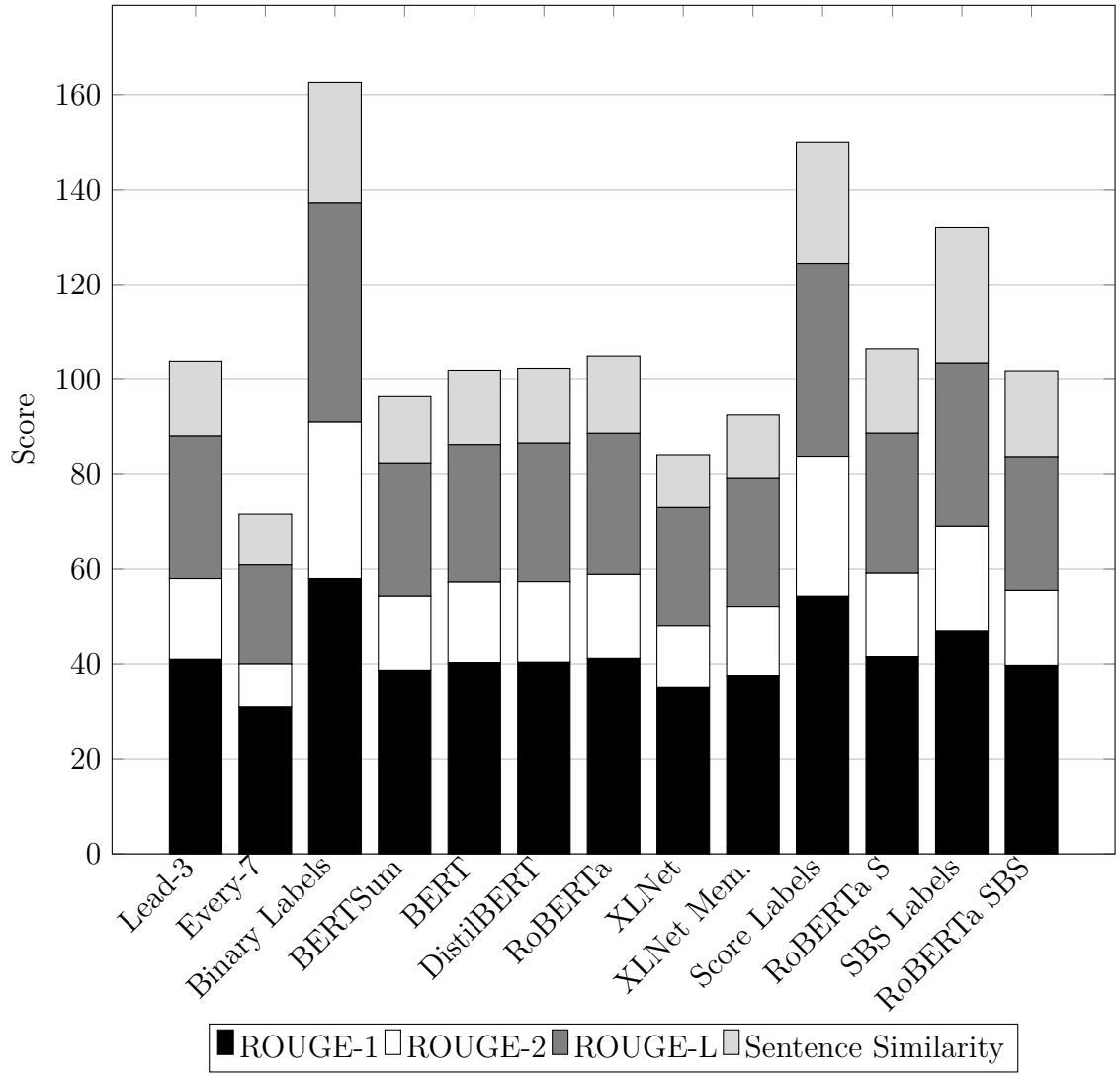


Figure 4.4: The combined scores on the full CNN/DM dataset. (S: Score, SBS: Sentence BERT Score)

state-of-the-art score for large models as well.

It is noteworthy how similarly DistilBERT performs to BERT, even surpassing it for some of the metrics. Considering the reduced model size and training time, this makes DistilBERT a very interesting choice compared to the other models.
XLNet did not perform well on the truncated dataset, especially considering the much longer training time it needed. When considering the full dataset, its results become even worse.

That the truncated dataset evaluation had better results than the full dataset could be partly be attributed to dataset bias explored further in Section 4.6. The full dataset might be considered a more "fair" evaluation, since the model is forced to pick from all sentences. But it could also be argued that this is not "fair" since the models were not trained for such a scenario, due to their token limit. Models have limits and exceeding them is not "fair" for the model.

The scores achieved on the different evaluation metrics did not change a lot between the various models. Since the label data produced scores that are quite a bit higher there should be room for improvement in the models. All of them were only a few points ahead of the Lead-3 baseline, though it is unclear what small score changes mean in terms of human readability. In the next section, this question will be explored further on the Academic Paper dataset summaries.

4.3 Academic Paper Dataset

In the following, the results of evaluating our models on the Academic Paper dataset will be presented. Our goal is to investigate how well the models are able to generate summaries of similar quality to the ones manually produced by us, and how well the models can transfer to the unfamiliar dataset, despite only having been fine-tuned on news data.. When looking at the following results, we ask the reader to keep in mind that the Academic Paper dataset consisted of only 10 papers with reference summaries and that any conclusions drawn from these results can therefore only be regarded as tentative. The summaries were generated without using tri-gram blocking as it resulted in lower scores for most models.

4.3.1 Results

As we had two reference summaries for each of the ten papers, the generated summaries were compared against both of them and the two scores were averaged for each of the models and metrics used. These averaged scores, as well as information on how the two original scores differed for each model, can be seen in Table 4.4. These results are visualised in Figure 4.5.

The Ref-CMP results show the scores of comparing the two reference summaries against each other. These scores show that there can be discrepancy even between

human summarisers. There is, for example, only 26% sentence overlap between the two reference summaries. We use these metrics as a rough upper bound on the scores we can expect from the generated summaries.

As a baseline we use Every-7, which selects every seventh sentence. This leads to summary lengths similar to the reference summaries. The baseline is surprisingly strong for this dataset, outperforming DistilBERT and almost reaching the same scores as BERT.

In Table 4.4 we also included the scores that each paper's abstract achieved compared against the reference summaries. The abstract can be seen as a kind of abstractive summary, although, as mentioned in the Introduction to this thesis report, often only a limited one. The ROUGE metrics seem to confirm this, and are much farther away from the Ref-CMP scores compared to the ones achieved by the models. This, however, can also be explained by the comparison of extractive against abstractive summaries, which can be expected to result in lower word-overlap and therefore lower ROUGE scores. The same is true for the Sentence Overlap metric, which is also very low for the abstracts. We don't see as big of a drop for the Sentence Similarity metric, which may indicate that it is able to capture similarity without exact word matchings.

We see a greater difference between the top and bottom performing models than we did on the CNN/DM dataset. Notably the DistilBERT model performs worse compared to the other models, than it did on the CNN/DM data. This could indicate that when transferring to a different dataset, the pre-training and the model architecture become more important and that DistilBERT is less able than the other models to adapt to different datasets.

The RoBERTa models performed well on this dataset, especially the score-and-select version, RoBERTa S, which performs slightly better than the other two.

The XLNet model's performance lies somewhere between BERT's and RoBERTa's, as was also the case on the truncated CNN/DM dataset. But when utilizing the memory functionality it becomes the top scoring model. We saw gains for XLNet Mem. on the full CNN/DM dataset as well, but for this dataset they are greater. A reason for this could be that the texts are much longer than those from CNN/DM, resulting in more blocks, and that therefore having access to context sharing between blocks becomes more important.

Table 4.4: The averaged scores the different models achieved on the Academic Paper dataset. In parenthesis, the difference between the score achieved when using reference summary 1 and the one achieved when using reference summary 2.

Method	ROUGE-1	ROUGE-2	ROUGE-L	Similarity	Overlap	Mean
Ref-CMP	61.22	49.72	57.36	44.25	26.49	47.81
Every-7	43.96 (2.34)	25.68 (3.86)	40.59 (2.71)	41.31 (1.57)	09.62 (0.88)	32.23
Abstract	32.27 (2.29)	12.67 (0.61)	23.77 (0.68)	39.97 (2.67)	00.00 (0.00)	21.74
BERTSum	44.92 (2.45)	26.43 (2.52)	41.41 (2.46)	41.39 (0.02)	09.89 (1.80)	32.81
BERT	46.37 (2.75)	28.67 (3.00)	43.11 (2.92)	41.21 (0.63)	11.96 (1.68)	34.26
DistilBERT	43.32 (1.03)	23.71 (1.70)	39.47 (1.27)	40.54 (0.19)	08.53 (0.65)	31.11
RoBERTa	49.75 (3.17)	32.34 (2.59)	45.72 (2.29)	42.50 (0.28)	14.69 (2.66)	37.00
RoBERTa S	51.31 (0.71)	34.60 (1.44)	48.04 (0.63)	42.51 (0.13)	13.99 (1.59)	38.09
RoBERTa SBS	50.71 (2.35)	35.02 (3.07)	46.95 (1.12)	43.42 (1.96)	15.10 (3.42)	38.24
XLNet	47.69 (4.32)	30.71 (4.70)	44.60 (4.80)	40.88 (1.55)	11.72 (2.54)	35.12
XLNet Mem.	54.26 (1.58)	39.08 (2.91)	50.94 (1.60)	43.48 (0.17)	18.25 (1.94)	41.20

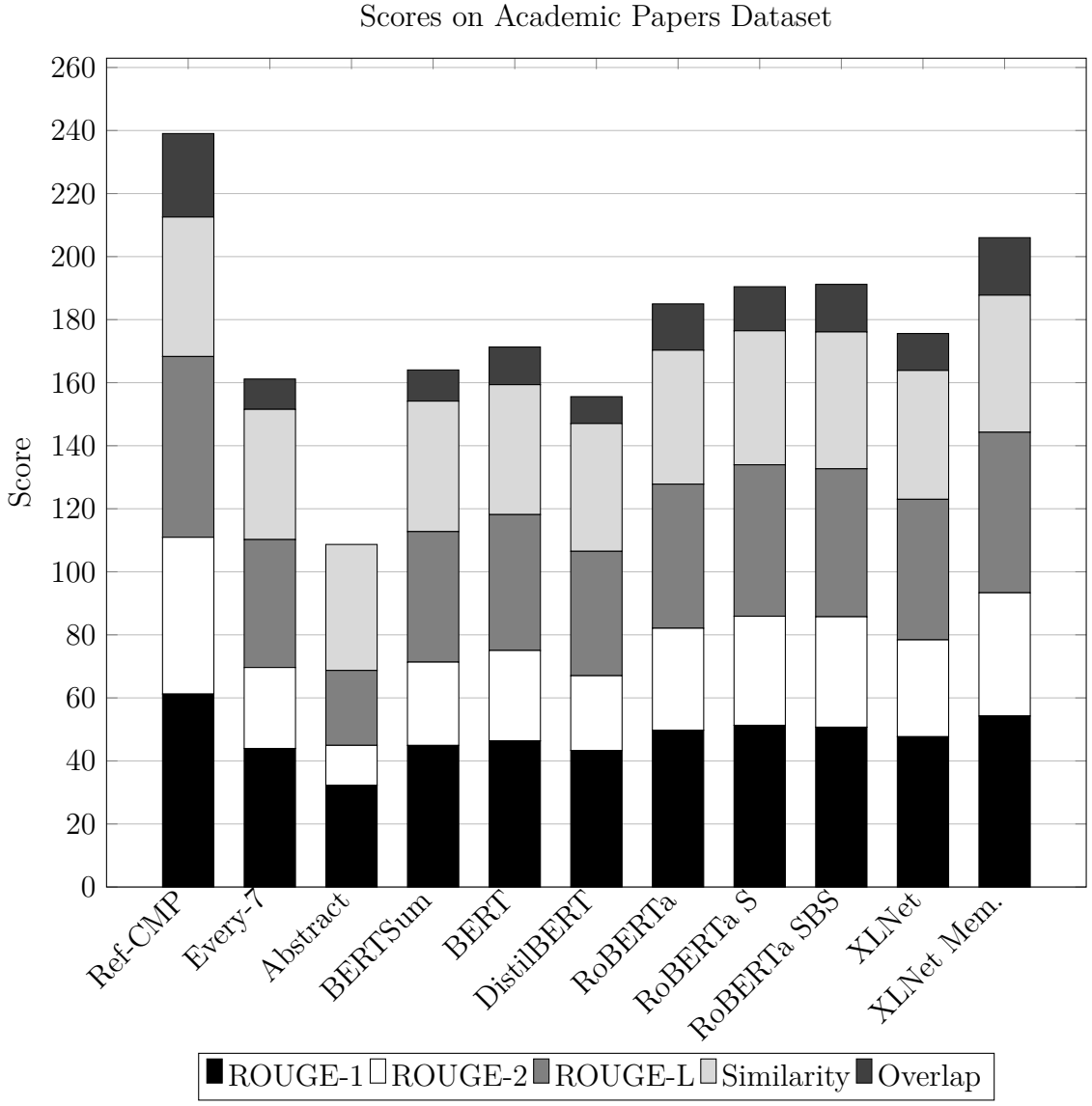


Figure 4.5: The scores on the Academic Paper dataset.

Table 4.5 shows rankings of each model based on each of the metrics. The top scoring model across all metrics is XLNet using memory. Generally, the metrics result in similar rankings, the ROUGE score even result in the exact same rankings. The Sentence Similarity metric ranks the models slightly differently. Predictably, it ranks the RoBERTa model "RoBERTa SBS", which has been trained to maximise the similarity metric, higher than the other two RoBERTa versions. BERTSum and Every-7 perform better by the standards of sentence similarity than those of any other metric, while the standard XLNet drops several ranks. For all metrics, the DistilBERT model is ranked at the very bottom.

Table 4.5: Rankings of the Academic Paper dataset summaries according to the different scores.

	Comb.	**R-1**	**R-2**	**R-L**	**Similarity**	**Overlap**
1	XLNet Mem	XLNet Mem	XLNet Mem	XLNet Mem	XLNet Mem	XLNet Mem
2	RoBERTa SBS	RoBERTa S	RoBERTa S	RoBERTa S	RoBERTa SBS	RoBERTa SBS
3	RoBERTa S	RoBERTa SBS	RoBERTa SBS	RoBERTa SBS	RoBERTa S	RoBERTa
4	RoBERTa	RoBERTa	RoBERTa	RoBERTa	RoBERTa	RoBERTa S
5	XLNet	XLNet	XLNet	XLNet	BERT-Sum	BERT
6	BERT	BERT	BERT	BERT	Every-7	XLNet
7	BERT-Sum	BERT-Sum	BERT-Sum	BERT-Sum	BERT	BERT-Sum
8	Every-7	Every-7	Every-7	Every-7	XLNet	Every-7
9	Distil-BERT	Distil-BERT	Distil-BERT	Distil-BERT	Distil-BERT	Distil-BERT

4.3.2 Single Sample Results

In addition to looking at the models' performance as measured by metrics, as we did above, we also wanted to evaluate the human-readability of the generated summaries. Because performing manual evaluation for every generated summary would have been too costly, we selected a random sample from the academic dataset instead. Table 4.6 shows how the summaries of this paper(full scores in appendix A.2), produced by the different models, were ranked by the automatic metrics, and how they were ranked by our human evaluation as described in Section 3.5.5. For this sample, the Every-7 baseline was very strong, at least according to the metrics, even beating all but the XLNet models on the combined scores. Manual evaluation, however, did not confirm this, as we found the summaries produced by Every-7 not very readable. Other than that we see many similarities of this sample with the combined averages presented in the previous section.

Some noteworthy observations that we made while reading the generated summaries and performing the manual ranking:

- We could not have said with complete certainty which of the generated summaries was our baseline "Every-7", but it generally read worse in terms of coherency. These summaries were the worst in both our rankings.
- Many of the summaries seemed to start out very good, but then often declined in quality as they went on.
- Sentences were often taken out of context. Sometimes sentences were put next to one another that even suggested wrong statements that were not actually in the original paper. A very alarming problem.

Table 4.6: Rankings of the sample summaries according to the different scores.

	Manual(1)	**Manual(2)**	**Combined**	**R-1**	**R-2**	**R-L**	**Similarity**	**Overlap**
1	RoBERTa SBS	RoBERTa	XLNet Mem	XLNet	XLNet Mem	XLNet	BERTSum	XLNet Mem
2	XLNet Mem	RoBERTa SBS	XLNet	Every-7	Every-7	XLNet Mem	XLNet Mem	XLNet
3	RoBERTa S	XLNet Mem	Every-7	XLNet Mem	XLNet	Every-7	RoBERTa S	RoBERTa
4	XLNet	RoBERTa S	RoBERTa	BERT	RoBERTa	RoBERTa	XLNet	Every-7
5	RoBERTa	XLNet	BERT	RoBERTa	RoBERTa S	BERT	Distil-BERT	RoBERTa S
6	BERT	BERT	RoBERTa S	RoBERTa S	BERT	RoBERTa S	RoBERTa SBS	BERT
7	BERTSum	BERTSum	BERTSum	BERTSum	BERTSum	BERTSum	BERT	BERTSum
8	Distil-BERT	Distil-BERT	Distil-BERT	Distil-BERT	RoBERTa SBS	Distil-BERT	RoBERTa	RoBERTa SBS
9	Every-7	Every-7	RoBERTa SBS	RoBERTa SBS	Distil-BERT	RoBERTa SBS	Every-7	Distil-BERT

- None of the summaries read very well. Partly, this is to be expected with extractive summaries, but our manually generated reference summaries demonstrate that in many places better flow of the text could be achieved.
- XLNet, as well as XLNet Mem., often seemed to pick rather "unique" sentences, which none of the other methods did. Sometimes these were good summary sentences, sometimes not.

Due to time constraints, we have not been able to read through all the generated summaries, or even a large fraction of them. We did, however, pick a few additional summaries at random and read them to corroborate our findings from the ranking. In doing this, we observed the following:

- There was no clear pattern of higher quality in the beginning of the summaries across the additional samples as there was in the single sample.
- Our observation that XLNet seems to pick different sentences than the other methods,seems to have been confirmed, or at least not disproved. Across the additional samples, XLNet and XLNet Mem. selected different sentences compared to the other models.
- Repetitions and sentences that were too out-of-context to be comprehensible could be found in most of the summaries, but especially in the ones produced by the lower-ranking methods.

Since these observations are based on only a small subset of the summaries, it is unclear with how much confidence any definitive statements about the quality of the summaries can be made.

4.3.3 Discussion

We were able to successfully utilize the block splitting method to produce summaries for the much longer academic texts. According to the metrics, the summaries produced by the XLNet Mem. model were quite close in quality to the ones produced by us. But when performing human evaluation, it became clear that many of the generated summaries were not very good in terms of readability. This is not something the evaluation metrics take into account, though we often found that they do reflect it to some degree. DistilBERT, for example, was consistently ranked very low by all the metrics, and when reading the summaries, we could confirm that these summaries read worst of all our generated summaries. The correlation between the scores and human judgement should, however, also not be overstated. Especially the ROUGE scores, but also the combined score, sometimes ranked the summaries produced by Every-7 rather high, as can be seen for our sample in Table 4.6 in Section 4.3.2. When reading the summaries, however, we found that Every-7 produced by far the worst summaries, often to the point of being very incoherent. This is confirmed by Kryściński's findings in [43], who performed experiments to determine the correlation between the ROUGE scores of automatically generated text summaries with human judgement and found that this correlation was weak, especially with regards to fluency and coherence in extractive models.

Some of the summaries for one of the academic papers, generated by the models,

as well its original abstract and one of our manual summaries for reference, can be seen in Appendix A.4.

4.4 Evaluation Metrics

Having reviewed the results on the two datasets, in this section we will discuss the usefulness of the different evaluation metrics.

4.4.1 ROUGE

The flaws of the ROUGE metric became apparent during the human evaluation for the Academic Paper dataset. The Every-7 method scored highly for the ROUGE metrics but was ranked as the worst summary text in the human evaluation. This makes us question the suitability of ROUGE as a metric for evaluating summaries. Since the model is attempting to maximise ROUGE scores during fine-tuning, this might limit the quality of resulting summaries in terms of human readability.

Despite being widely used, ROUGE has some important short-comings:

1. Lacking a way of representing semantics, ROUGE only considers exact matches of words/n-grams. This fails to capture more nuanced relations between them. For example, the words "sofa" and "couch" would not create a match, even though they refer to the same object. For another example, consider the following two sentences: "Animals have to eat." and "Dogs need food." They do not share any words and ROUGE would not detect any matches between them. But even though they do not say exactly the same thing, they clearly contain similar and related information.
2. ROUGE-1 also does not consider word order, which should be taken into consideration. For example, consider "Cop shoots criminal" and "Criminal shoots cop" these would have a perfect ROUGE-1 scores even tough they have very different meaning.
3. ROUGE only looks at word-/n-gram-matches, but has no way of explicitly evaluating how readable the texts as a whole are. A big problem in this regard can be unresolved *anaphors*. These are words like "it", "he", "which" and "this", which lose their meaning when taken out of context. They are a problem especially in extractive summarisation, as taking sentences out of their original context is how these summaries are created.

These limitations can lead to potentially good summaries being assigned a low ROUGE-score if they do not use the exact same words as the reference summaries. They can also cause summaries that don't make much sense to be assigned a high score, as long as they contain the right words.

4.4.2 Sentence Similarity

This metric is a rather experimental one, which is not established in the field of NLP like ROUGE is, and which we tried out of curiosity and in the hope that it

would ameliorate some of ROUGE's shortcomings. Namely, ROUGE's lack of taking semantic similarity into account. We could, however, observe very little difference between the Sentence Similarity scores of the various models, despite fluctuations in actual quality (as determined via human evaluation). This makes it a not very useful metric for our purposes. Since all of the scores were also very high, it is unclear what exactly a change of the magnitudes we observed indicates. Further evaluation of the metric would need to be performed. It is also uncertain how this metric correlates with the actual content and quality of the summaries. When compared against reference summaries of the wrong paper the score reflects this well, showing a greater difference than ROUGE. For the most part, however, the score does seem to correlate with the ROUGE metrics. Generating the score takes much more time than ROUGE, making it impractical when evaluating large datasets. A sentence similarity score is not something we have seen used or investigated much in the literature. This may be an interesting thing to do for future research, but given the results our experiments produced so far, we do not think it shows a lot of promise.

4.4.3 Human Evaluation

Human evaluation is often a tedious task, and especially so for summarisation tasks of long texts, as it requires a lot of time and concentration. In addition to this, the subjectivity of the task still introduces a lot of uncertainty in the obtained scores. This might be a reason for the reliance on automated metrics like ROUGE, but these metrics, as we discussed, are not a good substitute for human evaluation. Many of the features and problems with our generated summaries we only identified through human evaluation, as the scores did not sufficiently reflect them. This will remain a problem for summarisation tasks until perhaps a better metric is established. For now we can only stress the importance of human evaluation, as over-reliance on metrics such as ROUGE can give a distorted view of the actual quality of the generated texts.

4.5 CNN/DM Dataset Positional Bias

In this section we will investigate positional bias in sentence selection for the CNN/DM dataset. The presence of bias in this dataset was indicated by the Lead-3 vs. Every-7 baselines in our experiments and has also been mentioned in previous work by Kryściński [43]. In this section we will explore this bias further.

Figure 4.6 shows the averaged label data over each position for the test (a) split and the full (b) CNN/DM dataset respectively. The score indicates confidence levels as to whether or not the sentence should be included in the summary. It shows that both sets are clearly biased towards selecting early sentences. This is most likely due to the fact that news articles will often front load important information to hook a reader. This is something that Kryściński also notes in [43] as a potential problem with fine-tuning on the CNN/DM dataset. This bias could cause problems when fine-tuning a model for later use on different kinds of texts. The model might learn

to highly value early sentences, hurting its ability to transfer to a dataset which does not share this bias. We will look further into this in the next section.

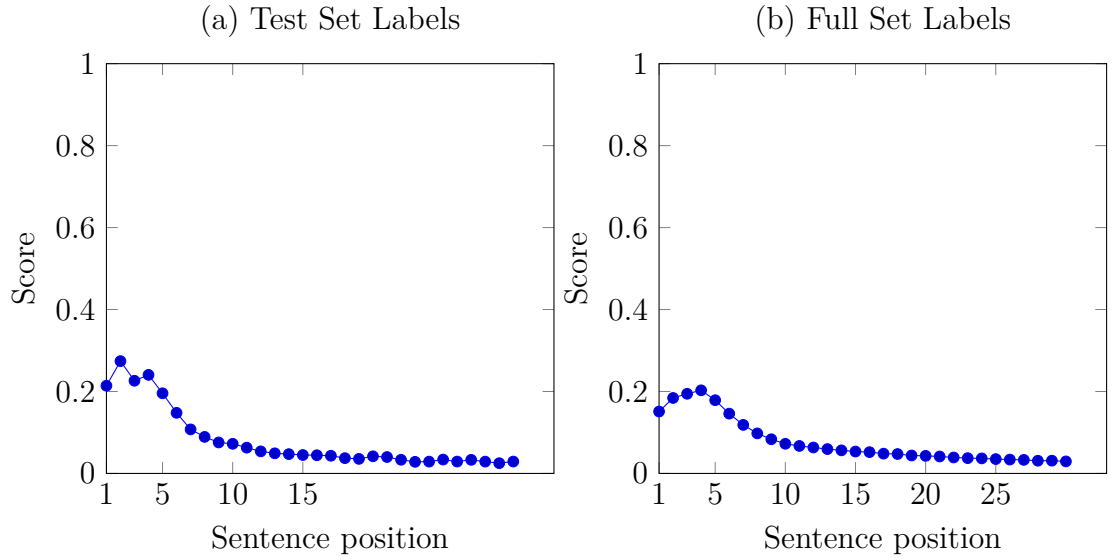


Figure 4.6: Selection scores of the sentences of the CNN/DM dataset with respect to sentence position.

4.6 Model Positional Bias

In this section we will investigate how the CNN/DM bias has affected the fine-tuned models and their ability to perform well on other types of documents. For this purpose we will compare the scores the models achieved on the CNN/DM dataset and on the Academic Paper dataset.

4.6.1 CNN/DM Dataset

The results in this section were obtained by using a subset of the CNN/DM test split data, where each sample had at least 18 sentences. This was done to get texts with an appropriate number of sentences for better comparison against the academic texts. The subset contains 4155 samples which is 36% of the total test split. Figure 4.7(a) shows the averaged model output over each position and shows that the models clearly favour early positions. This, however, is not necessarily indicative of learned bias. It could also be a result of the model correctly selecting the early sentences, as they are indeed the most important ones. To investigate how much the model relies on positional information, we performed another evaluation where the sentences were randomly shuffled into a different order, obscuring the original sentence positions. The results can be seen in Figure 4.7 (b). The models still show a slight bias towards the early sentences but less so than before. This means that the model has learned to somewhat favour early sentences but it is the combination of the position and other features of the sentence that lead to the scores in 4.7(a).

This indicates that the model is not overly reliant on position alone and the bias of the dataset has not fully transferred to the model. This is a good sign for the model's ability to transfer to other datasets.

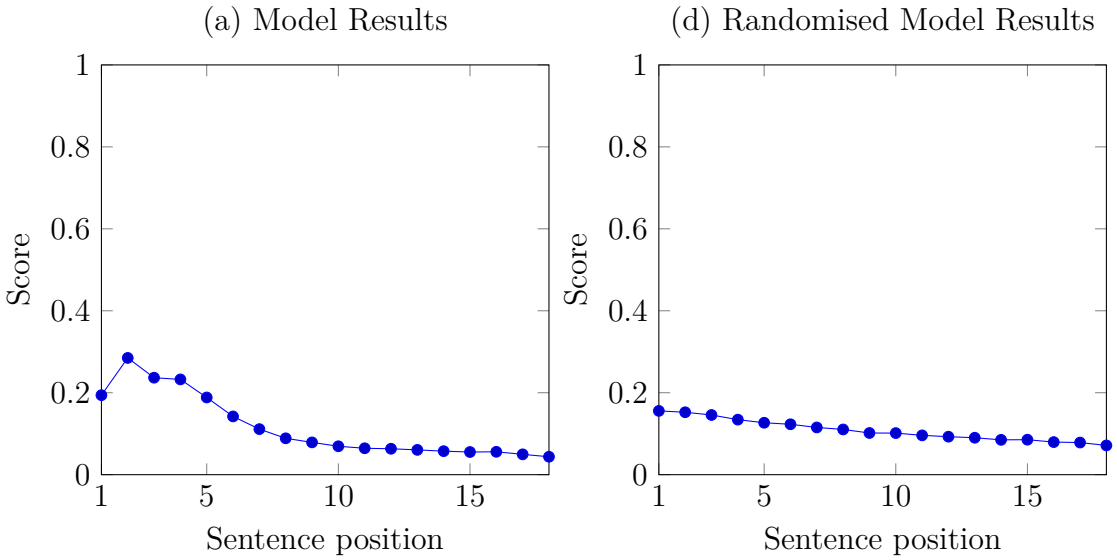


Figure 4.7: Selection scores of the sentences with and without randomised positions.

As we noted in Section 4.1, we also suspect slight overfitting of our models to the training data, which may exacerbate potential positional bias. It is likely that preventing overfitting will also ameliorate positional bias, though due to lack of time we did not try to confirm this experimentally.

4.6.2 Academic Paper Dataset

In this section, we are investigating whether or not the positional bias of the CNN/DM dataset affected the models' performance on the Academic Papers dataset. The XLNet Mem. model performed surprisingly well during our experiments on the Academic Paper data, which makes further inspection interesting. We will therefore in particular look at the effect the memory functionality had.

Figure 4.8(a) shows the scores over sentence positions on a subset of the Academic Paper dataset, exluding one of the texts, which was an outlier in terms of length. The red vertical lines signify a block split. If the model had positional bias, we should see the same pattern repeated within each block. But as can be seen, the highest scoring positions vary between blocks. Although the majority of the top scoring sentences are among the first three sentences, the pattern is not as clear as with the CNN/DM data. Therefore, even though the positional bias clearly is a risk in theory, in practise, it does not seem to have affected our models very much.

Figure 4.8(b) shows the scores when the memory functionality is utilised. The memory comes into play from the second block onward, and from that point on

we also start to see a difference in the scores. For example, in the second block the XLNet model has the two highest scoring sentences in the first four positions while XLNet Mem. sees the highest score at the ninth and fourth positions. The memory functionality seems to reduce positional bias. Additionally, the peaks are lower with memory. This could potentially make sentence selection more difficult, as there won't be as clear "best" sentences.

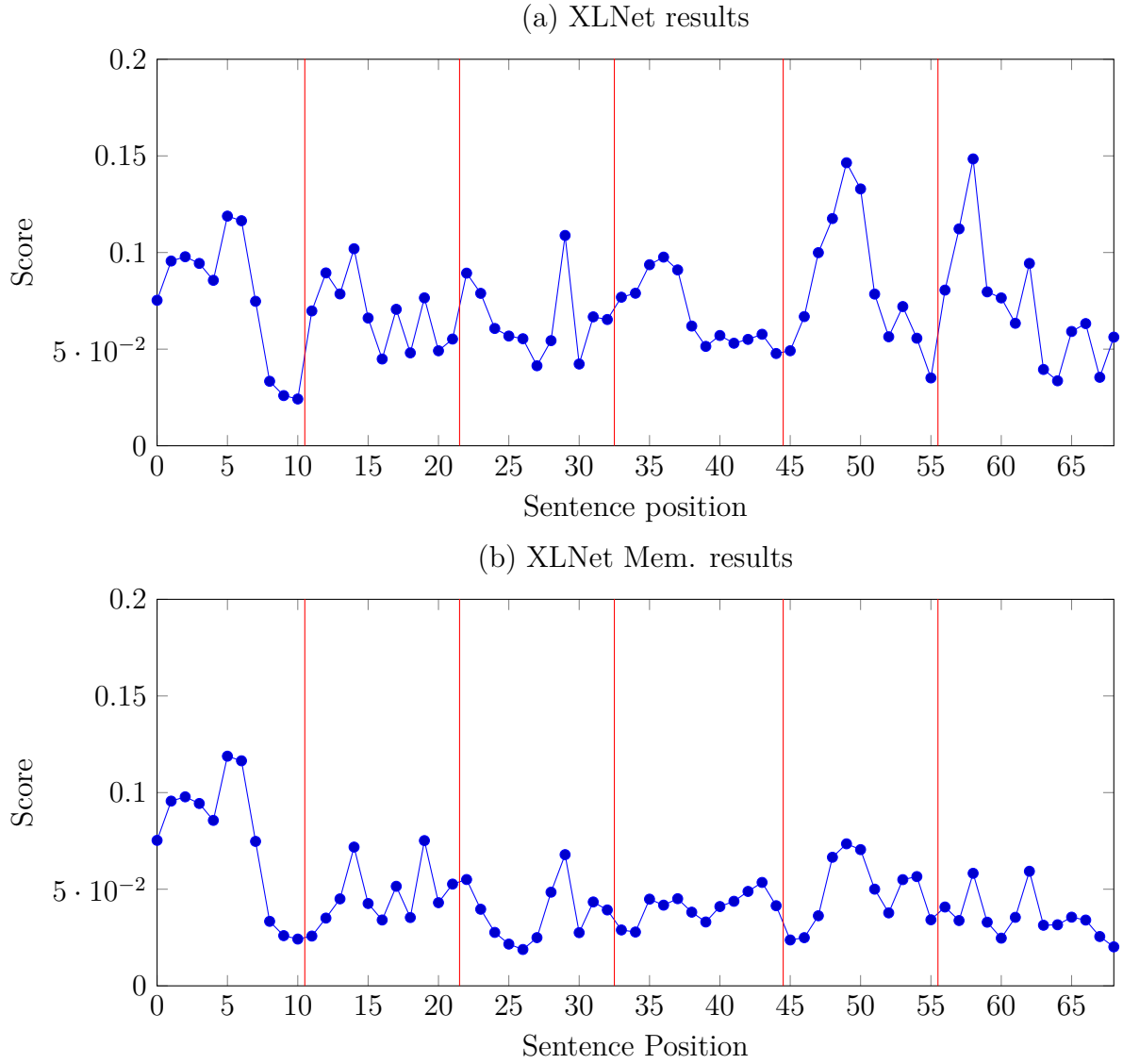


Figure 4.8: Averaged scores of XLNet and XLNet Mem. on the Academic Paper dataset with regards to sentence position. (Red Line signifies block split)

4.7 Model Confidence

In this section we will take a closer look at model confidence. The model output score for each sentence can be interpreted as a measure of how confident the model is that a sentence should be in the summary, ranging from 0, not confident at all, to 1, very confident. Figure 4.9 shows plots of the averaged sorted confidence scores, such that the first position contains the average score of the highest-ranking sentence from each paper, and so on. This gives us an idea of how confident the model is in the sentences it picks for the summary. The red vertical line indicates where the cutoff for the summary is made. We decided to only show the RoBERTa plots here. The full plots can be seen in Appendix A.3
The plots have similar distributions across the datasets, which is a good sign. If the scores for the Academic Paper dataset, for example, resulted in a horizontal line, it would have meant that the model could not differentiate between good and bad sentences to pick for the summary. In that case we would have had to conclude that the model was not able to generalize and could not transfer to a different dataset.

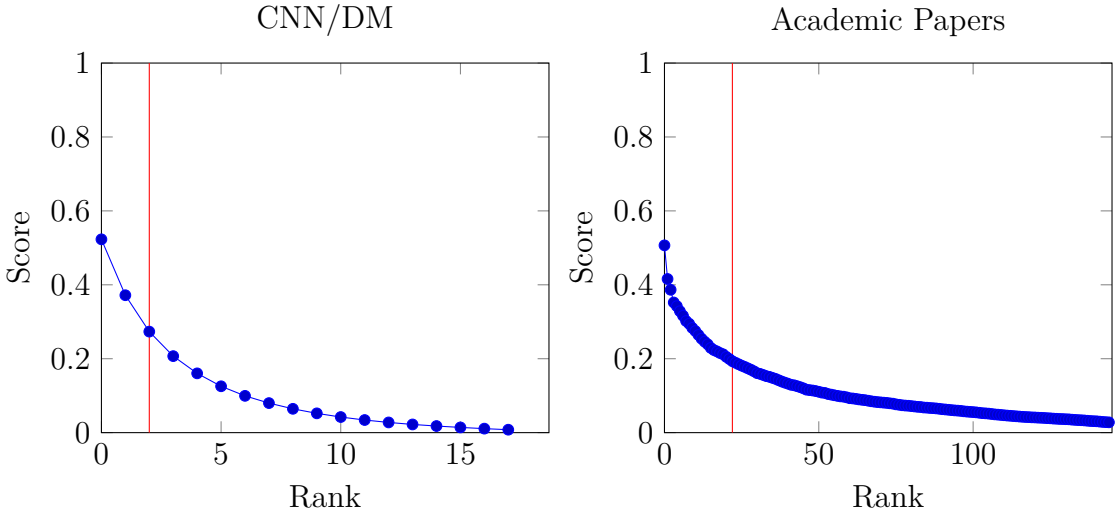


Figure 4.9: Plots of RoBERTa Confidence scores

The plots indicate that the model should be able to generate summaries longer than the limit of 3 it was trained on, although with each additional sentence it would become increasingly harder for it to pick the best sentences, as confidence scores become very low.

Table 4.7 shows the confidence scores of RoBERTa. We see very similar top confidence scores for both of the datasets. Ideally the maximum value would be as close to 1 as possible. But as long as there is a clear difference among the sentences' scores, a distinction between summary and non-summary sentences can be made with reasonable confidence. Based on the "Summary mean" confidence value, we can say that the model is more confident in its prediction of the CNN/DM summaries than the Academic Paper summaries, which is to be expected, due to it being

fine-tuned on the CNN/DM data. But the results are similar enough to say that the model seems to be able to transfer well to the very different Academic Paper dataset.

Dataset	Max	Min	Mean	Std.	Cutoff	Sum. mean	Sum. std.
CNN/DM	0.523	0.008	0.119	0.139	0.273	0.389	0.103
Texts	0.507	0.028	0.112	0.089	0.205	0.291	0.075

Table 4.7: Statistic of RoBERTa confidence scores

Looking at the confidence metrics for the XLNet models in Figure 4.10 on the Academic Paper dataset, we see some interesting results. Both XLNet models show lower confidence compared to RoBERTa. Even though the XLNet Mem. model achieved the best scores in the evaluation, it shows the lowest average confidence scores. Ideally we would like to see a higher confidence, but as the XLNet Mem. model nonetheless achieved the best results, the most important thing seems to be that a distinction between higher- and lower-scoring sentences can be clearly made.

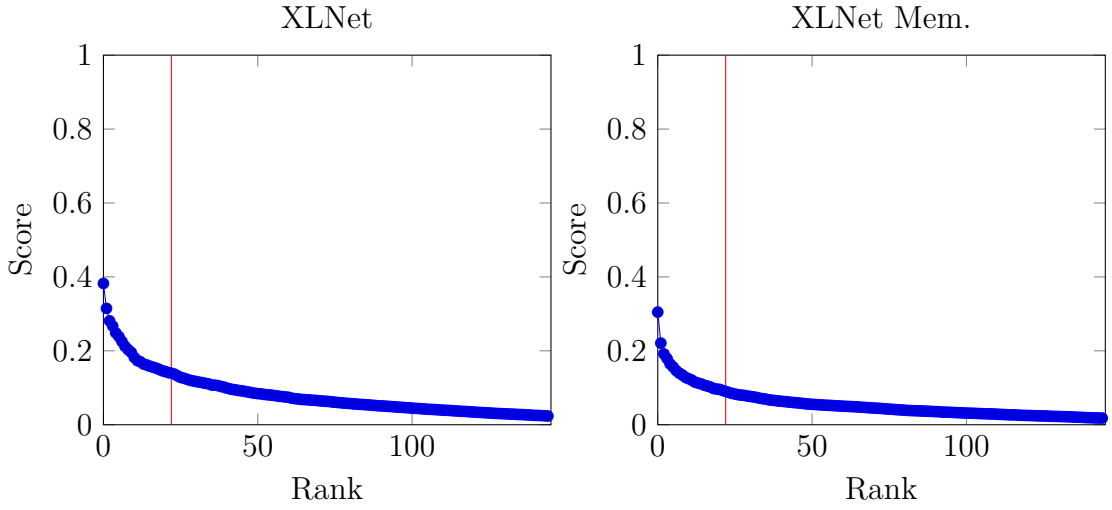


Figure 4.10: Plots of XLNet and XLNet Mem. Confidence scores

Figure 4.11 plots the average top scores across the datasets for the RoBERTa-S model. Comparing them to those of RoBERTa from Figure 4.9, we see a clear difference. The curve for the S model is almost linear, giving a more even distribution of the sentence scores. The purpose of training this model was that it would be better at generating longer summaries and this curve indicates that it is.

Based on the values in Table 4.8, RoBERTa S's confidence scores for the two datasets are quite similar, but the minimum value for the Academic Paper data is about twice as high as for the CNN/DM data. This could indicate that the model is not able to differentiate sentence at the lower level as well, but this is a minor problem, since it is unlikely that summaries of this length would need to be generated.

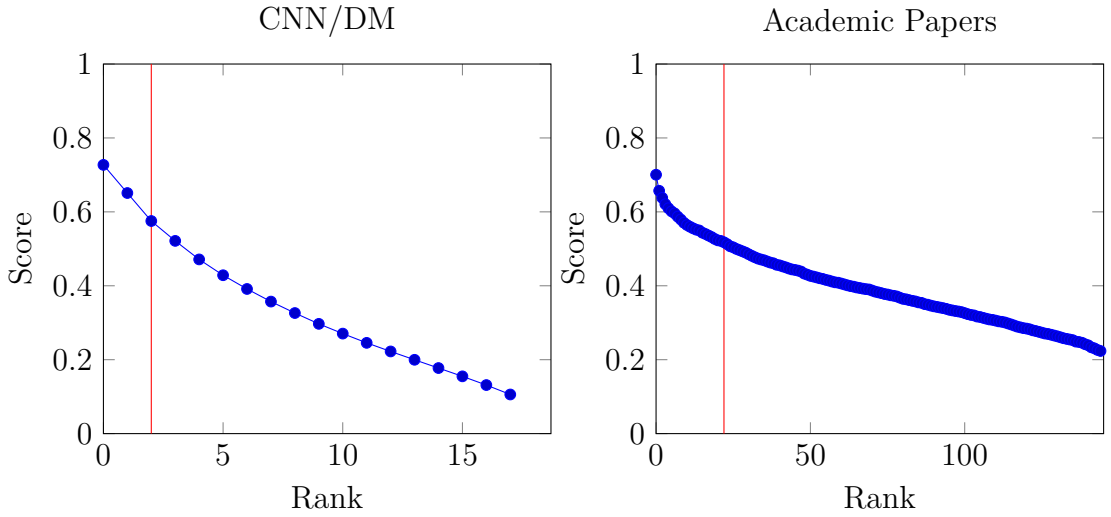


Figure 4.11: Plots of RoBERTa S Confidence Metrics

Dataset	Max	Min	Mean	Std.	Cutoff	Sum. mean	Sum. std.
CNN/DM	0.727	0.106	0.347	0.178	0.575	0.651	0.062
Texts	0.700	0.224	0.395	0.106	0.523	0.578	0.045

Table 4.8: Statistics of RoBERTa S Confidence Scores

5

Conclusions and Future Work

In this chapter we will present our closing thoughts. Section 5.1 addresses the ethical questions the topic of our thesis project poses. We then discuss the limitations of our project and findings in Section 5.2. This chapter concludes with our closing thoughts on the project in Section 5.3 and ideas for future work in Section 5.4.

5.1 Ethical Considerations

A possible ethical concern is that automatically generated summaries might misrepresent the text they summarise. This is an issue on multiple levels: Firstly, automatically generating a summary forces the original text's author(s) to give up a certain amount of agency over their text. When authors write their own summaries, they are, at least to some degree, in control of how their text is represented and will be perceived by a reader. When a machine does it, however, the summary might highlight things that the author(s) themselves consider to be of little importance while the main points the author tried to make, might be missed. This problem is compounded by the fact that sometimes not only is emphasis shifted, but the text may be summarised in such a way that the meaning is downright *changed* from the authors' original intent. This can happen in extractive summarisation when sentences are presented out of their original context. Causes and effects might be suggested between statements where originally none existed and important qualifiers can be left out. We have seen this in our experiments and consider it a serious problem.

The issue that we just described feeds into the second issue: Not only are there ethical concerns with regards to author agency, but also with regards to the information that is given to the reader. In manual summaries, it is possible for the authors to make judgement calls on which nuances of the text are important to preserve, in order not to give an over simplified version of the text or even a falsification. In automatic summarisation, however, this can be an elusive concept to model and the generated summaries might not always give an accurate impression of the complexity of the topic.

For some topics, even misrepresentations of the original texts might be of limited consequence, but in other areas, there might be serious implications. If we imagine a sequence of sentences like "Autism rates have been increasing since the wide-spread use of vaccines. However, no link between the two has been established.", it is easy

to see that including the first sentence in the summary, but leaving out the second one would falsify the entire statement, potentially leaving the reader with a downright dangerous conclusion about the topic.

Reading the summary might lead a reader to believe that they already know "all the important points" of the original text and suggest to them a false sense of expertise which they do not actually possess. If this sense of expertise leads to them deciding against reading the original text, there is a risk that automatically generated summaries could impede the transfer of information and knowledge rather than helping it.

Since the kind of judgements involved in the issues above can be difficult to model, and since neural networks tend to be opaque in their workings in general, it can be hard to calculate these risks, but they should always be considered and measures should be taken against them, especially when the summaries are intended for use by important decision makers.

5.2 Limitations

Shortcomings of the ROUGE evaluation metric
Despite being widely used, ROUGE has some important short-comings as discussed in Section 4.4. As the models are trained on label data generated using the ROUGE metric, the summaries quality will be affected by this.

Important aspects of the summaries not considered
We did not take factual correctness of the summaries into account when evaluating their quality. The automatic scores do not capture this property, and we did not have enough expert knowledge to judge the truthfulness of the summaries, nor the time to check every single statement against the original papers, as putting the original sentences into a different context may have distorted their meaning to the point of falsehood.

The size of the Acadmic Paper Dataset
As mentioned before, an important limitation of our experiments was the very small size of the Academic Paper dataset, which only consisted of 10 papers with two reference summaries each. From such a small dataset, it is hard to say which of our results are universally applicable and which are results of specific features of the few papers we collected, as even just a few outliers would have significant effects on the results.

Unclear definition of "a good summary"
What constitutes a good summary is very subjective, we worked with a rather fuzzy definition of what "a good summary" even is. Kryściński convincingly argues in [43] that the notion of "a good summary" depends very heavily on who that summary is intended for, and for what purpose. Someone new to the topic might have very different requirements for a "good summary" than an expert. The notion of a uni-

versally "good" or "bad" summary is a flawed concept.

Limited fine-tuning of the models
Our fine-tuning was performed on the truncated CNN/DM data-set. Utilizing the full texts might have led to better results. In particular the XLNet model might have performed better when trained on the full CNN/DM dataset utilizing memory. We did not perform this type of fine-tuning because of time limitations.

Fine-tuning on CNN/DM and transferring to the Academic Paper dataset does not help the model learn corpus-specific features. For example, sentences with references to figures and tables are commonplace in academic texts but should be avoided for summarisation. The models could only have learned this, had they been fine-tuned on academic paper data. This, however, was not possible for us due to the limited size of our dataset.

No investigation of different selection layer architectures
While we did test different language models for sentence embedding, and saw how those affected the quality of the generated summaries, the choice of selection layers could also be an important variable. We were stuck with the selection layers of the original BERTSum architecture and did not investigate different ones.

Limited human evaluation
Due to the many models and methods we tested, and the limited time we had available for this project, we were only able to manually look at a small subset of the generated summaries. The confidence with which we can draw conclusions about their quality from that is limited.

Since we did not perform human evaluation on the generated CNN/DM summaries, it is hard to compare the results of both datasets directly. It is hard to tell which of the discrepancies we observed between the scores of the generated Academic Paper summaries and our human judgement of them are due to the nature of the models themselves, and which are related to a potential lack of transfer learning. Ideally we would have liked to evaluate the transferability on a reference dataset, but few are freely available.

5.3 Conclusions

The goal of this project was to produce summaries for a set of academic papers. We achieved this by utilizing a modified BERTSum architecture fine-tuned on news data. We also set out to evaluate and compare several different pre-trained models as sentence embedders. This led to new SOTA results on the CNN/DM dataset when using the more robustly pre-trained RoBERTa model. But there were only minor increases in terms of ROUGE scores and there is still a large gap to the theoretical maximum scores of the label data, so there is still plenty room for improvements. When summarising the academic papers the differences in score between sentence embedders increased, indicating that the more robustly pre-trained models, such as

RoBERTa, is better at generalization.

Extractive summarisation is a complex problem that the current models seem to have difficulties with, as implied by their performance against the baselines. The selection layer might be a limiting factor for generalization, something we did not investigate. Using a deeper selection layer might be better at capturing sentence relations to better select correct sentences. Additional training could also improve the sentence selection, but steps would need to be taken to ensure its ability to generalize and avoid over-training on the news data.

The CNN/DM dataset might not be the best option for fine-tuning a general model, since the dataset has positional bias. This bias is to some degree replicated by the model, but does not seem to overpower other sentence features. Regardless of this, the model is able to make fairly clear decisions on sentences from a very different dataset, which indicates that the models have some ability to generalise.

Finally, let us assess the usefulness of the generated summaries for academic papers. Do these summaries have an advantage over the common practise of reading through abstracts, introductions and conclusions to gain an overview of the topic? If we go by the evaluation metrics alone, the XLNet Mem. model seems to create summaries that are almost on par with our human generated summaries. But the results of the human evaluation put this into question. During human evaluation it became evident that the summaries are flawed, not only in terms of readability and cohesion but also in terms of misrepresenting information. The summaries can, for example, combine sentences in such a way that they portray the opposite of the original meaning. Consider the following example, a sentence implies something and the following sentence negates it. If only one of them is selected the meaning will be misrepresented. The negating sentence might also be selected in combination with some earlier sentence, wrongly negating that one in the summary. This would not happen when reading the original text directly. Considering this, we think reading the abstract, introduction and conclusion to be the better choice.

Our manually created summaries, however, give us hope that better extractive summaries are possible and may be a useful tool, if ways can be found to automatically generate them. We also want to stress the need for better evaluation metrics than the ones currently available, as well as for more nuanced definitions of what a good summary is for a given purpose, and - due to the current lack of suitable metrics - human evaluation.

5.4 Future Work

Better methods and metrics for the evaluation of summaries, both manual and automatic, need to be explored. The metrics should take things like fluency, coherency and factual correctness into account and ideally be tailored towards more specific notions of what a good summary for a specific purpose and target audience is.

We only explored extractive summarisation, future work could also explore abstractive summarisation methods. There are also methods that combine both extractive and abstractive methods into a combined model, which might be interesting to investigate.

We focused on the encoding/embedding layer. Improvements to the selection layer could also be explored. The block splitting method could also be adjusted to combine the sentence embeddings before they are passed to the selection layer, avoiding the context splitting for that layer.

Other methods for producing sentence embeddings could also be explored, rather than using the [CLS]-token. There are alternatives like the mean of all the sentences' tokens embeddings.

We did not explore training on the "full" CNN/DM dataset. This would allow the model to utilise all the training data. In addition to this, the XLNet model's memory functionality could be utilised during training.

Additional pre-processing could be performed, which could take text structure into account when splitting texts into blocks. This could help split the context in a more sensible way, allowing context to be better maintained within paragraphs and sections of the text.

Performing training/fine-tuning solely on academic papers would also be an interesting avenue to explore. But for that to be possible, a suitable dataset would first need to be created. To generate such a dataset, we identified some obstacles that would need to be solved: Firstly, academic papers often come in the form of PDF documents, as such the text needs to be extracted to a more suitable format. For this purpose a competent PDF-text extractor is required. Secondly, extractive summarisation requires that the text is split into sentences. For this purpose, competent sentence tokenisers are required. Finally, label data is required and the manual generation of summaries as label data is a very time consuming process. Some method for generating label data for a large amount of texts is required. Another suggestion for dataset generation: Students will many times be tasked to produce a summary for a paper for their courses, perhaps this could be used as a method for gathering summaries.

We believe that the transferability between datasets discussed in this thesis is worth further exploration. This should ideally be done with more robust datasets that are similar in size to the original CNN/DM one and where reference scores are available, to better assess the effects of the transfer specifically. If two datasets were available, a cross-training/evaluation procedure could be employed, where models are fine-tuned and evaluated on each dataset followed by evaluating on the other dataset comparing the results.

Bibliography

[1] A. See. (2017). Taming recurrent neural networks for better summarization, [Online]. Available: http://www.abigailsee.com/2017/04/16/taming-rnns-for-better-summarization.html (visited on 09/18/2019).

[2] Y. Liu, "Fine-tune BERT for extractive summarization", *CoRR*, vol. abs/1903.10318, 2019. arXiv: 1903.10318. [Online]. Available: http://arxiv.org/abs/1903.10318.

[3] M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut, "Text summarization techniques: A brief survey", *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 10, 2017. DOI: 10.14569/IJACSA.2017.081052.

[4] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries", in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81.

[5] K. Al-Sabahi, Z. Zuping, and M. Nadher, "A hierarchical structured self-attentive model for extractive document summarization (hssas)", *IEEE Access*, vol. PP, pp. 1–1, Apr. 2018. DOI: 10.1109/ACCESS.2018.2829199.

[6] A. Fabbri, I. Li, T. She, S. Li, and D. Radev, "Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model", pp. 1074–1084, Jul. 2019. DOI: 10.18653/v1/P19-1102.

[7] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom, "Teaching machines to read and comprehend", *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., pp. 1693–1701, 2015.

[8] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation", *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311–318, 2002.

[9] E. Reiter, "A structured review of the validity of BLEU", *Computational Linguistics*, vol. 44, no. 3, pp. 393–401, Sep. 2018. DOI: 10.1162/coli_a_00322.

[10] E. Sulem, O. Abend, and A. Rappoport, "BLEU is not suitable for the evaluation of text simplification", *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 738–744, Oct. 2018. DOI: 10.18653/v1/D18-1081.

[11] J. Steinberger and K. Ježek, "Evaluation measures for text summarization", *Computing and Informatics*, vol. 28, no. 2, pp. 251–275, 2012.

[12] Z. S. Harris, "Distributional structure", *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[13] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space", *CoRR*, vol. abs/1301.3781, 2013.

[14] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation", *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[15] Q. Le and T. Mikolov, "Distributed representations of sentences and documents", *International conference on machine learning*, pp. 1188–1196, 2014.

[16] M. Campr and K. Ježek, "Comparing semantic models for evaluating automatic document summarization", *International Conference on Text, Speech, and Dialogue*, pp. 252–260, 2015.

[17] A. M. Dai, C. Olah, and Q. V. Le, "Document embedding with paragraph vectors", *ArXiv*, vol. abs/1507.07998, 2015.

[18] R. Józefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the limits of language modeling", *ArXiv*, vol. abs/1602.02410, 2016.

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding", in *NACL*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[20] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding", in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 5753–5763.

[21] J. Schmidhuber, "Deep learning in neural networks: An overview", *Neural networks*, vol. 61, pp. 85–117, 2015.

[22] Y. Kim, "Convolutional neural networks for sentence classification", in *EMNLP*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751. DOI: 10.3115/v1/D14-1181.

[23] S. El Hihi and Y. Bengio, "Hierarchical recurrent neural networks for long-term dependencies", in *Proceedings of the 8th International Conference on Neural Information Processing Systems*, ser. NIPS'95, Denver, Colorado: MIT Press, 1995, pp. 493–499.

[24] Y. Goldberg, "A primer on neural network models for natural language processing", *ArXiv*, vol. abs/1510.00726, 2015.

[25] J. L. Elman, "Distributed representations, simple recurrent networks, and grammatical structure", *Machine Learning*, vol. 7, no. 2, pp. 195–225, Sep. 1991, ISSN: 1573-0565. DOI: 10.1007/BF00114844.

[26] B. Hanin, "Which neural net architectures give rise to exploding and vanishing gradients?", *Advances in Neural Information Processing Systems*, pp. 582–591, 2018.

[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

[28] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation", in *EMNLP*, 2014, pp. 1724–1734. DOI: 10.3115/v1/d14-1179.

[29] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate", *CoRR*, vol. abs/1409.0473, 2014.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need", in *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., 2017, pp. 5998–6008.

[31] R. Al-Rfou, D. Choe, N. Constant, M. Guo, and L. Jones, "Character-level language modeling with deeper self-attention", in *Thirty-Third AAAI Conference on Artificial Intelligence*, 2019.

[32] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context", in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 2978–2988. DOI: 10.18653/v1/P19-1285.

[33] Z. a. Yonghui Wu, "Google's neural machine translation system: Bridging the gap between human and machine translation", *ArXiv*, vol. abs/1609.08144, 2016.

[34] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books", *Proceedings of the IEEE international conference on computer vision*, pp. 19–27, 2015.

[35] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach", *ArXiv*, vol. abs/1907.11692, 2019.

[36] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners", *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[37] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression", *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541, 2006.

[38] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network", *ArXiv*, vol. abs/1503.02531, 2015.

[39] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks", in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3982–3992. DOI: 10.18653/v1/D19-1410.

[40] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering", *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015. DOI: 10.1109/cvpr.2015.7298682.

[41] K. M. Hermann, T. Kočiský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom, "Teaching machines to read and comprehend", in *Proceedings of the 28th International Conference on Neural Information Pro-*

cessing Systems - Volume 1, ser. NIPS'15, Montreal, Canada: MIT Press, 2015, pp. 1693–1701.

[42] H. Xiao, *Bert-as-service*, `https://github.com/hanxiao/bert-as-service`, 2018. (visited on 02/25/2020).

[43] W. Kryscinski, N. S. Keskar, B. McCann, C. Xiong, and R. Socher, "Neural text summarization: A critical evaluation", *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 540–551, 2019.

[44] Y. L. Murphey, R. Milton, and L. Kiliaris, "Driver's style classification using jerk analysis", *2009 IEEE Workshop on Computational Intelligence in Vehicles and Vehicular Systems*, pp. 23–28, 2009.

A

Appendix 1

A.1 Full Text Scores

The Table A.1 shows the full scores on the Academic Texts datasets when the generated summaries are compared against both the reference summaries.

Method	Rouge-1	Rouge-2	Rouge-L	Similarity	Overlap
Ref-CMP	61.215	49.724	57.362	94.249	26.491
Every-7(1)	45.130	27.612	41.941	90.524	10.056
Every-7(2)	42.789	23.748	39.236	92.096	09.173
BertSum(1)	43.693	25.241	40.010	91.397	09.147
BertSum(2)	46.250	27.865	42.660	91.390	10.976
Bert(1)	44.995	27.167	41.650	90.895	11.114
Bert(2)	47.745	30.171	44.574	91.527	12.796
DistilBert(1)	42.800	22.861	38.830	90.440	08.206
DistilBert(2)	43.832	24.565	40.099	90.632	08.860
Roberta(1)	51.334	33.633	46.863	92.641	16.020
Roberta(2)	48.167	31.039	44.578	92.358	13.358
XLNet(1)	45.532	28.359	42.199	90.099	10.452
XLNet(2)	49.852	33.055	47.000	91.652	12.995
XLNet Mem(1)	54.658	40.022	51.306	93.202	18.603
XLNet Mem(2)	53.357	37.413	50.062	93.249	17.214
Roberta S(1)	51.664	35.320	48.347	92.444	14.787
Roberta S(2)	50.958	33.879	47.722	92.569	13.194
Roberta SBS(1)	51.884	36.551	47.513	94.398	16.811
Roberta SBS(2)	49.530	33.483	46.396	92.435	13.391

Table A.1: Scores on Text Dataset against both the reference summaries.(1: compaired against reference summaries 1, 2: compaired against reference summaries 2, GS: Greedy Score, SBS: Sentence Bert Similarity)

A.2 Academic Texts: Single Sample Scores

The Table A.2 shows the scores of the single sample of the Academic Texts dataset.

Table A.2: The scores the different models achieved on the selected sample of the Academic Paper dataset. In parenthesis, the difference between the score achieved when using reference summary 1 and the one achieved when using reference summary 2.

Method	Rouge-1	Rouge-2	Rouge-L	Similarity	Overlap	Comb.
Ref-CMP	55.56	41.57	52.16	43.12	16.67	41.82
Every-7	52.42 (00.60)	36.10 (01.04)	49.10 (04.67)	40.64 (00.92)	10.10 (03.06)	37.67
Abstract	30.47 (01.74)	11.25 (00.84)	20.86 (02.61)	40.39 (00.17)	00.00 (00.00)	20.59
BertSum	48.10 (01.14)	30.00 (03.96)	44.92 (01.03)	44.10 (00.62)	09.22 (05.10)	35.27
Bert	49.62 (04.43)	30.44 (03.46)	46.93 (03.94)	42.61 (01.80)	10.10 (03.06)	35.94
DistilBert	45.00 (05.05)	21.92 (06.21)	41.30 (05.45)	42.93 (01.56)	07.32 (03.53)	31.70
Roberta	49.43 (05.66)	31.15 (07.73)	47.63 (03.38)	41.95 (00.04)	11.43 (05.71)	36.32
Roberta S	48.27 (06.02)	30.74 (06.87)	46.60 (06.83)	43.37 (00.78)	10.10 (03.06)	35.82
Roberta SBS	43.08 (00.45)	23.38 (00.90)	40.04 (00.31)	42.88 (01.53)	08.83 (00.52)	31.64
XLNet	52.85 (06.56)	34.66 (06.67)	50.04 (08.46)	43.24 (03.66)	13.03 (02.52)	38.76
XLNet Mem	51.15 (12.53)	36.37 (14.17)	49.11 (13.53)	43.56 (01.26)	15.88 (08.23)	39.21

A.3 Full Confidence Metrics

The following figures, A.1 and A.2, show the confidence plots for all the models on the CNN/DM and Academic Texts datasets respectively.

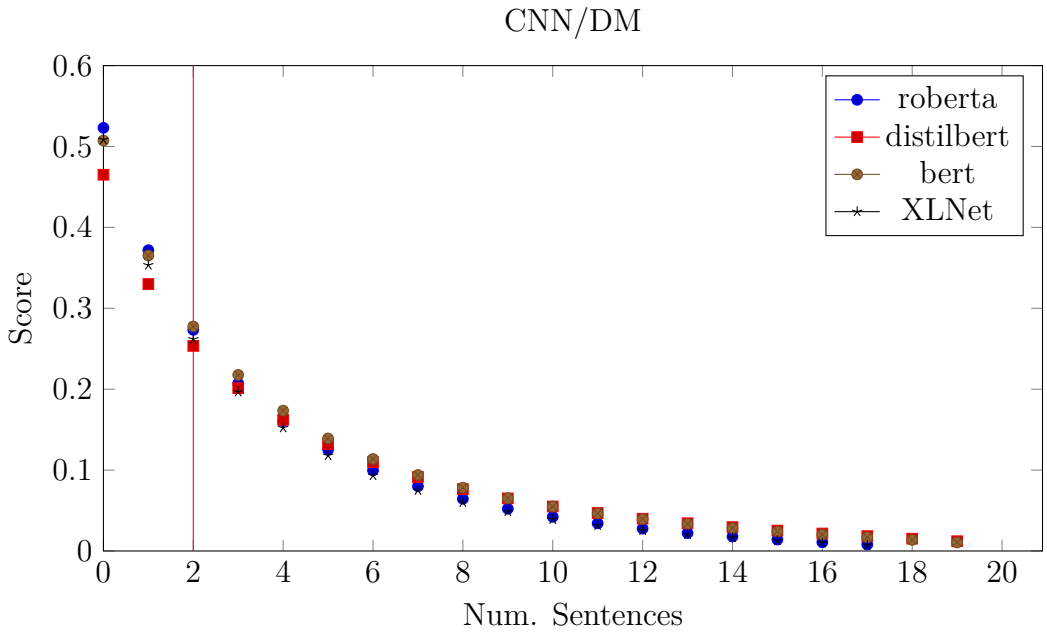


Figure A.1: The confidence score of all the models on the CNN/DM dataset

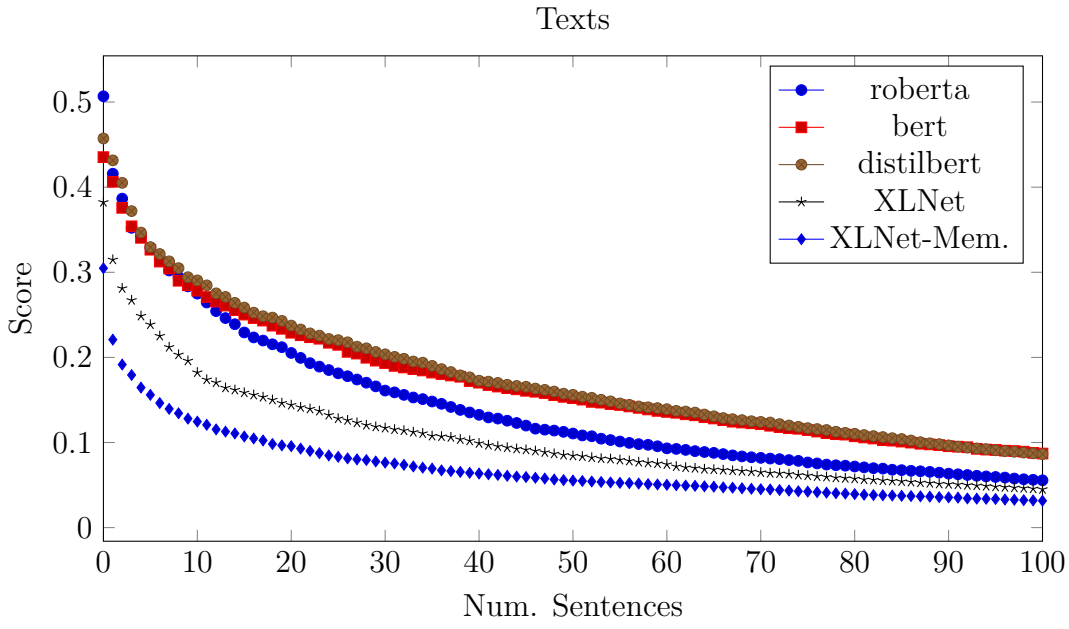


Figure A.2: The confidence score of all the models on the Academic Texts dataset

A.4 Text Excerpts

The following sections contain summaries of the paper [44], the first is the paper's original abstract, followed by one of our manually created summaries and several automatically generated summaries.

A.4.1 Abstract

This paper presents an innovative approach to classifying the driver's driving style by analyzing the jerk profile of the driver. Driving style is a dynamic behavior of a driver on the road.
At times a driver can be calm but aggressive at others. The information about driver's dynamic driving style can be used to better control fuel economy.
We propose to classify driver's style based on the measure of how fast a driver is accelerating and decelerating. We developed an algorithm that classifies driver's style utilizing the statistical information from the jerk profile and the road way type and traffic congestion level prediction.
Our experiment results show that our approach generates more reasonable results than those generated by using other published methods.

A.4.2 Manual Extractive Summary

We present in this paper an innovative approach to dynamically classifying driver's style by analyzing the online jerk profile of the driver combined with the statistics of driver styles specific to different roadway types.
We also propose a quantitative method to assess the performance of driver style classification.
We consider the driving style as a transient behavior: a driver can be aggressive at one time period but normal at others.
Jerk is defined, in physics, as the rate of change in acceleration or deceleration.
On a freeway with no traffic congestion, even an aggressive driver does not accelerate and decelerate that much.
However on a local road with heavily congested traffics, even a calm driver has to make many brake acceleration moves.
Therefore it is important to incorporate the driving statistics of roadway types into the driver style classification.
The average jerk has significant difference among different traffic congestion levels and different roadway types.
This means if the standard deviation of the jerk exceeds the average jerk of the road-type, then the driver will be classified as aggressive.
On the contrary, if the standard deviation of the jerk is much lower than the average jerk of a normal driver; it will be classified as calm, unless

the velocity is zero.
We model a driving trip as a time series of speed profile over a sequence of different road types such as local, freeway, arterial/collector, etc. augmented with different traffic congestion levels.
The classification algorithm, DS_Classification, is executed at every time step to perform online driver style classification.
One quantitative performance measure of driver style classification is to calculate the fuel rate with a given time segment.
We presented an innovative algorithm for online classification of driver's driving styles for application to vehicle power management.
The algorithm, DS_Classification, extracts jerk features from the current vehicle speed within a short window, and classifies the current driver style into three classes, calm, normal and aggressive, by comparing the extracted jerk feature with the statistics of the driver styles on the current roadway.
Through experiments conducted in the PSAT simulation environment, the DS_Classification algorithm has shown to be effective in classifying the driver's style: (a) the fuel-rate of a conventional vehicle has a positive correlation with spikes in the jerk profile of the cycle, (b) the fuel rates within the aggressive segments are higher than the normal and calm segments, (c) the fuel mileages of "calm" and "normal" segments are much greater than the "aggressive segments, and in most cases, the fuel mileages of"calm" segments are greater than "normal" segments.

A.4.3 Every-7

DRIVING patterns exhibited in a real world driver are the product of the instantaneous decisions of the driver to cope with the (physical) driving environment.
The goal is to make an intelligent prediction of the future as our research in the road way prediction for power management has proven successful. For example if it's predicted that the driver is being aggressive at the current drive cycle, more battery power may be used instead of engine power to help minimize the fuel consumption.
We conducted experiments on a number of drive cycles and the results will show that our driver style classification approach generates better results than those generated by the published method in [5, 6].
This driving style is less fuel efficient.
Jerk is defined, in physics, as the rate of change in acceleration or deceleration.
We propose a quantitative measure of ground truth for driver style classes: fuel rate measured in grams per second.
The current driver's dt driving style at time t is classified based on the following jerk features extracted within time period t E [(Ie - OJ), Ie) : the ratio of the standard deviation of the jerk profile and the typical jerk while driving on that particular road type, where te is the current

time and OJ is the window size.
On a freeway with no traffic congestion, even an aggressive driver does not accelerate and decelerate that much.
Each level of service represents a range of operating conditions and the driver's perception of those conditions; safety is not included in the measures that establish the service levels [10,11].
On the contrary, if the standard deviation of the jerk is much lower than the average jerkofa normal driver; it will be classified as calm, unless the velocity is zero jerk throughout the cycle.
The ds_classification algorithm relies on the output of a roadway type prediction program that predicts the current roadway type and traffic congestion level.
Step 5. If Velocity = 0 m/s then No Speed Else If r < nOrm threshold then DS = Calm Elseif nOrm threshold < r < aggthreshold then DS = Normal Elseif aggthreshold < r then DS = Aggressive Two thresholds are used, norm_threshold and agg_threshold.
The classification result is shown only when it is different from the result of the previous window.
Because the driver style is measured by the transient behavior, jerk, which is captured in a short time interval, i.e. within a window of time, window size is closely related to the classification results.
The driving classification results using different window sizes for approximately 200 seconds of the US06 drive cycle are shown in Fig.
Note only window size 6 and 3 detected the stop period, i.e. no speed period between 125sec and 150sec.
In both figures the classification results using the acceleration features are shown in the upper graph, and the classification results generated by the ds_classification algorithm are shown in the lower graph.
Another example is that the ds_classification detected an aggressive style at around 200 second, but the acceleration based algorithm detected the same segment as calm style.
We expect the correlation between the fuel mileage and the driving style as follows: MPG(calm) > MPG(normal) > MPG (aggressive).
But in US06 the average mileage for "calm" segments is much less than the "normal" segments.
Through experiments conducted in the PSAT simulation environment, the ds_classification algorithm has shown to be effective in classifying the driver's style: (a) the fuel-rate of a conventional vehicle has a positive correlation with spikes in the jerk profile of the cycle, (b) the fuel rates within the aggressive segments are higher than the normal and calm segments, (c) the fuel mileages of "calm" and "normal" segments are much greater than the "aggressive segments, and in most cases, the fuel mileages of"calm" segments are greater than "normal" segments.

A.4.4 DistilBERT

Research has shown that driver style, roadway type, and traffic congestion levels have various degrees of impacts on fuel consumption and emissions [1, 2].
Research has also shown that by incorporating the knowledge of driving environment into power management, fuel consumption can be significantly reduced [3, 4, 5, 6, 7].
The knowledge about driver style can be used in real time vehicle power control to minimizing fuel consumption in hybrid electric vehicles.
If the ratio is greater than 100We also propose a quantitative method to assess the performance of driver style classification.
Jerk is calculated as the derivative of the acceleration/deceleration or the second derivative of the velocity.
While an acceleration profile shows how a driver speeds up and slows down, a jerk profile shows how a driver accelerates and decelerates, which is more important in determining the driver's aggressiveness.
The jerk profile clearly depicts how the acceleration changes over time in the UDDS driving cycle and the spikes in the jerk profile occur only when there are big changes in the acceleration, negative or positive.
The reason we use J as a feature in the classification is that we believe a driver's driving style is strongly influenced by the roadway type and traffic congestion level the driver is on.
We use the speed profile of 11 standard drive cycles developed by Sierra Research [9] as the standard normal driver's driving style on the different roadway types and traffic congestion levels.
Table I shows the average jerk calculated from all 11 standard drive cycles.
These average jerks from Table I are calculated by averaging the absolute value of each entire sierra research jerk profile.
The absolute value of the jerk is used since it is desired to obtain the 'total' amount of local and ramps have higher jerk average than freeway in average.
These jerk average values are used in the driver style classification algorithm, ds_classification algorithm described in the next subsection, to represent a 'normal' driver on the respective roadway and the LOS.
This means if the standard deviation of the jerk exceeds the average jerk of the road-type, then the driver will be classified as aggressive.
Step 3. Detect the current road type and traffic congestion level using the roadway type prediction algorithm presented in [12].
Based on trial and error in experiments, we suggest to take norm_threshold = 0.5 and agg_threshold = 1.0.
A number of experiments have been conducted to evaluate the proposed driver's driving style classification algorithm, ds_classification.
If the window size is too small, it may not give enough time to capture the entire acceleration and deceleration event.
The ds_classification algorithm successfully detected the aggressive style

between 10 seconds and 50 seconds, but the acceleration based algorithm considered it as normal driving style.
Fig.7 shows the fuel rate curves for three 20-second drive cycle segments that were classified by the ds_classification algorithm as calm (magenta curve), normal (green curve), and aggressive (blue curve).
For the most time, the fuel rates within the aggressive segment are higher than the normal and calm segments, and the fuel rates within the normal segment are higher than the calm segment.

A.4.5 XLNet Mem.

Research has also shown that by incorporating the knowledge of driving environment into power management, fuel consumption can be significantly reduced [3, 4, 5, 6, 7].
During real world driving, the driving patterns need to be predicted accurately in real time so the results can be used by the online power controller [5, 6, 7].
Driving style is a dynamic behavior of a driver on the road.
At times a driver can be calm but aggressive at others.
The knowledge about driver style can be used in real time vehicle power control to minimizing fuel consumption in hybrid electric vehicles.
For example if it's predicted that the driver is being aggressive at the current drive cycle, more battery power may be used instead of engine power to help minimize the fuel consumption.
We also propose a quantitative method to assess the performance of driver style classification.
The aggressive drivers tend to have highest fuel rate, calm drivers have the minimum fuel rate, and normal drivers are in the middle.
A drive cycle is usually represented as the speed function of time t, i.e. DC(t), t=O, ..., teo The jerk function, J(t), can be derived by taking the second derivative d 2DC(t) of speed, DC(t), i.e. J(t) = 2.
We use the speed profile of 11 standard drive cycles developed by Sierra Research [9] as the standard normal driver's driving style on the different roadway types and traffic congestion levels.
This set of 11 drive cycles represents passenger car and light truck operations over a range of facilities and congestion levels in urban areas.
These average jerks from Table I are calculated by averaging the absolute value of each entire sierra research jerk profile.
This means if the standard deviation of the jerk exceeds the average jerk of the road-type, then the driver will be classified as aggressive.
The DSClassification algorithm relies on the output of a roadway type prediction program that predicts the current roadway type and traffic congestion level.
Because the driver style is measured by the transient behavior, jerk, which is captured in a short time interval, i.e. within a window of time,

window size is closely related to the classification results.
A good window size allows to accurately capturing this transient behavior.
The classification results generated by the window sizes of 15, 9, 6, and 3 seconds are superimposed on the speed profile of the drive cycle.
The DSClassification algorithm successfully detected the aggressive style between 10 seconds and 50 seconds, but the acceleration based algorithm considered it as normal driving style.
We expect the correlation between the fuel mileage and the driving style as follows: MPG(calm) > MPG(normal) > MPG (aggressive).
The average mileage for the "calm" segments are all greater than the "normal" segments on all drive cycles except for the two drive cycles, US06 and ARB02.
We presented an innovative algorithm for online classification of driver's driving styles for application to vehicle power management.
The algorithm, DSClassification, extracts jerk features from the current vehicle speed within a short window, and classifies the current driver style into three classes, calm, normal and aggressive, by comparing the extracted jerk feature with the statistics of the driver styles on the current roadway.