University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Multi-Document summarisation: A Pointer-Generator Network with Semantic Concept Generalisation

Eóin C. McMahon

April 6th, 2020

Abstract

Copious amount of information are available to people in the modern world. However, it is unrealistic for them to be able to digest all of this information in a short period of time. For this reason it is essential to summarise the information to make its access simpler. Manual text summarisation is a laborious and expensive task, so it is imperative that the field of automatic text summarisation continues to advance. This project aims to extend the pre-existing PG-MMR multi-document summarisation model by using a mechanism of semantic concept generalisation. Often in summarisation tasks, important information is lost. By adding generalisation to the model, important information that is crucial for the context of documents is preserved. Using common evaluation metrics in the field of text summarisation, such as ROUGE, BLEU and METEOR, the model scores slightly lower than the original PG-MMR model. However, examples demonstrated that generalisation does in fact bring value and aid in keeping important information. For this reason, it is argued that future work should focus on the implementation of generalisation to other neural models.

Acknowledgements

I would like to thank Dr. Joemon Jose for his advice and guidance throughout this dissertation, without whom, this project would not have been possible. I would also like to thank my family back home in Belfast, as well as my girlfriend Jeanne, for their unwavering encouragement and support.

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: Eoin McMahon Date: 13 April 2020

Contents

1 Introduction

Text summarisation aims at condensing text down to a smaller length, whilst still retaining its key information. As information on the World Wide Web is increasingly available, the quantity of data is overwhelming and impractical for common people to absorb. News articles, books and academic works are only a few examples of material that would benefit from a shorter, more condensed format. This diversity and the extent of the information available, demonstrates the necessity of condensing this information into a smaller more readable form for people to increase their information intake.

There are different ways of condensing texts down. Either, important sentences from the text can be used to represent a smaller version of it, or new text can be generated to paraphrase the content of the source document. These two methods are called extractive and abstractive text summarisation respectively.

Text summarisation usually refers to a *single* document being condensed into one smaller summary. Multi-Document summarisation however, can take *multiple* source documents as input and produce one summary that summarises information from them all. Usually, documents like news articles have different sources, each taking an alternate view. Multi-Document Summarisation is important as it can present all of the information presented across the different sources in a single summary.

This paper is centered around the idea of Multi-Document summarisation. This paper presents an extension of two different models; PG and PG-MMR, to help improve their ability to represent large amounts of information in a condensed form. The extension to the models is a form of generalisation called 'Named Entity Generalisation', it is a method of preserving named entities in the summary by finding and generalising them in the source text before passing them to the model.

This paper is structured as follows:

- **Chapter 2: Related Work** presents the research already performed in the field of text summarisation. It showcases attempts at extractive and abstractive summarisation and explains what previous studies have achieved and how they have achieved it.
- **Chapter 3: Analysis** outlines the reflection that allowed the scope of this project to be narrowed down to the PG-MMR with Generalisation model. It also illustrates how the datasets that were used in the development of the model were chosen.
- **Chapter 4: Methodology** describes in detail the high-level architecture proposed for the model. It also explains how each separate stage of the implementation is linked and interacts with one another.
- **Chapter 5: Implementation** outlines how exactly the model proposed in Chapter 4 was realised in code. It discusses how the various technologies used which enabled the successful implementation of the proposed model.
- **Chapter 6: Evaluation** discusses the metrics used for evaluation, as well as the results produced from each of the models proposed.
- **Chapter 7: Conclusion** summarises this paper, proposes future work that should be performed in the research field and reflects on the project as a whole.

2 Related Work

Current state of the art regarding text summarisation have two forms; Single Document Summarisation (SDS) and Multi-Document Summarisation (MDS), the former being summaries generated from a singular document and the latter from multiple documents. Although there is an overlap in how SDS and MDS are performed, each presents unique technological challenges in its utilisation. Previous work in text summarisation follows one of two distinct formal methods; *extractive* or *abstractive* summarisation.

The first method, *extractive summarisation*, only uses words and sentences that appear in the source text to generate the summary. Thus, the algorithm cannot paraphrase or generalise the source text that it is provided with, but can only repeat sections of it verbatim. It *can* however, be selective in what it chooses to relay into the summary. Previously successful algorithms only use the most important and the least redundant words or sentences. A significant advantage of extractive text summarisation is its ability to reproduce factual information without any information loss. The content of the summary produced will never contradict anything that appears in the source document as it is taken from it. An example of an extractive summary is presented below in Table 2.1.

The implementation of extractive text summarisation is a less complex task than abstractive summarisation and the algorithms are less computationally expensive . However, extractive summarisation presents some drawbacks. Extractive algorithms perform well only when given small documents. Shorter documents tend to have more sentences that represent the key ideas behind the source text, and thus these document are easier to condense down to a few sentences (Magooda and Marcjan (2020)). In the case of a long document, i.e a book, there will not be any sentences that can generalise a large enough portion of it to be useful in a short summary. This leads to a summary of sentences that are grammatically correct, but struggle to capture the essence of the source document.

Research into automatic (computer-generated) extractive summarisation dates half a century back to the late 1950s, with the first attempt by Luhn (1958) from IBM's Journal of Research and Development. It was shown that automatic text summarisation is feasible and is an area worth pursuing. Since then, there have been many attempts to better solve this problem, all of which make use of a range of technologies and design patterns to produce *less* redundant and *more* salient summaries. This will be touched upon later in section 2.1.

The second form of automatic summarisation is abstractive summarisation. This area of work is more complex than extractive summarisation as it requires the generation of new words and sentences, rather than copying them. Due to its complexity, it previously has had less research dedicated to its progression. Generally, the use of abstractive text summarisation techniques results in summaries that more closely resemble those written by humans. This is because abstractive techniques have the ability to paraphrase long sections of text by reordering, generalising and making use of words that do not appear in the source text. The result is a generated summary that is much more coherent and readable than that of an extractive summary. An example an abstractive summary is shown below in table 2.1.

Table 2.1: *Example Summaries for each form of summarisation*

Source Text	U.S. prosecutors have asked for a 20-day extension to provide Germany with paperwork. The paperwork is necessary to extradite a top lieutenant of Saudi terrorist suspect Osama bin Laden. This was reported on Saturday.
Extractive Summary	U.S. prosecutors have asked for a 20-day extension to provide Germany with paperwork. This was reported on Saturday.
Abstractive Summary	on Saturday it was said that prosecutors have requested a 20-day extension for the paperwork to extradite Saudi terrorist

NB: Green represents newly generated words. Pink and Orange represent extracted sentences from the source

Abstractive text summarisation is a much newer development in the field of summarisation. The first abstractive solution that produced comparable results to extractive techniques only dates back 5 years to the paper written by Rush et al. (2015). It was also one of the first papers to use a neural network for the generation of summaries, and it paved the way for other models that are now considered state-of-the-art.

In recent years, abstractive text summarisation has received more attention due to the increase in the availability of modern day computing power. Moreover, good datasets have began to surface like *CNN/DailyMail* from Hermann et al. (2015) as well as the *DUC* and *TAC* datasets, all of which are all used throughout in this project (Section 3.6).

2.1 Extractive Summarisation

2.1.1 Simple Sentence Ranking

As mentioned previously, the first attempt at automatic text summarisation from Luhn (1958) is an extractive technique. The method includes stripping *stopwords* [1] from the source text in order to only preserve the important words. It then creates a collection of *top words* by selecting a subset of the words that appear more frequently in the document than others to represent the key information of the text. Using this collection of top words, sentences are then ranked depending on how important they are. Whereby the sentences that contain more *top words* than others are ranked higher in the list. Depending on the required length of the summary, the top *k* sentences from the ranked list of sentences are then iteratively included in the summary being generated. This technique, although simple, produces readable results. The paper highlights one of the main advantages of extractive text summarisation, that is, the summaries generated will always contain correct factual information and correct grammar since its contents are taken straight from the source document.

> "abstracts have a high degree of reliability, consistency, and stability, as they are the product of a statistical analysis of the author's own words."

(Luhn 1958)

A later work in the area of automatic text summarisation is the *Maximum Marginal Relevance* criterion hereby referred to as MMR. Developed by Carbonell and Goldstein (1998), MMR is an

[1] Words that are common in the English language that add no semantic information. Examples of stopwords would be "a", "the", "is" etc.

attempt to consider both the *relevance* of a section of text, as well as its *redundancy* regarding what has already been generated. MMR was presented as a technique for Information retrieval (at the document level) and text summarisation (at the sentence level). At the time the paper was written, IR (Information Retrieval) systems had already been able to produce a set of documents in an ordered fashion based on relevance. Therefore the focus of the paper was more on returning documents that contained more *novel* information compared to others. In essence, a document receives a high marginal relevance score if it is both; relevant and novel. That is, relevant to the query as well as dissimilar to the previous documents that have already been selected. The method for selecting the most desirable documents is to iteratively select the document (in the case of IR) or sentence (in the case of summarisation) with the *Maximum* Marginal Relevance score. MMR produces encouraging results and holds its own even now when compared to more complex extractive techniques.

2.1.2 Graph-based Methods

Since extractive summarisation is a task of ranking sentences in a source document, the performance of the algorithm is *bounded* by the accuracy of the ranking algorithm. At the turn of the century, graph-based ranking algorithm; PageRank from Brin and Page (1998) had proven successful at modeling a graph structure of the World Wide Web. This was the main inspiration behind the TextRank algorithm from Mihalcea and Tarau (2004), a graph-based ranking algorithm for the processing of textual information.

Graph-based ranking algorithms work by constructing a directed graph of nodes, where each edge in the graph represents a *vote* or *recommendation*. In essence, an edge from one node to another is the sender node's recommendation of the receiving node. The higher number of recommendations/votes per node, the higher perceived *importance* of the node. In the scope of text summarisation, the importance of a node denotes its 'centrality' i.e how closely it resembles the centroid[2] of the document. The importance of the node *casting* the vote is also taken into account for the importance calculation. If it is perceived to have *less* importance in regards to the overall graph, then the vote it casts will be weighted lower, compared to a vote cast from a *high* importance node. This means that sentences which are less important to the document cannot have artificially high scores due to being recommended by other less important sentences. In the case of TextRank, a graph is constructed where each node in the graph represents a sentence in the source text. The strength of the relationship between each node is determined by their similarity to one another (the word overlap between them). The graph constructed using these relationships, is highly connected as well weighted appropriately. Each node is given a score, which is calculated as the sum of all the weighted votes is has received. Each of the sentences are then sorted by their score and arranged into descending order. For a summarisation task, the highest scored sentences are used to form the summary for the source text. Similarly to MMR mentioned previously, the length of the summary is variable since the sentences are picked from an ordered list. The algorithm can pick more or less sentences based on the required output length.

LexRank (Erkan and Radev (2004)) is another graph-based ranking algorithm for text summarisation. Both TextRank and LexRank were published in 2004 and they are very similar. The difference of these two approaches lies in how the similarity (overlap) between sentences are computed. TextRank uses the number of words that overlap between the two sentences and normalizes the result by the sentence length, whereas LexRank converts the sentence into a TF-IDF vector (Robertson (2004)) (TF-IDF is further explained in Section 4). The algorithm then computes the cosine similarity between each of the vectors in the graph to produce a similarity score for each pair of sentences. Both LexRank and TextRank are aimed at single-document summarisation. However, in the case of multi-document summarisation, LexRank employs a

[2]In a mathematical view of a document, the *centroid* would denote the *key* information that the document represents.

heuristic post-processing step to recompute the scores for the sentences with regards to what has already been picked for the summary. By doing this, sentences that are too similar to the generated summary are given reduced scores to discourage the algorithm from selecting them.

Choosing sentences from the ranked list will generate a summary that has captured the essence behind a source text to some degree. However, it does not guarantee that the sentences picked will form a coherent and readable summary. To circumvent this issue, Nayeem and Chali (2017) builds on top of the TextRank algorithm and includes a greedy algorithm to reorder the sentence to aid with the summaries continuity. Essentially, from the collection of top-scored sentences from TextRank, a random one is chosen to be the first sentence in the summary. Following this, the cosine similarity is iteratively calculated for each of the top sentences with the sentence already selected for the summary. The most similar of the top-scored sentences is selected to be the next sentence in the summary. This process iteratively continues until all of the top sentences have been inserted into the summary. This process was added because sequential sentences in a document will typically be similar to one another as sentences usually elaborate on what has previously been said. Grouping the summary sentences by their similarity assists the algorithm in producing a summary that flows better, in comparison to one produced by the original TextRank algorithm.

2.2 Abstractive Summarisation

2.2.1 Neural Approaches

In contrast to extractive summarisation, abstractive summarisation chooses to generate *new* words in order to paraphrase the source text in a similar fashion to how a human would. An example of a successful attempt at abstractive summarisation is Rush et al. (2015). Rush et al were among the first set of researchers to implement a Recurrent Neural Network (RNN) with an encoder-decoder architecture [3] for abstractive text summarisation. They make use of an attention-based encoder inspired by the attentional encoder from Bahdanau et al. (2014). This is then used to generate a context vector [4] softly aligned (weighted) with attention. Attention refers to an extra process in which the model is allowed to learn which words in the source text it should pay *attention* to. When a model without an attention mechanism encodes a long sequence of text into a context vector, some important information can be lost or forgotten (Further discussed in section 4). By using attention, the model focuses on a subset of the input, to better remember these details in the decoding phase. For the decoder, Rush et al use a Neural Language Model (NLM) that had shown success in Machine Translation (MT). The decoder uses the context vector provided, to produce a probability distribution across the input vocabulary[5]. This probability distribution is then used to generate the next word in the summary. The attentional encoder from Bahdanau et al. (2014) produces consistently better results and as such, it has since been used in many summarisation research projects.

Nallapati et al. (2016) also make use of an attentional RNN encoder-decoder model based on Bahdanau et al. (2014). Nallapati et al adapt the Large Vocabulary Trick(LVT) from Jean et al. (2014) which restricts the decoder to use only the vocabulary present in the current batch of the source document being processed. This removes a lot of noise in the final word prediction phase, as the softmax layer (the part of the decoder which predicts which word to generate next) has a far smaller vocabulary to choose from. This speeds up the generation process and does not detriment the quality of the output. Nallapati et al also make use of a Feature-Rich encoder which encodes linguistic features such as named entity tags, parts-of-speech tags as well as TF-IDF statistics

[3] An encoder is a RNN that is used to encode the sequence of text into a vector representation. The decoder is another RNN which then takes the vector representation as input and produces the intended output from it

[4] the encoded representation given to the decoder

[5] Collection of words that appear across the input data

for words into additional look-up based embeddings. These additional linguistic features add an extra layer of depth to the representation of words and help the encoder better capture the essence of each word. These lookup-up based embedding matrices are used by the decoder to retain extra information from the text when generating the summary. This method has been used in traditional extractive papers such as Wong et al. (2008), but this research provides the first instance of this approach being used in an abstractive domain. Nallapati et al make use a Pointer-Generator (PG) network, enabling the model to act in both an extractive and abstractive way. The PG network facilitates the ability to copy words directly from the source text or to generate new words. This appears useful for information like named-entities or rare/unseen words where faithfulness to the original source is crucial. The model that Nallapati et al propose is a Hierarchical attention model. Whereby it models the attention at both *sentence* and *word* level, as opposed to previous attempts which only modeled at the sentence level. This allows the model to apply specified focus to sentences in the text as well as individual words within those sentences.

The Pointer-Generator (PG) network in Nallapati et al's paper strikes a balance between text generation and extraction. It ensures that factual information is repeated correctly in the summary and helps deal with rare or unseen words. A further extension of PG is the work of See et al. (2017). They demonstrated that a common problem with neural approaches to text summarisation is that factual information is conveyed incorrectly and information is repeated leading to a summary filled with redundancies. See et al use a PG network and a Coverage mechanism to retain factual information of the text and reduce repetition of information respectively. Dissimilar to the method presented by Nallapati et al, PG from See et al is able to use the pointing mechanism for any word, rather than being constrained to only out-of-vocabulary (OOV) words and named entities. By proposing the coverage mechanism in their paper, the authors' model is able to keep track of the summary that has already been partially produced and discourages the model from producing sentences that are similar to those that have already been generated.

See et al's Pointer Generator network is written with a single source document in mind, although, it is a concept that could be applied to the multi-document setting. The work of Lebanoff et al. (2018) provides an example of such an implementation. Lebanoff et al extend the PG network to be applicable for the multi-document setting by combining it with the MMR algorithm from Carbonell and Goldstein (1998). First, the network is adapted by concatenating all the source documents into one 'mega-doc'. MMR is then applied to this long source document to extract the most important sentences. The attention weights are modified so that more attention is placed on these sentences during summary generation. One of the main benefits of using this architecture is the potential to be trained on a SDS model thanks to the MMR technique, which truncates the long document into one of comparable length to a singular document. This is beneficial, as much larger and much more renowned datasets are available in the single document domain compared to multi-document summarisation. Single document datasets are also much simpler to produce, since only a fraction of the work has to be done to collect an article-summary pairing (One-to-One in SDS, Many-to-One in MDS). As a result, the maintenance of this model in the future through the introduction of new training sets or the increase in size of the current training set will be easier.

Extending SDS to MDS is an increasingly important task due to the abundance of SDS datasets available. Zhang et al. (2018) provide another example of such an extension. They extend the sinABS model presented by Tan et al. (2017) by implementing a *Document Encoder*, which takes a set of individually encoded documents as input, and encodes them into a *set* representation. This document encoder is added as an intermediate step between encoding and decoding. This architecture was centered around the idea that a decoder takes in a vector representation as input, and produces a summary based on it. Thus, if the document encoder produces a vector representing the key information of the documents, the decoder will be able to generate a summary for it in a similar fashion to SDS. Since there is no dependence on the order of the documents in the set, a simple concatenation of weighted document vectors can be used to

produce the final encoding. The weighting is dependent on the content of the document, and therefore is calculated based on its centrality regarding the document set, i.e how well the document represents the key information conveyed across the entire collection. This vector representation is then given to the decoder as its initial state. In the original sinABS research, as well as many other attentional decoders present in neural models, the attention is calculated based on the vector representation the decoder is given as its initial input. However, Zhang et al alter this method to calculate attention based not on the document set encoding, but on all of the original documents. The issue with this method is that the context vector passed to each phase of decoding is very large as it captures information from *all* the source documents. To combat this, Zhang et al selectively apply attention weights to the top k ranked sentences. These weights are computed using the PageRank algorithm from Brin and Page (1998), where k is a specified hyperparameter.

2.2.2 Semantic Concept Generalisation

Both Neural and extractive approaches to text summarisation have their drawbacks. Neural approaches, whilst being able to paraphrase and generalise to a good degree, can lead to losses in factual information during decoding time. One of the reasons behind this, is that information can be presented in multiple different ways throughout a document. For instance, dates can be written in various ways throughout a piece of text, i.e *Tuesday the 7th of January* can be represented as *January 7th*, *7/01/2020* etc. To combat this issue, a semantic concept generalisation approach is proposed by Kouris et al. (2019). Two methods are introduced, a *Named Entity* generalisation (NEG) method and a *Level driven* generalisation method (LEG).

The NEG method replaces named entities that appear in the source text with with *concept tags* [6]. Generalisation is useful for entities which do not appear often in the text, therefore, only entities that appear less times than a predefined threshold are generalised. Once a summary is produced using this generalised text, an algorithm is used to replace the concept tags with their original meaning.

The LEG method is similar, the only difference lying in its ability to generalise *any* word in the source text rather than being confined only to named entities. Similarly to NEG, the words are only generalised if they appear less times than a frequency threshold. The algorithm is provided with a *level* parameter that determines how generalised the word should be. For instance, 'Earl Grey' can be generalised to '_tea_', which would be level one, '_beverage_' for level two, '_liquid_' for level three, and so on. Once the generalised summary has been produced, the same post-processing method is used as in NEG. Both forms of generalisation improve the quality of a given summary, however NEG was shown to perform better overall (Kouris et al. (2019)).

Semantic concept generalisation is a valuable mechanism, as it retains factual information much better than other approaches. This is crucial for summarising source texts that rely on their factual content. This method is optimal for domains like news summarisation or sports, where faithfulness to the source is vital. However, for more forgiving summarisation tasks, this process can be omitted.

[6] Special tags that replace the entities in the text, they are wrapped in underscores '_' so that they can be differentiated from the rest of the words in the text

3 Analysis

3.1 Introduction

Automatic text summarisation is a computational task that dates back to the late 1950s. In a world with copious amounts of information, it is vital to have an effective and efficient way of condensing it into a digestible form. Not only is the manual creation of summaries a labour intensive job, but it is also prone to human error and misinterpretation. Thus, work on leveraging computing power to automatically perform this task is imperative.

Summarising text is a process that produces variable results. There is an inherent subjectivity to the summary written, as different people's summarisation of the same document will most likely produce diverse results, each varying in abstractiveness and generality. Moreover, in the automatic domain, the datasets used for training and testing summarisation techniques vary in how abstractive they are (See figure 3.1). The different models mentioned in Section 2, will also produce different forms of summaries due to the nature of their architectures, each ranging from single-document to multi-document, abstractive to extractive, structure based to semantic based and so on. Thus, the production of good quality, human readable summaries relies on an optimal combination of the models presented.

3.2 Multi-Document vs Single-Document

Typically, information to be summarised comes from a multitude of sources. For example, many text summarisation models are trained on news data. News articles almost always cover information that is also present in articles from other news agents, each varying in their writing style. Some of these sources are more subjective and opinionated, whilst other are more objective and state the facts. Due to these differences, not all articles concerning the same news will contain the same information. A good attribute for a summariser to have is the ability to summarise not just the specific document itself, but the general topic in which the document refers to. In other words, a summarisation system should not be confined to a specific writing style, it should rather be able to provide a more general and balanced summary regardless of the source document's author. This can only be done with Multi-Document summarisation since multiple sources are provided to the model. In single document summarisation, the model is restricted to use only the text present in the source article, meaning that it is constricted to the style of writing presented in the document when generating a summary. Multi-Document summarisation presents a much wider scope of information for the model to pull from, making for better informed and more balanced summaries. The risk with using a Multi-Document dataset is that it can lead to redundancy. Key information about a topic will be mentioned in almost all sources, and thus it is likely to be repeated multiple times in the generated summary if these redundancies are not filtered out.

Single-Document summarisation has been the focus in previous work because of its simplicity. However, in practise, sources are often Multi-Document, where each source presents new information. Thus, Multi-Document summarisation will be the focus of this experiment.

3.3 Abstractive vs Extractive

Abstractive and extractive summarisation are the two distinct methods that summarisation techniques bifurcates into. Both have produced good results overall. The difference, however, is the form of the summaries that are produced. Extractive summaries are much more disjointed than their abstractive counterparts. The result of extractive summarisation is a summary containing sentences selected from various places in the source text, thus leading to a summary that is grammatically correct, but jarring to read. Every word in an extractive summary exists in the source document and thus any human error from the source document will also be carried over to the summary. This also means that factual information is perfectly preserved. Dates, names and locations will all be correct in reference to the source document which ensures that no false information is conveyed in the summary. Abstractive summarisation can struggle with this, as often factual information can be either incorrectly relayed onto the summary or left out all-together. However, there are measures to prevent this. Using a pointer-generator and concept generalisation can help limit the amount of false information, both of which will be discussed more later in this section. The key benefit of abstractive summarisation is its ability to generate new words, paraphrase and generalise documents all on its own. This creates summaries that resemble human written summaries more closely and as a result are more readable.

Whether a summary was automatically generated or not should not be noticeable. Text summarisation use cases lie in many real world applications, as they offer a way to summarise documents without having to ask a human to do so. Ideally, there would be no difference between the human written summary and the computer-generated one since that is what the model is designed to replace. Abstractive summarisation is the only way to generate summaries that resemble human-written ones, but abstractive techniques have received less focus in the past due to their complexity. However, with frameworks like TensorFlow from Abadi et al. (2015), the process of creating neural abstractive models is now much more accessible.

While good results have emerged from the field of extractive summarisation, the summaries are only as good as the source documents they are provided with. The ceiling for abstractive techniques is much higher and there are more new ideas to explore. With this in mind, this experiment will focus on abstractive techniques.

3.4 Model

As demonstrated in the background section, many different models perform well in the domain of abstractive text summarisation. Over previous years of research, certain features of neural models have become standard to include in a neural implementation, for example an attentional encoder as well as a pointing mechanism. Since their introduction, they have been used in many different projects as they are robust methods that consistently add to the quality of summarisation independently of the domain. Since they perform well and are now common-place among research, it is important that they are included in the implementation of this current study.

Working on a model that has been extended from SDS to MDS appears sensible since it enables the use of large single document datasets for training. Considering this, the choice of models for the project is narrowed down to that of PG-MMR from Lebanoff et al. (2018) and Multi-Document sinABS from Zhang et al. (2018). Both are well performing and trained on Single-Document datasets. Zhang et als model is a more abstractive model than PG-MMR since it has no PG mechanism to copy words from the source text. However, this does not necessarily mean that it will produce higher quality summaries. PG-MMR performs better even though it is a hybrid of abstractive and extractive methods. This can be explained by the fact that PG-MMR performs the actions of an abstractive system (paraphrasing, generalising and reordering) but still retains a good grammatical structure due to its extractive capabilities. PG-MMR strikes a good balance

between abstractive and extractive approaches and thus, it produces summaries comprised of grammatically correct sentences and words that are tied together well using generated words. Although it is a heavily extractive method of summarisation, it still performs the actions that an abstractive system would. For this reason, PG-MMR will be the focus of this project.

3.5 Generalisation

The ability to generalise concepts of an idea is commonplace in human written summaries. It reflects an inherent understanding of the key ideas behind the source text and contributes to the overall quality of the summary since it helps to compress the important information. Although PG-MMR is highly extractive, when generating abstractive words, the model might chose not to relay words that are mentioned often in the source document, even if they are integral to the information presented. Moreover, the same words can also be represented in different ways throughout the text. For instance, if an article includes the name "William Shakespeare" it will refer to that person as "Shakespeare" later in the document. This can lead to an under-representation of the given entity in a summarisation task. Each name is considered as its own separate entity, when in reality, the two words should be considered as the exact same. This is common in most documents as humans are intelligent enough to make the connection between the two different names. However, this sort of information is lost in the case of a neural network unless told otherwise. As stated above, if the name is only mentioned twice in the entire document, then it is likely that this information will not be relayed to the summary during decoding time. Considering this, it is important to ensure that these entities *are* in-fact treated the same and thus, will be seen as more analogous to the centrality of the document.

This project will implement a method proposed by Kouris et al. (2019) to ensure that these important objects are reinforced in the document representation. Kouris et al present two different methods of performing this generalisation. The first of which is *Named Entity* generalisation which only generalises names, locations and organisations in the text. This method swaps named entities with a singular tag so that the encoder is more likely to include the entities in the summary. The concept tags in the summary are then replaced by their original named entity counterpart via a matching algorithm. The second form of generalisation is *level-driven* generalisation. This means that any word is generalised if it appears a number of times under a predefined threshold. For example, the word "aubergine" could be generalised by replacing it with the tag "_vegetable_" if it appeared in the document less times that the given threshold. Since other words like "potato", "onion" etc, would also be generalised to "_vegetable_", the decoder is encouraged to use the phrases including this tag as it would occur more often than the non-generalised words separately. The issue with this method is that normal words like these are rare in the source text simply because they are not integral to the ideas behind the document. Because of this, the method can *over-generalise* and include information in the summary that is not central to the main ideas and thus add nothing of value. Named entity generalisation has been shown to be more performant in creating good summaries. With this in mind, this project will implement Named Entity Generalisation as it is more conducive to high quality summaries.

3.6 Datasets

3.6.1 Datasets for Training

For this project, multiple datasets will be used. Since the PG-MMR model is an extension of the original SDS PG model, it will also require a SDS dataset that is large enough for training. There are two datasets available consisting of a large number of article-summary pairs, the Gigaword from Parker and Graff (2003) and the CNN/Daily Mail dataset (CNN/DM) Hermann et al. (2015). The original Gigaword dataset was later annotated with linguistic features by Napoles et al. (2012) and in most cases, it is used instead of the original Gigaword dataset. Throughout the rest of this study, "Gigaword" will refer to the annotated version of Gigaword.

The Gigaword dataset is very large with over 4.1 million documents. However, the articles in Gigaword are quite small with an average length of 31.4 words in the input article and 8.3 words in the corresponding summary (Sebastianruder (2020)). This means that the model trained on it will only be able to produce summaries of roughly a sentence long which is undesirable. Moreover, since most articles requiring summarisation are much longer, the summary would need to be at least a paragraph long to be able to capture all of the key concepts from the source document. Using Gigaword for summarisation would be redundant as the input articles are already short enough to the point where they themselves could be considered a summary of the topic they refer to. Because of this, the Gigaword dataset is much more suited to the domain of Headline Generation (generating headlines for news snippets) so it will not be used for this project.

The other SDS dataset used in research is the CNN/DM Hermann et al. (2015). CNN/DM is a large single document dataset with 287,000 article-summary pairs which is sufficient to train a model as complex as PG-MMR. The CNN/DM dataset is also what is used by both PG See et al. (2017) and PG-MMR Lebanoff et al. (2018) and thus it will also be used in this project for consistency.

3.6.2 Datasets for Testing

In order to test the model in this paper, a MDS dataset must be used. Two of the most popular MDS datasets are the DUC and TAC datasets. They are both comprised of roughly 50 document sets, where each set contains 10 source articles and 1 corresponding target summary (human-written). For both the DUC and the TAC datasets, the average lengths are roughly 4,650 words and 100 words for input articles and target summaries respectively Fabbri et al. (2019).

Table 3.1: *Examples of N-grams*

Source Text	U.S. prosecutors have asked for a 20-day extension to provide Germany with paperwork.
Uni-grams **(N=1)**	U.S prosecutors have asked for a 20-day extension to provide Germany with Paperwork
Bi-grams **(N=2)**	U.S prosecutors have asked for a 20-day extension to provide Germany with Paperwork ...
Tri-grams **(N=3)**	U.S prosecutors have asked for a 20-day extension to provide Germany with Paperwork ...
4-grams **(N=4)**	U.S prosecutors have asked for a 20-day extension to provide Germany with Paperwork ...

NB: Each colour represents an n-gram.

As different people will write summaries in different ways, the target summaries for the DUC and TAC datasets will vary in their degree of abstractiveness. To measure the abstractiveness of a

dataset we can count the amount of n-grams [1] that appear in the summary which do not appear in the source text. An example of n-grams is shown in 3.1 above.

Single words (uni-grams) appearing in both the summary and source text capture to what degree the vocabulary of the source text is retained in the summary. The more novel high-degree n-grams (tri-grams and 4-grams) the summary contains, the more abstractive that summary can be measured as. Table 5.2 illustrates that the DUC and TAC datasets are more abstractive. This can be helpful for evaluating an abstractive model since when it is evaluated against these two datasets, extractiveness in the generated summary will be penalised. This is one of the reasons that the DUC and TAC datasets have been popular in the field of abstractive summarisation.

% novel n-grams	Multi-News	DUC03+04	TAC11	CNNDM
uni-grams	17.76	27.74	16.65	19.50
bi-grams	57.10	72.87	61.18	56.88
tri-grams	75.71	90.61	83.34	74.41
4-grams	82.30	96.18	92.04	82.83

Table 3.2: *Percentage of novel n-grams in target summaries that do not appear in the source documents, The higher the number of novel n-grams, the more abstractive a target summary is. This table has been borrowed from Fabbri et al. (2019)*

The extractiveness of a dataset must also be taken into account. For this, three measures were presented by Grusky et al. (2018). Fabbri et al. (2019) extended these measures to the MDS domain by concatenating all of the source documents into a single mega-doc. The first measure is the *Coverage* of a summary, which is defined as the percentage of the words in the summary that are directly taken from the source document, i.e how much of the source document the summary has *covered*. The second measure is the *Density* of a summary. This is measured as the average length of the extracted sentence in the source document from which each word in the summary was taken from. The higher the density of a given summary the more extractive it is. This is because each extracted word covers a higher percentage of the original sentence the word was taken from. Figure 4.6 below shows the relationship of extractive density and extractive coverage across a multitude of datasets. The third and final measure of extractiveness presented by Grusky et al is the *Compression* of a given summary. This is a ratio of the length of the article to the length of the summary. This is not shown in a figure as it is biased towards SDS datasets since one article mapped to one summary has a much lower compression rate than a collection of articles mapped to one summary.

The DUC dataset is by far the most popular dataset for evaluating MDS. Thus, to reliably compare the model in this paper to other research, it will be used for testing. Specifically, the DUC 2003 and DUC 2004 will be used, as well as the TAC datasets from 2008-2010. The DUC and TAC datasets have been the field standard for testing MDS systems for many years as they were considered the only reputable MDS datasets available. However, recently a new MDS dataset was released called Multi-News Fabbri et al. (2019). It is one of the first attempts at collecting a large MDS dataset. Whilst the dataset is relatively small in comparison to the available SDS datasets, it is sufficient for the training of a simple MDS neural model. Since PG-MMR is trained on a SDS dataset, a MDS dataset is only required for testing purposes. In which case, the size of Mutli-News is excessive. Because of this, the Multi-News dataset will not be used in this project.

[1] sub-sequences of sentences that are treated as one word

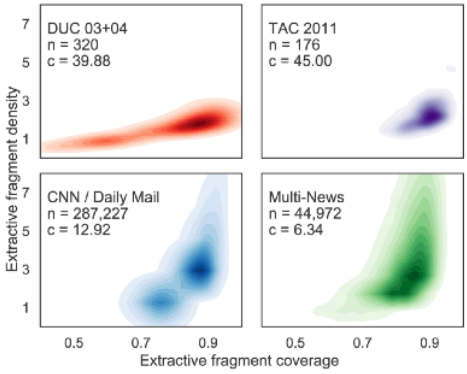

Figure 3.1: *the y-axis represents variance in the average length of extracted subsequences that appear in both the source text and the summary. The x-axis represents variance in the average length of the sequence of words in the source text that a extracted word in the summary it taken from. n represents the number of documents in the collection and c represents the average coverage of the dataset. This figure has been borrowed from Fabbri et al. (2019)*

3.7 Software Environment

To train a neural network, a lot of computing power is required, especially for the many matrix multiplications performed during training and testing of such a network. A GPU (Graphical Processing Unit) will be beneficial due to its high level of resources and high bandwidth to memory for performing complex calculations on large datasets.

Platforms like Google Colab are available on the web, which can connect to a hosted runtime, ran on a single Nvidia K80 GPU for up to 12 hours. This GPU offers enough resources to train the model. However, it soft disconnects [2] after an hour and a half and hard disconnects [3] after twelve hours. This could lead to issues when training a complex model such as PG-MMR as it takes a lot of time to run. Checkpoints could be used to save and reload the model progress at specific time intervals. However, the only form of storage accessible on Google Colab is Google Drive which requires an extra process of mounting. Google Drive also has a limited amount of storage per user and a shortage of this will cause the program to fail. The similar platform Kaggle also has GPU access, however it shares the storage issues mentioned previously.

The alternative to this is to use the University of Glasgow's High Performance Computing Cluster (HPCC). The HPCC consists of eight Nvidia GTX 1080s which can be used for running code. Moreover the HPCC is equipped with 1,550 virtual CPUs, this is enough to train the PG-MMR model. Storage is also less of an issue as the code is run locally on the cluster. This means that it can access any data stored locally on the cluster as well. There is also no storage limit per user so the potential for the program failing due to a lack of storage is greatly reduced. The downside of using the HPCC is that additional scripts have to be written in order to run code and request resources. Moreover, code will not be run immediately but rather added to a queue and be run after an amount of time depending on the level of resources requested. The more resources requested, the longer the job will stay in the queue before being run. This means that buggy code can be submitted and queued, but it will not be noticed until the code is run. This can slow down development drastically if requesting a large amount of resources with code that contains bugs.

[2] stops running but retains progress
[3] disconnects and does not retain progress

Assuming the codebase is written well and contains minimal bugs, the HPCC is the best choice as it allows for the most flexibility. Training PG-MMR on the CNN/DM dataset is far too large a task to be finished within twelve hours (On Google Colab/Kaggle) and as such, this project will use the University of Glasgow's HPCC.

3.8 Approach

To summarise, this project will focus on Abstractive Multi-document text summarisation using the PG-MMR model trained on a generalised version of the CNN/DM dataset and tested on the generalised DUC and TAC MDS datasets. The University of Glasgow's HPCC will be used to train this model due to its high performance. Testing will take place on any machine given its low resource requirements.

In order to successfully carry out this experiment there are several steps:

1. **Baselines** - Get results from PG and PG-MMR trained on the CNN/DM dataset and tested on the DUC and TAC datasets
2. **Semantic Concept Generalisation** - Implement generalisation and combine it with PG-MMR
3. **Model Analysis** - Analyse the effectiveness of applying generalisation in regards to the baseline models.
4. **Model comparison** - How well does PG-MMR-GEN models perform compared to other research in the field of MDS

4 Methodology

The aim of this study is to successfully generate a summary from a collection of documents, by exploiting not only PG-MMR, but also semantic concept generalisation. This section will go into detail about how PG-MMR works as it is the base of this project, it will then discuss the methodology of semantic concept generalisation.

4.1 Architecture

The proposed encoder-decoder architecture for this project is shown below in Figure 4.1. It is comprised of a Pre-processor that generalises the input documents - Section 4.3.2, the PG-MMR neural model - Section 4.2.4 and a Post-processing unit to transform the generalised summary; described in Section 4.3.3.

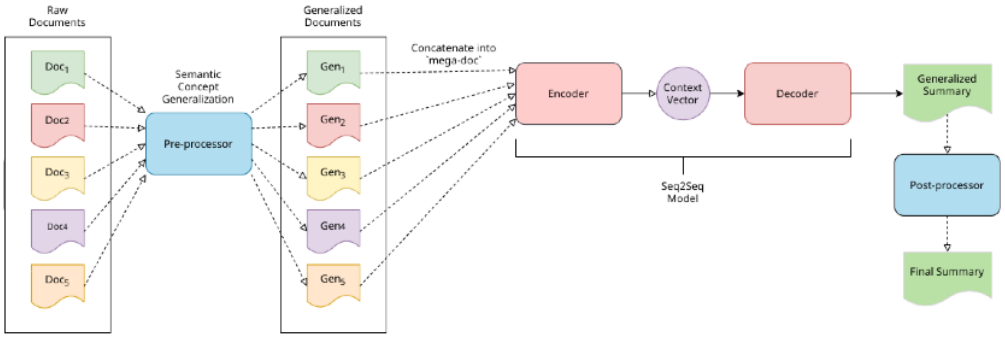

Figure 4.1: *Rnn Encoder-Decoder Architecture with a Pre-processor and Post-processor for Named Entity Generalisation*

4.2 Sequence to Sequence model

For a text summarisation task, a Sequence to sequence model (encoder-decoder) needs to be used in order to generate words depending on what has already been generated. This sequence to sequence architecture can be represented using Recurrent Neural Networks (RNN). Dissimilar to the conventional feed-forward network which simply passes data forward, RNNs also have the ability to remember previous inputs as each state of an RNN cell when passed forward to the next iteration. As a result, any decision made at time t is affected by the decision made at time $t - 1$. Figure 4.2 shows the architecture of an RNN.

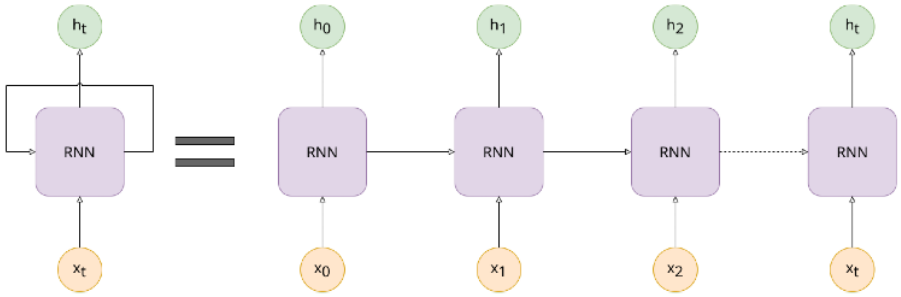

Figure 4.2: *The RNN is recurring as shown on the left-hand side, however we can unfold it to so show each individual RNN operation, x_t is the current input, h_t is the hidden state at timestep t*

4.2.1 Vanishing Gradient Problem

The appeal of RNN cells as shown in figure 4.2 is their ability to work well for sequential tasks like summarisation or machine translation. However, they can struggle to remember information from the past as the sequence of operations grows in length. This is a major problem when considering that our input documents are concatenated into 'mega-docs', which are very long by nature but still require information throughout to be represented. This issue is due to an effect named *the vanishing gradient problem*. The vanishing gradient problem occurs when the output at a specific time is dependent on input that was encoded many timesteps before. This large gap in the sequence means that the information captured much earlier in the sequence is harder to remember since the information in-between is essentially *noise* Olah (2015). Alternatively, this can be viewed as an issue of under-representation. That is to say, the information at timestep $t - N$ (where N is the length of the sequence bounded by the two time intervals) embodies *less* of the RNNs hidden state at time t. This view is represented in Figure 4.3

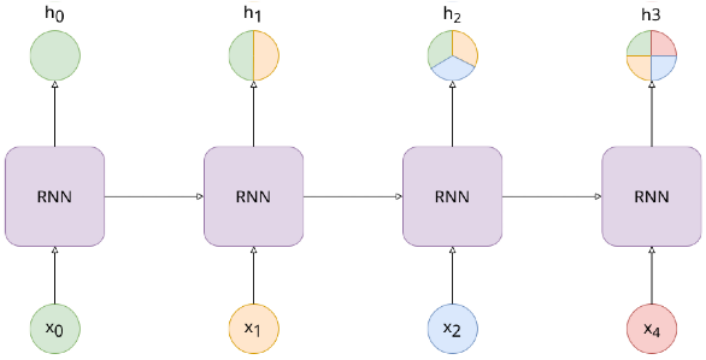

Figure 4.3: *After each timestep t the hidden state captures more information, thus each piece of information is represented less and less as each timestep passes*

To combat the issue of the vanishing gradient problem, Long Short Term Memory (LSTM) units were introduced, adding gates inside the RNN cells to control the information flow. LSTMs decide which information to *forget* and which information to *remember*. This decision is made by passing the input data through layers of gates, resulting in a *score* normalized between the range *0* and *1*. If the score is closer to *1* it is remembered, otherwise it is forgotten. This mechanism allows LSTMs to perform much better when encoding long sequences of data. In the PG-MMR implementation, used as a base for this paper, a bidirectional LSTM was used for the neural model. This is an altered LSTM which encodes the data backwards and forwards and then concatenates

the hidden states to produce the context vector. This enables the LSTM to train faster as well as better capture the context of the input data since information about the past *and* the future can be captured. The vanishing gradient problem is also part of the reason that *attention* was developed. By applying attention weights to an input, importance can be placed on certain sections and thus be reinforced in order to be remembered at a later stage.

4.2.2 Pointer Generator Model

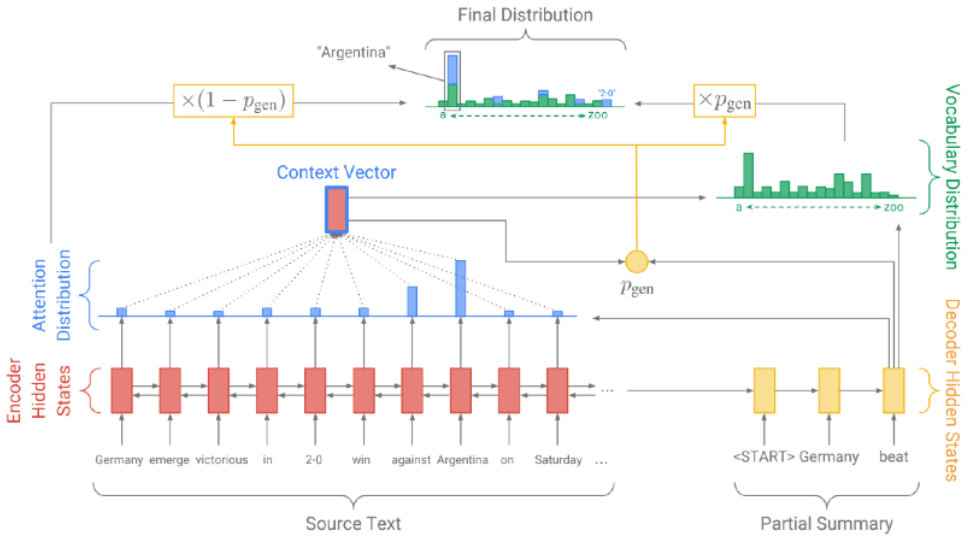

Figure 4.4: *Architecture of the Pointer Generator model. At each decoder timestep, the probability of generation is calculated. The vocabulary distribution on the right side and the attention distribution on the left side are summed to produce the final distribution. This figure is borrowed from See et al. (2017)*

$$e_i^t = v^T tanh(w_h h_i + w_s S_t + bias_{attn}) \tag{4.1}$$

$$\alpha^t = softmax(e^t) \tag{4.2}$$

In equation 4.1 above, the parameters; v, w_h, w_s and $bias_{attn}$ are parameters to be learned by the model. As shown in equation 4.2, the attention is calculated through the use of a softmax activation layer, in compliance with Bahdanau et al. (2014). This softmax function *squeezes* each input value between a range of real numbers bounded by 0 and 1. The attention distribution calculation provides a distribution across the words in the text. It is used to produce the context vector which is comprised of a weighted sum of the encoder hidden states. The calculation of the context vector is shown below in equation 4.3.

$$cv = \sum_i \alpha_i^t h_i \tag{4.3}$$

This context vector is a representation of what has already been encoded up until the current timestep. It is given to the decoder state S_t and used to produce the distribution of the source vocabulary P_{vocab} [1]. The vocabulary distribution is calculated as follows:

[1] This is a probability distribution denoting which word is most likely to be generated next.

$$P_{vocab} = softmax(V`(V[S_t, cv] + bias) + bias`) \tag{4.4}$$

Similarly to the calculation of the attention distribution, some of the parameters in this equation are learned, namely V, $V`$, $bias$, $bias`$. As said previously, this vocabulary distribution is used to calculate the probability of each word coming next in the generated summary, where the probability of a word coming next is defined as:

$$P(word) = P_{vocab}(word) \tag{4.5}$$

It should also be noted that during training the loss is calculated as the negative log likelihood of the probability of generating the target word from the reference summary (denoted as w_t^*). The loss for the entire sequence is calculated as a simple average of the loss calculated for each individual word. The total loss for the sequence is defined in equation 4.6.

$$loss = \frac{1}{T}\sum_{t=0}^{T} -log(P(w_t^*)) \tag{4.6}$$

Described so far is See et al's baseline model. To enable the pointing functionality, an additional probability must be used to determine whether the model should copy from the source text or generate a new word. This probability is denoted as p_{gen} and is shown in equation 4.7

$$p_{gen} = \sigma(w_h^T cv + w_s^T S_t + w_x^T x_t + bias_{pointer}) \tag{4.7}$$

In this equation, the weightings (w_h, w_s, w_x) and the scalar value ($bias_{poiner}$) are learnable, similar to previous equations. The sigmoid function σ, is similar to the softmax function mentioned previously. It too, *squeezes* data between a range of real numbers bounded by 0 and 1. The sigmoid function is characterized by its *s-shaped* curve and is shown Figure 4.5, the equation for the sigmoid curve is denoted in equation 4.8 (Weisstein (2002)).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.8}$$

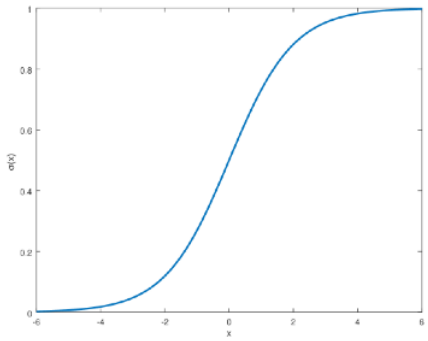

Figure 4.5: *The Sigmoid Curve. Borrowed from* Hvidberrrg@GitHub *(2020)*

Once the P_{gen} at timestep t has been calculated, it is used as a *switch* which chooses between generating a new word or copying one from the source text through the use of the previously mentioned attention distribution a^t.

Using the p_{gen} value, the *extended* vocabulary for each document can be computed. This is the formed from the amalgamation of all the words that appear in the source document as well as the entire vocabulary so far in encoding. This new probability distribution is calculated using equation 4.9.

$$P(w) = p_{gen} * P_{vocab}(w) + (1 - p_{gen}) * \sum_{w_i}^{w_i = w} \alpha_{w_i}^t \tag{4.9}$$

4.2.3 Coverage Mechanism from PG

See et al. (2017) also introduce a coverage mechanism to keep track of what has already been generated as part of the summary. It is adapted from the coverage model of Tu et al. (2016) and works by maintaining another vector known as the *coverage vector*. The coverage vector is defined as the sum of each attention distribution calculated at each decoder timestep. This vector is defined in equation 4.3

$$coverage^t = \sum_{t`=0}^{t-1} \alpha^{t`} \tag{4.10}$$

This coverage vector can be viewed as a distribution across the source texts vocabulary, which is dependant on how much *attention* each word has received while generating the summary thus far. Initially this vector is equal to 0 since at timestep 0, no words have been generated yet. Thus the coverage is 0.

This coverage vector is supplied as an extra parameter to the initial attention mechanism changing 4.1 to 4.11 below.

$$e_i^t = v^T tanh(w_h h_i + w_s S_t + w_c C_t + bias_{attn}) \tag{4.11}$$

w_c is also a learnable parameter. The addition of this parameter helps to make sure that previous decisions the decoder has made are taken into account when calculating attention. Hence, encouraging the decoder to not repeat itself during the generation of the summary.

Considering the addition of said coverage vector, the loss function is altered in order to penalise repetition. The loss is altered by adding *coverage loss* defined in equation 4.12 below. This coverage loss is weighted by a defined hyperparameter λ when calculating the overall loss. This altered loss function is defined below in 4.13

$$cov\_loss_t = \sum_i min(\alpha_i^t, C_i^t) \tag{4.12}$$

$$loss = \frac{1}{T} \sum_{t=0}^{T} (-logP(w_t^* + \lambda cov\_loss_t)) \tag{4.13}$$

4.2.4 PG-MMR

PG-MMR extends the PG model to the MDS domain, by introducing the MMR algorithm into the attention mechanism.

Maximum Marginal Relevance (MMR)

The MMR algorithm attempts to reduce redundancy whilst retaining relevance when reordering sentences. In the case of extractive text summarisation, the algorithm extracts the most *valuable* sentence from the source text and includes it in the generated summary until a length threshold is reached. The value of a sentence is determined by calculating its importance and penalising its redundancy. For sentence importance, a similarity measure is used, as typically important sentences resemble the *centroid* of the text. That is to say, the more central a sentence is regarding the context of the documents semantics, the more important information it includes. The importance calculation for a sentence is shown in equation 4.14 below.

$$MMR \stackrel{def}{=} \underset{s_i \in D}{argmax}[\lambda Sim_1(S_i, D) - (1-\lambda) \underset{\forall Sum_j \in Sum}{maxSim_2}(S_i, Sum_j)], \tag{4.14}$$

where D is the source document, Sum is the partially generated summary and λ is a balancing agent. Its value can be varied to prioritise either sentence novelty or sentence importance.

MMR in PG-MMR

In the context of the PG-MMR framework, MMR is used to iteratively select the top k sentences from the source text. The k sentences picked are then used by the PG model to generate a sentence for the summary. Once a sentence is generated in the summary, the MMR scores for each sentence are then updated. Now that the summary is not empty, the source sentences will be scored on importance, and more crucially on their current redundance with respect to the partially generated summary. That is to say, the source sentences that are more similar to what has already been generated will be penalized and thus receive a lower MMR score.

For the PG model to only make use of the selected sentences, the attention weights for each word belonging to sentences *not* picked must be dynamically be altered. The attention weights for the words in sentences not picked must be set to 0 so that the model does not focus on them at all. This is shown in equation 4.15.

$$a_i^{t\,new} = \begin{cases} \alpha_i^t MMR(S_k), & \text{if } i \in \{S_k\}_{k=1}^K \\ 0, & \text{otherwise} \end{cases}, \tag{4.15}$$

Where $i \in \{S_k\}_{k=1}^K$ is true if word i belongs to one of the top k sentences.

If the word does not belong to one of the top k sentences then in effect it will be *muted*. If it does then its attention will be weighted against its importance, thereby *re-normalizing* the attention weights to enable the PG model to further focus on important pieces of text during the next timestep.

Following Lebanoff et al. (2018), the diagram showcasing the PG-MMR architecture is illustrated below.

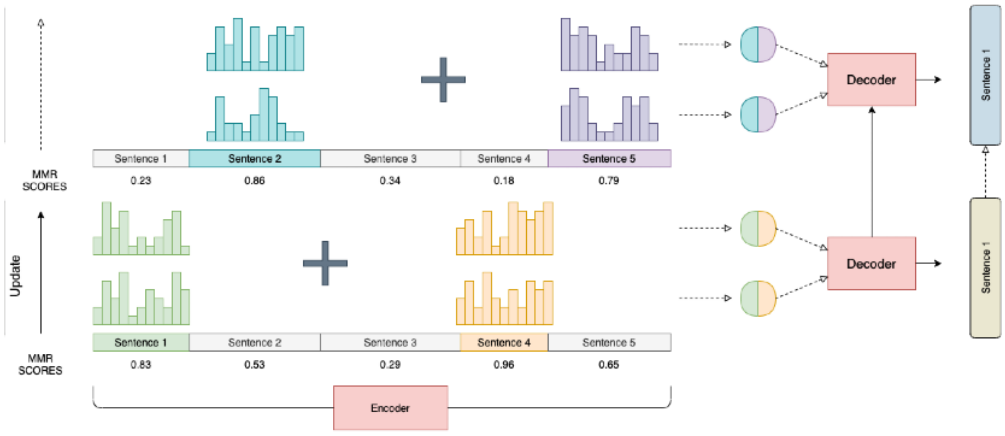

Figure 4.6: *altered for clarity from Lebanoff et al. (2018)*

Sentence importance

To compute the MMR score for a given sentence, an importance measure must be defined. In this project, Term Frequency - Inverse Dense Frequency (TF-IDF) is used. TF-IDF is a way to compute weightings for words depending on their importance regarding a corpus or document of words. The algorithm is processed in two separate parts, the first being Term Frequencey *(TF)* and the Inverse Dense Frequency *(IDF)*.

TF is a measure of how often a word is used in a document. This measure alone can be biased towards longer documents since there is a higher chance that the words will appear more often. To combat this issue, the TF score is divided by the document length to normalize it.

$$TF = \frac{\text{number of occurences of word w}}{\text{number of words in the document}} \tag{4.16}$$

IDF is a measure of how *important* a word is in regards to a collection of documents. In detail, it is the number of given documents divided by the number of documents that contain the given word.

$$IDF = \frac{\text{number of documents in the collection}}{\text{number of documents in the collection that contain the word w}} \tag{4.17}$$

Combining these two metrics, forms the TF-IDF algorithm, shown in equation 4.18. Manning et al. (2008)

$$TF - IDF = TF * IDF \tag{4.18}$$

To help TF-IDF form a true understanding of the sentence, pre-processing must be performed in order to remove *noise* from the sentence. The first of which is the process of stemming. Throughout a document, different forms of the same word will be used. From Manning et al, "Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational

affixes.". In other words, stemming is the process of splicing semantically identical words to their *root* form, i.e *{organising, organises}* ⟶ *organise*. This lets the TF-IDF algorithm treat each of these different words as the same, even though their representations are different in the text.

The second process is the removal of stopwords. Stopwords are common words in the English language such as "and", "is", "a". If not removed, these words will be used in the calculation of TF-IDF even though they add nothing to the meaning of a piece of text and therefore will *dilute* the word weightings.

TF-IDF computes scores for each word in a document. To extend this to a sentence-level weighting, the average of each individual score is taken.

$$\overset{sentence}{TF-IDF} = \frac{1}{N}\sum_{n=0}^{N} \overset{word}{TF-IDF}(w_n) \tag{4.19}$$

To be able to calculate the importance of sentences within a document, the document itself must also be represented as TF-IDF score. This is done similarly to above, but instead of getting the average TF-IDF score for each word, it is altered to compute the average of the *sentence* level TF-IDF scores. Then, the importance of each sentence can be calculated as how close it is to the document TF-IDF score. Wherein the score that is mathematically closest to the the document level TF-IDF is deemed the most important sentence.

Sentence Redundancy

Sentence redundancy also has to be calculated as part of the MMR algorithm. To do this, the ROUGE-L metric Lin (2004) is used (Further detailed in Section 6.1.2). ROUGE-L measures the longest common sub-sequence (LCS)[2] between a sentence in the source text and the partial summary generated thus far by the model. The length of the LCS is then divided by the length of the source text. From this, it can been understood that a high ROUGE-L score would imply a large amount of content overlap between the source text and the partially generated summary. Since the MMR score is offset by this metric, a high ROUGE-L score will in turn, produce a lower MMR score, meaning that the sentence is less likely to be used by the PG model.

Considering all the separate parts of PG-MMR mentioned previously, the formal algorithm for the process of text summarisation using the PG-MMR model is shown in algorithm 1.

4.3 Named Entity Generalisation

In text generalisation, text is often referred to as a *concept*. Text on its own is not generalisable since the word *text* is generic and does not imply any inherent meaning attached to the word. For the text to be generalisable, it needs to exist as a concept, that is, a piece of text in which the semantics are known. If the meaning behind a piece of text is unknown, then it cannot be generalised since there is no way to correctly chose another word that will subsume the meaning of the given text. Similarly if the text *can* be conceptualized, but there is no other concept that can subsume its semantics, then the text is still non-generalisable.

[2]This can also be viewed as the highest order N-gram from the source text that also exists within the partially generated summary. See Table 3.1.

Data: *SDS Data*, the SDS data that will be used to train the original PG model.
$\{S_i\}$, the collection of MDS sentences that will be used to generate a summary
λ, The balancing agent that will be used to compute the MMR score for a sentence.
L_{max}, maximum length of summary.
Result: *summary*, the final generated summary.
begin
    Train PG model on *SDS* dataset
    $I(S_i) \longleftarrow TFIDF(S_i)$
    $R(S_i) \longleftarrow 0$ `// nothing has been generated yet, so redundancy is 0`
    $MMR(S_i) \longleftarrow \lambda I(S_i) - (1-\lambda)R(S_i)$
    $summary \longleftarrow \{\}$
    $t \longleftarrow$ index of summary words
    while $t < L_{max}$ **do**
        $top_k = \{S_k\}_{k=1}^{K}$ with highest MMR scores
        Compute new α_i^t based on top_k
        Run PG decoder for one step to get $\{w_t\}$
        $Summary \longleftarrow summary \cup w_t$ `// generate a word in the summary`
        if $\{w_t\} \equiv$ "." **then**
            $R(S_i) \longleftarrow maxSim(S_i, summary), \forall i$
            $MMR(S_i) \longleftarrow \lambda I(S_i) - (1-\lambda)R(S_i), \forall i$
        end
    end
end

Algorithm 1: PG-MMR: Pointer Generator network with Maximum Marginal Relevance, borrowed from Lebanoff et al. (2018)

Moreover, for a piece text to be generalisable, it must have an extractable *taxonomy of concepts* from which the semantic structure of the text can be graphed. In essence, the taxonomy of concepts contains concepts and their associated *hypernyms* [3]. Given this graph, the *taxonomy path of concepts* can be extracted, which structurally orders the concepts. An example of a taxonomy of concepts is shown below in Figure 4.7. In the example, the word *Beef* could be generalised by the word *Meat* or *Food*. The taxonomy path of concepts for *Beef* would be defined as $Path_{c_5} = \{C_5, C_3, C_1, C_0\}$ and as such the taxonomy depth of concepts has a length of 3 (Path size excluding the root node). Full definitions of all the terms defined previously can be found in Section 4.3.1.

4.3.1 Definitions

Hypernym, A hypernym of any given concept C_o, is another concept C_h, for which the semantics of C_h subsume that of the associated concept C_o

Taxonomy of Concepts, A taxonomy of concepts is a graphed relationship of each concept from a given text along with their associated hypernyms. For each concept in the Taxonomy of Concepts, a concept at *Level N* can be generalised a the concept associated with it in *Level N-d* where d is a predefined depth.

Taxonomy Path of Concepts, A taxonomy path of concepts $Path_{C_n}$ for a given concept C_n is an ordered set of Concepts in the form $\{C_n, ..., C_0\}$ where C_0 is the root of the taxonomy of concepts. Any concept bounded by C_n and C_0 can be used to generalise C_n.

Taxonomy Depth of concepts, The taxonomy depth of concepts is defined as the depth of a concept within its taxonomy of concepts. For a given concept C the taxonomy depth of concepts is equal to the length of $Path_c$ - *1*

Generalisable Concept, A generalisable concept is a concept which has at least one hypernym within its taxonomy of concepts.

[3] Words which subsume the given concept. Alterntavely they can be though of as synonyms.

Generalisable Text, Text is generalisable when it contains one or more *Generalisable Concepts*.

Concept Tags, A concept tag is a tag that replaces a generalisable concept in the text. this is denoted by any word encased in underscores '_'. In compliance with Figure 4.7 above, The concept tag of *'Brie'* would be *'_Cheese_'* since 'Cheese' is the hypernym of 'Brie'.

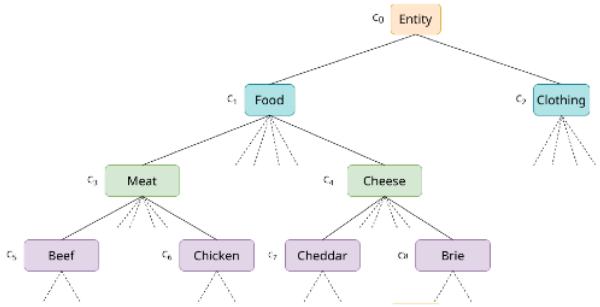

Figure 4.7: *Taxonomy of Concepts. Each Colour represents a different Taxonomy Depth on concept*

4.3.2 Pre-processing

The *Named Entity Generalisation* (NEG) algorithm presented by Kouris et al. (2019) is used in this project. NEG enables the generalisation of named entities in a given source text, ensuring that the neural network relays this information in the summary, since the tag that generalises all the entities will appear more times than any individual entity alone.

To implement the NEG task, there are two independent steps that need to be taken; pre-processing and post-processing. The pre-processing step performs the initial *generalisation* on the source text. It is the first stage in the overall architecture of this project shown in 4.1. The method is applied to each document in the input data collection; CNN/DM and DUC/TAC for training and testing respectively. The algorithm to perform the pre-processing loops over each word in the document, checks if it is a named entity or if there exists a named entity in its taxonomy path of concepts. It also checks to see whether the word appears less times than the predefined threshold. If the criterion is met, the word is considered generalisable. Thus, the word is replaced by a concept tag i.e *_person_, _location_ or _organisation_*, depending on the entity type. Algorithm 2 outlines the steps for pre-processing.

For the NEG task, pre-processing has to be performed on the training data as well as the test data. Generalising the train data *can* be omitted. However, generalising the test data and using it with a model that has not been trained on a generalised dataset will produce summaries that are nonsensical. This is due to the model not knowing how to deal with the concept tags. The concept tags are wrapped in underscores '_' to differentiate them from normal words in the text. As a result the concept tags will most likely not appear across the data's vocabulary. If a model is not trained on a generalised dataset, it will have never seen these concept tags before and thus, it will have a difficult time dealing with them. The term used to describe words that the model has not seen before is *Out of Vocabulary* (OOV).

Once the dataset is pre-processed, it can then be used in the model in the same fashion as the original non-generalised dataset. The text will be encoded and decoded into a summary as normal. However, the summary produced, if processed correctly, will contain concept tags. An example of such a summary is shown below in table 4.1

Data: $text$, the text to be generalised.
θ, the defined frequency threshold.
C, The list of Concepts, each entry in the list is a tuple containing (i) the concept, (ii) the taxonomy path for the given concept and (iii) the frequency of the given concept.
$NamedEntities$, a dictionary of concept tags in the form $\{namedEntity : replacementConceptTag\}$.
Result: $genText$, the text in its generalised form.
begin
    $genText \longleftarrow text$
    forall $(Concept_i, Path_i, Freq_i) \in C$ **do**
        if $Freq_i < \theta$ & $\exists C \in Path_i : C \in NamedEntities$ **then**
            $genText \longleftarrow$ replace $Concept_i$ with C
        end
    end
    return gen
end

Algorithm 2: NEG: Pre-processing Matching Algorithm, borrowed from Kouris et al. (2019)

Table 4.1: *Example Summaries from a normal model, and a model given a generalised dataset. text coloured in Green denotes the generalisable concepts. Pink denotes the concept tags that have replaced the generalisable concepts.*

Source Text	U.S. prosecutors have asked for a 20-day extension to provide Germany with paperwork. The paperwork is necessary to extradite a top lieutenant of Saudi terrorist suspect Osama bin Laden. This was reported on Saturday.
Normal Summary	prosecutors have asked for a 20-day extension to provide Germany with paperwork necessary to extradite terrorist suspect
Generalised Summary	_location_ prosecutors have asked for a 20-day extension to provide _location_ with paperwork necessary to extradite terrorist suspect _person_ _person_ _person_

4.3.3 Post-Processing

Assuming the model was given a generalised dataset, the summary produced will be in a generalised form. The post-processing stage takes the generalised summary produced by the model as input and produces the final summary as output. To do so, it makes use of an algorithm to find the concept tags in the summary and replace them with their original meaning.

To do this, a collection of replacement candidates must be produced. That is, a list denoting what piece of the source text each concept tag could have generalised, ranked by the maximum likelihood. From this collection, the most likely candidate can be chosen, and replace the concept tag in the summary.

Algorithm 3 shows the post-processing algorithm in an high-level manner. First, it loops over each word in the generalised summary and checks whether it is a concept tag. If it is, it is then appended to a list of concept tags to keep track of it. With this concept tag in mind, the algorithm loops over each word in the original (non-generalised) source text associated with the summary. Each word in the original source text is then checked to determine whether it is a named entity or if its taxonomy path of concepts contains the concept tag found in the summary.

If it does, then the word is a *candidate replacement* for this concept tag. Following this, the similarity between the context of the concept tag, as well as the generalisable concept in the source text is calculated[4]. A tuple consisting of (*i*) the concept tag, (*ii*) the original word, and (*iii*) the similarity score is then appended to a list of candidates. This process is repeated for each concept tag in the generalised summary and once finished, each concept tag in the summary is replaced with the word from the original text with the highest similarity score.

Data: *genSum*, The generalised summary which needs to be post-processed.
text, The original text from which the summary was generated.
T, The taxonomy of concepts.
Result: *summary*, The final summary with concept tags replaced by their counterparts from the source text.

```
begin
    candidates ⟵ {}
    conceptTags ⟵ {}
    summary ⟵ genSum
    forall token_sum ∈ genSum do
        if token_sum is generalised then
            append token to conceptTags
            forall token_orig ∈ text do
                if ∃c ∈ P_token_orig : token_sum = c then
                    similarity ⟵ calculateSimilarity(token_sum, token_orig)
                    append (token_sum, token_orig, similarity) to candidates
                end
            end
        end
    end
    sort candidates in descending order of similarity
    forall (token_sum, token_orig, similarity) ∈ candidates do
        summary ⟵ replace token_sum with token_orig
        conceptTags ⟵ remove token_sum from conceptTags
    end
end
```

Algorithm 3: NEG: Post-processing Matching Algorithm, borrowed from Kouris et al. (2019)

Once the post-processing stage is finished, the summary should contain no concept tags. Thus it is a finalised summary. The summary produced will be of a higher quality as it will ensure that no important information is omitted.

4.3.4 Subsequence Similarity

In regards to how the similarity between concept tags in the summary and the text they generalise is calculated, a sliding window mechanism [5]is used. Moreover, the similarity score that a candidate is given, is based on how *similar* the words surrounding it are to each sliding window containing a generalisable concept in the source text. Once two subsequences have been selected from the summary and the source text, there are many ways in which the similarity between them can be calculated. In this case, GloVe word embeddings Pennington et al. (2014) were chosen, as they not only show the similarity between words syntactically, but they can also measure the similarity between the semantics of the words.

[4]In this case, *context* is the sub-sequence of words (of a defined length) that the word being analysed belongs to.

[5]A process to consider an entire sub-sequence of words at once. For a window of size N at word index i, the sliding window would consist of a range of words from index $i - \frac{N}{2}$ to $i + \frac{N}{2}$. The next sliding window would consist of words at index $i + 1 - \frac{N}{2}$ to $i + 1 + \frac{N}{2}$ and so on.

5 Implementation

This project implements a system which creates abstractive summaries of text given a collection of source documents. A python impementation provided by Lebanoff et al. (2018) was used as the base for the project and it has been extended to use Named Entity Generalisation.

5.1 PG-MMR Implementation

As mentioned above, publicly available code for PG-MMR was available on GitHub (Ucfnlp (2019)) and was used as the base for this project. The code features a fully functional PG-MMR model written in python using the TensorFlow library developed by Abadi et al. (2015). The PG-MMR model was chosen as the base for this project due to its high-performant summarisation capabilities. Since Lebanoff et al's code is publicly available, using it as a base for this project was the most sensible option, rather than re-implementing it from scratch. The code is built on top of the implementation publicly provided by See et al. (2017), and as such, PG-MMR is built to use a trained version of See et al's model.

See et al provide a pre-trained model on their GitHub repository (Abisee (2019)). This model provided is not exactly the same trained model as used in their paper. As a result, it performs slightly worse than what is presented in the paper. The difference between the performance of the pre-trained model *versus* the model used in their paper is small. Moreover, re-training this model would take a long time and produce comparable results. For this reason, the pre-trained model provided by See et al was used.

5.1.1 Pre-processing the input data

To efficiently read large amounts of input data for such a task, it can be helpful to serialize this data and store it in small individual binary files that can be read linearly. This is done in the PG-MMR implementation through the use of TensorFlow Examples ('tf.Example'). At its core, Tensorflow Examples are a mapping in the form *{"string": tf.feature}*. The string label is used when the file is being read to differentiate each feature, and the tf.feature is a binary encoded representation of the input data that will be used by the model to generate summaries. A section of code from the PG-MMR algorithm which serializes the input documents is shown in Appendix A.1. In the code, the cleaned article is passed to the *write_binary* function, alongside the abstract of the article[1] and the original uncleaned article split into a list sentence-wise. A Tensorflow Example is created and each parameter is encoded to a byte string and added to the example. Once the data has been encoded, the example is written to a binary *.bin* file.

When the data is loaded to be processed by the model, it is given to a *Batcher* class to split the data into mini-batches. Specifically, each mini-batch contains N Tensorflow Examples where N is the selected *Beam size*. This is used in the decoding phase which implements a Beam Search algorithm (Discussed in Section 5.1.3).

[1] Each document in the DUC and TAC datasets has an abstract included which was used as the headline for the given article.

5.1.2 Encoder

As mentioned in Section 4.2.2, a bidirectional LSTM is used as the encoder for the model. In the code where the model is built, two Tensorflow LSTMS are instantiated. One for encoding the data forwards and one for encoding it backwards. The two LSTMs are then used to create a singular Bi-LSTM Tensorflow object. Once the input data is encoded using this Bi-LSTM, two separate states are produced representing each direction of encoding. A function called *_reduce_states* is defined in order to take each of the aforementioned states and combine them into one through an additional linear *Relu* layer (similar to sigmoid mentioned in Section 4.2.2). This is necessary as the decoder is a unidirectional LSTM and thus can only take one state as input. The *_add_encoder* function which instantiates the Bi-LSTM is shown in appendix A.2.

5.1.3 Decoder

The code to create the decoder is almost identical to the encoder. The differences are that only one LSTM cell is created (since the decoder is a Unidirectional LSTM) and some extra values are stored, namely, previous MMR score and previous coverage. The LSTM cell and extra values are passed to another function called *attention_decoder*. This function is where all of the features of our decoder are added i.e attention, MMR scoring, coverage and the pointing mechanism. Alongside this function, another one called *attention*() defines multiple helper functions which are used to calculate scores for MMR, coverage, p_gen as well as the attention distribution. This is shown in the appendix A.3.

Appendix A.3 shows the code for the steps mentioned in Section 4. This function is called at each timestep to collect each score from the previous timestep. These values are all used to produce the next decoder output. The *P_gen* probability is also calculated for each timestep, the code for which is shown in appendix A.4. The calculation of *P_gen* makes use of the context vector, the hidden state of the decoder and the previous decoder output. It is the implementation of equation 4.7 from Section 4.

5.1.4 Beam Search

When the decoding process is run, output words need to be generated. These words are generated using a beam search algorithm. Traditionally, what is called a *greedy search* algorithm is used. In the case of producing a summary, a greedy search algorithm would simply pick the word at each timestep with the highest probability. This simple algorithm can work well, however, it can lead to *important* words being picked repeatedly. While this is good for outputting important information, it produces sentences that are not well structured. In a typical human-written sentence, there are important words, but also words that are used to tie the ideas presented together i.e stopwords. The greedy search algorithm can ignore these words and thus produce a summary that isn't readable. To combat this, Lebanoff et al. (2018) use a beam search algorithm for decoding. Beam search works by consolidating multiple *hypotheses* from the potential outputs by expanding all of the possible steps for them, it then only makes use of the most likely hypothesis (Brownlee (2019)). In other words, it picks the k most likely words across the distribution, and keeps track of the state for each of them. At each timestep, the successors for each selected word are generated. It then evaluates the set of successors for each initial word that was picked and selects the best group of successors to be used in the summary Russell and Norvig (2002).

The number of concurrent hypotheses made is determined by a predefined hyperparameter, i.e if *beam_size* = 2 then it will evaluate 2 different potential sentences at the same time. It is important to note that if *beam_size* = 1, the beam search algorithm behaves in a greedy manner. Regarding the implementation of beam search, the function *run_bream_search* is called in the main *decode* function in *decode.py*. It iterates from range 0 to the *max_decoder_steps* (the maximum word length of a generated summary, in this case, set to 50.). In this loop, the beam search algorithm

gets the latest tokens produced as part of a hypothesis, as well as the previous coverage and MMR scores. If it is the first decoding step, then these values are empty. It runs one decoding step in order to produce a singular output distribution. This distribution of word probabilities is then used to form the hypotheses. It then loops over the range of hypothesis required. Inside this loop, the successors are generated from each initially selected word. *Hypothesis* is a defined class in the code and has the attribute *avg_log_proba*, which is the sum of the log probabilities of the tokens so far. This probability is then normalized by the length of the hypothesis generated since there will always be a bias in giving longer sequences a higher probability. Once all of the hypotheses have been generated, they are sorted by their *avg_log_proba* in descending order. This list is then filtered to contain only hypotheses that end (i.e end with a '.'), as well as hypotheses that meet at least the minimum length quota (defined as the hyperparameter: *min_decoder_steps*). This list of ranked hypothesis is returned back to the *decode* function mentioned previously. From this list, the first hypothesis at index 0 is selected, and each number inside is converted back to its original word and appended to the summary[2]. This summary is then written to file.

5.2 NEG Implementation

Named entity generalisation was implemented in this project following a method inspired by the work of Kouris et al. (2019). As mentioned in Section 4.3, for generalisation to be possible, a taxonomy of concepts must be created. There are many forms of software that can be used for this but the most renowned and the one used in this project is WordNet (Miller (1998)), a large lexical database for the English language.

5.2.1 Wordnet

Wordnet comes as part of Python's Natural Language Tool Kit (NLTK) (Loper and Bird (2002)) and it is used to create the structured taxonomy of concepts. In Wordnet; nouns, verbs, adjectives and adverbs are grouped into collections of cognitive synonyms. Wordnet calls each of these words in the collection a *synset* (which we call generalisable concepts). Wordnet has a function *wordnet.synsets*() which takes a word as the parameter and returns a list of hypernyms for that word (*What is WordNet?* (2020)). It is important to note that the *synsets*() function specifically returns a list of lists, where each element is a list of hypernyms for a given synset. A list of lists is returned as WordNet groups synsets by their semantic meaning, not by the word itself. This is done as words can have different meanings depending on the context they are used in. For example the use of *wordnet.sysnset("Program")* returns the hypernyms for the word 'program' in relation to a *plan* or *schedule*, but it also returns the hypernyms in relation to a *computer-system* or *software*. Using the context of the sentence to predict which hypernyms to use would be a complex task. Moreover, Raganato et al. (2017) have demonstrated that the first option in the list (called Wordnet's *first sense*), is a hard baseline for word sense disambiguation approaches. Thus in this implementation, the first item in the returned synset lists was chosen as the taxonomy.

5.2.2 NER Tagging

We can extract a taxonomy of concepts using Wordnet, however, it is impossible to tell whether or not a word in the source text is a named entity without a Named Entity Recognition tool. There are multiple options available for this software, the main ones being; Spacey (Honnibal and Montani (2017)), StanfordNER (new API) and StanfordNER(old API) Finkel et al. (2005). All are viable options, each with advantages and drawbacks. A comparison of each of these NER tagger is shown in table 5.1 below. As can be seen from the table, Spacey is favourable

[2]When pre-processing, a *vocab* file is produced which maps words to word IDs. This vocab is parsed into a python dictionary to be used at decode time. $word2id(id)$ performs a lookup for the id provided and returns the corresponding word.

to StanfordNER (new API) as it received a higher score as well as being over 1000% faster. However, the StanfordNER(old API) had a much higher accuracy score. For this reason it was used in the project.

Table 5.1: *Results for each NER parser*

	Spacey	StanfordNER (old API)	StanfordNER (new API)
Time to Complete	7 minutes	24 hours	85 minutes
Found Entities	7946	15503	7582
Percentage of cities found out of all in text	12.5%	24.5%	11.9%

NB: Each scanned a document where the only named entities included were the names of cities.

The StanfordNER tagger is written in Java and in order to get it running, a folder needs to be downloaded containing all of the necessary JAR files. To interact with these JAR files, python's NLTK provides a *StanfordNERTagger* class which takes the path to the JAR files as input to initialise the *tagger* object. This tagger object has an instance method *tag*, which, when given a list as input, returns a list of the same length containing word types. For example the list: *['University', 'of', 'Glasgow', 'Scotland', 'Nicola', 'Sturgeon']* when passed to the *tagger* object returns *[('University', 'O'), ('of', 'O'), ('Glasgow', 'LOCATION'), ('Scotland', 'LOCATION'), ('Nicola', 'PERSON'),('Sturgeon', 'PERSON)]*.

5.2.3 Pre-processing

In order for PG-MMR to be able to deal with the concept tags from the generalisation process, the model has to be retrained on a generalised dataset. To do this, the original PG implementation was used See et al. (2017). However, instead of training it on the original CNN/DM dataset, the dataset was pre-processed in order be generalised. The code for the generalisation of the testing data is identical. In detail, a *Generalise* class was created, taking as input; the cleaned article text, the abstract for the article, the raw article list of sentences, a hyperparameter *search_hypernyms*[3] as well as the frequency threshold. There are three main functions to pre-process the text; *get_generalised_article*, *get_generalised_abstract* and *get_generalised_article_sents* for generalising the article, abstract and raw sentences respectively. There are also a multitude of helper functions defined to make the code cleaner and easier to modify. Code for *get_generalised_article* is presented in appendix A.5.

5.2.4 Caching

It is important to note that caching is used in the pre-processing stage. This is done because the StanfordNER tagger, whilst being accurate, is very slow (as shown in table 5.1). Even with caching the words seen, the entire pre-processing stage took just over two weeks whilst running on the university's HPCC. Considering this, it is clear that caching these words for quicker use is imperative as failing to do so could have led to pre-processing potentially taking months. A graph showcasing pre-processing progress while leveraging caching is shown below. During generalisation, every word seen are cached along with their corresponding type (returned by the *StanfordNerTagger*), as well as their hypernyms (from *Wordnet*) using python dictionaries. These dictionaries are then stored locally using python pickle as 'vocab.p' and 'hypernyms.p'. A benefit of this is that even at the testing phase, when generalising the test data, the same cached

[3]Each word in the source text is either it is a named entity or not. However, even if it is not, there could still be a named entity as a hypernym in its taxonomy of concepts, so when this parameter is set to True, the hypernyms are searched to check whether any of them are named entities. If there is, then the original word is generalised

dictionaries are used, which greatly speeds up the process of generalising the DUC and TAC datasets.

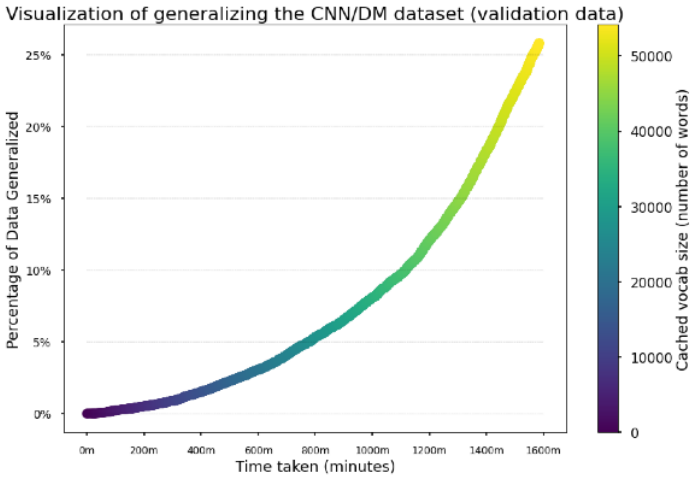

Table 5.2

It should be noted that once the data is generalised, it is pre-processed the same way as the regular datasets for both training and testing. I.e it is split up into to TensorFlow Examples in order to be linearly read by the model (as mentioned in Section 5.1.1).

5.2.5 Post-processing the generated summaries

As mentioned in Section 4.3.3, a matching algorithm is used in order to replace concept tags in the generated summary with the original corresponding word. This is done using a sliding window technique wherein a sliding window of a specified size is *slid* over the summary. If the window contains a concept tag, then a sliding window is performed over the source text. If a token within this sliding window is generalisable, then it is a potential candidate for replacement, and thus the similarity is computed between the sliding window in the summary and the sliding window in the source text. This similarity score is then appended to a list containing all the candidates for replacement. Once the source text has been iterated over, the list of similarities is sorted in descending order and the maximum one is used to replace the concept tag. This is done for each concept tag in the summary. Code for this algorithm is shown in appendix Section A.6.

5.2.6 Challenges

Many challenges were faced during the course of this project and as such a lot of time was wasted on certain aspects of it. The most prevalent example is that originally this project was going to be built upon an image sequence captioning network (visual storytelling) which incorporated an encoder-decoder model. Two to three months were spent on this existing implementation in order to try and extend it to the domain of text summarisation. This proved unfruitful and after three months of work, it was abandoned and an extension of the PG-MMR model was favoured. As a result, a lot of time was wasted and thus, hindered the progress of the project. Challenges were also faced when trying to run this project on the University of Glasgow HPCC. As mentioned in Section 3.7, the HPCC is very unforgiving when submitting code with bugs. The code would fail after waiting in the job queue for hours which drastically slowed down progress.

6 Evaluation

This section of the report is an appraisal of the summaries generated by each of the aforementioned models. Each metric used in order to evaluate the models is described and the subsequent results attained for each model are presented. Scores attained by some of the models mentioned in Section 2 are also presented.

6.1 Metrics

There id a multitude of evaluation metrics available that can be used to evaluate a summarisation task. Some of which are taken from the similar domain of Machine Translation (MT). The three evaluation methods that are used in this project are BLEU (Bi-Lingual Evaluation Understudy) from Papineni et al. (2002), ROUGE (Recall Oriented Understudy for Gisting Evaluation) from Lin (2004) and METEOR (Metric for Evaluation of Translation with Explicit ORdering) from Banerjee and Lavie (2005). ROUGE is the only evaluation metric built from the ground up for text summarisation, whereas BLEU and METEOR have been borrowed from MT.

6.1.1 BLEU

BLEU was the first evaluation metric developed out of the three. The key idea behind it is that, "The closer a machine translation is to a professional human translation, the better it is". This is the central idea behind the BLEU metric. To represent this *closeness to reference* metric, the concept of precision is used. To calculate the precision for a given summary, the number of words in the generated summary that also appear in the reference summary are counted, and then divided by the length of the generated summary. This is the acknowledged standard precision algorithm, however, MT systems (or in our case, summarisation systems) can over-generate generic and common words (for example 'the', 'when', 'is' etc.). The over representations of these generic words can lead to summaries that are given a very high precision score but are actually of low quality. To combat this problem, BLEU introduces the concept of *modified uni-gram precision (MUP)*. To calculate MUP, firstly the maximum number of times a word appears in the references is counted. Then, each word in the generated summary is *clipped* to match its count in the reference[1]. These *clipped* counts are then added up and divided by the full length of the *un-clipped* generated summary. An example is shown in Table 6.2. Using the standard precision metric would give the generated summary a score of (7/7) 100%.

Table 6.1: *Modified N-Gram Precision*

Generated Summary	the the the the the the the the the
Reference Summary	the cat is on the mat

Modified uni-gram Precision = 2/7

[1] I.e, the clipped count is the *min(Count_in_generated, Count_in_reference)*, the rest of the words are ignored in regards to appearing in the reference summary.

For multi-sentence summaries, the whole summary is treated as one large sentence. This modified n-gram precision can be computed similarly for any n (See Table 3.1). For the BLEU metric, n-gram sizes up to $n = 4$ are computed and weighted in order to produce the final score. In their paper, Papineni et al showed that the precision decays exponentially with an increasing n. As a result, to combine the n-gram values, a weighted average of the log of modified precisions is used. BLEU also introduces a brevalty penalty for summaries that are too long. Generated summaries that are too long are already penalised by the modified n-gram precision so there is no need to further penalise. However, for high-scoring generated summaries, the summary must match the length of the reference summary. When the length of the generated summary matches the length of the reference summary, the brevalty penalty is 1.0. Otherwise, the brevalty penalty is calculated using equation 6.1 below.

$$BREV = \begin{cases} e^{1-\frac{R}{G}}, & \text{if } G \leq R \\ 1.0, & \text{otherwise} \end{cases}, \tag{6.1}$$

Where G denotes the length of the generated summary and R denotes the length of the reference summary.

The full BLEU metric calculation is performed using the equation below.

$$BLEU = BREV * exp(\sum_{n=1}^{N} w_n \log p_n) \tag{6.2}$$

To extend the BLEU metric to take multiple references into account, the score is computed for each of them and then the maximum is used as the final BLEU score.

6.1.2 ROUGE

The paper that introduced ROUGE evaluation Lin (2004) presents four separate measures that can be used, ROUGE-{N, L, W, S}. For abstractive summariaztion, ROUGE-N and ROUGE-L are the commonly used metrics, and thus they will be focused on.

ROUGE-N

ROUGE-N measures the n-gram similarity between words. Specifically, it measures the n-gram *recall* between a generated summary and a set of reference summaries. The equation to compute ROUGE-N is as follows:

$$\text{ROUGE-N} = \frac{\sum_{S \in ReferenceSummaries} \sum_{gram_n = inS} Count_{match}(gram_n)}{\sum_{S \in ReferenceSummaries} \sum_{gram_n = inS} Count(gram_n)}, \tag{6.3}$$

Where n denotes the degree of the n-gram, $gram_n$ and $Count_{match}(gram_n)$ is the maximum number of overlap of n-grams between the generated summary and reference summaries.

In simpler terms, ROUGE-N is the ratio between n-grams that are present in both the generated and reference summaries to the total number of n-grams that occur in the reference summaries. ROUGE-N is closely related to the aforementioned BLEU metric, however, instead of measuring precision, ROUGE-N measures recall. In the equation above, the score is only computed for a single reference summary. To apply this method to multiple references, similarly to BLEU, the ROUGE-N score is computed for the generated summary in relation to each reference summary

available. The maximum score attained is then used to represent the ROUGE-N score for the summary. The equation to calculate ROUGE-N for multiple references is shown below.

$$\text{ROUGE-N}_{multi-reference} = argmax_i(\text{ROUGE-N}(ref_i, S)) \tag{6.4}$$

ROUGE-L

ROUGE-L measures the Longest Common Sub-sequence (LCS) for the generated summary in regards to the reference summaries, i.e the largest degree n-gram that appear in both the generated summary and the reference summary. An interesting characteristic of the ROUGE-L metric is that it does not require the words apart of the n-grams to be sequential in the text. As long as the word level ordering of the words in the text is retained, then it is still considered as an n-gram. It can be seen in the table below that ROUGE-L is a very good metric for calculating the similarity between two pieces of text, as it takes the ordering of words into account along with the n-gram overlap.

Table 6.2: *ROUGE-L: Longest Common Sub-sequence*

Generated Summary	The cat was found under the bed
Reference Summary	The cat was under the bed

in this case, ROUGE-L = 6

In their paper, Lin proposes the calculation of LCS as an F-measure[2] between the generated summary and reference summary. The F-Measure score denoted in equation 6.7 is what is referred to as the ROUGE-L metric. Similarly to ROUGE-N and BLEU mentioned previously, ROUGE-L can be extended to be used with multiple references simply by computing the score for the generated summary and each of the references and taking the maximum score.

$$Recall_{lcs} = \frac{LCS(gen, ref)}{length(gen)} \tag{6.5}$$

$$Precision_{lcs} = \frac{LCS(gen, ref)}{length(ref)} \tag{6.6}$$

$$F - Measure_{lcs} = \frac{(1 + \beta^2) Recall_{lcs} Precision_{lcs}}{Recall_{lcs} + \beta^2 Precision_{lcs}}, \tag{6.7}$$

where $\beta = \frac{Recall_{lcs}}{Precision_{lc}}$.

[2]An F-Measure is the harmonic mean of the precision and recall, it is a good measure of a piece of text's overall *accuracy*

6.1.3 METEOR

METEOR works by defining an alignment between two pieces of text as a mapping of uni-grams between the generated summary and the reference summary. Each uni-gram in either of the summaries maps to exactly *zero* or *one* uni-gram in the other summary. This mapping is produced over the course of multiple stages. An example of such mapping is shown below in Figure 6.1

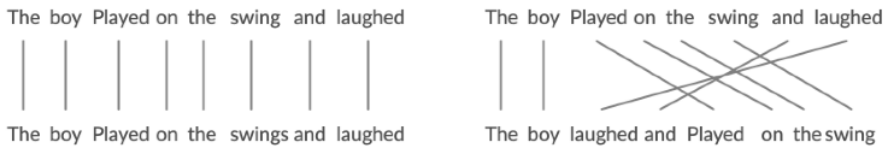

Figure 6.1: *A mapping of uni-grams over two pieces of text. N.B the number of uni-gram crosses for the mapping on the left and right hand side are* 0 *and* 8 *respectively*

For the first phase, a list of all possible uni-gram mappings are created by using an assortment of three different modules; the *exact module*, the *Porter stem* module and the *WN synonyms* module where each of these modules creates its own mapping. The exact module only maps identical uni-grams between the two pieces of text. The porter stem module stems each word in the pieces of text (See Section 4.2.3) and maps identical stemmed uni-grams. Finally, the WN synonyms module maps all uni-grams which are synonyms of one another. Using these three different modules allows for much more flexible summaries. The WN synonyms module is especially imperative as it appropriately scores abstractive summaries where synonyms can be used to generalise text by the model.

In the second stage, the largest subset of the uni-gram mappings is selected wherein the resulting subset forms an alignment (a mapping where each uni-gram maps to at *most* one other uni-gram). If there are more than one of such alignments, then the alignment with the least uni-gram crosses is selected (demonstrated in Figure 6.1).

Similarly to ROUGE-L, the METEOR score for a generated summary and a reference summary is calculated by using an F-Measure. First, the precision and recall for the generated summary are calculated. Precision in this case is a simple ratio of words in the generated summary that appear in the reference summary to the total number of words in the *generated* summary. Similarly, recall is measures as the ratio of words in the generated summary that appear in the reference summary to the total number of words in the *reference* summary. The precision and recall for METEOR is calculated the same was as above for ROUGE-L. However, instead of the longest common sub-sequence being used as the numerator, the number of uni-grams in the generated summary that also appear in the reference summary is used. METEOR also *hard-sets* the value for β as 3.

The F-Measure value forms the basis of the METEOR score. However, METEOR additionally defines a penalty for variance in length between the generated summary and reference summary. This penalty is computed by chunking all of the uni-grams in each summary into the highest possible n degree n-grams. The higher the n for each n-gram, the less chunks there will be. Once finished, each chunk will represent an n-gram where n = length(chunk). Once chunked, the penalty can be computed as:

$$penalty = 0.5 * (\frac{\text{number of chunks}}{\text{number of matched uni-grams}}) \tag{6.8}$$

The METEOR score is then calculated using the *F-Measure* and the *penalty* in the following equation:

$$Score = F - Measure * (1 - penalty) \tag{6.9}$$

The external modules used to compute the METEOR score are all used in conjunction with one another. Specifically, the *exact* module firstly maps all of the uni-grams between the generated and reference summaries. Then, if there are any uni-grams that still have not been mapped, another mapping stage is performed using the *porter stemmer* module. Similarly, if there are still uni-grams in the generated summary that do not have a mapping to a uni-gram in the reference summary, then the *WN synonyms* module is used. In their paper, Banerjee and Lavie found that using all three modules in this systematic way performed the best when compared to human evaluations.

6.1.4 Metric Issues

Each of the aforementioned metrics have issues that hinder their ability to accurately evaluate summaries. The BLEU metric has generally received the most criticism as a metric. This is for a multitude of reasons, mainly being that the ordering of summaries is not taken into account. For example, the text "the weather is nice" and "weather the nice is" will receive identical scores using the BLEU metric. The BLEU metric also does not consider the semantics of words. Therefore, the summaries will be penalised for using synonyms which is undesirable behaviour. Although, this issue is not specific to BLEU as ROUGE also does not consider semantic content. As shown in Agarwal and Lavie (2008), METEOR was shown to a have a very high correlation in regards to human evaluations of summaries. METEOR is also slightly newer than ROUGE and BLEU and thus it appears less across research in text summarisation. However, due to its benefits, it could be argued that METEOR is the optimal metric and its use should become commonplace in the evalutaion of text summarisation.

6.2 Results

Results for the addition of generalisation to PG-MMR were mixed. The evaluation results were typically lower compared to the original PG-MMR model. However, it can be argued that the summaries produced were of higher quality.

Each of the evaluation metrics used in this project has individual flaws. However, a shared fundamental flaw is that they work by comparing the generated summary to a set of reference summaries (human-written). Human-written summaries are subjective by nature. Different people summarizing the same document will approach it in different ways. Some will choose to be more extractive in their approach whereas some will be more abstractive. As shown in Table ??, the reference summaries provided by the TAC and DUC datasets are very abstractive by nature. PG-MMR-GEN, whilst being fundamentally an abstractive model, is highly extractive. This means that the model will be heavily penalised even if good *extractive* summaries are produced. The generalisation of the dataset will detriment the evaluation score further, as it forces the summary to be more extractive when it comes to matching the concept tags during post-processing.

To reiterate, this does not necessarily mean that the summaries produced are of lesser quality, but simply that the summaries produced are less similar to the heavily abstractive reference summaries provided for the DUC and TAC datasets.

Contradictory to the evaluation scores, generalisation does improve the performance of the model. This can be seen clearly in the table shown below. In the first few words of the summary, PG-MMR chooses to omit 'Peter's second name 'Kennedy'. Kennedy appears less than the other names in the source text and thus is forgotten at decode time which is undesirable. By using generalisation, Kennedy was preserved by replacing it with a concept tag before encoding.

This is the main reason for implementing named entity generalisation. Details such as 'Peter's second name are important to the context of the summary and need to be included in order to

be considered high quality. Models that do not use generalisation can forget these important details and thus detriment the summaries information content.

Generalisation changes the way in which words are given attention. The addition of generalisation not only made named entities more prevalent in the generated summaries but it also changes sentences that do not include any. An example of this is also present in table 6.3 (full table with non-post-process summar is shown in appendix A.1). At the end of the PG-MMR-GEN summary, the words 'for the hostages' are included. This section of words does not contain any named entities, however, it does add to the fluidity of the text.

Table 6.3: *Model Results*

Model	**Generated Summary**
PG-MMR	Britons Peter , Darren Hickey , and Rudolf Petschi , and Stanley Shaw 3 in Grozny . hundreds of Chechen were found lined up along a highway Tuesday outside Chechnya 's capital Grozny . They had been abducted by unidentified gunmen Oct. 3 in Grozny . hundreds of people have been kidnapped in the breakaway republic and neighboring regions by armed gangs who are mostly motivated by ransom . the men are still searching for the men 's bodies and returning to Chechnya 's capital city Grozny from a small village nearby , news reports . the one alleged kidnapper had been arrested and authorities gleaned enough information to launch a rescue operation .
PG-MMR-GEN post-processed	Britons Peter Kennedy, Darren Hickey , and Rudolf Petschi , and Stanley Shaw 3 in Grozny . hundreds of Chechen were found lined up along a highway Tuesday outside Chechnya 's capital Grozny . They had been abducted by unidentified gunmen Oct. 3 in Grozny . hundreds of people have been kidnapped in the breakaway republic and neighboring regions by armed gangs who are mostly motivated by ransom . the men are still searching for the men 's bodies and returning to Chechnya 's capital city Grozny from a small village nearby , news reports . the one alleged kidnapper had been arrested and authorities gleaned enough information to launch a rescue operation for the hostages .

In table 6.4, less impressive results are shown. Both PG-MMR and PG-MMR-GEN struggle to form grammatically correct sentences. What is clear to see however, is that PG-MMR-GEN does include more important information and is somewhat better at structuring its sentences.

Using generalisation fundamentally changes the attention weights applied to each word at decode time. Due to the abundance of concept tags that are in the generalised input data, more focus is placed on them and hence, more focus is placed on sentences that contain them. This heavily influences the pointing mechanism to extract sentences that contain multiple concept tags. Therefore, the summaries produced are more likely to extract sentences that have a high number of named entities and then generate the rest of the words. An example of this is shown in the first sentence shown in table 6.4 below (full table with intermediate summary is shown in appendix A.2). First, PG-MMR chooses to use 'Philippine presidents' whereas PG-MMR-GEN chooses to be specific and use the presidents full name. The post-processing algorithm struggles here however, as _person_ _person_ is replaced by "Estrada" "Estrada". The reason behind this is because when two concept tags are next to each other in the summary, and the first is replaced, the replacement will not be taken into account for finding the replacement for the second concept tag. Thus the most likely candidate will be the same as for the previous concept tag. This is a flaw in the implementation of post-processing.

The second sentence in each of the summaries also refer to the same topic. However, in PG-MMR-GEN, the ideas are presented better. PG-MMR-GEN includes much more important

information and also is grammatically correct. The second sentence in the PG-MMR-GEN summary (highlighted in red) is an extracted sentence from the source document that was extracted due to the generalisation mechanism prioritising named entities. This shows that using generalisation alongside PG-MMR works especially well, as it influences the pointer-generator's decisions, as well as retaining named entities. Later in the summaries, both models struggle to form a correct sentence, however it can be see that PG-MMR-GEN chooses to include important information in the text rather than PG-MMRs repeated use of 'New'.

Table 6.4: *Model Results*

Model	Generated Summary
PG-MMR	Philippine presidents have hinted they might not attend the meeting of the 18 Asia-Pacific Economic Cooperation forum . the challenges are no connection , then had him expelled from Malaysia 's ruling party . The is scheduled to go to Malaysia for next week 's summit in the heart of Southeast Economic forum .the Philippines , Brunei , Chile , Chile , China , New , New , New , New , New , New , New , New , New , New , New , New , New , New , New , Taiwan , Thailand and the United States .
PG-MMR-GEN post-processed	Estrada Estrada said he was considering not going to the Asia-Pacific Economic forum because of Anwar Ibrahim 's arrest . he said " there is no connection " between Anwar 's case and the summit meeting in Malaysia of the Estrada Estrada Economic forum . The is scheduled to go to Malaysia for the meeting of the 18 Asia-Pacific Economic Cooperation forum . the Estrada , Estrada , Estrada , Estrada , Estrada Estrada , Estrada , South Korea , New , New , New , New , New , New , New , Malaysia , Malaysia , Malaysia , Malaysia , Malaysia , Malaysia , Malaysia , Malaysia and the United States .

The evaluation scores for the proposed model are shown in the two tables below, alongside other text summarisation models scores. The table below contains the results attained for the DUC 2004 dataset. The DUC 2003 and TAC dataset scores are shown in Appendix A.3, A.4, A.5, A.6 respectively.

Table 6.5: *DUC 2004 RESULTS*

Model	ROUGE-1	ROUGE-2	ROUGE-L	BLEU	METEOR
SumBasic	29.48	4.25	-	-	-
ABS	26.55	7.06	22.05	-	-
ABS+	28.18	8.49	23.81	-	-
LexRank	34.44	7.11	-	-	-
TextRank	35.6	7.81	-	-	-
Nallapati et al. (2016)	28.61	9.42	25.24 -	-	-
Zhang et al. (2018)	36.7	7.83	-	-	-
Kouris et al. (2019)	28.73	9.87	26.12	-	-
PG	31.16	5.96	27.97	87.25	24.27
PG-MMR	35.57	7.69	31.46	89.53	27.01
PG-MMR-GEN	35.29	7.75	27.68	85.51	26.17

7 Conclusion

7.1 Results

In this project, the effectiveness of semantic concept generalisation on neural encoder-decoder RNN model was explored. In order to assess this, we built on top of the work of Lebanoff et al. (2018) in order to combine their work with a generalisation method proposed by Kouris et al. (2019).

Using generalisation with the PG-MMR model produced encouraging results. Whilst it was given slightly lower scores by the evaluation metrics, it can be seen that it does in fact add value to the summaries produced. Summaries produced by the model typically contained more important information by way of named entities. Moreover, the extractive aspects of the model put more focus on extracting sentences with named entities in them. This not only meant that more important words were used in the summaries but also that the sentences containing them were well structured. More fine tuning of the generalisation mechanism is needed in order to out-perform the original PG-MMR model. This can be done by re-training the PG model with different levels of generalisation, which this project would have included if time permitted. The example summaries produced show that generalisation does in fact have a place in text summarization and should be further researched in order to increase its performance.

7.2 Reflections

This project is the most comprehensive work that I have worked on thus far. A lot of challenges were faced along the way. I now have a far better understanding about how natural language processing is researched and implemented. I am now also far more knowledgeable in the tools that I used to create the project.

Looking back over what could have been done differently, I would have spent less time trying to extend the image sequence captioner to the domain of text summarisation. It drastically slowed down the process of the project and at the end there was nothing to show for it. In terms of the rest of the project, I am happy with how it was developed into a fully fledged model. I now have a far greater appreciation of the immense challenges that are faced when working on a large project similar to this.

7.3 Future Work

There are many possibilities for future work to be based off of this project. Semantic concept generalisation is a method which can be applied to any summarisation model and as such it could work with any of the models mentioned in the background as well as more recent state-of-the-art models. PG-MMR is a multi-document summarisation model built to extend a single document model due to the lack of datasets available in the MDS domain. However, with the more recent release of the multi-news dataset from Fabbri et al. (2019), training PG-MMR on a multi-document dataset could be a viable option. Moreover, doing so could potentially improve

its evaluation scores. The MMR scoring method could also be explored deeper. MMR is a robust method for sentence ranking, however, it is not the most recent tool to do so. LexRank and TextRank (mentioned in Section 2) are both similar methods of performing sentence ranking and were shown to have encouraging results when put to the task of extractive text summarisation. Considering this, a potential extension to PG-MMR would be to experiment with either of the aforementioned algorithms as the sentence importance mechanism instead of MMR.

Alternative methods of generalisation could also be researched. In this paper, named entities are generalised, however this could be extended to also generalise other forms of entities and find a somewhat middle ground between NEG and LEG. Text by nature can be paraphrased and reworded, however, numerical information must be repeated verbatim in order to convey the same information. Considering this, *dates*, *time*, *money* could also be generalised to enforce the summary to be faithful to the source when relaying them. Internally in the generalisation algorithm, other similarity metrics could be used, in this project word2vec Pennington et al. (2014) was used, however there are alternative ways to embed words in order to represent them numerically.

A Appendices

```
def _add_encoder(self, encoder_inputs, seq_len):
   with tf.variable_scope('encoder'):
      cell_fw = tf.contrib.rnn.LSTMCell(self._hps.hidden_dim,
          initializer=self.rand_unif_init, state_is_tuple=True)
      cell_bw = tf.contrib.rnn.LSTMCell(self._hps.hidden_dim,
          initializer=self.rand_unif_init, state_is_tuple=True)
      (encoder_outputs, (fw_st, bw_st)) =
          tf.nn.bidirectional_dynamic_rnn(cell_fw, cell_bw, encoder_inputs,
          dtype=tf.float32, sequence_length=seq_len, swap_memory=True)
      # concatenate the forwards and backwards states
      encoder_outputs = tf.concat(axis=2, values=encoder_outputs)
   return encoder_outputs, fw_st, bw_st
```

Listing A.2: *Function to create the encoder for the model*

```
def attention(decoder inputs, MMR & Coverage Scores, etc):
   # Take softmax of parameter e, pad it and re-normalize it
   def masked_attention(e):
      attn_dist = nn_ops.softmax(e) # take softmax. shape (batch_size,
          attn_length)
      attn_dist *= enc_padding_mask # apply mask
      masked_sums = tf.reduce_sum(attn_dist, axis=1) # shape (batch_size)
      return attn_dist / tf.reshape(masked_sums, [-1, 1]) # re-normalize
   # Apply the calculated mmr score to the attention distribution
   def apply_mmr_score(pre_attn_dist, mmr_score):
      post_attn_dist = pre_attn_dist * mmr_score
      post_attn_dist = post_attn_dist /
          tf.expand_dims(tf.reduce_sum(post_attn_dist, axis=1), axis=1)
      return post_attn_dist
   # Multiply coverage vector by w_c to get coverage_features.
   coverage_features = nn_ops.conv2d(coverage, w_c, [1, 1, 1, 1], "SAME")
   # Calculate v^T tanh(W_h h_i + W_s s_t + w_c c_i^t + b_attn)
   e = math_ops.reduce_sum(v * math_ops.tanh(encoder_features + decoder_features
       + coverage_features), [2, 3]) # shape (batch_size,attn_length)
   # Calculate attention distribution
   pre_attn_dist = masked_attention(e)
   # Added for PG-MMR: apply mmr_score to change distribution
   attn_dist = apply_mmr_score(pre_attn_dist, mmr_score)
   # Update coverage vector
   coverage += array_ops.reshape(attn_dist, [batch_size, -1, 1, 1])
```

```
def write_binary(self, article, abstracts, raw_article_sents, writer):
      tf_example = example_pb2.Example()
      tf_example.features.feature['article'].bytes_list
         .value.extend([str.encode(article)])
      for abstract in abstracts:
         tf_example.features.feature[b'abstract'].bytes_list
            .value.extend([str.encode(abstract)])
      for sent in raw_article_sents:
         tf_example.features.feature[b'raw_article_sents'].bytes_list
            .value.extend([str.encode(sent)])
      tf_example_str = tf_example.SerializeToString()
      str_len = len(tf_example_str)
      writer.write(struct.pack('q', str_len))
      writer.write(struct.pack('%ds' % str_len, tf_example_str))
```

Listing A.1: *Function to encode data in to serialized tensorflow examples*

```
   # Calculate the context vector from attn_dist and encoder_states
   context_vector = math_ops.reduce_sum(array_ops.reshape(attn_dist, [batch_size,
       -1, 1, 1]) * encoder_states, [1, 2]) # shape (batch_size, attn_size).
   context_vector = array_ops.reshape(context_vector, [-1, attn_size])
   return context_vector, attn_dist, coverage, pre_attn_dist
```

Listing A.3: *Function to create the decoder for the model*

```
# state.c is the hidden state and state.h is the output
 p_gen = linear([context_vector, state.c, state.h, x], 1, True) # Tensor shape
     (batch_size, 1)
p_gen = tf.sigmoid(p_gen)
p_gens.append(p_gen)
```

Listing A.4: *calculation of the $p_g$enprobability*

```
def get_generalised_article(self):
   article = self.text
   # iterate over every word in the text
   for i, word in enumerate(self.text):
      # If the word has already been seen, then get its tag
      if word.lower() in self.words_seen:
         tag = self.words_seen[word.lower()]
      else:
         # otherwise, use the StanfordNERTagger to perform NER on the word
         tag = self.get_ner_tag(word)
         # Add it to the words we have seen
         self.words_seen[word.lower()] = str(tag)
      # If the word is a named entity OR any of its hypernyms are a named entity
          AND it appears less times than the threshold
```

```
        if (self.is_named_entity(tag) or self.has_named_entity_hypernym(word)) and
            self.word_counts[word] < self.freq_thresh:
            # Generalise it
            article[i] = self.ner_tags[tag]
    # store caching dictionaries for later use
    pickle.dump(self.words_seen, open('./cache/vocab.p', 'wb'), protocol=2)
    pickle.dump(self.synsets_and_hypernyms, open('./cache/hypernyms.p', 'wb'),
        protocol=2)
    # Return the article in its generalised form
    return ' '.join(article)
```

Listing A.5: *Function to create the decoder for the model*

```
def process_generalised_summary(self, summary, raw):
    copy = []
    generalised_concepts = []
    candidate_replacements = []
    chunks = self.slidingWindow(summary.split(), self.summary_win_size,1)
    # iterates over the summary in chunks where each chunk is a tuple of size
        $summary_win_size$
    for chunk in chunks:
        word = chunk[0]
        if self.is_generalised(word):
            generalised_concepts.append(word)
            context = self.slidingWindow(raw.split(), self.source_win_size,1)
            # iterates over the source text in chunks where each chunk is a tuple of
                size $source_win_size$
            for slide in context:
                # If the token in the source text is generalisable or contains a
                    generalisable hypernym, it is a potential candidate
                if self.is_named_entity(tag):
                    # if its generalisable, get the similarity between its context and
                        the context of the tag in the summary
                    sim = self.similarity(chunk, slide)
                    if sim != 0:
                        candidate_replacements.append((word, token,sim))
                else:
                    continue
            # Sort similarities in descending order
            sort = sorted(candidate_replacements, key=self.sec_elem, reverse=True)
            max_sim = sort[0]
            # concept tag with most likely candicate
            copy.append(max_sim[1])
            continue
        copy.append(word)
        # Return preprocessed summary
    return ' '.join(copy)
```

Listing A.6: *Function to create the decoder for the model*

Table A.1: *Model Results*

Model	Generated Summary
PG-MMR	Britons Peter , Darren Hickey , and Rudolf Petschi , and Stanley Shaw 3 in Grozny . hundreds of Chechen were found lined up along a highway Tuesday outside Chechnya 's capital Grozny . They had been abducted by unidentified gunmen Oct. 3 in Grozny . hundreds of people have been kidnapped in the breakaway republic and neighboring regions by armed gangs who are mostly motivated by ransom . the men are still searching for the men 's bodies and returning to Chechnya 's capital city Grozny from a small village nearby , news reports . the one alleged kidnapper had been arrested and authorities gleaned enough information to launch a rescue operation .
PG-MMR-GEN un-post-processed	Britons Peter _person_ , Darren Hickey , and Rudolf Petschi , and Stanley Shaw 3 in Grozny . hundreds of Chechen were found lined up along a highway Tuesday outside Chechnya 's capital Grozny . They had been abducted by unidentified gunmen Oct. 3 in Grozny . hundreds of people have been kidnapped in the breakaway republic and neighboring regions by armed gangs who are mostly motivated by ransom . the men are still searching for the men 's bodies and returning to Chechnya 's capital city Grozny from a small village nearby , news reports . the one alleged kidnapper had been arrested and authorities gleaned enough information to launch a rescue operation for the hostages .
PG-MMR-GEN post-processed	Britons Peter Kennedy, Darren Hickey , and Rudolf Petschi , and Stanley Shaw 3 in Grozny . hundreds of Chechen were found lined up along a highway Tuesday outside Chechnya 's capital Grozny . They had been abducted by unidentified gunmen Oct. 3 in Grozny . hundreds of people have been kidnapped in the breakaway republic and neighboring regions by armed gangs who are mostly motivated by ransom . the men are still searching for the men 's bodies and returning to Chechnya 's capital city Grozny from a small village nearby , news reports . the one alleged kidnapper had been arrested and authorities gleaned enough information to launch a rescue operation for the hostages .

Table A.2: Model Results

Model	Generated Summary
PG-MMR	Philippine presidents have hinted they might not attend the meeting of the 18 Asia-Pacific Economic Cooperation forum . the challenges are no connection , then had him expelled from Malaysia 's ruling party . The is scheduled to go to Malaysia for next week 's summit in the heart of Southeast Economic forum .the Philippines , Brunei , Chile , Chile , China , New , New , New , New , New , New , New , New , New , New , New , New , New , New , New , Taiwan , Thailand and the United States .
PG-MMR-GEN un-post-processed	_person_ _person_ said he was considering not going to the Asia-Pacific Economic forum because of Anwar Ibrahim 's arrest . he said " there is no connection " between Anwar 's case and the summit meeting in Malaysia of the _location_ _location_ Economic forum . The is scheduled to go to Malaysia for the meeting of the 18 Asia-Pacific Economic Cooperation forum . the _location_ , _location_ , _location_ , _location_ , _person_ _person_ , _location_ , South Korea , New , New , New , New , New , New , New , Malaysia , Malaysia , Malaysia , Malaysia , Malaysia , Malaysia , Malaysia , Malaysia and the United States .
PG-MMR-GEN post-processed	Estrada Estrada said he was considering not going to the Asia-Pacific Economic forum because of Anwar Ibrahim 's arrest . he said " there is no connection " between Anwar 's case and the summit meeting in Malaysia of the Estrada Estrada Economic forum . The is scheduled to go to Malaysia for the meeting of the 18 Asia-Pacific Economic Cooperation forum . the Estrada , Estrada , Estrada , Estrada , Estrada Estrada , Estrada , South Korea , New , New , New , New , New , New , New , Malaysia , Malaysia , Malaysia , Malaysia , Malaysia , Malaysia , Malaysia , Malaysia and the United States .

Table A.3: DUC 2003 RESULTS

Model	ROUGE-1	ROUGE-2	ROUGE-L	BLEU	METEOR
LexRank	35.7	8.1	-	-	-
TextRank	35.3	7.2	-	-	-
PG	31.09	6.26	29.03	86.56	23.10
PG-MMR	35.95	7.11	32.90	91.82	28.64
PG-MMR-GEN	34.58	8.66	29.72	79.62	24.56

Table A.4: TAC 2008 RESULTS

Model	ROUGE-1	ROUGE-2	ROUGE-L	BLEU	METEOR
ICSI-2	37.9	38.3	-	-	-
PG	31.05	6.36	27.77	92.88	25.55
PG-MMR	35.58	8.75	31.50	95.05	30.02
PG-MMR-GEN	34.89	8.36	27.93	91.89	27.47

Table A.5: TAC 2009 RESULTS

Model	ROUGE-1	ROUGE-2	ROUGE-L	BLEU	METEOR
PG	31.84	7.38	21.50	93.19	27.02
PG-MMR	35.97	9.63	23.87	94.76	30.16
PG-MMR-GEN	35.62	8.61	18.62	92.30	28.05

Table A.6: TAC 2010 RESULTS

Model	ROUGE-1	ROUGE- 2	ROUGE-L	BLEU	METEOR
PG	29.49	6.29	26.52	90.86	24.31
PG-MMR	33.86	8.44	29.76	93.59	27.36
PG-MMR-GEN	34.28	7.47	27.91	90.52	27.08

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X. (2015), 'TensorFlow: Large-scale machine learning on heterogeneous systems'. Software available from tensorflow.org.
URL: *http://tensorflow.org/*

Abisee (2019), 'abisee/pointer-generator'.
URL: *https://github.com/abisee/pointer-generator*

Agarwal, A. and Lavie, A. (2008), Meteor, m-bleu and m-ter: Evaluation metrics for high-correlation with human rankings of machine translation output, *in* 'Proceedings of the Third Workshop on Statistical Machine Translation', pp. 115–118.

Bahdanau, D., Cho, K. and Bengio, Y. (2014), 'Neural machine translation by jointly learning to align and translate', *arXiv preprint arXiv:1409.0473* .

Banerjee, S. and Lavie, A. (2005), Meteor: An automatic metric for mt evaluation with improved correlation with human judgments, *in* 'Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization', pp. 65–72.

Brin, S. and Page, L. (1998), 'The anatomy of a large-scale hypertextual web search engine'.

Brownlee, J. (2019), 'How to implement a beam search decoder for natural language processing'.
URL: *https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/*

Carbonell, J. and Goldstein, J. (1998), The use of mmr, diversity-based reranking for reordering documents and producing summaries, *in* 'Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval', pp. 335–336.

Erkan, G. and Radev, D. R. (2004), 'Lexrank: Graph-based lexical centrality as salience in text summarization', *Journal of artificial intelligence research* **22**, 457–479.

Fabbri, A., Li, I., She, T., Li, S. and Radev, D. (2019), Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model, *in* 'Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics', Association for Computational Linguistics, Florence, Italy, pp. 1074–1084.
URL: *https://www.aclweb.org/anthology/P19-1102*

Finkel, J. R., Grenager, T. and Manning, C. (2005), Incorporating non-local information into information extraction systems by gibbs sampling, *in* 'Proceedings of the 43rd annual meeting on association for computational linguistics', Association for Computational Linguistics, pp. 363–370.

Grusky, M., Naaman, M. and Artzi, Y. (2018), 'Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies', *arXiv preprint arXiv:1804.11283* .

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M. and Blunsom, P. (2015), Teaching machines to read and comprehend, *in* C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama and R. Garnett, eds, 'Advances in Neural Information Processing Systems 28', Curran Associates, Inc., pp. 1693–1701.

Honnibal, M. and Montani, I. (2017), 'spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing'. To appear.

Hvidberrrg@GitHub (2020).
URL: *https://hvidberrrg.github.io/deep_learning/activation_functions/sigmoid_function_and_derivative.html*

Jean, S., Cho, K., Memisevic, R. and Bengio, Y. (2014), 'On using very large target vocabulary for neural machine translation', *arXiv preprint arXiv:1412.2007* .

Kouris, P., Alexandridis, G. and Stafylopatis, A. (2019), Abstractive text summarization based on deep learning and semantic content generalization, *in* 'Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics', pp. 5082–5092.

Lebanoff, L., Song, K. and Liu, F. (2018), 'Adapting the neural encoder-decoder framework from single to multi-document summarization', *arXiv preprint arXiv:1808.06218* .

Lin, C.-Y. (2004), ROUGE: A package for automatic evaluation of summaries, *in* 'Text Summarization Branches Out', Association for Computational Linguistics, Barcelona, Spain, pp. 74–81.
URL: *https://www.aclweb.org/anthology/W04-1013*

Loper, E. and Bird, S. (2002), 'Nltk: the natural language toolkit', *arXiv preprint cs/0205028* .

Luhn, H. P. (1958), 'The automatic creation of literature abstracts', *IBM Journal of research and development* **2**(2), 159–165.

Magooda, A. and Marcjan, C. (2020), 'Attend to the beginning: A study on using bidirectional attention for extractive summarization', *arXiv preprint arXiv:2002.03405* .

Manning, C. D., Raghavan, P. and Schütze, H. (2008), *Introduction to information retrieval*, Cambridge university press.

Mihalcea, R. and Tarau, P. (2004), Textrank: Bringing order into text, *in* 'Proceedings of the 2004 conference on empirical methods in natural language processing', pp. 404–411.

Miller, G. A. (1998), *WordNet: An electronic lexical database*, MIT press.

Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B. et al. (2016), 'Abstractive text summarization using sequence-to-sequence rnns and beyond', *arXiv preprint arXiv:1602.06023* .

Napoles, C., Gormley, M. and Van Durme, B. (2012), Annotated gigaword, *in* 'Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-Scale Knowledge Extraction', AKBC-WEKEX '12, Association for Computational Linguistics, USA, p. 95–100.

Nayeem, M. T. and Chali, Y. (2017), 'Extract with order for coherent multi-document summarization', *arXiv preprint arXiv:1706.06542* .

Olah, C. (2015), 'Understanding lstm networks'.
URL: *https://colah.github.io/posts/2015-08-Understanding-LSTMs/*

Papineni, K., Roukos, S., Ward, T. and Zhu, W.-J. (2002), Bleu: a method for automatic evaluation of machine translation, *in* 'Proceedings of the 40th annual meeting on association for computational linguistics', Association for Computational Linguistics, pp. 311–318.

Parker, R. and Graff, D. (2003), 'Jumbo kong, ke chen, and kazuaki maeda. 2011', *English Gigaword Fifth Edition. Linguistic Data Consortium, Philadelphia* .

Pennington, J., Socher, R. and Manning, C. D. (2014), Glove: Global vectors for word representation, *in* 'In EMNLP'.

Raganato, A., Camacho-Collados, J. and Navigli, R. (2017), Word sense disambiguation: A unified evaluation framework and empirical comparison, *in* 'Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers', pp. 99–110.

Robertson, S. (2004), 'Understanding inverse document frequency: on theoretical arguments for idf', *Journal of documentation* .

Rush, A. M., Chopra, S. and Weston, J. (2015), 'A neural attention model for abstractive sentence summarization', *arXiv preprint arXiv:1509.00685* .

Russell, S. and Norvig, P. (2002), 'Artificial intelligence: a modern approach'.

Sebastianruder (2020), 'sebastianruder/nlp-progress'.
URL: *https://github.com/sebastianruder/NLP-progress/blob/master/english/summarization.md*

See, A., Liu, P. J. and Manning, C. D. (2017), 'Get to the point: Summarization with pointer-generator networks', *arXiv preprint arXiv:1704.04368* .

Tan, J., Wan, X. and Xiao, J. (2017), Abstractive document summarization with a graph-based attentional neural model, *in* 'Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)', Association for Computational Linguistics, Vancouver, Canada, pp. 1171–1181.
URL: *https://www.aclweb.org/anthology/P17-1108*

Tu, Z., Lu, Z., Liu, Y., Liu, X. and Li, H. (2016), 'Modeling coverage for neural machine translation', *arXiv preprint arXiv:1601.04811* .

Ucfnlp (2019), 'ucfnlp/multidoc_summarization'.
URL: *https://github.com/ucfnlp/multidoc_summarization*

Weisstein, E. W. (2002), 'Sigmoid function'.

What is WordNet? (2020).
URL: *https://wordnet.princeton.edu/*

Wong, K.-F., Wu, M. and Li, W. (2008), Extractive summarization using supervised and semi-supervised learning, *in* 'Proceedings of the 22nd international conference on computational linguistics (Coling 2008)', pp. 985–992.

Zhang, J., Tan, J. and Wan, X. (2018), 'Towards a neural network approach to abstractive multi-document summarization', *arXiv preprint arXiv:1804.09010* .