UPPSALA UNIVERSITET

UPTEC IT 21004

Examensarbete 30 hp
April 2021

Abstractive Summarization of Podcast Transcriptions

Hannes Karlbom

Institutionen för informationsteknologi
Department of Information Technology

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Abstractive Summarization of Podcast Transcriptions

Hannes Karlbom

In the rapidly growing medium of podcasts, as episodes are automatically transcribed the need for good natural language summarization models which can handle a variety of obstacles presented by the transcriptions and the format has increased. This thesis investigates the transformer-based sequence-to-sequence models, where an attention mechanism keeps track of which words in the context are most important to the next word prediction in the sequence. Different summarization models are investigated on a large-scale open-domain podcast dataset which presents challenges such as transcription errors, multiple speakers, different genres, structures, as well as long texts. The results show that a sparse attention mechanism using a sliding window has an increased average ROUGE-2 score F-measure of 21.6% over transformer models using a short input length with fully connected attention layers.

Handledare: Ann Clifton
Ämnesgranskare: Joakim Nivre
Examinator: Lars-Åke Nordén
ISSN: 1401-5749, UPTEC IT 21004
Tryckt av: Reprocentralen ITC

Acknowledgements

I would like to thank my thesis advisor Joakim Nivre for all the helpful and constructive feedback, good suggestions and guidance throughout the project. I'm also very grateful to the Lab in Language Technologies at Spotify for all the help and support, especially my primary supervisors Ann Clifton and Rosie Jones. Finally I would like to thank my wife, Jisoo Maeng, for all her support during the whole process.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	**A**rtificial **I**ntelligence
BERT	**B**idirectional **R**epresentations **E**ncoder from **T**ransformer
BART	**B**idirectional and **A**uto-**R**egressive **T**ransformer
BoW	**B**ag-**of**-**W**ords
BPE	**B**yte **P**air **E**encoding
CBOW	**C**ontinuous **B**ag **Of** **W**ords
DL	**D**eep **L**earning
TF-IDF	**T**erm**F**requency-**I**nverse**D**ocument **F**requency
GPT	**G**enerative **P**re-trained **T**ransformer
LSTM	**L**ong **S**ort-**T**erm **M**emory
ML	**M**achine **L**earning
MLM	**M**asked **L**anguage **M**odeling
NLP	**N**atural **L**anguage **P**rocessing
RNN	**R**ecurrent **N**eural **N**etwork
ROUGE	**R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation
Seq2Seq	**S**equence-**to**-**S**equence
TREC	**T**ext **RE**trieval **C**onference

Chapter 1

Introduction

Text summarization is a part of computational linguistics and computer science with the aim being to compress a longer text into a shorter one, where both the input and output lengths are variables of the model. The summarized text has to have multiple attributes to be considered a good summary such as containing no redundant information, being written in good language, being concise and most importantly containing all the important information from the source text that is being summarized [1].

As the internet has developed, a massive amount of text documents have become available and it is becoming more and more important to be able to extract and store the relevant information in a fast and efficient way. For humans, the process of summarizing documents manually is both time consuming and expensive, making it a good motivation as to why automatic text summarization is becoming more important as the volume of text documents in circulation increases.

Podcasts are a fast growing audio-only medium and with the availability of automatic speech-to-text systems more and better transcriptions are produced. The Spotify podcast dataset is the first large-scale set of podcasts with transcripts that has been released publicly[1] with over 100,000 transcribed podcast episodes. Podcasts as a format, and their transcriptions, are varied with respect to category, structure, speakers, length, etc. [2], and therefore present a challenge for current summarization models.

There are many different approaches to text summarization with them mainly being divided into abstractive summarization, extractive summarization or a mix of the two [1, 3]. Abstractive summarization models can be good at correcting linguistic errors and paraphrasing information in documents, and therefore is a good candidate to use for podcast transcriptions summarization.

[1]As of when this was written: *2021-04-11*

1.1 Motivation

Summarizing texts is nothing new where the work has mostly been focused on summarizing shorter texts, e.g. news articles, where there are many summarization datasets available [4–6]. Podcast text summarization is different from tasks such as news article summarization due to the nature of podcasts, which are generally longer and more diverse in structure. They often include multiple speakers and have different formats, such as interview, monologue, conversation, etc. and for this reason it is not guaranteed that models that work well for other text sources will work well for transcribed podcast texts.

When deciding whether to listen to a podcast episode it is hard to gauge if the content is interesting or relevant as a listener and in this situation the description is an important part in the decision of whether to listen or not. Creating good descriptions can be both hard and time consuming. Therefore, automatic transcription and summarizing is valuable to both the podcast creators and the podcast listeners.

1.2 Purpose

The thesis objective was to implement and apply deep learning natural language models to the podcast transcriptions, to generate and evaluate summaries created by the models. Both extractive and abstractive summarization techniques have previously been shown to work well in compressing text whilst keeping the relevant information [7]. This gives good motivation for applying established models to the podcast transcriptions.

The focus is put on the abstractive summarization methods due to the fact that spoken words are transcribed automatically by a system which is not perfect. An abstractive summarization model might be able to correct for the transcription errors but it wouldn't be possible when using an extractive model. Additionally the conversational format of podcasts is less advantageous to extractive algorithms as it might not always be coherent both in the context of a conversation or in the context of whole podcast episode.

Podcast episodes, with an mean input length of 6780 tokens, are significantly longer than most state of the art abstractive summarization can process due to how computational complexity scales with the input length. The main objective of this thesis is to investigate how current abstractive summarization models can be extended to handle longer inputs and how this compares to other models when summarizing podcast episodes. Recently summarization of longer documents such as scientific articles, have seen big improvements through model architecture improvements and reduction in computational complexity [8, 9] and these advancements are applied to create a model for podcast summarization.

In more detail the project set out with the following objectives:

- Create a set of baseline models and measure the performance on the podcast dataset.
- Quantify the performance of state of the art abstractive summarization models which largely truncate the input texts on the podcast dataset.
- Extend abstractive summarization models to handle longer text input lengths and compare the new model against the other models benchmarks.
- In addition to the most common automatic evaluation metrics for summarization, ROUGE score, investigate how other scores compare in capturing the quality of a podcast summary.
- Participate in the podcast track of the Text REtrieval Conference (TREC) conference 2020 [10].

The project looks to answer the following research questions:

- Can abstractive summarization models be trained to successfully summarize podcast transcriptions?
- Can the state of the art abstractive summarization model "BART" be extended handle longer text input and is this beneficial to podcast summarization?

Finally the project aims to investigate the following hypotheses:

- Abstractive summarization is a good approach to summarize podcasts due to the conversational format and error rate of the automatic transcription system.
- Salient information does not only exist in the beginning of a podcast episode and being able to use information from as much of the input text as possible is essential to creating a good podcast summary.

- The evaluation of podcast summaries produced by abstractive summarization models can benefit from other evaluation metrics, in addition to the n-gram based evaluation metric ROUGE, when evaluating the quality of summaries.

1.3 Context

The project was done at Spotify AB, a digital music, podcast, and video streaming service that gives you access to millions of songs and other content from artists all over the world. Spotify has released a dataset with transcribed texts from 100,000 episodes from different podcast shows[2], which will be used as a basis for creating and evaluating text summarization models [11].

In addition to the experiments on the podcast dataset, a part of the project was to submit a run for the shared summarization task, of the podcast track, at the Text REtrieval Conference (TREC) 2020 [10].

1.4 Delimitation

The project looks at the domain of transcribed texts from podcasts which contains many subproblems e.g. speaker summarization, discussion summarization, themes etc. and as a delimitation the thesis focuses on problem of how do deal with the long input sequences of the podcast episode transcriptions.

No audio features of the podcast episode are considered, although the audio files are available as part of the dataset, due to the significant time increase of processing and analysing audio in addition to the texts.

1.5 Outline

The thesis is structured as follows. Chapter 2 introduces a comprehensive background starting with an introduction to automatic summarization, then going into concepts and techniques related to natural language processing. This is followed by a section on deep learning where the models used in the thesis are described, as well as the models that they were evolved from and

[2]https://podcastsdataset.byspotify.com - *Accessed: 2021-04-11*

built upon. After that a more general section about model training is introduced where concepts such as overfitting and regularization are covered. Finally there is a section related to model evaluation where evaluation metrics specifically relevant to summarization are described.

Chapter 3 covers specific methods, details, choices and experimental setup used in the project. It starts off by describing the dataset in detail followed by a section where the architecture of the new model created for this thesis is covered. The chapter then describes what preprocessing was applied and the experimental setup. The next sections covers an introduction to TREC and the manual evaluation of the shared task at TREC, as well as the automatic evaluation from the experiments. The last section is a set of short descriptions of tools and libraries used as part of the project work.

Chapter 4 states the results of the experiments as graphs and tables with accompanying text explaining how to interpret them. Lastly chapter 5 and chapter 6 discusses the results of the experiments and the conclusion of the thesis with relation to the motivation and purpose posed in chapter 1.

Chapter 2

Background

This chapter presents the background and theory that are of interest to the text summarization work explored in this thesis. Foundational natural language processing concepts and machine learning concepts are described, as well as how the transformer models and their attention mechanism works. Techniques to preprocess the data, how to split the dataset, and more general concepts of modelling such as overfitting are also covered.

2.1 Automatic Summarization

Automatic text summarization is nothing new and researchers have been trying to improve upon different techniques since the 1950s [12]. The aim is, for a given input text, to produce an output text that is significantly shorter and contains all the important information from the input text [7]. In addition to compression and capturing the important information, a good summary has to be linguistically well written with respect to features such as grammar and spelling. There are also other softer metrics which are more or less important depending on the summarization context and can include named entity mentions, topical awareness, structure and format consistency.

When it comes to methodology, automatic summarization is mainly divided into two different top-level categories: extractive summarization and abstractive summarization. Extractive summarization can generally be divided into three different steps needed to construct a final summary of the input text [13].

Given a set of documents $D = \{D_1, ..., D_j\}$ where each document D_i consists of a set of sentences $S_i = \{s_1, ..., s_{ji}\}$ then the whole text input is given by the complete set of sentences $S = \bigcup_{k=1}^{j} S_k$. The extractive summarization steps are described as follows:

1. Create an input representation for each sentence $s_i \in S$ such that it is helpful in determining sentence salience, e.g TF-IDF or word embeddings.

2. Use the representation to train a model to assign a score to each sentence $s_i \in S$.

3. Select the top k scoring sentences $\{s_1, ..., s_k\} \in S$ and create a summary by merging them together.

Abstractive summarization methods create summaries through natural language understanding where the models have to build a representation of the input text and then rephrase and paraphrase it when generating the summary [14]. Abstractive summarization is newer and less mature than its counterpart extractive summarization, but in recent years research attention has shifted towards abstractive summarization, or a mix of extractive and abstractive summarization [15, 16].

Deep learning methods have been proposed for abstractive summarization by framing the learning problem of summarizing text as a sequence-to-sequence (seq2seq) problem, and this has shown promising results [4]. The automatic summarization approaches using deep learning can learn language models specific to the generation related to the source training documents, and they have the ability to create completely new descriptions from the input text [17].

An upside to the deep learning seq2seq models is that they are trained end-to-end and as such need much less data preparation, specific preprocessing or feature selection by experts [5]. The models can also avoid the need for sub-models or domain specific representations, and rely more heavily on a large amounts of data for training. A drawback to relying on having more data is that in some domains, e.g. medical research, data can be scarce and expensive to come by. More data also increase resource requirements needed for model training and inference.

2.2 Natural Language Processing

Natural language processing (NLP) is a field of linguistics, computer science and artificial intelligence concerned with computer processing of natural language, that is any language naturally evolved through use by humans.

The processing of natural language data differs from the processing of numerical datasets. When processing texts, the words have to be tokenized and transformed into a numerical representation that can be used as input to the machine learning algorithms. The algorithms face challenges such as how to encode the position of a word relative to other words, word importance in the complete set of documents, out of vocabulary words etc., and how well a model performs can be greatly impacted by how the input text document is represented [18].

This section describes the NLP techniques used in the models, the preprocessing and evaluation of the text data that was used and produced in the experiments.

2.2.1 Tokenization

Tokenization, also known as word segmentation, is the process of splitting a given document into meaningful pieces, or *tokens*, that can be used for further analysis [19]. Tokens may be words, numbers, or punctuation marks.

How to approach the task of tokenization is not always obvious as one has to decide what pieces of a text contains meaning and how to segment the document. Intuitively, a text might be split on white spaces and punctuation. Consider the example string "this is a text about tokenization". This input string might be processed into the following tokens, according to the rule of splitting on white spaces and punctuation:

$\boxed{\text{this}}\ \boxed{\text{is}}\ \boxed{\text{a}}\ \boxed{\text{text}}\ \boxed{\text{about}}\ \boxed{\text{tokenization}}$

This particular example does not look dubious, but that set of rules does in fact introduce a whole new set of problems. For instance, the city "New York" would be split up to two separate tokens, $\boxed{\text{New}}$ and $\boxed{\text{York}}$, which gives the text a different semantic meaning. Another tricky case in the English language is how to handle apostrophes, e.g when handling a word like "haven't". It might be tokenized as:

$\boxed{\text{haven't}}$, $\boxed{\text{havent}}$, $\boxed{\text{haven}}\ \boxed{\text{t}}$, or $\boxed{\text{have}}\ \boxed{\text{n't}}$

A different example of when tokenization is not straight forward is in the case of abbreviations where punctuation marks denote the abbreviated word instead of the end of a sentence.

$\boxed{\text{e.g.}}$, $\boxed{\text{e.g}}\,\boxed{.}$ or $\boxed{\text{e}}\,\boxed{.}\,\boxed{\text{g}}\,\boxed{.}$

The problems that occur when deciding how to tokenize text is very language specific, and is often tackled by applying some heuristic rules to the language that is subject for analysis [19].

2.2.2 Stemming and Lemmatization

The information that might be semantically important in a word is often found in the stem of the word. Inflection is just a grammatical way of expressing it [19, 20]. Stemming and Lemmatization aims to remove all inflectional forms of a word, leaving just the stem of the word in the case of stemming, or lemma form in the case of lemmatization. This can often be advantageous, for instance when counting word frequencies algorithms often use stems in order to find matching words in the collection of texts that is to be analyzed [20]. A couple of examples of lemmatization and stemming are:

(1) am, are, is $\rightarrow$ be
(2) train, trains, train's, trains' $\rightarrow$ train

Stemming is the not very refined process of removing affixes from words leaving just the stem [21]. The process of stemming will not always be accurate, but is rather done with the hope of being right most of the time. A problem with stemming might occur when stemming the word *sing*, that could result in the word *s*, which is clearly incorrect.

In contrast to stemming, lemmatization is a more sophisticated approach that uses morphological analysis and a dictionary to reduce words to their lemma. Lemmatization of the word *saw* will result in *see*. Example (2) above shows how a lemmatizer will transform all the inflections of the word *be* to its lemma.

It is not always a clear choice between stemming and lemmatization as they both have advantages and drawbacks that comes to light depending on the text. Just as with tokenization rules, the choice between stemming and lemmatization is largely language and application dependent [19]

2.2.3 WordPiece and BPE Tokenization

Byte pair encoding (BPE) [22] and WordPiece [23] are two algorithms which works in a similar way to segment words into subword-level symbols. Subwords are useful as tokens because they allow a model to handle misspellings, rare words and out of vocabulary (OOV) words well. The model can e.g. learn something about a common subword that might be helpful to understanding the meaning of rare words that contains that subword.

Both techniques start by initializing the vocabulary with all the individual characters from the dataset that the tokenizer is created from. The algorithms iteratively updates the vocabulary by combining the most frequently occurring symbols in the vocabulary, creating new entries for the combinations. WordPiece and BPE follow the same methodology but differ in how they the merge symbols in the vocabulary, which can have different results since the number of merges performed is a hyperparameter that can be tuned.

Given an input text:

I am a talker, I even talked about not having anything to talk about.

The input vocabulary would be initialized with all the characters from the input $V = \{,,.,I,a,b,d,e,g,h,i,k,l,m,n,o,r,t,u,v,y\}$.

Using the BPE algorithm the first 3 merges would look like:

1. t a => ta
2. l k => lk
3. ta lk => talk

At this point the vocabulary V would contain three new symbols $\{V \cup \{ta, lk, talk\}\}$ and merges would continue for as many iterations as specified. This type of tokenization is helpful e.g. when handling out of vocabulary (OOV) words because any word which doesn't occur in the input data can be broken down into subwords that the algorithm has seen during model training. In addition to handling OOV words more gracefully than other tokenization methods, the algorithms can also help when dealing with rare words where they will be split into subwords which have been seen more frequently.

The idea is that even if the words *talked* and *talking* are two completely different words, they have something in common and segmenting them in to *"talk"*, *"ed"* and *"talk"*, *"ing"* allows models to better deal with these words if *"talk"* occurs many more times in the training set. This is similar to stemming and lemmatization (described in a future section 2.2.2) but both parts of the word are kept as tokens.

2.2.4 Text Cleaning and Filtering

Text data is often unstructured, noisy and of varying quality and as such there are multiple ways to improve the training dataset through data analysis and cleaning the texts before transforming them to their model input representations. Data from the internet or data created through platforms can contain boiler plate text common to a majority of the input documents, e.g. advertisements, HTML code, or e-mails, and these sections of the texts can either make it take longer for the model to learn to generalize well or cause the model to overfit to that output. Different types of boiler plate text snippets can be cleaned using regular expressions and simple rules or more advanced models can be trained to detect it.

Filtering in a cleaning context refers to removing data points from the dataset in contrast to keeping them and cleaning the data. This can include removing inputs that are too short to make sense, different languages that the model can't handle, documents that are too similar to other documents in the dataset (indirect oversampling), and this can be achieved through rules e.g. $length(input) > 10$ or more advanced models such as language detection [24, 25].

2.2.5 Corpus, Dictionary and Bag-of-Words

When analyzing text data the set of documents used for the analysis is called a corpus, it is created by passing all the text through some preprocessing and is then represented as a list of tokens.

To be able to analyze a corpus according to some preferred statistical method, we need to have a numerical representation for each token in the set. The process of transforming text into its numerical representation starts with the creation of a vocabulary V which is a mapping between the tokens in the corpus and integer IDs. This means that every unique token in the text corpus is represented by a unique integer ID.

When the vocabulary has been created the text can be further processed into a set of input features called bag of words (BoW). A bag of words representation is a sequence of tuples containing a token ID and some word weight for that particular word. A bag of words is usually created for every document in the corpus, where the token ID is taken from the dictionary and the word weight is calculated according to a schema that sets a particular word in relation to the whole corpus. A simple but common BoW model represents a token t_i as a vector $\vec{v}$ where the size is the number of tokens in the vocabulary $|V|$, the value of the token is one $v_i = 1$ and the other values zero.

2.2.6 TF-IDF

Word frequency methods aims to assign importance to words by giving weights to each word in the corpus based on a frequency count. TF-IDF is such a method that uses the *word frequency - inverse document frequency* as the frequency count. Instead of using only the term frequencies, they are combined with the *inverse document frequency* to produce better results.

Term frequency is a measure of how many counts of a specific word the whole corpus contains [26]. Term frequency is a simple first step to take, but it does not have any semantic insight of the corpus and is thus insufficient when aiming for a more complex analysis of the text [21].

Inverse document frequency is an alternative way at looking at word frequency that, in contrast to regular word frequency measures, assumes that low frequency words are rich in content [27]. The assumption is that a word that occurs in a small set of documents, in relation to the size of the corpus contains a lot of information about that specific document. Given a set of documents $D = \{d_1 ... d_i\}$ and a term t, the inverse document frequency $idf(t, D)$ is mathematically defined as:

$$\boldsymbol{idf(t, D) = -log(\frac{df_t}{|D|}) \iff log(\frac{|D|}{df_t})}$$

Where df_t is the document frequency, or in other words the number of documents containing the term t, and $|D|$ is the number of documents in the collection [27]. This mathematical formula punishes the term frequency weight of a term that occurs in many documents since the frequency df_t is in the denominator, meaning that the more documents a term occurs in the lower weight it will have.

It has been found that if the inverse document frequency is combined with regular term frequency a highly effective weighting scheme is created [28]. TF-IDF is a value of the relative frequency of a word in a document when compared to the inverse document proportion of that same word, relative to the entire corpus [29]. Given that the count of term t in a document $d_i \in D$ is defined as $tf(t, d_i)$ then TF-IDF can be defined as:

$$\boldsymbol{tfidf(t, d_i, D) = tf(t, d_i) \times idf(t, D)}$$

What TF-IDF aims to do is to find words that occur frequently in a small set of documents and assign importance to them. Consequently, words that occur frequently in the document set will be given low importance. The result of TF-IDF weighting is a term-by-document matrix **X**, where each row is a document, and each column holds the TF-IDF value of a specific term. Each word in the corpus is represented in the matrix, and words that does not occur in a particular document is represented by a zero in that word's column.

While the TF-IDF measure has proven to be effective, it also has a few shortcomings; mainly that it fails to reduce the dimensionality of the feature set. Additionally, the TF-IDF weighting scheme gives little or no information about the statistical structure of the document, or the statistical structure of the relationships between the documents in the corpus [30].

2.2.7 Word2Vec

A word vector is a way of representing a word as a numerical vector, which is needed if we want to apply a machine learning algorithm. A special case of the BoW approach is called one-hot encoding where the word is encoded by checking if it is present in a vocabulary or not. A word encoded in this way is given the value 1, and all other words in the vocabulary are given the value 0. To illustrate this with a simple example, given the vocabulary $V = \{King, Queen, Man, Woman, Youth\}$ and the word $w_i = Queen$ to be encoded, the representation would look as follows:

$$\vec{w_i} = (0_{King}, 1_{Queen}, 0_{Man}, 0_{Woman}, 0_{Youth})$$

For documents each word in a document has a frequency count, or a binary check if present in the vocabulary, and the document is represented as the whole vocabulary vector. Using the one-hot representation for words does not allow for any meaningful comparison between words other than to check if two words are the same.

In distributed word embeddings each word is represented by N weights across the elements of a N-dimensional vector. Each element in this vector contributes to the meaning of the words, in contrast to the one-to-one mapping of the one-hot encoding. To continue the example above, if a word vector's first element relates to the semantic concept of royalty (although no such explicit labels are given in the algorithm) then the first element for a vector embedding of the word "king" will have a big weight whilst the same element for the word "man" will have a small weight.

The representation of words as vectors in a semantic vectors space allows for vector decomposition to answer questions such as [31]:

$$\vec{v}(king) - \vec{v}(man) + \vec{v}(woman) = ?$$

$$\vec{v}(Paris) - \vec{v}(France) + \vec{v}(Italy) = ?$$

It is also possible to answer questions such as "what is to man as queen is to king?" and singular to plural relationships.

Another reason to learn word embeddings, other than semantic text analysis and similarity measures, is that traditional representations do not capture the relationship between symbols of different words well. Word embeddings also work as a dimensionality reduction technique and since vocabularies can contain hundreds of thousands of words, representations such as BoW lead to very sparse data which often require more training samples to create good models [31].

Word embeddings is not a new concept. In 1975 Salton et al. suggested a vector space model where a vector represents a document, and the elements of the vector represent the words of the document [32]. Word embeddings were popularised in 2013 through the Word2Vec algorithm which is a computationally efficient algorithm for learning word embeddings from raw text where the authors suggested two architectures for learning the embeddings, Continuous Bag of Words (CBOW) and the Skip-gram model [31, 33].

2.2.8 Embedding Layer

When using a neural network model another common way to represent text input is to use an embedding layer as the first part of the network where the job of the embedding layer is to transform the input from e.g. a continuous bag of words representation to a more dense vector of lower dimensionality, similarly to the word embeddings described in section 2.2.7. This approach can be mixed and matched with the pre-trained word embeddings approach where the embedding layer can be initialized randomly or using word embeddings trained on general language data. The embedding layer is then updated as every other layer in a neural network using e.g. backpropagation and as such can learn a representation best suited for the specific task. Depending on the task the embedding layer might not learn as rich representations as a general word2vec pre-trained word embedding [34].

2.3 Deep Learning

Deep learning (DL) is a subset a of methods and algorithms using neural networks within the area of machine learning (ML) and where the "deep" part of deep learning comes from the fact the networks have many hidden layers. Although mathematical theory has shown that a neural network with a single hidden layer can be a universal function approximator [35], in practice using many hidden layers has proven very effective at solving complex problems such as image recognition or natural language understanding.

This section describes deep learning models and architectures used in the experiments and how they are combined to handle the longer text input from the podcast transcriptions. In addition a basic introduction to supervised learning and neural network models is given together with an overview description of different neural network architectures on which the models are built upon or has evolved from.

2.3.1 Supervised Learning

ML and DL are usually grouped into three distinct types of learning, unsupervised learning, supervised learning, and reinforcement learning. Supervised machine learning refers to learning problems where there is a label or ground truth for every data point which can be used as feedback for the model to adjust its parameters, and that is how the model learns. The difference between what the model outputs and the label can be measured and used to optimize the objective function, sometimes called the cost function or the loss function, during training. There are many different optimization methods for different models, e.g. for neural networks a common optimization method is stochastic gradient decent which is used to optimize the objective function. The optimization is done together with backpropagation which calculates the gradients of the objective function with respect to the parameters of the network and uses this to update the network weights [36].

The goal is to learn a function mapping a set of inputs data points $X = \{\vec{x_1}...\vec{x_i}\}$ to their paired outputs $Y = \{\vec{y_1}...\vec{y_i}\}$ where Y is known during training time but unknown at inference time. Every element $\vec{x_j} \in X$ is a k-dimensional vector where k is determined by manually created input features or as a hyperparameter passed at model when training [37].

Input features to a model is a representation handcrafted by experts or a transformation learned as part of the model training in an end to end process. Data many times comes in a format that has to be transformed before being used as an input to a model, if the input is an an animal, e.g. "cat", then that might be represented with BoW or if the input is "three" it might be represented as the number 3.

Features and their representations are important for the reason that they can have a big impact on the discriminatory and predictive power of the model. An example of this would be using weight and height to classify an animal where using only those two values would be enough to discriminate between some animals but not all of them. Furthermore, how the weight is represented is also important e.g. using gram for one animal while using kilogram for another can make it hard for the model to learn the meaning of the variable and possibly misinterpret it, leading to misclassifications. Features can also be more complex structures than numbers, e.g images, sentences or time series [38].

2.3.2 Artificial Neural Networks

The human brain is made up of a collection of more than 80 billion interconnected neurons and can handle difficult tasks such as face recognition, speech and motor control. Signals are transmitted between neurons via synapses, with the goal being to raise or lower the electrical potential at the receiving cell. If a certain electrical potential limit is reached the the signal will propagate to other neurons connected to it.

Artificial neural networks, or feed forwards neural networks, were created to mimic the behavior of the brain in order to make a machine learn a specific task [39]. The artificial neurons are connected by weights, and each neuron might be connected to one or multiple neurons. The input to a neuron is the sum of the input weights, transformed by an activation function.

If the value of the activation function exceeds a certain threshold an output will be sent to the next neuron in the structure.

A neural network can be made up of several layers, with the basic structure consisting of an input layer, a hidden layer, and an output layer, where each neuron is connected to every neuron in the adjacent layer.

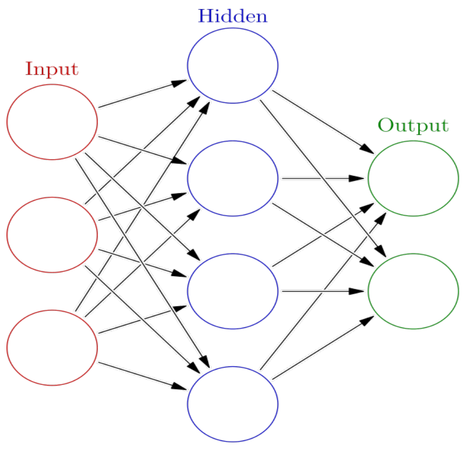


FIGURE 2.1: *Neural network with a 3 node input layer, a 4 node hidden layer and a 2 node output layer [40]*

When training an artificial neural network the weights in the network are adjusted in such a way that the objective function of the output layer is optimized, and the training phase produces a set of weights which optimizes the objective function.

A common way to adjust these weights is called *backpropagation*. Backpropagation calculates and propagates the objective function, together with an optimization function, backwards through the network, hence the name backpropagation. The gradient of the objective function is calculated with the negative of the partial derivative with respect to each weight in the network. The gradient is optimized using an optimization method, the most common one being stochastic gradient descent (SGD). The training phase repeats this process of feeding training examples forward through the network, back-propagating the objective function and adjusting the weights until some the stopping criteria is reached. The stopping criteria can be set in different ways and is a hyperparameter of the model training e.g. stopping after a certain number of epochs or stopping when the objective function increases or decreases too much on a validation set.

2.3.3 Recurrent Neural Networks

A recurrent neural network (RNN) addresses one of the main shortcomings in artificial neural networks, namely memory. The recurrent neural network allows information to persist in the network, making the output dependent on a chain of inputs. A recurrent neural network can be thought of as in figure 2.2, where multiple identical networks are interconnected, and the output of a network is the input of the next network in the sequence. This property has made RNNs hugely successful in many NLP tasks. Given that you want to predict a word based on a previous sequence of words, a regular artificial neural network would not be able to handle this as the previous sequence is given a label, but not remembered. RNN has a memory function, so if the sequence is "the trees are in the..." it would be able to derive the word *forest*.

One of the main problems with a RNN is the influence of a given input on the hidden layer, and the output of the network, it either decays or blows up exponentially when it goes around in cycles through the recurrent connections. This problem is called the vanishing or exploding gradient problem which is one of most important issues the Long Short-Term Memory architecture (LSTM) is designed to solve [41].

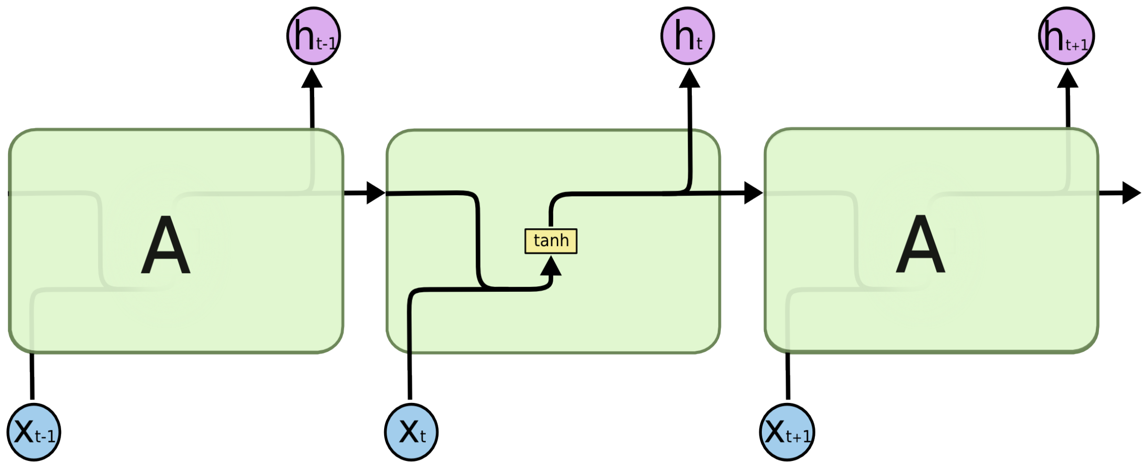


FIGURE 2.2: *A RNN cell showing a single tanh layer. Diagram by Chris Olah [42].*

LSTM

While recurrent neural networks have memory, it is unable to handle long-range dependencies. Long Short-Term Memory is a specific type of RNN architecture that was designed to model such long-range dependencies. LSTM contains memory blocks in the recurrent hidden layers. These memory blocks are capable of remembering dependencies that would decay in a RNN as the information flows through the sequence of networks. The memory cells are also capable of determining the importance of a specific input in the sequence, and pass that along in the chain, preventing a decay of that specific piece of information as it propagates forward through the network [43, 44].

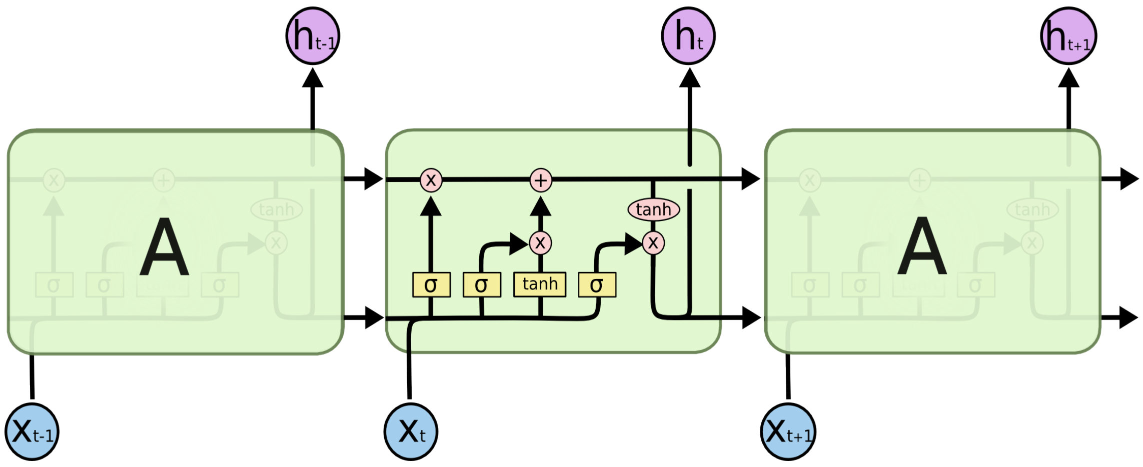


FIGURE 2.3: *LSTM memory cell showing 4 interconnected layers. Diagram by Chris Olah [42].*

An unfolded LSTM block shows that each LSTM memory state consists of four interacting layers, two tanh layers and three *sigmoid* (σ) layers. The sigmoid layers outputs a number between zero and one and can be seen as the proportion of information that is meant to be stored in the memory cell. The first layer in the chain is called the *forget gate*. It determines how much information from the previous state will be carried through to the memory block. The forget gate looks at the hidden state h_{t-1} and the input x_1 and concatenating those inputs, the output of the forget gate layer (f_t) is then given by:

$$f = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

where W_f and b_f are the weights and bias of the layers of the network.

When this step is done the next step is to decide how much new information is going to be let through to the cell state. This is decided by a *input gate* layer and a $tanh$ layer. The output of the input gate layer (i_t) decides which values are to be updated, and the $tanh$ layer produces a vector V_t of new candidate values.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$V_t = \tanh(W_V \cdot [h_{t-1}, x_t] + b_V)$$

The new output state (C_t) is produced as: First the old state C_{t-1} is then multiplied f_t to use the learned forgetting of information, and is then added together with $i_t V_t$ which are the candidate values at time t, scaled by the input gate i_t to decide how to much to update state's values V_t.

$$C_t = f_t \cdot C_{t-1} + i_t \cdot V_t)$$

The final output h_t will be a filtered version of the cell state which will decide what parts of the cell will be output. The entire cell state is ran through another $tanh$ layer, and then multiplied with the values of the third sigmoid layer o_t.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \cdot \tanh(C_t)$$

2.3.4 Seq2Seq Encoder-Decoder Models

Sequence-to-sequence learning is what it sounds like, learning to map one sequence to another e.g. learning to map a news article to a shorter summary or mapping an English sentence to a Swedish one. Sequence-to-sequence learning was introduced in 2014 by Sutskever et al. [45] as a way to map variable length input to variable length output by using two components, an encoder and a decoder. Both the encoder and the decoder consists of stacked recurrent units, most commonly LSTM cells where in the encoder each cell accepts one input element, token in the case of summarization, and propagates the information of that token forward. The decoder works in the same way were each cell passes a long its hidden state and produces one output y_t at the time step t.

The Encoder and Decoder are combined into one Encoder-Decoder model through the final hidden state produced by the encoder which is fed to the decoder (as seen in fig 2.6), this vector is sometimes referred to as the embedding vector, context vector or thought vector as it encapsulates all the information from the input elements to help the decoder make accurate predictions.

The main issues with this architecture for summarization, and Seq2Seq tasks in general, lies in dealing with longer inputs. The reason is that the

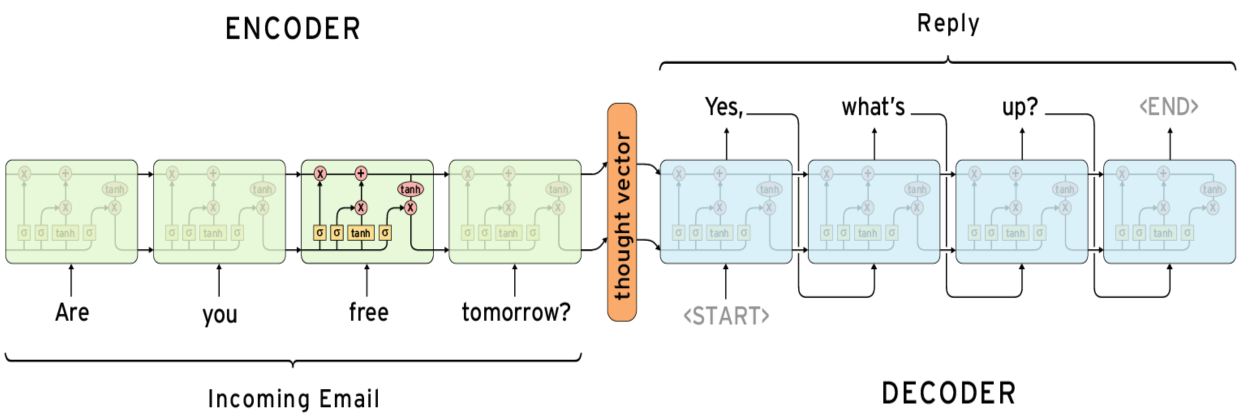


FIGURE 2.4: *Figure showing multiple LSTM cells connected in an Encoder-Decoder architecture. Diagram by Chris Olah [42]*

longer the input text is, the more information is lost as the output information of the encoder is compressed into the fixed length thought vector. LSTM networks do better than RNNs to handle long dependencies remembering or forgetting important information in the memory cell state, however they still do not perform well on long texts.

One of the main reasons for this is that the likelihood of keeping the context information about a word is exponentially decreased the further it gets from the current word being processed, meaning that when LSTMs process long texts words in distant positions are often forgotten by the network. The encoding of a token is dependent only on the previous hidden state and the memory state and as such there is no explicit modeling of dependencies, long or short.

Another issue with LSTMs is that they are sequential, every new time step needs the hidden state from the previous step making it hard to parallelize the training, leading to very long training times.

To remedy some of these problems the concept of attention was introduced as a way to model dependencies without regards to distance of the position they have in the input or output sequence [46, 47]. In these cases recurrent units (e.g. LSTM) are still used but instead of encoding the whole input text into a hidden state passed to the decoder in every encoding step, the hidden state of each word is passed a long all the way through the encoder to the decoding steps. In each decoding step the hidden state of the decoder is used together with the hidden state of every word passed to the encoder, where a neural network model is trained as part of the whole model to learn which inputs are the most important given the hidden state of the current decoder step.

This addresses the issues that words would lose importance depending on the position in the input sequence since the decoder now has access to all the words in the input at every decoding step. In addition, every word is represented by its hidden state and rather than allowing the decoding step to use information from specific words, the full sentence representation is compressed into one thought vector. The problem that attention couldn't solve in the recurrence based encoder-decoder models is the sequential nature, where each step is dependant on the previous one, and to improve upon that a new architecture was invented which resulted in a family of models called "transformers" [48].

2.3.5 Transformers

The start of transformer models came from the paper called "Attention Is All You Need" [48] by Vaswani et al. where the name of the paper alludes to a new innovation where the model architecture without recursion, such as the LSTM, is used and instead of recursion only attention is needed. The transformer model architecture seen in fig 2.5 consists of an encoder (the left part of the figure receiving the input) and a decoder (the right part of the figure receiving the output generated so far).

At every time step the transformer takes the complete text as input and calculates the probability of the next output token, because of this it does not suffer from any long range dependency issues. Since it processes the whole text for every prediction there is no risk of forgetting important information when using past hidden states. Finally, because transformers do not depend on previous hidden states' calculation the model allows for calculations to be parallelized leading to much reduced training in addition to the better predictive performance.

For the transformer models to work well it still needs to understand the order of words in a text which a LSTM encoder-decoder model got inherently due its sequential nature. The transformers solves this through adding a positional encoding and multi-head attention where the positional encoding is either a fixed non learnable sinusoid embedding [48] or a learned positional embedding as introduced by Gehring et al [49].

In each time step all the tokens are mapped into a continuous embedding space which is combined together with the positional encoding to form the input for the attention layers. The attention module applies self-attention

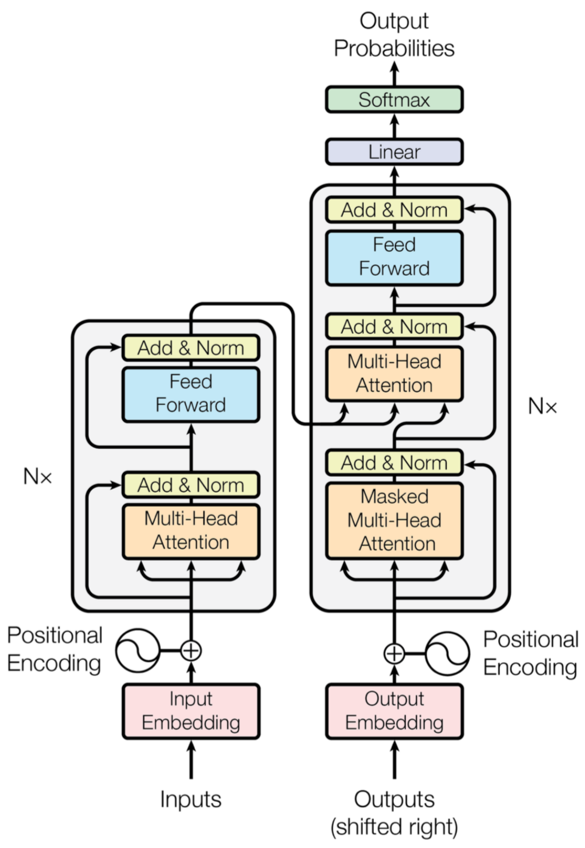


Figure 1: The Transformer - model architecture.

FIGURE 2.5: *Figure of the transformer architecture used in "Attention Is All You Need" by Vaswani et al. [48]*

which is a technique for relating different positions of an element in a sequence, e.g. a token in a text, to compute a representation of that sequence.

In the transformer model the first step to calculating self-attention is to convert the input vectors (one for each embedded token) to three new vectors called *Query* (Q), *Key* (K) and *Value* (V) vectors. The $\vec{Q}$,$\vec{K}$,$\vec{V}$ vectors are created by multiplication of three different weight matrices W_Q, W_K, W_V, projecting the learned embeddings into a lower dimensional space to improve computation. The parameters of the weight matrices W are learned as part of the end-to-end training of the transformer.

The self attention score for each token is calculated by multiplication of Q and K, as seen on the left of figure 2.6, where the set of all queries and keys of dimension d_k are stacked in matrix and calculated simultaneously. The results are scaled and passed through a softmax function resulting in a normalized score for each token. The score represents the importance of different tokens in the input sentence as each token is encoded at a specific position. Finally the score matrix is multiplied with the value matrix and this is the output which is used as input to a regular feedforward neural network layer, summarized as [48]:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

In the transformer models a variation of attention called multi-head attention is often used (show on the right side of figure 2.6) where the number of heads is an implementation choice of the architecture. This improvement to attention makes it possible for the transformer to attend to the information at different positions using different representation subspaces.

The attention calculation for each head is the same as the scaled dot product attention but instead of one set of weight matrices $W = \{W_Q, W_K, W_V\}$ there is a set of weight matrices for each attention head n: $W_n = \{W_{Qn}, W_{Kn}, W_{Vn}\}$. The Q,V,K matrices are passed through linear layer before going through attention score calculation and in the end the output is N matrices where N is number of attention heads. The feedforward layer after the attention layers expects one matrix with a vector representing each words to achieve this the output of the different attention heads are concatenated and multiplied again with weight matrix that is learned as part of training.

The encoder and decoder part of the transformer work very similarly where the difference is that the decoder has two self attention modules, one to pay attention to the input (the connection to the encoder) and the other one uses a mask to pay attention to only the output generated so far.

Transformer models have been outperforming traditional encoder-decoder models for sequence-to-sequence task and in summarization they have shown great success in the abstractive summarization field, currently holding state of the art benchmarks [15]. One major drawback of the original transformer model for summarization is that the self-attention calculation has a quadratic time complexity $\mathcal{O}(n^n)$ in relation the input sequence length. Many texts are long, ranging from scientific articles to full length novels and spoken word transcription of podcasts.

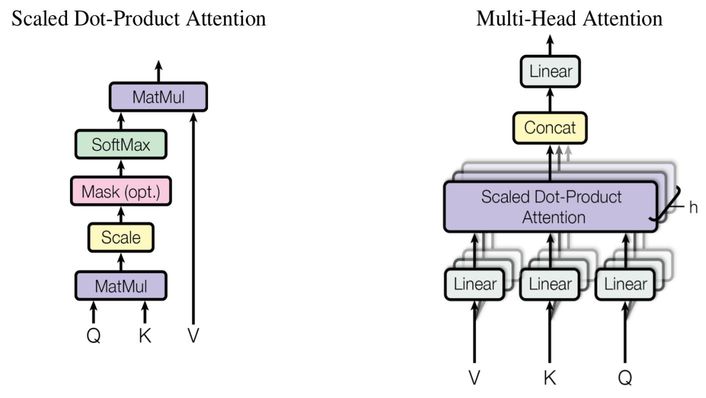


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

FIGURE 2.6: *Figure of attention used in "Attention Is All You Need" by Vaswani et al. [48]*

BERT

Bidirectional Encoder Representations from Transformer or BERT [50] for short is a transformer based model using only the encoder part of the transformer. The innovation BERT brought to the transformer models was a way to train a language model in a bidirectional way. This was in contrast to previous models such as Generative Pre-trained Transformer (GPT) [51] which had been trained in a unidirectional way where every token was only allowed to attend to previous tokens in the self attention layers.

The pre-training of BERT is done on unlabeled text data by conditioning on both the right and left context in all layers, and as such creating a bidirectional representations of the text. Through this pre-training the BERT model achieves state of the art results on many tasks by only adding a single output layer and fine tuning the model on the target domain and task.

To pre-train the model in a bidirectional way a method called masked language modelling (MLM) is used where each token in the unlabelled training text data has a 15% chance to be replaced with a [MASK] token. The training then consists of trying to predict the original token that was masked, based on the context of the other non-masked tokens in the text. To do the prediction a softmax layer over the vocabulary is added to end of the encoder stack and the vectors of the masked tokens are used as input to get a set of probabilities over which token is the correct one.

The second pre-training step is similar, but instead of predicting masked tokens the model has to predict if a sentence B is a continuation of a sentence A, where the sentences are separated by a specific [SEP] token. The training setup is done by swapping sentence B 50% of the time with another random sentence in the training corpus with the assumption that this sentence will be disconnected from sentence A [50].

The BERT model has seen good success in both fine tuning on specific tasks after pre-training and using the word embeddings that are learned by the encoder layers. The word embeddings can be used as features for other models and in contrast to e.g. word2vec, they are context-dependent and the embedding is instance-specific, meaning that a word will be represented differently depending on the context in appears in.

BART

Bidirectional and Auto-Regressive Transformer, or BART [52] for short, combines the BERT bidirectional encoder with an autoregressive decoder, such as GPT, into a full encoder-decoder architecture, same as the original transformer model with some minor changes. Figure 2.7 shows an example of pre-training by masking tokens in the input encoded by a bidirectional encoder and using an autoregressive decoder to predict tokens of the original document, where the input text to encoder doesn't need to align with the decoder outputs.

BART can be seen as a generalization of other pre-training schemes used by BERT and GPT with main advantage over the other methods being that any noising function can be applied to the input text, including altering the length of the input text. The arbitrary noising of the input texts allows for multiple composable approaches of which the following were used in the

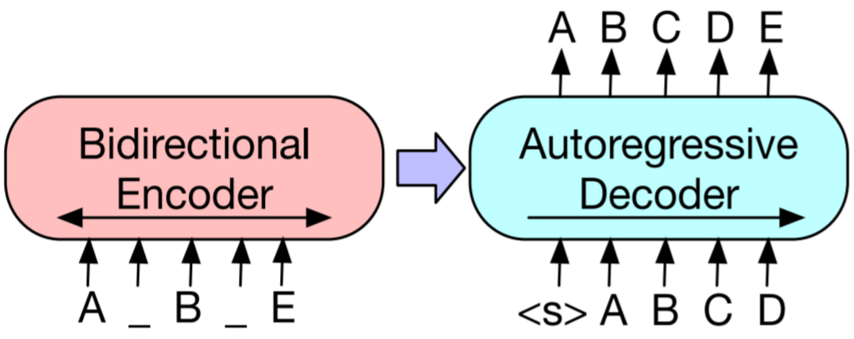


FIGURE 2.7: *Figure of training setup used in BART paper by Lewis et al. [52]*

pre-training of BART: token masking, sentence permutation, document rotation, token deletion and text infilling, as illustrated in figure 2.8.

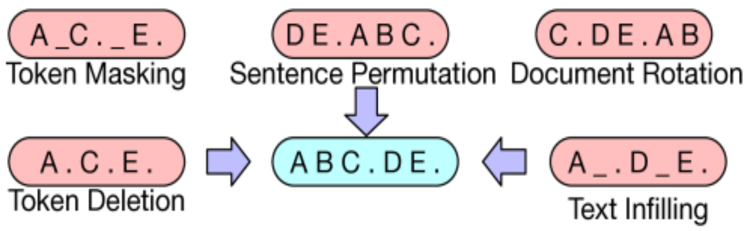


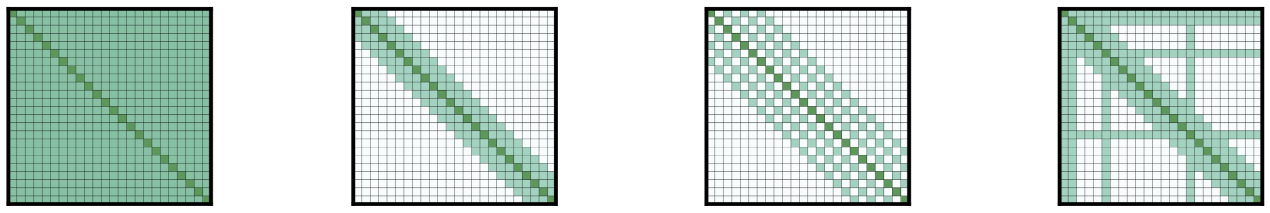
FIGURE 2.8: *Figure input sequence noising from the BART paper by by Lewis et al. [52]*

BART is especially good for text generation tasks such as summarization because the task itself is very similar to the pre-training objective, which is to copy the information in the input with some transformation added. Another benefit to using BART for summarization is that it has an encoder and a autoregressive decoder so it can be fine tuned on any sequence generation task, e.g. summarization directly [52].

Longformer

The Longformer, short for the "Long-document transformer" is a transformer model which aims to address the limitation of the quadratic time complexity of the self-attention in the original transformer [8]. To improve upon the self-attention's computational complexity the self-attention matrix is made sparse through different patterns reducing number of tokens attended to at different locations.

The longformer model has three different types of attention patterns, sliding window, dilated sliding window and global attention which can be used



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

FIGURE 2.9: *Figure of attention used in the Longformer paper by Beltagy et al. [8]*

in combination with the former ones, visualized in figure 2.9. The quadratic complexity of regular self-attention comes from the multiplication of the query and key matrices QK^T in the attention calculation $softmax(\frac{QK^T}{\sqrt{d_k}})V$ since both matrices are of size n, where n is the number of tokens in input text.

By using a window of size w where each token only attends the $\frac{1}{2}w$ tokens on each side of it, the attention calculation becomes $\mathcal{O}(n \times w)$. Using a fixed window size w this scales linearly with the input size n. As the transformer has multiple layers stacked on top of each other the final layer will have a receptive field of the input tokens that is the size of $l \times w$ for the sliding window attention pattern, where l is the number of layers.

Finally the global attention pattern can be used on specific tokens depending on the task objective, and it has been shown to be crucial for some tasks, where these tokens will use the full self-attention instead of the window attention. As long as the number of tokens are constant, and ideally low, the computational complexity is not affected by the global attention. In the extreme case where global attention is set on every token then that would be equivalent to using regular self-attention.

The sparse attention mechanism of the Longformer can act as a drop in replacement of the regular self-attention mechanism in any pre-trained transformer model and improve the performance on multiple NLP tasks.

2.4 Model Training

In this section general concepts, challenges and pitfalls related to training machine learning models are covered, as well as possible ways to improve generalization performance of the model. One of the hardest problem in machine learning is to create a model which generalizes well so that it can do accurate predictions on unseen data. Many challenges when training a model, irrespective of domain, comes from the fact the training dataset is a sample of the true distribution.

Some of the challenges include ensuring that the training sample is representative, e.g. data from one country might be different from another country when it comes to behaviour, preferences, etc. Another challenge is concept drift where the population changes and the sample that the model was trained on is no longer representative.

Furthermore bias can be introduced by the people creating the training dataset. There can also be issues in the sample itself which are problematic such as under or over-represented classes getting dis-proportionally high weight if the dataset is not large enough.

2.4.1 Bias Variance Trade-off

The bias variance trade-off is the problem of keeping two sources of error, the *bias* and the *variance*, low in a supervised learning task. The expected mean squared error of a statistical learning method can be decomposed into three quantities, the variance, the bias and the variance of the error terms [53]

$$E(y_0 - \hat{f}(x_0))^2 = Var(\hat{f}(x_0)) + [Bias(\hat{f}(x_0))]^2 + var(\epsilon),$$

where $E(y_0 - \hat{f}(x_0))^2$ is the expected mean squared error (MSE) of a point x_0. To build a good model that performs well on the test set, a good balance between variance and bias is required such that it minimizes the total error.

Variance is a measure of how a model performs when there are small changes in the training set. A model with high variance has large fluctuations when predicting a point *p* as small changes are introduced in the data, meaning that the mapping function $\hat{f}$ does not capture the underlying structure of the data. A model that has high variance can lead to random noise being modeled as relevant to the outputs, this is known as *overfitting* the model to the data.

The bias error is the error that occurs when the model makes simple assumptions about a complex problem which might result in sub-par predictive power e.g. linear regression, which assumes a linear relationship between the label *Y* and the input *X*, which is often not the case in real life[53]. A model that has high bias can result in the algorithm not learning important relations between the inputs and outputs, this is also referred to as *underfitting* the model to the data.

While it is easy to create a model with low variance and high bias, and vice versa, it is challenging to create a model with both low variance and low bias, hence the trade-off.

2.4.2 Overfitting

When fitting a model on the training data it is important to consider how well it generalizes and predicts new unseen data. In supervised learning the goal is to train a model that learns a general underlying function that maps the input data to its output.

To make sure that the model generalizes well it is important to make sure that it does not try to fit every variation in the training data, as that is more likely to be noise than the true signal. There is a greater risk of this when training highly flexible models with a high number of input dimensions and many parameters as part of the models, which is the case of text data and deep neural networks this is the case.

If a model has been trained on the training data too long it can erroneously see noise as a general pattern that is important to predicting the output and that is when it can be said that the model is *overfitting*.

A model can be 100% accurate on the training data, but due to overfitting it predicts new data poorly. An example of this is given in figure 2.10. The blue polynomial curve fits the data perfectly but oscillates heavily and is overfitting. The linear curve clearly captures the general pattern of the data better and will be able to predict new unseen data better than the polynomial curve.

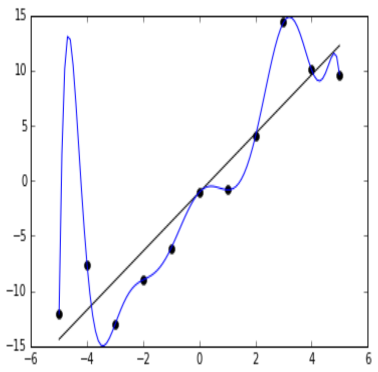


FIGURE 2.10: *A polynomial function and a linear function fit to noisy data that is approximately linear [54].*

2.4.3 Regularization

Regularization is a technique to solve the overfitting problem described above. When a model grows in complexity it tends to overfit. Regularization penalizes the complexity of the models in order to help them generalize better.

To make sure that a model is accurate a loss measurement is often used, however the loss function might show that the model is accurate, even though it is a complex model that is prone to overfitting. Regularization adds a penalty to the loss function or error function by adding a multiple of a specific norm, typically ridge (also known as L2 regularization) or lasso (also known as L1 regularization), to the input vectors. This multiple is controlled by the free parameter λ. The impact of the regularization term on the generalization error can be significant, thus the regularization parameter λ effectively controls the complexity of the model.

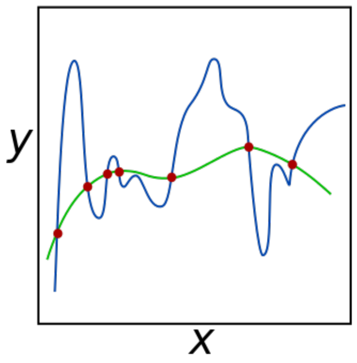


FIGURE 2.11: *Regularization example: The line with smaller oscillations shows the fitted curve after regularization [55]*

The blue line in Figure 2.12 shows an oscillating curve before regularization. Even though it captures every single data point it does not truly represent the general pattern in the data. The green line shows a line after regularization. It captures the general pattern in the data which is what is desired.

Another common regularization technique which produces good results, that is less general than L1 and L2 regularization, as it can only be used when training neural networks, is called dropout [56]. The technique works by selecting some nodes each iteration at random in a layer of the network together with the outgoing and incoming connections, and then removes them. Dropout can be configured individually for each layer in a neural network and the chance to remove a node at each iteration is a hyperparameter of the model which can be optimized.

Dropout can force the nodes of a layer to take on less or more responsibility for inputs in a probabilistic way by having the effect of introducing more noise into the training process. In certain situations some nodes can fix mistakes of other nodes, which can lead to complex co-adaptions, which in turn leads to overfitting because these co-adaptions do not generalize well [56]. By introducing noise into the training process through dropping nodes we can prevent places in the network where layers co-adapt and the model becomes more robust.

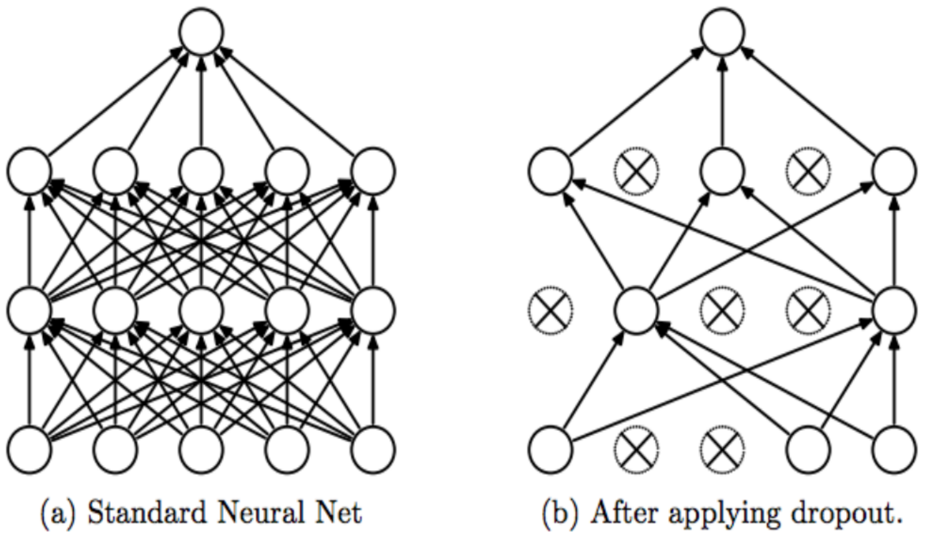


FIGURE 2.12: *Dropout as shown in the paper by Srivastava et al. [56]*

Although it is not a direct way of regularization, one of the most effective ways to decrease overfitting is to increase the amount of training data, and usually for advanced text models a lot of data is needed due to the high dimensional inputs and complex models. This can sometimes be hard if labelled data is time consuming or expensive to generate, however for some domains such as image classifications extra data can be created through augmentations, such as rotations and shifting of the images.

Another way to counteract overfitting was mentioned in section 2.4.2 where a validation set can be used during the training phase to do early stopping. Since the model evaluation is done at an interval on the validation set which is separate from the training data, this gives a better estimate of the models ability to generalize in contrast to only looking at the loss function on the training set. The case can happen where the training error keeps decreasing but the validation error increases and this where the model starts to overfit. Normally early stopping is decided based on a parameter tied how much the validation error is allowed to increase before stopping.

2.4.4 Dimensionality Reduction

Dimensionality reduction refers to the task of reducing the dimensions of the feature space while retaining the information that the feature set contains, or in some cases improving the features at hand. The number of features in a real world dataset can often be very high, and with each feature the dimensionality of the machine learning problem increases. However, it is not unusual that the output of the classifier only depends on a few of the many features used [57].

Dimensionality reduction can be done in many ways. Features that do not add new information might be scrapped in order to reduce the feature space and reduce computational overhead, e.g. if there are two features that describes weight, one in kg and the other in lbs, one of the two is redundant and could be dropped from the feature set without losing information. In the case of text data, reducing the input dimensions is more complex where the input is commonly represented as a sparse vector in the size of the vocabulary of the model. The vector can then be transformed into a dense vector of lower dimensionality through algorithms such as Word2Vec (2.2.7) or learnt as layer, part of a neural network (2.2.8).

2.5 Model Evaluation

Model evaluation, model choice and model training, can make a big difference in how well good models can be trained. Without good evaluation there can be no statement of quality about a model which is believable, and because of this standardized ways of evaluating models are a key part of the machine learning domain.

There are publicly shared datasets on which research can be compared and common metrics for evaluation, and for text summarization the most popular metric is called Rouge score. In this section the Rouge score, as one of the main evaluation metrics in the experiments, is described more in detail. In addition, two more evaluation metrics, BERT score and BLEURT score, are described, as well as techniques such as dataset splitting to better evaluate the generalization ability of a model. How well a model performs on unseen samples is one of the most important features of a good model and therefore important to evaluate as accurately as possible.

2.5.1 Dataset Splitting

To evaluate the final accuracy of a model the ideal estimation method would be one the has both low variance and low bias [58]. There are multiple ways to assess the accuracy of a model, each with their own variations, strengths and weaknesses. The method used to evaluate the models in the experiments is a mutually exclusive split of the dataset into three sets called training set, validation set and testing set. The draw back of this method is that both the validation set and the test set are only used to evaluate the model and therefore can not be used to directly train the model, this can be a problem due to the fact that high quality labelled data is both time consuming and expensive to create.

There are other types of methods such as k-fold cross-validation and bootstrap sampling which can utilize all the samples during training whilst still providing a good estimate of how the model generalizes [58]. Cross-validation is mostly used for model selection since it is more biased than using a full hold-out set for estimation, however in situation of low data volume it can be worth the risk of overfitting to have more data for the model to train on. The drawback with cross-validation is that it requires high computational cost because a model as to be trained and evaluated on k folds and this is also the reason as why a more simple dataset split is used to evaluate the models trained for the experiments.

To split the dataset $\mathscr{X}$ it is divided into a training set, validation set and a test set e.g 75% training, 10% validation and 15% test, will be chosen by randomly sampling data points into each set, both the the test and validation holdout estimation will be affected by how this division is done. The test set will be used for final evaluation of the model and cannot be used or in any way influence the model training and tuning [58]. The main purpose of the test set is to give a true estimation of performance and this also circumvents some of the problems of overfitting since the test set will be a sample of unseen test data. The validation set is not always needed but for neural network models it is often useful to be able to do early stopping to prevent the model from overfitting. For early stopping the validation set can be used similarly to the test set to estimate the models performance after n number of steps or epochs of training.

Using the assumption that the accuracy of a classifier becomes higher as the number of data points seen by the classifier increases, it can be said that the holdout set estimation method is a pessimistic estimate of the model's accuracy because the model has only seen a subset of the data during training. The fewer data points kept for the test set the wider the confidence interval for the model's accuracy becomes, and if more samples are kept for the holdout test set then the more biased the estimation becomes [58].

2.5.2 ROUGE Score

ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. It contains a set of evaluation metrics to automatically estimate the quality of a generated summary by comparing it to a golden reference summary created by a human. The metrics include a count of the number of overlapping word pairs, word sequences and n-grams for comparison between the golden human created summaries and the computer generated summaries that are to be evaluated, where the original paper introduces 4 different metrics: ROUGE-N, ROUGE-L, ROUGE-W, and ROUGE-S [59].

Looking closer at ROUGE-N and ROUGE-L, the evaluations used in this thesis, ROUGE-N is an overlap of n-grams where ROUGE-1 compares how many shared uni-grams that exists in the system generated summary and the reference summary, ROUGE-2 compares matching bi-grams and so on. In ROUGE-L the L refers the longest common sub-sequence (LCS), where a benefit is that LCS does not require matches that are consecutive, but matches that are in-sequence which reflect sentence level word ordering. Another advantage is that no predefined n-gram length is needed since it automatically uses the longest in-sequence common n-grams.

Given the following summaries $\{s_1, s_2, s_3\}$ and reference $\{r_1\}$:

- r_1: sheriff shot the outlaw
- s_1: sheriff shoots the outlaw
- s_2: the outlaw shoots sheriff
- s_3: the outlaw sheriff shot

In the example presented both summaries s_1 and s_2 would receive the same ROUGE-2 scores since they both have one bi-gram in common, "the outlaw", with the reference, for ROUGE-1 they both share three uni-grams, "sheriff", "the" and "outlaw". The example also illustrates how ROUGE-L can be useful where summary s_1 is correctly conveying the same information as the reference whilst s_2 is providing the opposite of what happened. In this case using ROUGE-L summary s_1 would get a higher score than s_2 due to the common sequence "sheriff, the, outlaw" in s_1 and the reference. Finally s_3 shows a weakness with ROUGE-L in that it only counts the main in-sequence words and therefore no other sequences provide any input to the results, for summary s_3 only "the outlaw" or "sheriff shot" are matched with the references as the LCS, but not both.

The ROUGE score have been found to achieve high correlation with human judgements in summarization tasks [59], however even though ROUGE has been shown effective at capturing n-gram overlaps between the summary and the reference, it has limited ability to capture rephrasing, synonyms, topical content and other metrics which can reflect the true quality of a summary [60].

2.5.3 BERT Score and BLEURT

ROUGE score is the most commonly used automatic evaluation metric for summarization however, it is not a perfect metric since it relies only on matching n-grams between the generated text and the reference text. This means that if the words in the n-gram do not exactly match then they are not counted towards the final score even though the word might mean something very similar in that context. Abstractive summarization models aim to capture the salient parts of the input text and generate a summary, where a strength of the model is that it can formulate or phrase the summarized text in a different way than the reference without losing the meaning of the text. In these cases ROUGE score unfairly disadvantages such summaries.

As models have improved, evaluation has stayed the same for a long time and there has been some criticism that ROUGE score does not reflect the true quality of a summary [60, 61]. Recently other automatic evaluation metrics have been suggested which can improve in the areas of preserving meaning and evaluating semantic and lexical diversity when comparing a summary to the reference. Two of the metrics used in the evaluation the podcast summaries are called BERT score [62] and BLEURT [63], which aims to improve the automatic summarization evaluation metrics.

BERT Score

BERT score works by calculating a similarity score between each token in candidate text and the reference text by using a contextual word embedding such as BERT and comparing it to other tokens using cosine similarity [62]. The score produced for each token can additionally be weighted with inverse document frequency scores, and although the similarity is calculated per token in isolation the context of the text comes from the embedding itself.

The final score between the candidate and the reference is an F1 measure, which is the harmonic mean of precision and recall. To produce a precision score, each token in the candidate is matched to a token in the reference and vice versa for recall, were each token in the reference is matched to a token in the candidate summary. The matching is done by a greedy algorithm where the token is matched by the highest consine similarity measure in the other text.

Finally the BERT score allows for weighting, with the reasoning that previous work have shown that rare words can have greater value in determining if two texts are more similar or not [64, 65], e.g. applying an inverse document frequency weighting calculated from the reference summaries.

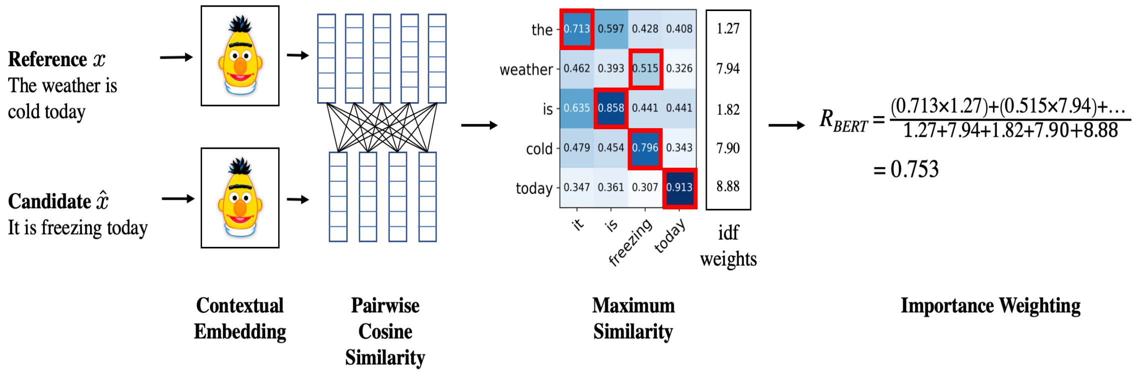

Reference x
The weather is cold today
Candidate $\hat{x}$
It is freezing today
the 0.713 0.597 0.428 0.408
weather 0.462 0.393 0.515 0.326
is 0.635 0.858 0.441 0.441
cold 0.479 0.454 0.796 0.343
today 0.347 0.361 0.307 0.913
it
is
freezing
today
1.27
7.94
1.82
7.90
8.88
idf weights
$R_{BERT} = \frac{(0.713\times1.27)+(0.515\times7.94)+\ldots}{1.27+7.94+1.82+7.90+8.88}$
$= 0.753$
Contextual Embedding
Pairwise Cosine Similarity
Maximum Similarity
Importance Weighting

FIGURE 2.13: *Figure of the calculation of BERT score recall from the original paper where the greedy matching is highlighted in red [62]*

BLEURT Score

BLEURT is a trained automatic evaluation metric based on BERT that aims to model human judgments. In contrast to BERT score which is a hybrid metric mixing trained elements such as the BERT contextual embeddings with encoded logic, e.g. token alignment, BLEURT score is a fully learned score. The challenges when learning an evaluation metric from human scores are multiple e.g. training data is hard to come by and a learned scoring model can suffer from both domain drift and quality drift [63].

A key part of the BLEURT score is that improves robustness and expressivity by pre-training the metric on large amounts of synthetic data and then it is fine-tuned on the target human ratings. The pre-training consists of first generating synthetic sentence pairs by perturbing segments randomly from Wikipedia and then training on different natural language generation (NLG) domain tasks [63].

Finally the fine-tuning is done by using the BERT transformer together with a [CLS] token which is passed through a linear layer at the end to predict the target variable, in this case the human assigned score. The function to be learned can be expressed as:

Given a candidate sentence prediction $\vec{x} = (x_1, ..., x_n)$ and a reference sentence $\vec{\tilde{x}} = (\tilde{x}_1, ..., \tilde{x}_n)$ passed through the BERT transformer $BERT(\vec{x}, \vec{\tilde{x}})$ a set of contextualized embedding vectors $\{\vec{x}_1, ..., \vec{x}_n\}$, $\{\vec{\tilde{x}}_1, ..., \vec{\tilde{x}}_n\}$, $\vec{v}_{[CLS]}$ is returned.

The training learns a function $f(\vec{x}, \vec{\tilde{x}}) \to y$ where both the parameters of the BERT model are updated (finetuned), as well as the final linear layer and bias applied added on top the [CLS] token vector $W\vec{v}_{[CLS]} + b$.

2.6 Related Work

There have been many different approaches, as surveyed by Tay et al. [66], to ease the quadratic computational complexity of the transformer models. The work in this thesis was done in relation to the strategies that focuses on not needing full attention and in that way reduces memory and computational requirements. Sparse Transformers by Child et al. [67] lowered the computational complexity to $\mathcal{O}(n\sqrt{n})$ and then the Reformer by Kitaev et al. [68] got it down to $\mathcal{O}(n\log(n))$ by using a nearest neighbors computation.

Multiple models have recently achieved a linear complexity $\mathcal{O}(n)$ including but not limited to: Linformer (Wang et al.) [69], Longformer (Beltagy et al.) [8], Performers (Choromanski et al.) [70], Linear Transformers (Katharopoulos et al.) [71], and BigBird (Zaheeret al.) [9].

The work in this thesis closely resembles that of the BigBird transformer and is almost exactly the same as the Longformer-Encoder-Decoder (LED) described in the Longformer paper, with differences in the experiments being model checkpoint initialization, model length and the podcast dataset finetuning. Both models use a local sliding window attention which can be combined with global attention, and additional random attention patterns in the BigBird model. In both cases a bidirectional encoding and auto-regressive decoders are used and the models have been successfully evaluated on summarization tasks.

Chapter 3

Method

The following chapter describes how the methods and algorithms were used in the experiments. An overview of the final model is given, as well as a detailed specification of the dataset and how it was created. Finally the experimental setup, from data splits to evaluation metrics, is described and what tools were used in the process.

3.1 The Podcast Dataset

The dataset consists of 105,360 transcribed podcast episodes encompassing nearly 60,000 hours of speech [2]. Podcasts come in a variety of formats, styles, structures, cadence and content, offering multiple new challenges in natural language understanding, including specific challenges for summarization. The variety and diversity of the podcast format can be described in different dimensions that can be combined in different ways, a few examples are listed as follows:

- **Format**: Fiction/non-fiction, documentary, interview, conversation, debate, story, etc.
- **Genre**: Education, Science, Lifestyle and Culture, News, Sports, Politics, etc.
- **Structure**: Scripted/improvised, formal/informal, narration, multiple speakers etc.
- **Production**: Professional, amateur.

In addition to the diverse content in the episodes there are other challenges when working with transcriptions for the summarization task. One of them is speech recognition errors as the model used for transcribing is not prefect with a noted word error rate of 18.1%. Another is the conversational nature of the podcasts, leading to utterances becoming fragmented and other disfluencies appearing, e.g from other speakers interrupting each other. Finally the creators' descriptions of episodes, which are used as the target when training, are noisy and sometimes not good summaries of the episodes.

When creating the Spotify podcast dataset, the episodes were sampled uniformly at random where 10% came from professional producers and the rest came from amateur podcast creators. In addition to the sampling used to create the dataset the following filters were applied to ignore some episodes [2]:

- The dataset is restricted to English as the only language. The filter is based of creator provided metadata tags and an additional n-gram language identification script is used to remove non English podcasts episodes.
- Any episodes over 90 minutes in length that are not professionally produced have been removed.
- To remove any podcast that has more music than speech (music > 50%) of the content, a proprietary speech detection algorithm was used to identify such podcast episodes.

The dataset characteristics can be summarized from the original paper [2] as follows:

- The 100k episodes come from 18,376 different podcast shows.
- There are more than one episode from 52% of the shows that are represented.
- The average length of a podcast episode is 5,700 transcribed words or 33.8 minutes of audio, with a large variance in the whole set.
- The descriptions of the podcast episodes as provided by the creators have on average a length of 85 words.
- Ranked in the order of most common to least common categories and weighted by episode length: Comedy, Sports, Health & Fitness, Society & Culture, and Education, Science, News & Politics, Government & Organization, and Fiction. Where the category was added by the podcast creators in the RSS feed.

In addition to baseline description of the dataset an extra table, Table 3.1, is provided in this thesis, looking at the lengths of the podcasts in tokens using the same BART tokenizer as used in the models for the experiments. This is because the tokenizer for BART uses the BPE algorithm which often results in more tokens per example than splitting the text on spaces, furthermore the input length is a focus of the experiments and is therefore one of the more important descriptive measurements of the dataset related to the results.

Statistic / # Tokens	Mean	Median	25%	75%	95%	Min	Max
Transcription	6780	6149	2447	10231	15729	18	49840
Episode description	75	57	32	98	206	4	1886

3.2 The Combined Model

As part of the experiments a combined model using a pre-trained checkpoint of BART and the Longformer's attention layers is investigated. The architecture of BART is the same except for the replacement of the self-attention layers in the encoder with the sparse attention layers used by the Longformer. BART has been shown to be effective on seq2seq tasks, especially on text generation objectives such as summarization due to its pre-training objectives [52], however it runs into the quadratic computational complexity problem of self-attention when faced with longer input texts.

Longformer made it possible to efficiently calculate attention, scaling linearly with the input sequence length, allowing for longer texts as input to the transformer model. The original Longformer paper started from a pre-trained checkpoint of RoBERTa [72] where the fully connected self-attention was replaced by sparse attention and the position embeddings were extened up to the size of 4096.

Instead of initializing the new position embeddings randomly they were initialized by copying the embeddings from the pre-trained model side by side multiples times. Previous analysis of the attention heads for BERT have show that they contain a strong learned bias toward attending to the local context and therefore copying works well because it preserves this local structure everywhere except at the partitions boundaries, which can be learned during fine-tuning [73].

The Longformer attention mechanism can be used as a drop-in attention, replacing the regular self-attention in pre-trained transformer models, leading to gains on multiple NLP tasks [8]. That idea and the same technique of copying the positional embeddings is used in the combined Longformer-BART model where the BART encoder's attention layers are replaced with the Longformer silding window attention and the positional embeddings are extended to 4096. The learned positional embeddings with a max position of 1024 tokens for BART is copied four times side by side, allowing the positional embeddings to encode an input sequence of up to 4096 tokens.

The decoder part of BART still keeps the full self-attetion with the motivation that the target texts are not expected to be long by analysis the podcast creator descriptions. The final LongformerBART model is finetuned on the podcast transcripts.

Independent of this work the Longformer paper's authors have since the original paper released an updated paper with where they introduce the Longformer-Encoder-Decoder (LED) [8]. The LED model uses the same methodology as previously described about the combined LongformerBART model with differences being that they used up to 16k tokens as input length as well as starting from a different BART checkpoint. The authors show that the model achieves state of the art results on the arXiv summarization dataset [74] using an input length of up to 16k tokens.

3.3 Preprocessing

Preprocessing is the task of preparing the raw data and transforming it into a representation which can be used by the model. It includes transforming the data to formats beneficial for the target domain, filtering bad data points, cleaning noisy data, normalizing dimensions and other task specific processing of the data. Real world data often is noisy, inconsistent, and incomplete and preprocessing has become a necessary step in machine learning to ensure that the quality of the data is good enough for learning and analysis [57]. Analyzing data that has not been properly preprocessed might produce misleading results as the data itself might be misleading.

Many of the labels, descriptions uploaded by the podcast creators, used when training the model are noisy and they can be too long, to short or not a good summary, e.g. not summarizing relevant content. Additionally the podcast creators' descriptions also contains boilerplate text with metadata about which tools were used and which platform the episode was uploaded to.

The creator descriptions are treated as the labels in the supervised fine-tuning of the models and to increase the quality of the labels multiple pre-processing steps were applied to the descriptions. The thresholds set are a relaxation of the "brass" dataset filtering, suggested in by authors of the podcast dataset, and they are the same as were used in the TREC podcast track overview paper. The filtering was applied as follows:

- Data points were filtered from the training set based on the creator description length. Texts that were are either too short or too long, being below the set minimum of 10 characters and the maximum of 1300 characters, were discarded from the training set.

- The creator descriptions were run through a TF-IDF vectorization and the resulting vectors were compared using cosine distance as a measure of similarity. If a set of data points were above threshold of 0.95 then they were considered too similar and one data points was kept at random whilst the others were removed from the training set.

- Many creator descriptions had issues with boilerplate text being present and to improve the quality, a BERT [50] based classifier was used to classify which sentences were boilerplate sentences. The sentences flagged by the model were removed but the training data point and description was kept in the training set. The boilerplate classification model training was done by manually labelling 1000 podcast episode descriptions and using that as a dataset [75].

The tokenization method used in the experiments is the BPE algorithm with the same modifications described in the GPT-2 paper where the algorithm is prevented from doing merging between character categories in any byte sequence with the exception of spaces. This is to avoid different versions of common words taking up limited vocabulary slots, e.g. *cat*, *cat?* and *cat!* [76].

3.4 Shared Task at TREC 2020

The Text REtrieval Conference (TREC) is a series of workshops within different areas of information retrieval (IR) called tracks in the TREC context. It was started in 1992 [77] with the purpose of supporting research in the information retrieval community through making infrastructure available for large-scale evaluation of methods within text retrieval.

The TREC conference is co-sponsored by the National Institute of Standards and Technology (NIST) and the U.S. Department of Defense. The workshop series have 4 goals officially stated as follows [10]:

- Encourage research in information retrieval based on large test collections.
- Increase communication among industry, academia, and government by creating an open forum for the exchange of research ideas.
- Speed up the transfer of technology from research labs into commercial products by demonstrating substantial improvements in retrieval methodologies on real-world problems.
- Increase the availability of appropriate evaluation techniques for use by industry and academia, including development of new evaluation techniques more applicable to current systems.

At TREC 2020 a podcast track was introduced for the first time with two challenges, search and summarization [78]. As a part of this thesis a run was submitted for the summarization part of the podcast track.

3.5 Evaluation

The main evaluation metrics used are Rouge scores (described in section 2.5.2), more specifically Rouge-1, Rouge-2 and Rouge-L. The scoring was used together with the commonly used performance metrics *precision*, *recall* and *F-measure* which are defined as follows:

$$tp = true\ positive, \quad tn = true\ negative, fp = false\ positive, \quad fn = false\ negative$$

$$Precision = \frac{tp}{tp + fp} \tag{3.1}$$

$$Recall = \frac{tp}{tp + fn} \tag{3.2}$$

$$F\!-\!measure = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3.3}$$

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{3.4}$$

For Rouge score, recall measures how many n-grams in the reference summary appear in the automatically generated summary, it can be seen as how much of the reference summary that the generated summary is captures.

$$Recall = \frac{number\_of\_overlapping\_ngrams}{number\_of_n grams\_in\_reference} \tag{3.5}$$

Although recall is a good measure, an automatic summary can be very long and noisy and still capture all the n-grams in the reference, this would be a bad summary. This is where precision comes in as it measures ratio of how many n-grams in the automatically generated summary appear in the reference summary, it can be seen as how much of the generated summary is relevant in relation to the reference summary.

$$Recall = \frac{number\_of\_overlapping\_ngrams}{number\_of\_ngrams\_in\_system} \tag{3.6}$$

Finally F-measure is the harmonic mean of recall and precision, however the meaning of this score is harder to interpret.

In addition to ROUGE two more evaluation metrics, BERT [62] score and BLEURT [63], are used to grade the summaries and this is because n-gram based evaluation metrics such as ROUGE have flaws that are better addressed by these scores. One area in which ROUGE score fails, is e.g to match paraphrasing correctly where the score will be an underestimation of semantically equal phrases because they don't match on a shallow n-gram level, however this can be captured through contextualized token embeddings [50].

Another reason as to why ROUGE score, as well as other n-gram based scores, are not enough to evaluate a summary on its own is that it fails to capture long range dependencies and re-orderings and as such semantically relevant orderings are penalized [79]. An example would be where cause and effect can be swapped between one phrase P_1 and another phrase P_2, with small penalization for n-gram based methods, and becoming smaller the longer the phrases are.

- P_1: "A because of B"
- P_2: "B because of A"

In contrast, contextualized embeddings are trained to handle distant ordering and dependencies. In abstractive summarization re-ordering and rephrasing is more common as the decoder generates text, which is different to extractive summarization where input text is preserved as it was. The summaries generated in this thesis are based mostly on abstractive methods and such it becomes more important to have good evaluation metrics the can capture how good the summary is.

3.5.1 Manual Evaluation

As a part of the TREC 2020 submission the combined LongformerBart model produced summaries for 179 transcribed podcast episodes and these episodes were then manually evaluated by humans to produce high quality labels. The evaluators were asked to grade the summaries according to a quality score and in addition to that they also answered a set of yes/no questions related to gauging the quality of the summaries.

The organizers of the podcast tracks sent out the following yes/no questions and quality score descriptions to the evaluators, where these scores and descriptions will act as a reference for the results presented in section 4.2.

Quality scores:

- **(3) Excellent**: The summary accurately conveys all the most important attributes of the episode, which could include topical content, genre, and participants. In addition to giving an accurate representation of the content, it contains almost no redundant material which is not needed when deciding whether to listen. It is also coherent, comprehensible, and has no grammatical errors.
- **(2) Good**: The summary conveys most of the most important attributes and gives the reader a reasonable sense of what the episode contains with little redundant material which is not needed when deciding whether to listen. Occasional grammatical or coherence errors are acceptable.
- **(1) Fair**: The summary conveys some attributes of the content but gives the reader an imperfect or incomplete sense of what the episode contains. It may contain redundant material which is not needed when deciding whether to listen and may contain repetitions or broken sentences.

- **(0) Bad**: The summary does not convey any of the most important content items of the episode or gives the reader an incorrect or incomprehensible sense of what the episode contains. It may contain a large amount of redundant information that is not needed when deciding whether to listen to the episode.

Yes/No Questions:

- **Q1**: Does the summary include names of the main people (hosts, guests, characters) involved or mentioned in the podcast?
- **Q2**: Does the summary give any additional information about the people mentioned (such as their job titles, biographies, personal background, etc)?
- **Q3**: Does the summary include the main topic(s) of the podcast?
- **Q4**: Does the summary tell you anything about the format of the podcast; e.g. whether it's an interview, whether it's a chat between friends, a monologue, etc?
- **Q5**: Does the summary give you more context on the title of the podcast?
- **Q6**: Does the summary contain redundant information?
- **Q7**: Is the summary written in good English?
- **Q8**: Are the start and end of the summary good sentence and paragraph start and end points?

3.6 Experimental Setup

The experiments are structured by first establishing a set of baseline models that do not require any training and then continuing to more advanced models starting from different pre-trained checkpoints followed by finetuning on the podcasts dataset. The final combined model (described in section 3.2) created as part of this thesis is compared to the increasingly complex baseline models using three different types of evaluations metrics, Rouge, BERT score and BLEURT score. The baseline models and the final model are described as follows:

Lead-50

A Lead-3 model has been shown produce very good results on the CNN/-dailymail summarization dataset [17] where the model simply picks the first three leading sentences [80]. The lead-50 models used in the experiments is inspired from this one but instead of choosing the 3 first sentences it produces the 50 first tokens extracted through splitting the text using spaces. The reason to use tokens split by spaces is that the model tokenization is based on subwords which doesn't make much sense in a summary, and the number 50 is chosen as it roughly matches the average target description length in the dataset of 75 subword tokens.

This model works especially well for news datasets because the first section usually summarizes what the article is about. This reasoning is extended to podcast episodes as well where a host might introduce what the episode is about before diving into a discussion and therefore **lead-50** serves well as a first baseline model to improve upon.

TextRank

TextRank [81] is a graph-based model that can be used as an unsupervised method to extract sentences from a text and it falls within the area of extractive summarization methods.

To find the most important sentences in a text the algorithm constructs a graph where each sentence is represented as a vertex and the edges between sentences are based on how many tokens are shared between the two sentences. Using the graph representation of the text, the sentences can be fed into the Pagerank algorithm [82] which ranks the sentences according to importance. The summary produced by the textrank algorithm for the experiments combines the top two sentences extracted to form a summary where the number 2 was chosen as it most closely, on average, matched the mean target length.

Bart-cnn

The model referred to as bart-cnn is a pre-trained checkpoint of the **bart-large-cnn**[1] model which has been trained on the CNN/dailymail dataset. The model has good performance when summarizing news articles and the reason this is used as model for comparison is that although the model does

[1]https://huggingface.co/facebook/bart-large-cnn, Accessed: 2021-04-11

not know anything about the podcast domain it has learnt the task summarizing text and his might generalize to a reasonable performance on the podcast dataset as well.

Bart-podcasts and Bart-cnn-podcasts

Building on the **bart-cnn** pre-trained checkpoint the **bart-cnn-podcasts** model uses the same starting point and is then finetuned on the podcast dataset, where the idea is that the model will benefit from the already learned task of summarization from the pre-trained checkpoint. To contrast this, a **bart-podcasts** model is also trained and uses the a more general purpose pre-trained checkpoint **bart-large**[2] to start from, and is then finetuned on the podcast dataset.

LongformerBart

Both the **bart-podcasts** and **bart-cnn-podcasts** models were compared to choose which pre-trained model would be best suitable to extend when creating the **LongformerBart** model, and in the end **bart-large-cnn** was chosen as the base to start from. The **LongformerBart** model extends the positional embeddings from 1024 to 4096 and the embeddings from **bart-large-cnn** are copied over multiple times as described in section 3.2 and it is then finetuned on the podcast dataset. The main reason to not use longer sequences than 4096 tokens is the memory limits of the GPUs used when training and as the mean token length per podcast episode is 6780 this input length still truncates some part of the transcribed text.

During the development and finetuning of all the BART based models, the training arguments and hyperparameters were only subject to minor changes from the default settings of the library used, which is in turn based on the values used in the original BART paper [52]. The hyperparameters and training arguments used to train all the models are kept the same to introduce less variability over the different models. They are defined as follows:

[2]https://huggingface.co/facebook/bart-large, Accessed: 2021-04-11

Hyperparameter	Value
learning_rate	3e-5
gradient_accumulation_steps	32
adam_epsilon	1e-6
warmup_steps	200
weight_decay	0.01
batch_size	1
num_train_epochs	6
max_target_length	200
max_source_length	1024 - 4096
encoder_layerdrop	0.1
decoder_layerdrop	0.1
dropout	0.1
attention_dropout	0.1
val_check_interval	0.5

TABLE 3.1: Hyperparameters and training arguments for fine-tuning of the BART based models.

The training of the models were set to 6 epochs (3 epochs for the TREC submission due to time constraints) with evaluation on the validation set being performed once every half epoch. The models used an early stopping criteria using the F-measure for the Rouge-2 score on the validation set, where a decrease two times in a row would stop training and the best checkpoint based on the validation set would be chosen as the final model used for evaluation on the test set. Due to constraints on resources a target length of 200 tokes were chosen when evaluating summaries generated on the validations set during training however most labels in the dataset would not get truncated by this limit.

For the final evaluation on the test set, in addition to Rouge scores, BERT score is calculated using the **roberta-large** pre-trained model and BLEURT is calculated using the **bleurt-base-512** model released as part of the original paper. The three evaluation metrics are introduced as way to evaluate the summaries in different ways where Rouge measures n-grams overlap, BERT score uses a contextualized embeddings to match tokens and BLEURT predicts a score aimed at reflecting human judgments. Using different evaluation metrics with different strengths gives more confidence about the quality of the summaries produced by the different models.

Pre-trained models and embeddings

As a part of the original BART paper multiple models where released with weights and vocabulary which could be loaded as checkpoints to continue training from. Leveraging pre-trained models and using them to continue training the models on domain and task specific datasets is something that has been shown to work well [83]. Two models were used as pre-trained starting checkpoints when training: **BART large**[3] and **BART large CNN**[4].

The combined model was created by combining the Huggingface transformers library [84] source code[5] together with the Longformer's [8] attention source code[6]. The model training runs where done on virtual machines with 4 x NVIDIA Tesla P100 attached.

3.7 Tools

This section describes which tools and libraries that were used to run the experiments, a short overview of each tool and what it does is also provided.

Python

Python [85] was selected as the programming language because many machine learning libraries are available, as well as text processing ones. Having access to established libraries helps reduce time and ensure that the code for the algorithms is well implemented and tested. The libraries that were used are the following:

- **Numpy** [86]: Adding support for large arrays, in addition it provides high level mathematical functions to perform operations on these arrays.
- **Pandas** [87]: A library primarily used for data manipulation and analysis, optimized for performance.
- **Huggingface Transformers** [84] is a state-of-the-art natural language processing library which focuses on deep learning neural network models with the transformer architecture. It provides thousands of models and API:s to easily download, modify and finetune models on research data.

[3]https://huggingface.co/facebook/bart-large, Accessed: 2021-04-11
[4]https://huggingface.co/facebook/bart-large-cnn, Accessed: 2021-04-11
[5]https://github.com/huggingface/transformers, Accessed: 2021-04-11
[6]https://github.com/allenai/longformer, Accessed: 2021-04-11

- **PyTorch** [88] is a package that on a high level provides two things, GPU accelerated tensor computation and deep neural networks built on tape-based autograd system[7].

- **Sci-kit** [89] learn is built on top of scientific python (SciPy), it is a library which contains multiple different machine learning algorithms and common preprocessing steps used when training models, e.g. TF-IDF.

- **Rouge score** A python library for calculating Rouge-N and Rouge-L scores [90], it is a native python program designed as a re-implementation of the original Perl script [59].

- **Huggingface Datasets** [91] is a natural language processing library which provides efficient ways to pre-process and load data, sharing datasets, loading datasets, and loading and using evaluation metrics.

- **PyTextRank** [92] is an implementation of the TextRank algorithm [81] used for graph-based text processing such as extracting top ranked sentences from a document.

[7]https://github.com/pytorch/pytorch, Accessed: 2021-04-11

Chapter 4

Results

This chapter presents the results from the different experiments as described in section 3.6, in the Method chapter. It also includes a subsection which covers the results of the submission to the TREC 2020 shared task for the summarization task of the podcast track.

4.1 Experiments

The results of the experiments are presented on a per evaluation metric basis where the score for one metric is reported for all the models in one table. The metrics are first reported on an overview level e.g. the mean scores followed by a visualization of the score distribution for each model per metric.

In Table 4.1 the mean of precision, recall and F-measure, represented as P, R and F respectively in the tables, for three different Rouge scores, Rouge-1, Rouge-2 and Rouge-L, are shown. The **LongformerBart** model with an input sequence length of 4096 has the highest Rouge scores across the board followed by the BART model starting from a checkpoint pre-trained on the CNN/dailymail summarization dataset [17] and then fine-tuned on the podcast dataset, marked as **bart-cnn-podcasts**.

Figure 4.1 visualizes the distribution of F-measure for the Rouge-2 score metric. Table 4.2, Table 4.3 and Table 4.4 present a detailed view of Table 4.1 where each of the respective Rouge scores, Rouge-1, Rouge-2 and Rouge-L, have values reported for precision, recall and F-measure in a 95% confidence interval.

Model	Rouge-1			Rouge-2			Rouge-L		
	Rouge scores (Mean)								
	P	R	F	P	R	F	P	R	F
lead-50	0.2129	0.2326	0.1888	0.0459	0.0512	0.0405	0.1376	0.1624	0.1253
textrank	0.1639	0.2405	0.1605	0.0227	0.0340	0.0219	0.1002	0.1585	0.0997
bart-cnn	0.2537	0.2413	0.2087	0.0699	0.0617	0.0547	0.1701	0.1698	0.1420
bart-podcasts	0.3340	0.2576	0.2473	0.1137	0.0895	0.0848	0.2386	0.1940	0.1805
bart-cnn-podcasts	0.3449	0.2911	0.2706	0.1261	0.1107	0.1010	0.2476	0.2209	0.1992
LongformerBart	**0.3597**	**0.3248**	**0.2967**	**0.1440**	**0.1370**	**0.1228**	**0.2635**	**0.2501**	**0.2226**

TABLE 4.1: Rouge scores.

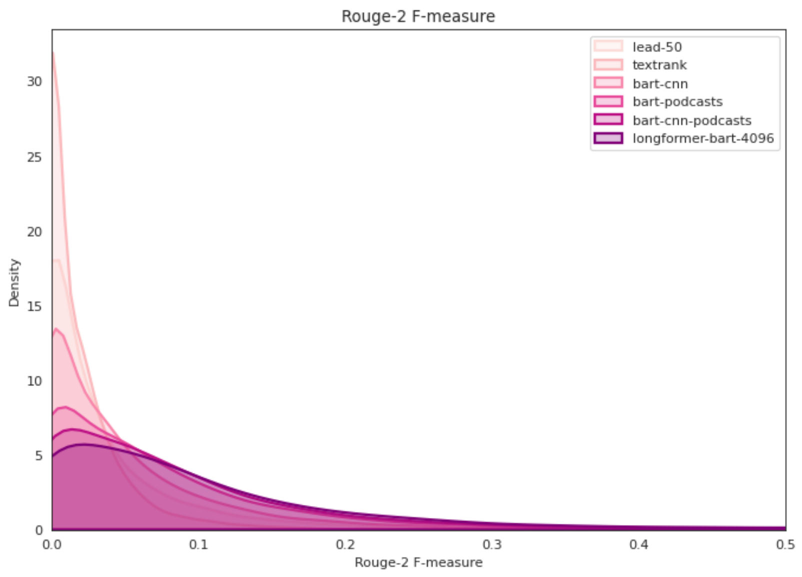


FIGURE 4.1: *Rouge-2 distribution plot using F-measure*

Model	Rouge-1 (95% confidence interval)		
	P	R	F
lead-50	0.2109 - 0.2151	0.2305 - 0.2347	0.1873 - 0.1902
textrank	0.1622 - 0.1657	0.2385 - 0.2425	0.1592 - 0.1617
bart-cnn	0.2514 - 0.2562	0.2393 - 0.2433	0.2072 - 0.2103
bart-podcasts	0.3312 - 0.3368	0.2551 - 0.2600	0.2453 - 0.2492
bart-cnn-podcasts	0.3421 - 0.3478	0.2884 - 0.2936	0.2685 - 0.2728
LongformerBart	**0.3570 - 0.3627**	**0.3217 - 0.3278**	**0.2944 - 0.2991**

TABLE 4.2: 95% confidence intervals for Rouge-1 scores.

Model	**Rouge-2 (95% confidence interval)**		
	P	R	F
lead-50	0.0448 - 0.0472	0.0496 - 0.0528	0.0395 - 0.0416
textrank	0.0220 - 0.0235	0.0330 - 0.0351	0.0213 - 0.0226
bart-cnn	0.0683 - 0.0716	0.0603 - 0.0632	0.0536 - 0.0559
bart-podcasts	0.1116 - 0.1159	0.0875 - 0.0916	0.0831 - 0.0866
bart-cnn-podcasts	0.1237 - 0.1284	0.1084 - 0.1131	0.0991 - 0.1031
LongformerBart	**0.1414 - 0.1464**	**0.1345 - 0.1397**	**0.1205 - 0.1249**

TABLE 4.3: 95% confidence intervals for Rouge-2 scores.

Model	**Rouge-L (95% confidence interval)**		
	P	R	F
lead-50	0.1360 - 0.1391	0.1606 - 0.1643	0.1242 - 0.1265
textrank	0.0992 - 0.1013	0.1568 - 0.1600	0.0989 - 0.1005
bart-cnn	0.1683 - 0.1720	0.1680 - 0.1716	0.1407 - 0.1433
bart-podcasts	0.2363 - 0.2409	0.1917 - 0.1963	0.1788 - 0.1823
bart-cnn-podcasts	0.2451 - 0.2502	0.2183 - 0.2233	0.1972 - 0.2012
LongformerBart	**0.2609 - 0.2660**	**0.2472 - 0.2529**	**0.2202 - 0.2247**

TABLE 4.4: 95% confidence intervals for Rouge-L scores.

Table 4.5 presents the mean values of the BERT score calculated for each generated summary and reference pair and in Figure 4.2 the different models' distributions of F-measure values are plotted as visualization of the BERT score metric. BERT score allows for calculation of precision, recall and F-measure (described in section 2.5.3) and a deep dive of the quantiles of their values are presented in Table 4.6 (precision), Table 4.7 (recall) and Table 4.8 (F-measure).

BERT score (roberta-large) (mean)			
Model	Precision	Recall	F-measure
lead-50	0.8207	0.8207	0.8204
textrank	0.8103	0.8197	0.8147
bart-cnn	0.8392	0.8313	0.8349
bart-podcasts	0.8629	0.8448	0.8534
bart-cnn-podcasts	0.8669	0.8507	0.8582
LongformerBart	**0.8699**	**0.8581**	**0.8635**

TABLE 4.5: Mean BERT scores for precision, recall and F-measure.

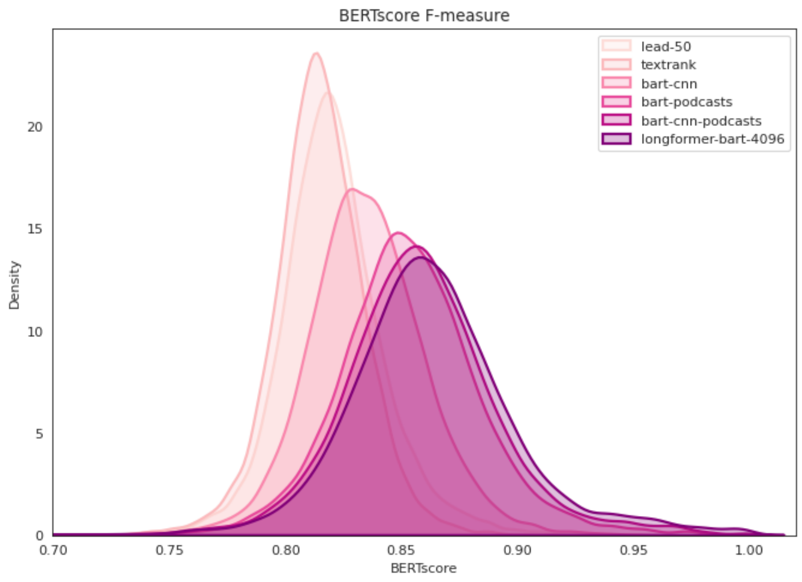


FIGURE 4.2: *BERT score distribution plot using F-measure*

BERT score - Precision							
Model	Mean	Median	25%	75%	95%	Max	Min
lead-50	0.8207	0.8182	0.8074	0.8315	0.8596	0.9650	0.6953
textrank	0.8103	0.8092	0.7976	0.8220	0.8442	0.9381	0.6081
bart-cnn	0.8392	0.8371	0.8206	0.8556	0.8854	0.9834	**0.6983**
bart-podcasts	0.8629	0.8630	0.8423	0.8827	0.9174	**1.0000**	0.6597
bart-cnn-podcasts	0.8669	0.8672	0.8454	0.8877	0.9241	**1.0000**	0.6764
LongformerBart	**0.8699**	**0.8695**	**0.8472**	**0.8907**	**0.9322**	**1.0000**	0.6066

TABLE 4.6: Summary statistics of the precision metric for BERT score.

BERT score - Recall							
Model	Mean	Median	25%	75%	95%	Max	Min
lead-50	0.8207	0.8210	0.8029	0.8386	0.8713	0.9697	0.6034
textrank	0.8197	0.8209	0.8038	0.8378	0.8632	0.9574	0.6030
bart-cnn	0.8313	0.8318	0.8127	0.8511	0.8844	0.9810	0.6032
bart-podcasts	0.8448	0.8439	0.8222	0.8667	0.9090	**1.0000**	0.6085
bart-cnn-podcasts	0.8507	0.8491	0.8265	0.8729	0.9230	**1.0000**	0.5947
LongformerBart	**0.8581**	**0.8555**	**0.8318**	**0.8805**	**0.9386**	**1.0000**	**0.6069**

TABLE 4.7: Summary statistics of the recall metric for BERT score.

BERT score - F-measure							
Model	Mean	Median	25%	75%	95%	Max	Min
lead-50	0.8204	0.8192	0.8068	0.8324	0.8592	0.9499	0.6773
textrank	0.8147	0.8143	0.8031	0.8265	0.8465	0.9438	0.6733
bart-cnn	0.8349	0.8341	0.8188	0.8504	0.8775	0.9645	0.6796
bart-podcasts	0.8534	0.8522	0.8343	0.8711	0.9048	**1.0000**	0.6804
bart-cnn-podcasts	0.8582	0.8568	0.8379	0.8765	0.9146	**1.0000**	**0.6812**
LongformerBart	**0.8635**	**0.8612**	**0.8419**	**0.8822**	**0.9261**	**1.0000**	0.6736

TABLE 4.8: Summary statistics of the F-measure metric for BERT score.

Finally the quantiles of the BLEURT score are reported in Table 4.9 and the distribution of all the models' BLEURT scores are visualized in a plot seen in Figure 4.3.

BLEURT scores (bleurt-base-512)							
Model	Mean	Median	25%	75%	95%	Max	Min
lead-50	-1.1745	-1.1794	-1.3233	-1.0209	-0.7302	0.5987	-2.5323
textrank	-1.1628	-1.1617	-1.3292	-0.9903	-0.7149	0.3304	-2.3415
bart-cnn	-1.0644	-1.0760	-1.2663	-0.8609	-0.5374	0.4957	-2.4768
bart-podcasts	-0.8807	-0.8931	-1.1308	-0.6420	-0.2566	0.9988	-2.4406
bart-cnn-podcasts	-0.8150	-0.8181	-1.0776	-0.5684	-0.1870	**1.0231**	-2.5764
LongformerBart	**-0.7687**	**-0.7682**	**-1.0319**	**-0.5159**	**-0.1049**	**1.0231**	**-2.4496**

TABLE 4.9: Summary statistics of BLEURT scores using the the bluert-base-512 model for evaluation.

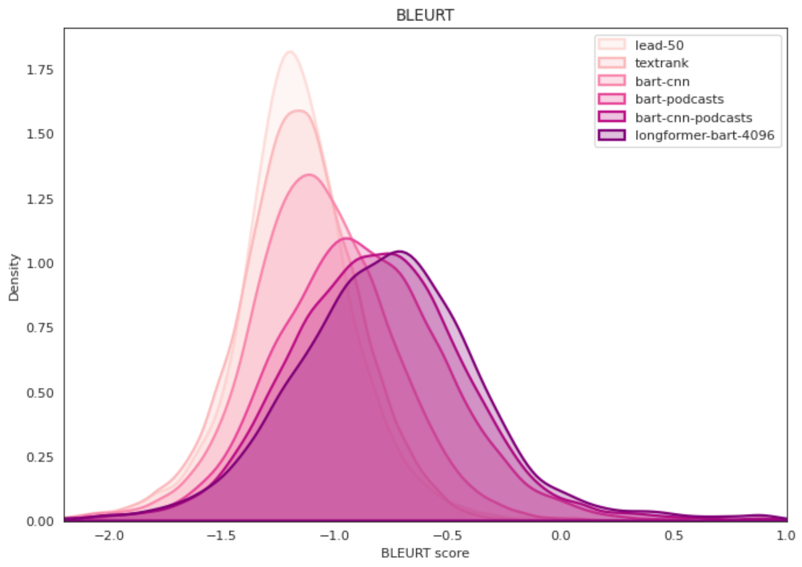


FIGURE 4.3: *BLEURT scores distribution plot*

4.2 Shared Task at TREC 2020

The results presented in this section are as received as a participant in the TREC conference, no further evaluation has been calculated for the generated summaries. For the submission, one run of the combined model, referred to as **LongformerBart**, was submitted. The results of the human evaluation, *yes/no* questions and *quality* scores, on 179 podcast episode transcriptions are visualized as bar plots in Figure 4.4 and in Figure 4.6. The LongformerBart model performance is shown in comparison to the creator descriptions and the creator descriptions filtered from boiler plate information (same method as described in the preprocessing section 3.3).

Figure 4.4 displays a bar for each quality rank of the summaries where the y-axis shows the episode count with a total of 179 episodes.

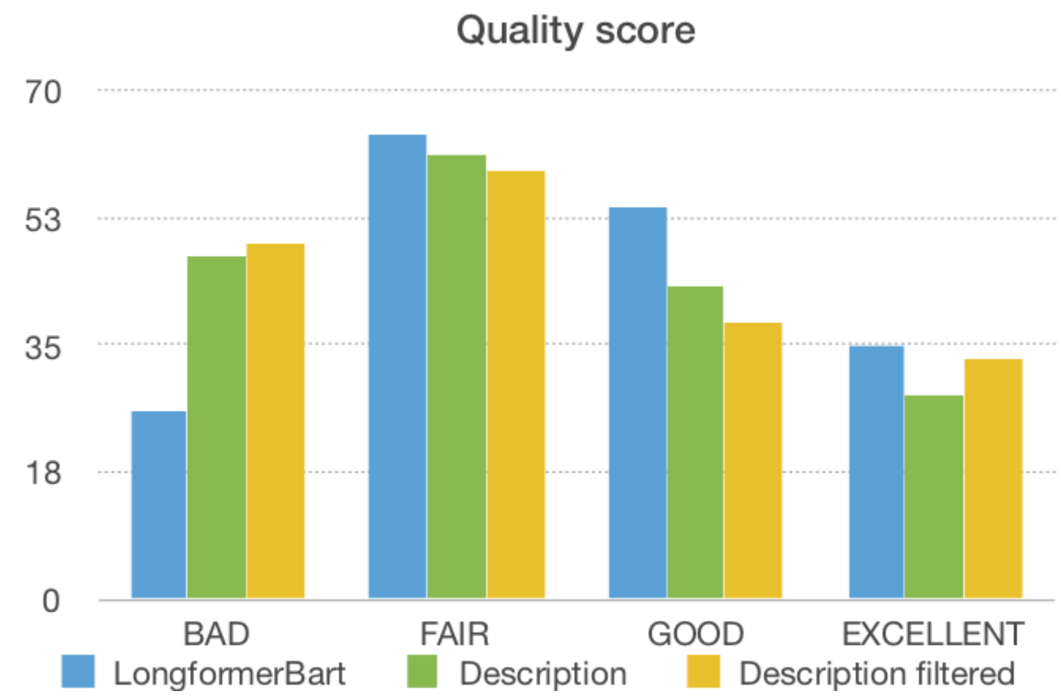


FIGURE 4.4: *Quality scores achieved compared to creator descriptions*

Figure 4.5 displays same plot as Figure 4.4 but with scores added from the baseline models presented in the TREC podcast overview paper [78]. The baseline models from TREC are similar in methodology and name to the ones used in the experiments but they are not the same models.

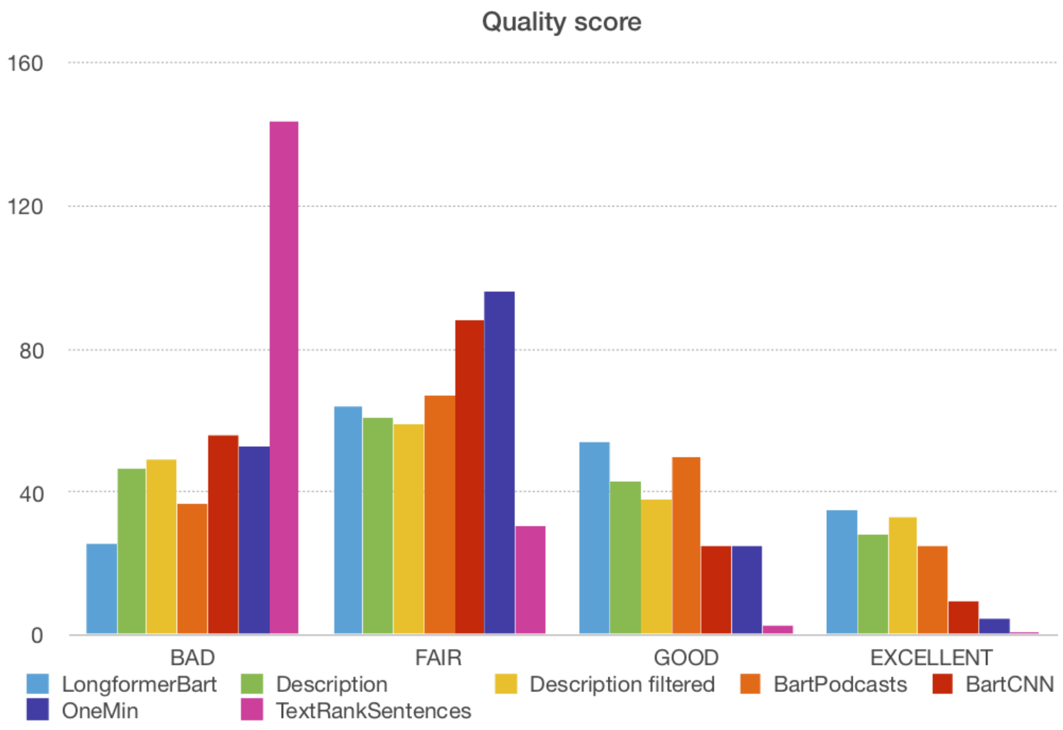


FIGURE 4.5: *Quality scores achieved compared to creator descriptions and baselines from the TREC podcast track overview paper [78]*

In Figure 4.6 the y-axis represents the percentage of podcast episode summaries that received the answer yes to each respective questions. The set of bars representing each question is described in detail in section 3.5.1 with the corresponding enumeration.

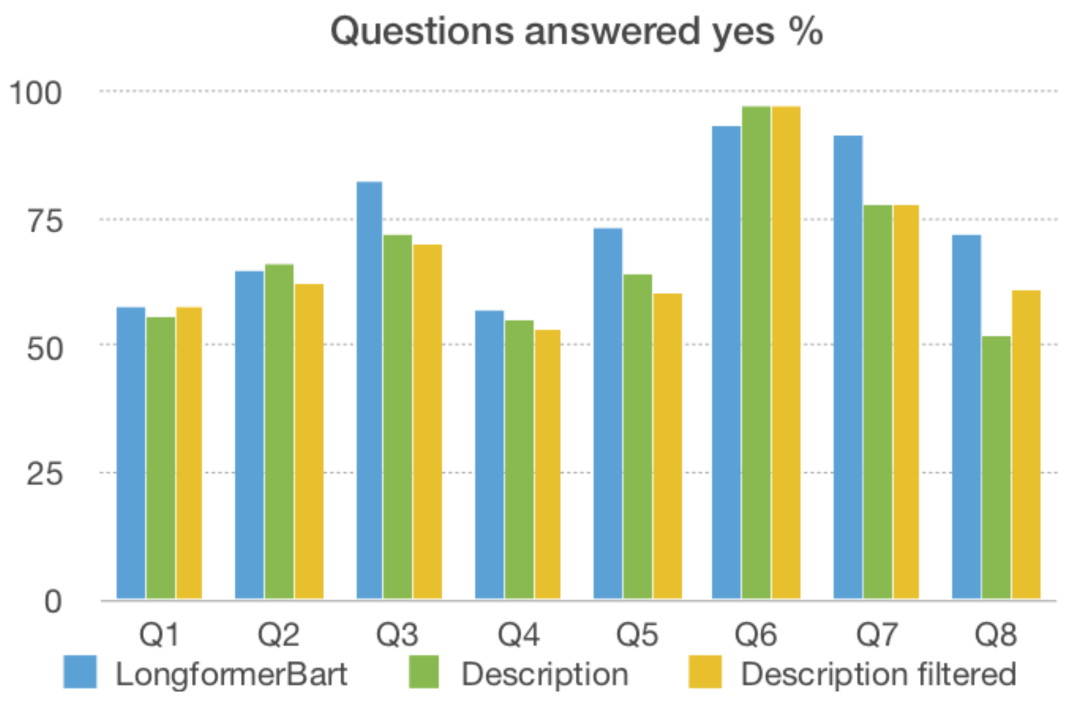


FIGURE 4.6: *Per question the percentage of yes answers compared with creator descriptions*

Table 4.10 presents the mean ROUGE-L scores for recall, precision and f-measure for the LongformerBart model's summaries against the noisy, non-filtered, creator descriptions, along with values for the 95% confidence interval of the evaluation metrics.

ROUGE-L Average scores		
Recall	Precision	F-measure
0.18983	0.26462	0.19234
Recall (95%-conf.int)	Precision (95%-conf.int)	F-measure (95%-conf.int)
0.25362 - 0.27536	0.25362 - 0.27536	0.18083 - 0.19886

TABLE 4.10: The average Rouge-L scores for the 179 episodes evaluated

Chapter 5

Discussion

Starting by looking at the TREC results presented in section 4.2, they show that the LongformerBart models produces higher *quality* scores and more favourable answers in the response to the *yes/no*-questions. This is interesting since the creator descriptions are treated as the labels or gold truth when training the model but the summaries produced by the model could potentially be better than the target summaries themselves. This is counter-intuitive since it by definition goes against the objective of the training, however it starts to make more sense with a closer look at the podcast creator provided descriptions.

The creator descriptions of the podcast episodes are noisy and of highly varying quality which is noted both in the TREC podcast overview paper [78] and the podcast dataset paper [2], where they are, after some filtering, referred to as "brass" instead of golden. The LongformerBart model managed to generalize well enough to produce a significantly lower amount of bad summaries, as seen in Figure 4.4, compared to the sample descriptions and this can also be related to the pre-training done for the checkpoint from which the model training started. In the pre-training of the BART model the task of masked language learning is done for a large amount of text data where the model, among other things, learns what a good English sentence looks like.

Looking at Figure 4.6 where question 6-8 are related to the language of the summary, e.g. if it is written in good English, the LongformerBart model does quite a bit better than the descriptions, suggesting that the model can use the attribute of being a good language model to improve upon noisy creator descriptions.

Furthermore looking at question 3 and 5 asking about the context and the main topics of the episode, the LongformerBart model also manages to outperform the descriptions by the creators themselves. Overall the LongformerBart model performed well on the summarization task at TREC 2020, outperforming the baselines and multiple other submissions presented in the overview paper, where Figure 4.5 shows the quality score in comparison to the score of the baselines from the overview paper.

Looking at the main experiments in section 4.1 the worst performing models are lead-50 and textrank which are the ones extracting text directly from the source transcriptions. The results get improved as models get more advanced and it is clear that finetuning the models on the dataset is important. Additionally it is interesting to see that there seems to be a clear benefit to start training from a pre-trained model checkpoint which has already been trained on the task of summarization, although in a different domain. Finally the results show that there is significant improvement using the LongformerBart models over the regular bart-cnn model, where we can see an improvement of 21.6% in the the Rouge-2 score F-measure metric.

Both the the BERTscore and BLEURT metrics show similar results with the graph in Figure 4.3 having slightly less overlap between the models compared to Figure 4.2 showing the BERTscore distribution. Comparing the two distribution plots to the plot of Rouge score seen in Figure 4.1 makes it more clear how the different models compare to each in the case of BERTscore and BLEURT, whilst for Rouge is harder to interpret because of the long tail and the high concentration of scores between 0.0 and 0.1. The values of e.g. BLEURT score doesn't make much sense by themselves and do not have any meaningful interpretation but seem to be useful in comparing models against each other to understand how their summaries' quality differ.

Chapter 6

Conclusion

This thesis set out with the goal to quantify the performance of abstractive summarization models on a challenging dataset of over 100,000 transcribed podcast episodes. To start off, a couple of extractive summarization models requiring no training were used as baselines to improve upon. The first one extracting the first 50 tokens of the transcription and the other one using page rank to find the most relevant sentences and merging them to create a summary. The next step was to apply previously proven state of the art abstractive summarization models to see how they compared against the naive extractive summarization baselines.

The BART transformer model was chosen as a base for further experiments since it works well for summarization. Two different pre-trained BART checkpoints were used, one trained for general purpose use and one trained on news article summarization, both were finetuned on the podcast dataset and additionally the one trained on news summarization was also used directly for inference. Finally the BART model was extended by replacing the attention layers with sparse attention from the Longformer model, creating a new model called LongformerBart. In relation to the research question of whether an abstractive summarization model could be successful, all the abstractive models finetuned on the podcast dataset showed that, in both generated examples (see apendix A) and automatic evaluation metrics (see section 4.1), the models should be considered to be summarizing podcast transcriptions well.

The purpose of the combined model was to investigate the hypothesis that information in the later parts of the podcast is important when summarizing an episode, and using longer input texts from the podcast transcriptions can improve the summarization. There is a distinction between a model being better because it has wider context for each token and a model being better because it can capture something important that was discussed at a later part of podcast episode, however the thesis mainly investigated impact of increasing the input length. The hypothesis was investigated by altering the BART model in such a way that the architecture stays the same except for the how many tokens the model can attend to, which directly affects how many tokens can be used by the encoder and decoder when generating a summary.

The experiments show a high increase of 21.6% for the Rouge-2 score when comparing the LongformerBart model to the baseline of the finetuned BART pre-trained checkpoint, which was the same one that the Longformer-Bart model was created from. This strongly suggests that being able to use longer parts the transcriptions as input is highly relevant to creating good podcast episode summaries. This also answers the research question of whether BART can be extended to handle long inputs and how well that works where the results show that this is not only possibly but also offers a significant improvement over the regular finetuned BART models for podcast summarization.

Another objective of the project was to participate in the shared task of TREC for the podcast track, and as part of the work a run was submitted using the the LongformerBart model. The results were presented at the main conference workshop sessions and a paper was submitted to the TREC 2020 proceedings [93]. The final results of human evaluation of the model shows that it was a success, surpassing the quality scores of the creator descriptions as well as performing comparatively well to the other submissions for the podcast summarization task.

A hypothesis was also that abstractive summarization would be a good approach due to the podcast formats and the fact that the text being summarized is transcribed from an automatic system. The abstractive models manages produce good summaries which are shown both in the experimental results and in the manual evaluation of the TREC submission. Looking at the qualitative data of the human judgements the LongformerBart model performs very well on questions related to linguistic quality of the summary. This gives plausability to the hypothesis that abstractive models are good choice for summarizing podcast transcriptions because they can rephrase the noisy format and smooth over any disfluencies and transcriptions errors well. Although only basic extractive models are evaluated as the baselines they perform very badly suggesting that an extractive approach might be limited by the noisy texts of the transcriptions.

Finally the thesis looked at three alternative automatic evaluation metrics, Rouge, BERT score and BLEURT, to estimate the quality of the summaries produced by the different models. From the results in the experiments we can see that using all the three scores together provided a more complete and robust view of how the models compared to each other. The Longformer-Bart came out on top for all the evaluation metrics but since the best summaries were produced by abstractive summarization models it would have been hard to say that LongformerBart was better using only the Rouge score.

The conclude the thesis, it can be said that to produce good quality summaries of podcast transcription it is essential to use as much of the text input as possible. Transformer models with sparse attention using a sliding window is an effective way to allow for longer text inputs whilst generating good summaries. Adding additional automatic evaluation metrics when running experiments gives a more robust view of summary quality when comparing abstractive summarization models to each other.

6.1 Future Work

Due to prohibitive resource requirements and training time, only one model with input sequence length of 4096 was trained but the model could be trained for different input lengths e.g. 2048, 6144, 8192 and 12,288 to study more closely how much the input length has an impact on the model performance. There is room make use of extra information in the podcast episodes since the mean number of tokens in a transcription is 6780.

The Longformer paper [8] showed that the best performing version of their LED model, which is very similar to the LongformerBart model from the experiments, on the arXiv dataset was using a sequence length of 16,384. Further ablation studies would also be a possible avenue for further work by modifying hyperparameters or others parts of the model, e.g. attention window, to see what effect the have on the result, however the training time might be prohibitive. Furthermore additional testing of effective transformers, recently surveyed by Tay, Yi, et al. [66], such as the BigBird transformer [9] can be applied to the podcast dataset to determine which model architectures work best for text transcriptions in the podcasting domain.

Another avenue for future work could be to introduce more information related to the content of the podcast such as topics or name entities to see if summaries can specialized when it comes to a specific topic or mentioning of e.g. important people that were discussed in the podcast episode.

Appendix A

Model generated summaries

This section showcases a few summaries generated by the models, and contrasts them with the episode description by the podcast creator. The summaries generated were picked from test set that was used as part of the final model evaluations.

Table A.1 is an example of an episode where the creator provided description is bad, essentially telling a potential listener little about what the episode is about, other than that it is probably about Josh Gordon. The episode has a length that is slightly longer 4096 tokens and as such the Longformer-Bart model can use most of input whilst the other abstractive summarization models truncate a big part of the transcription.

Looking at the summaries the ones created by the extractive models are hard to make sense of, and even the non-finetuned bart-cnn model is hard to read coherently. From there on it gets progressively better and it is hard to say definitely which one is better, bart-cnn-podcasts or LongformerBart. The LongformerBart model captures more names and the discussion about how he can help them win the super bowl, as well as covering the other information more concisely.

Model	**Summary**
lead-50	Hello, everyone. It is Bryson and a fell here Return of the champ. Shut the fuck up. Oh my God, he's just really pretty. We'ii. Just were we everyone everyone else gave up everyone else gave up. I kept it. He gave up Alonzo. I always kept it open for
textrank	I'm very sorry for the lack of free Josh Gordon son. But let's let's put them in there have Josh Gordon NICU
bart-cnn	Bryson: Josh Gordon is back and we're going to record a little you know, recap with a Patriots Titans practices along with week. One preseason along with the week two preseason prove you after this, but right now it is free Josh Gordon.
bart-podcasts	Josh Gordon is back and we're going to recap a little you know, recap with a Patriots Titans practices along with week 1 key plays.
bart-cnn-podcasts	Josh Gordon is back and we recap the Patriots' joint practices with the Tennessee Titans. We also talk about the Week 1 preseason game against the Titans and what to expect from Josh Gordon this season.
LongformerBart	Brycen and Aidan break down the return of Josh Gordon and how he can help the Patriots win the Super Bowl. They also recap the Titans/Patriots joint practices and the week 1 preseason.
Creator description	JOSHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH GORDOOOOOOOOOOOOOOOOONNNNNNNNNNNNNNNNNNNNN

TABLE A.1: Summaries generated by the different models on an example from the test set, including a bad creator description of the episode.

In Table A.2, the description provided by the podcast creator is good and serves as a gold truth of how a human would summarize the episode. In this case there are also interesting differences between the different models, where the LongformerBart model might be the most similar to the creator description but e.g. misses the point about them being Texans, something that the creator description mentions. In this case the lead-50 baseline also works okay since they episode starts out with an introduction.

Model	Summary
lead-50	Welcome to my stupid podcast a John Mayer podcast. My name is Angelo Gonzalez. Hey, guys, I'm Jeff Whitman and we are starting a John Mayer podcast. This is our episode 1 our first episode first episode on this Venture here, and we wanted to,Art a John Mayer podcast. There's
textrank	oh you're the dude that knows John Mayer stuff, like because not a lot of people did yeah guitarist respected that and you other people that didn't listen to him or like what the Your Body Is a Wonderland guy you kidding me. So, you know, we're both John Mayer fans.
bart-cnn	Jeff Whitman and Angelo Gonzalez are starting a John Mayer podcast. They are both John Mayer fans and live in Texas. It's going to be a similar format where we will touch on everything.
bart-podcasts	In this episode, Jeff and Angelo talk about their plans for a John Mayer podcast. They also talk about the Houston Texans' loss to the Chiefs, and what it means to be a Texans fan.
bart-cnn-podcasts	Welcome to the first episode of My Stupid Podcast, a John Mayer Podcast. This is our first episode and we talk about our plans for the podcast and what it will be about.
LongformerBart	In this episode, Angelo and Jeff introduce themselves and tell their John Mayer stories. They talk about how they got into John Mayer and why they started listening to John Mayer.
Creator description	Jeff Widman and Angelo Gonzalez are two Texans who both share a love of John Mayer. They connected on a national John Mayer Facebook Group and are starting this podcast about all things Mayer. Episode 1 is an introduction to both of them, who they are, and how they got on board the Mayer Train. Enjoy!

TABLE A.2: Summaries generated by the different models on an example from the test set, including a good creator description of the episode.

Bibliography

[1] M. Maybury, *Advances in automatic text summarization*. MIT press, 1999.

[2] A. Clifton, S. Reddy, Y. Yu, A. Pappu, R. Rezapour, H. Bonab, M. Eskevich, G. Jones, J. Karlgren, B. Carterette, *et al.*, “100,000 podcasts: A spoken english document corpus”, in *Proceedings of the 28th International Conference on Computational Linguistics*, 2020, pp. 5903–5917.

[3] J.-M. Torres-Moreno, *Automatic text summarization*. John Wiley & Sons, 2014.

[4] R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang, *et al.*, “Abstractive text summarization using sequence-to-sequence rnns and beyond”, *arXiv preprint arXiv:1602.06023*, 2016.

[5] A. M. Rush, S. Chopra, and J. Weston, “A neural attention model for abstractive sentence summarization”, *arXiv preprint arXiv:1509.00685*, 2015.

[6] S. Narayan, S. B. Cohen, and M. Lapata, “Don’t give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization”, *arXiv preprint arXiv:1808.08745*, 2018.

[7] M. Gambhir and V. Gupta, “Recent automatic text summarization techniques: A survey”, *Artificial Intelligence Review*, vol. 47, no. 1, pp. 1–66, 2017.

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

[9] M. Zaheer, G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, *et al.*, “Big bird: Transformers for longer sequences”, *arXiv preprint arXiv:2007.14062*, 2020.

[10] TREC. (1992). Text retrieval conference - overview. [Online; accessed 11-Apr-2021], [Online]. Available: `https://trec.nist.gov/overview.html`.

[11] A. Clifton, A. Pappu, S. Reddy, Y. Yu, J. Karlgren, B. Carterette, and R. Jones, “The spotify podcasts dataset”, *arXiv preprint arXiv:2004.04270*, 2020.

[12] W. S. El-Kassas, C. R. Salama, A. A. Rafea, and H. K. Mohamed, “Automatic text summarization: A comprehensive survey”, *Expert Systems with Applications*, p. 113 679, 2020.

[13] A. Nenkova and K. McKeown, *Automatic summarization*. Now Publishers Inc, 2011.

[14] J. K. Yogan, O. S. Goh, B. Halizah, H. C. Ngo, and C. Puspalata, “A review on automatic text summarization approaches”, *Journal of Computer Science*, vol. 12, no. 4, pp. 178–190, 2016.

[15] S. Gupta and S. Gupta, “Abstractive summarization: An overview of the state of the art”, *Expert Systems with Applications*, vol. 121, pp. 49–65, 2019.

[16] P. Mehta, “From extractive to abstractive summarization: A journey.”, in *ACL (Student Research Workshop)*, Springer, 2016, pp. 100–106.

[17] A. See, P. J. Liu, and C. D. Manning, “Get to the point: Summarization with pointer-generator networks”, *arXiv preprint arXiv:1704.04368*, 2017.

[18] A. K. Uysal and S. Gunal, “The impact of preprocessing on text classification”, *Information Processing & Management*, vol. 50, no. 1, pp. 104–112, 2014.

[19] C. D. Manning, P. Raghavan, H. Schütze, *et al.*, *Introduction to information retrieval*, 1. Cambridge university press Cambridge, 2008, vol. 1.

[20] J. B. Lovins, “Development of a stemming algorithm.”, *Mech. Transl. Comput. Linguistics*, vol. 11, no. 1-2, pp. 22–31, 1968.

[21] C. D. Manning and H. Schütze, *Foundations of statistical natural language processing*. MIT Press, 1999, vol. 999.

[22] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units”, *arXiv preprint arXiv:1508.07909*, 2015.

[23] M. Schuster and K. Nakajima, “Japanese and korean voice search”, in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2012, pp. 5149–5152.

[24] J. W. Osborne, *Best practices in data cleaning: A complete guide to everything you need to do before and after collecting your data*. Sage, 2013.

[25] J. Tang, H. Li, Y. Cao, and Z. Tang, "Email data cleaning", in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005, pp. 489–498.

[26] M. Yamamoto and K. W. Church, "Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus", *Computational Linguistics*, vol. 27, no. 1, pp. 1–30, 2001.

[27] K. Church and W. Gale, "Inverse document frequency (idf): A measure of deviations from poisson", in *Natural language processing using very large corpora*, Springer, 1999, pp. 283–295.

[28] D. Metzler, "Generalized inverse document frequency", in *Proceedings of the 17th ACM conference on Information and knowledge management*, ACM, 2008, pp. 399–408.

[29] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries", in *Proceedings of the first instructional conference on machine learning*, Citeseer, vol. 242, 2003, pp. 29–48.

[30] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation", *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[31] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space", *arXiv preprint arXiv:1301.3781*, 2013.

[32] G. Salton, A. Wong, and C.-S. Yang, "A vector space model for automatic indexing", *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[33] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality", in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[34] Y. Goldberg, "Neural network methods for natural language processing", *Synthesis Lectures on Human Language Technologies*, vol. 10, no. 1, pp. 1–309, 2017.

[35] G. Cybenko, "Approximation by superpositions of a sigmoidal function", *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[36] A. P. Engelbrecht, *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[37] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[38] C. M. Bishop, “Pattern recognition”, *Machine Learning*, vol. 128, pp. 1–58, 2006.

[39] N. K. Kasabov, *Foundations of neural networks, fuzzy systems, and knowledge engineering*. Marcel Alencar, 1996.

[40] W. Commons. (2016). Colored_neural_network.png — wikimedia commons, the free media repository. [Online; accessed 11-Apr-2021].

[41] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, “A novel connectionist system for unconstrained handwriting recognition”, *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 5, pp. 855–868, 2008.

[42] C. Olah. (2015). Understanding lstm networks. [Online; accessed 11-Apr-2021], [Online]. Available: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[43] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[44] A. Graves and J. Schmidhuber, “Framewise phoneme classification with bidirectional lstm and other neural network architectures”, *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[45] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks”, *arXiv preprint arXiv:1409.3215*, 2014.

[46] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate”, *arXiv preprint arXiv:1409.0473*, 2014.

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

[48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need”, *Advances in neural information processing systems*, vol. 30, pp. 5998–6008, 2017.

[49] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, “Convolutional sequence to sequence learning”, in *International Conference on Machine Learning*, PMLR, 2017, pp. 1243–1252.

[50] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding", *arXiv preprint arXiv:1810.04805*, 2018.

[51] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training", 2018.

[52] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension", *arXiv preprint arXiv:1910.13461*, 2019.

[53] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 6.

[54] W. Commons. (2016). Overfitted data.png — wikimedia commons, the free media repository. [Online; accessed 11-Apr-2021].

[55] ——, (2016). Regularization.svg.png — wikimedia commons, the free media repository. [Online; accessed 11-Apr-2021].

[56] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[57] H. Lu, S. Yuan, and S. Y. Lu, "On preprocessing data for effective classification", in *ACM SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery*, Citeseer, 1996.

[58] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection", in *Ijcai*, Stanford, CA, vol. 14, 1995, pp. 1137–1145.

[59] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries", in *Text summarization branches out*, 2004, pp. 74–81.

[60] K. Ganesan, "Rouge 2.0: Updated and improved measures for evaluation of summarization tasks", *arXiv preprint arXiv:1803.01937*, 2018.

[61] N. Schluter, "The limits of automatic summarisation according to rouge", in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, 2017, pp. 41–45.

[62] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert", *arXiv preprint arXiv:1904.09675*, 2019.

[63] T. Sellam, D. Das, and A. P. Parikh, "Bleurt: Learning robust metrics for text generation", *arXiv preprint arXiv:2004.04696*, 2020.

[64] R. Vedantam, C. Lawrence Zitnick, and D. Parikh, "Cider: Consensus-based image description evaluation", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 4566–4575.

[65] M. Denkowski and A. Lavie, "Meteor 1.3: Automatic metric for reliable optimization and evaluation of machine translation systems", in *Proceedings of the sixth workshop on statistical machine translation*, 2011, pp. 85–91.

[66] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey", *arXiv preprint arXiv:2009.06732*, 2020.

[67] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers", *arXiv preprint arXiv:1904.10509*, 2019.

[68] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer", *arXiv preprint arXiv:2001.04451*, 2020.

[69] S. Wang, B. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity", *arXiv preprint arXiv:2006.04768*, 2020.

[70] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, *et al.*, "Rethinking attention with performers", *arXiv preprint arXiv:2009.14794*, 2020.

[71] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention", in *International Conference on Machine Learning*, PMLR, 2020, pp. 5156–5165.

[72] 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 preprint arXiv:1907.11692*, 2019.

[73] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, "What does bert look at? an analysis of bert's attention", *arXiv preprint arXiv:1906.04341*, 2019.

[74] A. Cohan, F. Dernoncourt, D. S. Kim, T. Bui, S. Kim, W. Chang, and N. Goharian, "A discourse-aware attention model for abstractive summarization of long documents", *arXiv preprint arXiv:1804.05685*, 2018.

[75] S. Reddy, Y. Yu, A. Pappu, A. Sivaraman, R. Rezapour, and R. Jones, “Detecting extraneous content in podcasts”, in *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, 2021.

[76] 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.

[77] D. K. Harman, *The first text retrieval conference (TREC-1)*, 500. US Department of Commerce, National Institute of Standards and Technology, 1993.

[78] R. Jones, B. Carterette, A. Clifton, M. Eskevich, G. Jones, J. Karlgren, A. Pappu, S. Reddy, and Y. Yu, “Overview of the trec 2020 podcasts track”, in *The 29th Text Retrieval Conference (TREC 2020) notebook*, NIST, 2020.

[79] H. Isozaki, T. Hirao, K. Duh, K. Sudoh, and H. Tsukada, “Automatic evaluation of translation quality for distant language pairs”, in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 2010, pp. 944–952.

[80] R. Nallapati, F. Zhai, and B. Zhou, “Summarunner: A recurrent neural network based sequence model for extractive summarization of documents”, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017.

[81] R. Mihalcea and P. Tarau, “Textrank: Bringing order into text”, in *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004, pp. 404–411.

[82] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web.”, Stanford InfoLab, Tech. Rep., 1999.

[83] S. Rothe, S. Narayan, and A. Severyn, “Leveraging pre-trained checkpoints for sequence generation tasks”, *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 264–280, 2020.

[84] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, “Huggingface’s transformers: State-of-the-art natural language processing”, *ArXiv*, vol. abs/1910.03771, 2019.

[85] G. VanRossum and F. L. Drake, *The Python Language Reference*. Python software foundation Amsterdam, Netherlands, 2010.

[86] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation", *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[87] W. McKinney, "Data structures for statistical computing in python", in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 51–56.

[88] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035.

[89] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[90] A. See, P. J. Liu, and C. D. Manning. (2017). Python rouge implementation. [Online; accessed 11-Apr-2021].

[91] T. Wolf, Q. Lhoest, P. von Platen, Y. Jernite, M. Drame, J. Plu, J. Chaumond, C. Delangue, C. Ma, A. Thakur, S. Patil, J. Davison, T. L. Scao, V. Sanh, C. Xu, N. Patry, A. McMillan-Major, S. Brandeis, S. Gugger, F. Lagunas, L. Debut, M. Funtowicz, A. Moi, S. Rush, P. Schmidd, P. Cistac, V. Muštar, J. Boudier, and A. Tordjmann, "Datasets", *GitHub. Note: https://github.com/huggingface/datasets*, vol. 1, 2020.

[92] P. Nathan, *PyTextRank, a Python implementation of TextRank for phrase extraction and summarization of text documents*, 2016. DOI: `10.5281/zenodo.4540784`. [Online]. Available: `https://github.com/DerwenAI/pytextrank`.

[93] H. Karlbom and A. Clifton, "Abstract podcast summarization using bert with longformer attention", in *The 29th Text Retrieval Conference (TREC 2020) notebook*, NIST, 2020.