Approaches to solving Single Document Extractive Summarisation

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science
in
Computational Linguistics by Research

by

Ramkishore Saravanan
20161092
ramkishore.s@research.iiit.ac.in

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad - 500 032, INDIA
March 2022
International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled 'Approaches to solving Single Document Extractive Summarisation' by Ramkishore Saravanan, has been carried out under my supervision and is not submitted elsewhere for a degree.

_________________________  __________________________
Date                        Adviser: Prof. Vasudeva Varma
To my family.
Acknowledgments

I would like to thank Prof. Vasudeva Varma for being my advisor. He always gave me freedom in exploring and let my research be my own. He also helped me a lot when I was stuck somewhere, and his guidance has always helped me proceed further. He was also a constant source of encouragement and support throughout the time I was a part of IRE-lab.

I also thank my other (unofficial) mentor, Litton J. Kurinsinkel, for introducing me to the field of Summarization. I’m really grateful for all the late night meetings and discussions we had, which really shaped up my research and laid the foundations for this thesis. And often, his critique was not just limited to my research work, but also extended to my lifestyle choices, in a good way. He was both a friend and a mentor I could always reach out to at all times.

I also extend my thanks to Dr. Manish Gupta, Nishant, Sumukh, Pappu, Bakthiyar, and Nikhil for all of their advise and help when I needed it. I thank Dr. Radhika Mamidi, Dr. Soma Paul and Dr. Dipti Mishra for introducing me to the field of linguistics. I thank Dr. Naresh Manwani for giving me strong foundations in Reinforcement Learning. I also would like to thank Richard Socher and Christopher D. Manning, whose YouTube lectures were my first introduction to NLP and Deep Learning.

I thank my parents and countless other people without whom this thesis would not be possible. Finally, I would also like to thank IIIT-Hyderabad and it’s vibrant community for giving me a wonderful learning experience.
Abstract

Summarization involves reducing the size of the document without the loss of important content. The reduced content is called the summary of a document. Summarization can be divided into two broad categories: Extractive and Abstractive. In Extractive Summarization, the summary of a document is constructed by extracting phrases and sentences from the document itself. In this work, we are mainly concerned with single document extractive text summarization and employing deep-neural-network-based solutions for the same.

While on the surface, extractive summarization seems like a trivial problem since summarizing here involves copying content from the source document, in reality, it is far from trivial. One main reason for the complexity of the task is the absence of gold labels for what sentences/phrases to extract. Typically, extractive summarization systems are evaluated using human-written summaries. However, this summary cannot be directly used to train the neural networks. Indirect methods of labeling or training are commonly used in many of these cases.

Extractive Summarization can be divided into two steps: Document Understanding and Document pruning. A document is not simply a bunch of sentences put together. There is an underlying structure between the way sentences are arranged, which makes a document coherent and meaningful. Moreover, summarisation can be said to utilize this exact structure (among other things) to determine which sentences are more important and relevant. Most works assume that the structure of the document is linear and use recursive networks to encode them. But it was shown by Rhetorical Structure Theory (RST) that documents can be represented by tree-like structures. One of the first things we explore is to see if we can use non-linear tree-based networks to encode the document. Moreover, since learning to summarize involves learning to understand documents implicitly, we see if we can try to capture such structures implicitly when learning to summarize. We also explore if such structures can aid in summarisation.

Another reason for the complexity of the task is that extractive summarization is different from sentence ranking. Sentence ranking can be defined as ranking sentences based on how good a sentence is for representing the document. However, in summarisation, the selection of each sentence also depends on the other sentences selected. This makes the task very complex due to the exponential number of possibilities in selecting a subset of sentences to represent the document. Each of these possible subsets of sentences can be called a candidate summary of the document. A document of length 25 has over
two thousand such candidate summaries, only when considering those of length 2 and 3 sentences. This creates a combinatorial explosion problem that we should overcome by using indirect means.

Many works deal with this combinatorial problem in their own way. Here we also explore different novel and unique methods for doing the same. We introduce a new pipelined architecture for extractive summarization, which helps in dealing with precisely this problem. We also see how we can adopt different complex reinforcement learning (RL) formulations to extractive summarisation. Many works use one of the two formulations in training their models: either policy-gradient-based algorithms or deep-Q-networks. In this work, we see how we can improve both of them. We also experiment with neural-network-based loss functions for summarisation.

At last we see the problems that can arise in training extractive summarisation models when we try to circumvent the combinatorial explosion problem. Many works try to simplify their training objective by approximating summarisation as sentence classification, or use very limited set of samples to train their models. This creates a lot of issues in what the model actually learns. We show this by experiments and analysis, and also provide a glimpse at a solution for solving the same.
## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>1.1</td>
<td>Extractive Summarisation</td>
</tr>
<tr>
<td>1.2</td>
<td>Sentence Understanding</td>
</tr>
<tr>
<td>1.2.1</td>
<td>Structure of a Sentence</td>
</tr>
<tr>
<td>1.2.2</td>
<td>Sentence Representation</td>
</tr>
<tr>
<td>1.3</td>
<td>Document Understanding</td>
</tr>
<tr>
<td>1.3.1</td>
<td>Structure of a Document</td>
</tr>
<tr>
<td>1.3.2</td>
<td>Document Representation</td>
</tr>
<tr>
<td>1.4</td>
<td>Document Pruning</td>
</tr>
<tr>
<td>1.5</td>
<td>Challenges in Extractive Summarisation</td>
</tr>
<tr>
<td>1.6</td>
<td>Contributions of this Thesis</td>
</tr>
<tr>
<td>1.7</td>
<td>Thesis Organisation</td>
</tr>
<tr>
<td>2</td>
<td>Related Work</td>
</tr>
<tr>
<td>2.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2.2</td>
<td>Dataset</td>
</tr>
<tr>
<td>2.3</td>
<td>Evaluation Metrics</td>
</tr>
<tr>
<td>2.4</td>
<td>Definitions and Notations</td>
</tr>
<tr>
<td>2.5</td>
<td>Neural Approaches to Summarisation</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Sequential Decision Making</td>
</tr>
<tr>
<td>2.5.1.1</td>
<td>Document Order</td>
</tr>
<tr>
<td>2.5.1.2</td>
<td>Free order</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Parallel Decision Making</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Summary Ranking</td>
</tr>
<tr>
<td>2.5.4</td>
<td>Adopting Transformer Language Models for Summarisation</td>
</tr>
<tr>
<td>2.6</td>
<td>Post processing</td>
</tr>
<tr>
<td>2.6.1</td>
<td>Trigram Blocking</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Modifying sentences after Extraction</td>
</tr>
<tr>
<td>3</td>
<td>Implicitly learning document structures using Extractive Summarisation</td>
</tr>
<tr>
<td>3.1</td>
<td>Related Work</td>
</tr>
<tr>
<td>3.2</td>
<td>Tree Neural Networks</td>
</tr>
<tr>
<td>3.3</td>
<td>The Joint task: Learning to Summarise and implicitly identify Discourse Structures</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Model Architecture</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Modifications to the Tree Network</td>
</tr>
<tr>
<td>3.4</td>
<td>Label generation and Training</td>
</tr>
<tr>
<td>-----</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>3.5</td>
<td>Results</td>
</tr>
<tr>
<td>3.6</td>
<td>Issues due to Lead bias</td>
</tr>
<tr>
<td>3.7</td>
<td>Improving quality of learned document structures</td>
</tr>
<tr>
<td>3.8</td>
<td>Evaluating the quality of implicitly learned Discourse Structures</td>
</tr>
<tr>
<td>3.9</td>
<td>Conclusion</td>
</tr>
<tr>
<td>4</td>
<td>Neural Approaches to Solving Combinatorial Explosion problem in Extractive Summarisation</td>
</tr>
<tr>
<td>4.1</td>
<td>Motivation: Optimality for ROUGE in Extractive Summarisation</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Sentence-Level vs Summary-Level Extractors</td>
</tr>
<tr>
<td>4.2</td>
<td>Hierarchical Summarisation Framework: Jointly Extracting sentences and Scoring Summaries</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Architecture</td>
</tr>
<tr>
<td>4.2.2.1</td>
<td>Sentence Representation</td>
</tr>
<tr>
<td>4.2.2.2</td>
<td>Document Representation</td>
</tr>
<tr>
<td>4.2.2.3</td>
<td>Sentence Extractor</td>
</tr>
<tr>
<td>4.2.2.4</td>
<td>Summary Ranker</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Training</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Data creation and Labelling</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Results</td>
</tr>
<tr>
<td>4.3</td>
<td>Adapting Summarisation to Episodic Reinforcement Learning Framework</td>
</tr>
<tr>
<td>4.3.1</td>
<td>The Episodic RL Framework</td>
</tr>
<tr>
<td>4.3.1.1</td>
<td>Deep Q networks</td>
</tr>
<tr>
<td>4.3.1.2</td>
<td>Policy Gradients</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Limitations of ROUGE in RL framework</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Adapting ROUGE to Reinforcement Learning Framework</td>
</tr>
<tr>
<td>4.3.3.1</td>
<td>New Reward Design $R^*$ based on ROUGE</td>
</tr>
<tr>
<td>4.3.3.2</td>
<td>Using $R^*$ with Deep Q learning</td>
</tr>
<tr>
<td>4.3.3.3</td>
<td>Using $R^*$ with Policy Gradients</td>
</tr>
<tr>
<td>4.3.3.4</td>
<td>Differences with similar works</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Model Architecture</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Choosing Trajectories</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Results</td>
</tr>
<tr>
<td>4.4</td>
<td>DNN based loss function for learning to sort summaries</td>
</tr>
<tr>
<td>4.4.1</td>
<td>SoDeep: A DNN based loss function for sorting</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Results</td>
</tr>
<tr>
<td>4.5</td>
<td>Conclusion</td>
</tr>
<tr>
<td>5</td>
<td>Issues specific to training Neural Extractive Summarisation Models</td>
</tr>
<tr>
<td>5.1</td>
<td>Dataset Overview</td>
</tr>
<tr>
<td>5.2</td>
<td>Issues with existing models</td>
</tr>
<tr>
<td>5.2.1</td>
<td>BertSum</td>
</tr>
<tr>
<td>5.2.2</td>
<td>MatchSum</td>
</tr>
<tr>
<td>5.3</td>
<td>Importance of training with variance</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Model details</td>
</tr>
</tbody>
</table>
CONTENTS

5.3.2 Evaluation .................................................... 52
5.4 Results and Observations .................................... 52
5.5 Experimental Details ........................................ 53
5.5.1 Models ...................................................... 53
5.5.2 Hyper-parameters ......................................... 53
5.6 Discussion ..................................................... 54
5.7 Conclusion ...................................................... 54

6 Conclusions and Future Work .................................. 56

Bibliography .......................................................... 60
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>7</td>
</tr>
<tr>
<td>2.1</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>14</td>
</tr>
<tr>
<td>2.3</td>
<td>16</td>
</tr>
<tr>
<td>2.4</td>
<td>17</td>
</tr>
<tr>
<td>2.5</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>31</td>
</tr>
<tr>
<td>4.2</td>
<td>31</td>
</tr>
<tr>
<td>4.3</td>
<td>34</td>
</tr>
<tr>
<td>4.4</td>
<td>36</td>
</tr>
<tr>
<td>FIGURE</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>4.5</td>
<td>The architecture of the model for Episodic Summarisation. The encoder is similar to the way Bert was used in BertSum. The token embeddings in the Transformer decoder are replaced with Sentence Vectors. Other functions remain the same. Note that the diagram provided is for Policy Gradient based training. In the DQN based version, the softmax layer is removed and the output of the product is used directly.</td>
</tr>
<tr>
<td>4.6</td>
<td>The architecture for using the SoDeep loss function. Here $\Theta_A$ represents the neural network used for ranking summaries. $y$ is the output scores of the network. These outputs are fed to another deep Neural Network $\Theta_B$ which predicts the ranks $r$ of the scores $y$. These ranks are then compared with the actual ranks $y^*$, which gives the loss in sorting. This loss is backpropogated through the entire network (both $\Theta_A$ and $\Theta_B$).</td>
</tr>
<tr>
<td>5.1</td>
<td>Here we show the average of ROUGE scores of summaries when sorted according to their rank on randomly selected 10,000 documents. Notice that the top 5 candidates are within a 0.02 margin of each other, which makes them qualitatively very hard to distinguish.</td>
</tr>
<tr>
<td>5.2</td>
<td>Here we show the number of actual top 20 candidate summaries that are among those that can be constructed from the top 5 sentences of BertSum on Test Set, as training progresses. We notice that the number is decreasing with increasing epochs. This could either mean 1) That the model is learning to predict the correct first summary with increased accuracy, while ignoring others, or 2) The model is overfitting the first summary, since that is the only summary it ever sees.</td>
</tr>
<tr>
<td>5.3</td>
<td>Here we show the BertSum’s bias for the first summary during training, which is not as prevalent on test set. Notice how the slope is lower on the testset. This could be due to the model not being able to distinguish as much between 1st, 2nd and 3rd summaries, since they are similar, while they are over-fitting the training data.</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Examples of Nucleus-Satellite RST relations. The Nucleus is considered more important than the Satellite.</td>
<td>8</td>
</tr>
<tr>
<td>1.2 Examples of Multi-Nuclear RST relations</td>
<td>8</td>
</tr>
<tr>
<td>2.1 CNN/DailyMail dataset statistics</td>
<td>12</td>
</tr>
<tr>
<td>3.1 The results of the NDS model and related works on the CNN/DailyMail dataset. NDS (bilinear) and NDS (MLP) represent the modified variants shown in equations (3.6) and (3.7) respectively. Note: Works that were not published at the time of this work are not included.</td>
<td>25</td>
</tr>
<tr>
<td>3.2 Results for the paragraph prediction accuracy. The variant of the NDS used uses the original TreeLSTM model with no modifications (NDS(original)). random is the result when predicting the result randomly.</td>
<td>26</td>
</tr>
<tr>
<td>4.1 An example for showing sentence level scores of individual sentences with the human-written summary for a given document. Only showing first 16 sentences out of 29.</td>
<td>28</td>
</tr>
<tr>
<td>4.2 Showing top 18 candidate summaries of length 2 and 3 for the document shown in Table 4.1. Notice that the 3rd best sentence occurs for the first time only in 9th best candidate and that the best summary is made of sentences with ranks 1, 13, 10.</td>
<td>28</td>
</tr>
<tr>
<td>4.3 Showcasing the computational complexity of the Summarisation problem. Notice that an increase in only 5 sentences from 20 to 25 almost doubles the amount of possible candidate summaries.</td>
<td>30</td>
</tr>
<tr>
<td>4.4 Results on the CNN/DailyMail dataset</td>
<td>35</td>
</tr>
<tr>
<td>4.5 This table shows an example for the way in which rewards are designed based on the ROUGE score. The first column denotes the summary being selected, the second column denotes the ROUGE score of the summary. The third column denotes the order in which sentences are selected in each timestep. The fourth column shows how individual rewards and cumulative rewards are computed.</td>
<td>40</td>
</tr>
<tr>
<td>4.6 Results on the CNN/DailyMail dataset. ES-DQN and ES-PG represent the models presented in this section, based on DQN and Policy Gradient respectively.</td>
<td>44</td>
</tr>
<tr>
<td>4.7 Results for training the MatchSum Network with SoDeep loss</td>
<td>46</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>5.1</td>
<td>This table shows the ability of the model to rank high quality summaries higher. In each column, the number at the top represents the average top k summaries per document was among the top 20 summaries selected by the model. The number in the bottom shows the percentage of documents which had at least one summary among the actual top k in its top 20 summaries.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In today’s age, the speed of technological advancements has become so rapid that "change is the new constant". In no other period of history have human societies changed and evolved so much like the last few decades. And a few decades is nothing compared to three hundred thousand years, the approximate period of time for which humans have lived on earth. Yet, the technological advancements made in the last few decades far outweigh all of those made in the last three hundred thousand years. Our computers have become intelligent enough to beat humans in games like Chess and Go, which were once considered to be the pinnacle of human intelligence, something the machines can never achieve. Furthermore, with the advancements in the last few years, machines are overtaking many areas that were previously labeled "humans only".

The credit to these technological advancements mainly goes to the breakthroughs in computing power, storage, and networking. The first hard drive to have more than 1 GB in capacity was the IBM 3380 in 1980. It was the size of a refrigerator, weighed 250 kg, and was priced from $81,000 to $142,400. Today, we could store Terabytes of data inside chips so small that they can fit inside our palms, and only for the price of tens of dollars. Our computing power has increased by over a trillion times in the last few centuries, while the chips which do the computing became more compact and energy efficient. Not to mention the improvements in networking which let us now transfer vast amounts of data in milliseconds, that Content-on-Demand and Streaming services have become more popular than DVDs and Blu-Rays were a decade ago.

With all these advancements also came information overload. Content generation and sharing have become easier than ever. Consider this: by September 2020, 1.3 million articles were being published on Medium (a blog-sharing website) every month. Three hundred hours of video is being uploaded to YouTube (a video-sharing website) every minute. And it is becoming more and more challenging to analyze the kind of content that is being uploaded and shared.

Any information that a person needs is now at their fingertips. Moreover, the rate at which we are consuming and creating information is increasing as each day passes. Due to this, information has become so much powerful and influential that it can be manipulated to win elections.
warfare just using social media websites. This alone calls for the need for understanding and analyzing the kind of information that is being created and spread.

Given the amount of content involved, manual means of evaluation can no longer keep up. However, the advances and breakthroughs in technology have also given us the powers and tools to make our machines do the analysis for us. Automatic Summarisation is one such tool to deal with this information overload. Advances in Summarisation can aid us in generalizing vast troves of information and shrinking them without loss of important content.

In this work, we mainly focus on Summarization of text documents. Summarization of text documents can be broadly divided into two categories: Extractive and Abstractive. In Extractive Summarization, the summary is constructed by extracting phrases, words or sentences from the document, while in Abstractive, the information contained in a document is paraphrased. Both approaches are different means to the same end: Reduction and Generalisation of content without loss of information. We specifically focus on Extractive Summarisation: this work describes different attempts at Extractive Summarisation from vastly different areas of AI and linguistics.

1.1 Extractive Summarisation

Extractive Summarization involves selecting a subset of content from a document while preserving essential information. But a document is not simply a set of words put together. Documents have structure and order, which enables us to interpret and understand them. And while understanding news articles is an effortless task for humans, the processes behind the scenes are far from simple. Machines have a difficult time understanding Natural Language, much less processing and understanding the information. In fact, we don’t really know how humans do it either. A human cannot explain the physiological processes that actually go on which enable understanding in concrete terms, much less replicate it in a machine. But that hasn’t stopped anyone from trying to attempt it.

Chomsky first showed the world that Grammar has innate rules and that language can be represented as a Formal System. Given certain axioms and rules for a language, we can generate all possible sentences that are correct according to that language. This can be represented by tree-like structures (e.g., Syntax trees). Later works like Rhetorical Structure Theory (RST) showed similar structural arrangements are present in documents. But unlike the structure of sentences, which are more clearly defined, the structure of documents is less rigid. This makes it challenging to approach the problem from a Classical Linguistics Perspective. Today’s machines, like BERT, have shown that they can capture the sentence structures from just data by simply filling in the blanks and learning if a sentence follows another. Maybe, they can also tackle the problem at the level of documents and maybe even facilitate and help human understanding.

From a theoretical perspective, Extractive Summarisation can be divided into multiple steps as follows:

From a theoretical perspective, Extractive Summarisation can be divided into multiple steps as follows:
1. **Document Understanding**: Involves understanding how sentences and paragraphs in the document are related to each other, generalizing what the document is about from the contents, title etc.

   (a) **Sentence Understanding**: Involves understanding the contents of a sentence, individually or in the context of the document.

   (b) **Understanding relationship between sentences**: Involves understanding how sentences are related to other sentences, how they support each other to present the information, and which of them have a independent role and which of them have a supporting role etc.

2. **Document Pruning** Involves removing content from the document which could be redundant, unimportant or inessential and in selecting sentences/phrases which are essential and have the capacity to represent the essence of a document.

The modern (or computational) Extractive Summarisation framework is very much dependent on this formulation, and they are realized using Deep Neural Networks. The steps involved in this can be divided into three sequential steps: Sentence Representation, Document Representation, and Document Pruning. (A Representation here means converting the textual information into a numerical format like vectors or matrices for processing). A typical organization of the framework is as follows:

1. **Sentence Representation**: Encoding information from word/sub-token distributed representations into one unit information representative of the sentence.

2. **Document Representation**: Computations that represent understanding relationships between sentences, and sometimes also involves generalizing the content of the document.

   (a) **Constructing context aware Sentence Representations**: This step aims to encode information into each sentence representations based on how they are related to other sentences.

   (b) **Constructing Document representation (optional)**: Encoding all the information in the sentences into one unit information representative of the document.

3. **Document Pruning**: This step constitutes the summarization phase of the framework, in which important sentences/phrases to be included in the summary are selected.

   (a) **Extracting Sentences/phrases**: Selecting some sentences/phrases representative of the document or those that contain important information etc. Other factors like redundancy, novelty can also be involved.

   (b) **Post-Processing (optional)**: Operations on the output of the previous step to improve the quality of the summary. These can be as simple as n-gram based methods to as complex as employing specialized neural networks.

We will look at these steps listed in detail below, and also try to show how the theoretical framework is realised in the computational framework.
1.2 Sentence Understanding

According to Wikipedia, "A sentence is the basic unit of language which expresses a complete thought." In English, a sentence is made up of words with strict constraints. For example, not every random bunch of words that can be strung together becomes a valid sentence of the language. There are various rules which humans implicitly use when constructing sentences that facilitate us to produce meaningful utterances. We will first look at the Sentence Structures and then at the way sentences are encoded and represented in Deep Neural Networks for Summarisation.

1.2.1 Structure of a Sentence

![Diagram showing Constituent and Dependency structures](image.png)

**Figure 1.1** Image showing the Constituent structure (a) and Dependency structure (b) of a simple sentence: "My vote is for Mexico". The parse trees were obtained using Stanford CoreNLP [42].

The structure of the words in a sentence has been studied under two famous categories: Syntactic/Constituency Structure [14] and Dependency Structure [24]. Both of these structures are represented with trees (constituency tree/dependency tree), and the means of obtaining these trees given a sentence is referred to as Parsing (constituency parsing/dependency parsing). An example image of both these structures for a simple sentence is shown in Figure: 1.1. A constituent parse shows the syntactical structure of a sentence, while a dependency parse shows the relations of the words in a sentence with each other. Generally, dependency trees are associated with representing semantic information. For example, the sentences "He ate an apple" and "An apple was eaten by him" have the same dependency tree but different constituency trees.

The **Constituency tree** is a representation of the sentence derivation using the formal system introduced by Chomsky [14]. This is shown in figure 1.1(a). Each node in the tree and its immediate
children represent the rules in the system. An example of these rules is shown in equation (1.1). NP
represents Noun Phrase, VP represents Verb Phrase, PRP$ represents personal noun with possession,
etc. Note that this structural representation does not show the relationship of words with each other. For
more details regarding syntax structures of sentences, we refer the reader to [14].

\[
\begin{align*}
S & \implies [NP, VP] \\
NP & \implies [PRP$, NN] \\
VP & \implies [VBZ, PP] \\
PRP$ & \implies [My]
\end{align*}
\] (1.1)

The Dependency tree, on the other hand, shows how different words are related to each other. For
example, in the same figure [11] (b), notice that this version of parsing shows how each of the words
modifies others, and this also shows subject-object relations, unlike the Constituency tree. The relations
\textit{nmod} : \textit{poss} mean Noun Modifier with Possession, \textit{nsubj} means nominal subject, etc. For more details
of the dependency structure, we refer the reader to [50].

An interesting difference between these two versions is that: in constituency trees, only the leaf
nodes contain the words of the sentence it represents. Other nodes represent abstract concepts on how
these nodes are combined. In the dependency tree, all the nodes in the tree are made of the words in the
sentence.

These structures are important because they can aid us a lot in various NLP tasks. For example,
we can formulate the problem of paraphrasing a sentence as ’Modifying or changing the Syntax tree
of a sentence without relationship changes in its corresponding dependency tree.’ Or we can formulate
the problem of Sentence Summarization by word deletions as: ’deleting sub-trees in the constituency
tree while upholding the following conditions: 1) the resulting constituent tree is a valid one and 2) no
relations in the dependency trees are changed. For example, when we say that a particular sentence is
valid, we are actually saying that this particular sentence can be derived from the Syntax rules of the
language. For more such formulations on how these theories can help, we refer the reader to [30].

1.2.2 Sentence Representation

In Deep learning networks for summarisation, a sentence is usually represented numerically by a
vector. This numerical representation is called ’sentence representation,’ This is usually computed
from the numerical representations of units that make up the sentence. These units can be words or
sometimes even character n-grams. Works like Word2Vec [44] and Glove [52] show how words can be
represented using vectors. FastText [8] is one example of a work that uses n-grams as the smallest units:
representation of a word is built by the average of each of the n-gram representations it contains.

A typical flow for obtaining sentence representations is as follows: A given sentence is split into
tokens. These tokens are then mapped to their corresponding numerical representations. This gives us a
two-dimensional matrix for each sentence. Let’s call this matrix \( W \). Optionally, additional information like positions can also be added here.

The next step is to convert this matrix \( W \) into the ‘sentence representation’ vector. This process can be divided into two abstract steps, as shown in equations (1.2) and (1.3).

\[
\text{Contextual Vectors} = \text{Encode}_\text{Context}(W) \quad (1.2)
\]

\[
\text{Sentence Representation} = \text{Encode}_\text{Sentence}(\text{Contextual Vectors}) \quad (1.3)
\]

Both the steps in equations (1.2) and (1.3) can be realised in many different ways. For example, they can be made of Convolutional Networks [32, 12], LSTM network [49], LSTM network with Attention layers [1], Transformer layers [64, 38] etc.

There are also tree-based networks, which can encode a sentence from word representations statically using pre-computed parse trees [61] or dynamically constructed trees [13].

1.3 Document Understanding

A document is made up of paragraphs, which in turn is made up of sentences. And much like sentences, documents also have an implicit structure. It is this structure that helps us in understanding the relationship between sentences. In this section, we will first look at document structure from a linguistics perspective and then look at how DNNs represent documents.

1.3.1 Structure of a Document

The structure of the document is defined by the relationships between the sentences it contains. This structure was first studied by Mann et al. [41] and was called ‘Rhetorical Structure Theory (RST).’ RST is intended to describe texts rather than describe the process involved in creating, reading, or understanding them. An example document structure based on sentence relations from RST is shown in figure 1.2. The relations need not just be between sentences but also between phrases in the sentences.

The RST relations between sentences can be divided into two categories: Nucleus-Satellite relation and Multi-Nuclear relations. In Nucleus-Satellite relations, the nucleus is more important than the satellite. In many cases, the satellite can be deleted without deleting the nucleus while maintaining the coherence of the document. But deleting a nucleus without deleting the satellite always makes the document incoherent. Some examples of Nucleus-Satellite relations are shown in table 1.1. In Multi-Nuclear relations, all the sentences in the relationship are equally important. Examples for Multi-Nuclear relations are shown in table 1.2.

RST can be directly optimized for Summarisation without any training [27]. Since RST denotes the relationship of sentences with each other and most of them are in a hierarchical relation (Nucleus-Satellite), finding the most important sentences that other sentences depend on becomes very easy. But RST by itself is not enough in most cases. For example, consider the same sentence written in two ways:
Figure 1.2 An example RST discourse Tree for a sample document. The example is from the dataset provided by Allen et al [2]. The original order of the sentences can be obtained by going top to bottom and left to right.

"He took a holiday because his mother was sick" and "His mother was sick, so he took a holiday." The first sentence has the relation "Non-Volitional Cause" while the second sentence can be considered to have the relation "Non-Volitional Result". In both cases, the cause being 'The mother being sick' and the result being 'The person taking a holiday'. Depending on how the text was interpreted, either of these could be Nucleus or Satellite. And identifying such cases is not easy. There are also other relations like 'Explanation', 'Cause' etc., which could be attributed to the same sentence. The authors of the RST themselves have shown and studied this issue [41].

But RST clearly captures the relationships among sentences in a document which facilitates document understanding. While these relations alone are not enough for Summarisation, they can aid greatly in summarisation [63, 68].

1.3.2 Document Representation

Documents are usually represented either by a vector or a matrix and in many works, the process of encoding words into a sentence representation resembles the process of encoding sentences into a document representation [12, 48].

Some works like MatchSum [72] ignore the concept of sentences altogether. They concatenate tokens of all the sentences in a document together and tries to encode them into a single vector. It can be said that they treat the entire document like a single sentence.

Many works do not explicitly construct a document representation [21, 1, 38, 39]. Some also call the context encoded sentence vectors as the document representation [73]. In this case, the document
<table>
<thead>
<tr>
<th>Relation Name</th>
<th>Nucleus</th>
<th>Satellite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antithesis</td>
<td>ideas favored by the author</td>
<td>ideas disfavored by the author</td>
</tr>
<tr>
<td>Background</td>
<td>text whose understanding is being facilitated</td>
<td>text for facilitating understanding</td>
</tr>
<tr>
<td>Justify</td>
<td>text</td>
<td>information supporting the writer’s right to</td>
</tr>
<tr>
<td></td>
<td>action or situation whose occurrence results</td>
<td>express the text</td>
</tr>
<tr>
<td></td>
<td>from the occurrence of the conditioning situation</td>
<td>conditioning situation</td>
</tr>
<tr>
<td>Evidence</td>
<td>a claim</td>
<td>information intended to increase the reader’s</td>
</tr>
<tr>
<td>Elaboration</td>
<td>basic information</td>
<td>belief in the claim</td>
</tr>
<tr>
<td>Otherwise</td>
<td>action or situation whose occurrence results</td>
<td>conditioning situation</td>
</tr>
<tr>
<td>(anti conditional)</td>
<td>action or situation whose occurrence results from the lack of occurrence of the conditioning situation</td>
<td>conditioning situation</td>
</tr>
<tr>
<td>Motivation</td>
<td>an action</td>
<td>information intended to increase the reader’s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>desire to perform the action</td>
</tr>
</tbody>
</table>

Table 1.1 Examples of Nucleus-Satellite RST relations. The Nucleus is considered more important than the Satellite.

<table>
<thead>
<tr>
<th>Relation Name</th>
<th>Nucleus</th>
<th>Other Nucleus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contrast</td>
<td>one alternate</td>
<td>the other alternate</td>
</tr>
<tr>
<td>List</td>
<td>an item</td>
<td>a next item</td>
</tr>
<tr>
<td>Sequence</td>
<td>an item</td>
<td>a next item</td>
</tr>
</tbody>
</table>

Table 1.2 Examples of Multi-Nuclear RST relations.

representation is a matrix of two dimensions. The formulations for obtaining context-aware sentence representation are similar to equation (1.2).

There are also works that construct context-aware sentence representations directly from token representations [38, 39]. The way this is done is explained in the next chapter, under section 2.5.4.

1.4 Document Pruning

This step represents the Summarization part of the framework. In this step, some sentences/phrases from the document are selected to represent the document as its summary. The factors considered for this step can include novelty, salience, amount of content, etc. Additional post-processing of selected sentences like Trigram-Blocking, Paraphrasing, Shortening are also done. The majority of the next chapter deals with this topic.
1.5 Challenges in Extractive Summarisation

Extractive Summarisation models can be broadly categorized into two categories: Sentence Extractors and Summary Extractors. Put simply, Sentence extractors are those which select a subset of sentences from the document (a set of sentences), while Summary Extractors are those which select a subset of sentences from among all possible subsets of sentences. This immediately shows the issue in trying to solve the problem of extractive summarization using Summary Extractors. The complexity is exponentially higher when we try to approach the problem using Summary Extractors compared to using Sentence Extractors. For example, for a document with 25 sentences, a Sentence Extractor should only make 25 decisions, while a Summary Extractor should ideally select one summary out of over 2000 candidate summaries.

But why should we approach the problem as Summary Extractors? Zhong et al. [72], introduced the concept of Pearl Summaries. Pearl Summaries are the best extractive summaries of a given document and are made up of sentences that have relatively low quality when considered individually (as compared to sentences in any other candidate summary). So in datasets that have a very high number of pearl summaries, Sentence Extractor based methods would perform poorly.

Given the complexity of the problem when there is a high concentration of pearl summaries, many works try to solve the problem using unique formulations. And so, the problem of Extractive Summarization has been formulated as Seq2Seq [12], Sequence Labelling [1 48 38 39], Reinforcement Learning [49], Multi-Arm Bandit [21], Learning to Rank [72], ILP [27] among other many more non-standard methods [29].

This problem is explored in more detail in Section 4.1. All of the works in this Thesis deal with datasets that have a very high concentration of Pearl Summaries.

1.6 Contributions of this Thesis

In this thesis, we first look at how Neural Summarization can aid in Document Understanding. Since Summarisation can be formulated as Document Understanding + Document Pruning, we formulate the problem in such a way that Summarisation can optimize for finding optimal Document Structure during Document Encoding. This should enable us to implicitly learn and take advantage of such document structures. We then look at different Neural Approaches to solving the Combinatorial Explosion problem introduced in the previous section. Finally, we look at issues in training Neural Extractive Summarisation models, try to reason why they occur, and show some simple experiments on how they can be solved. These contributions are summarized as follows:

- Propose a model which can learn implicit document structures indirectly by learning to Summarize, and show some proof for the structures learnt.

- Adopting Episodic Reinforcement Learning Framework (using Policy Gradients and Deep Q Networks) to Extractive Summarisation based on a new unique Reward formulation.
• A framework and a model for a 2-stage pipelined Extractive Summarisation model.

• Issues in existing Extractive Summarisation works due to poor problem formulation, restrictive training, and some experiments to show how the same can be solved.

1.7 Thesis Organisation

Chapter 1 (this chapter) introduces the field of Extractive Summarization, how classical linguistic theories provide a framework for summarization, a brief look at the challenges involved, and an overview of the contributions of this thesis.

Chapter 2 describes other Related works in the field of Extractive Summarization based on their problem formulation.

Chapter 3 describes the work on leveraging dynamically learned document structures for summarisation. Documents have an implicit discourse structure that facilitates human understanding. Typically linear models are used in neural summarization. This chapter describes attempts at modeling documents using non-linear structures and show how it can aid in summarisation. We also see if an end-to-end model trained for summarization can learn structures inherent in a document.

Chapter 4 describes different attempts in optimizing Neural Summarization models directly for the evaluation metric and dealing with the combinatorial-explosion problem that plagues the task. It describes three different approaches for solving the same problem.

Chapter 5 talks about issues that are less obvious in Neural Extractive Summarisation models due to poor problem formulation, training, etc. Here we show how two very different classes of models have learned hacks to summarize the documents with experiments and analysis.

Chapter 6 concludes the document. It summarises the contributions and presents an overview of possible future improvements.
Chapter 2

Related Work

2.1 Introduction

On the surface, Extractive Summarisation looks like a simple problem since parts of the document are directly copied to construct the summary. But that is not true. One of the reasons why Extractive Summarization is a complicated task is because, unlike Abstractive Summarization, there are no gold outputs for what to predict as a summary from the document. The objective of the task is formulated as extracting a summary representing a document that best matches the corresponding human-written summary. This makes the task non-trivial since the gold output labels for training Extractive Summarisation models should be prepared indirectly. Different works use different formulations for assigning labels to phrases/sentences that they want to extract, which often depends on their problem formulation. This makes the problem more open to various approaches and more freedom in solving the problem. Not to mention that Extractive Summarisation works also have to deal with the Combinatorial Explosion problem introduced briefly in section 1.5. Each work handles each of these issues differently. In this chapter, we will look at how different works approach Extractive Summarization, their problem formulations, their labeling methods, and their solutions.

2.2 Dataset

The work presented in this document mainly depends on the dataset introduced by Hermann et al. [26], popularly known as the CNN/DailyMail dataset. The dataset contains news articles and article highlights from two famous news networks, CNN and DailyMail. The dataset was originally released as a Question Answering dataset but was soon adopted as one of the standard datasets for Summarisation. The highlights of the News articles were considered to be the gold standard summary. The highlights were on average around 3-4 sentences, while the articles were on average around 25 sentences. Other statistics are given in table 2.1.

News articles, in general, have a structure. They start with a summarised version of the content in the first paragraph and slowly proceed to add more content. In fact, the first three sentences of the article,
Table 2.1 CNN/DailyMail dataset statistics

<table>
<thead>
<tr>
<th></th>
<th>CNN</th>
<th>DailyMail</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train</td>
<td>Valid</td>
</tr>
<tr>
<td>#months</td>
<td>95</td>
<td>1</td>
</tr>
<tr>
<td>#documents</td>
<td>90,266</td>
<td>1,220</td>
</tr>
<tr>
<td>avg #tokens</td>
<td>762</td>
<td>763</td>
</tr>
<tr>
<td>#vocab</td>
<td>118,497</td>
<td>208,045</td>
</tr>
</tbody>
</table>

called LEAD-3, is a very strong summarisation baseline for the task, and it took a good amount of time for Summarisation techniques to beat it.

2.3 Evaluation Metrics

The most common evaluation metric for the task is ROUGE proposed by Lin et al. [36]. Specifically, three versions of ROUGE are used for the task: F1 scores of ROUGE-1, ROUGE-2, and ROUGE-L.

ROUGE-n denotes the F1 score of n-gram overlap between the predicted and the gold summary.

ROUGE-L denotes the F1 score of the longest common sub-sequence between the predicted and the gold summary.

Although many works criticize ROUGE since it cannot handle semantic similarity and is a very simple metric based on lexical similarity, it is a decent metric for this task due to the nature of the dataset. Because the highlights were written based on the articles, there is a large amount of overlap in the content between the article and the highlights. This largely eliminates the problems of synonyms. But other problems like co-references remain. ROUGE cannot resolve co-references. But this is the most popular metric used by the works for this dataset. Some works do rely on human evaluation, but that is a metric that is very difficult to standardize.

2.4 Definitions and Notations

To maintain consistency in discussions, we define the following terms in this context:

- **Gold Summary/Abstract Summary/Human written summary**: This represents the highlights of news articles in the CNN/DailyMail dataset.

- **Sentence notations**: $s_i$ represents the $i^{th}$ sentence of a document $D$. Unless mentioned, it represents the numerical representation of the sentence. Sometimes, $e_i$ is used to represent context-encoded sentence representations, but they can be used interchangeably with $s_i$.

- **Candidate Summary**: Here they always represent a subset of sentences from the document. Any combination of sentences from a document can be a Candidate Summary. These are represented
by \( c_i \). Note that the \( i \) in \( c_i \) is only used for identification, unlike sentence representations \( s_i \), where \( i \) represents sentence position in the document.

- \( \theta \) represents the parameters of the model under discussion.
- **Timestep**: \( i^{th} \) timestep represents \( i^{th} \) step in a sequence of similar/repeated/recurrent computations. These computations usually happen on a sequence of inputs, and the \( i^{th} \) element of the sequence is the input to the \( i^{th} \) timestep.

### 2.5 Neural Approaches to Summarisation

![Diagram of Neural Extractive Summarisation](image)

**Figure 2.1** An arrangement of the Related works in the field of Extractive Summarisation based on their problem formulation. The examples included (from top-to-bottom, then left-to-right) are [72, 21, 73, 69, 12, 49, 29, 48]

Most of the works in Extractive Summarisation which use Neural Networks, can be classified based on the expressiveness of their model into two categories: 1) Sentence Ranking, 2) Summary Ranking. Here expressiveness denotes the number of decision points each model can give per document. For example, a model that predicts one value per sentence can make \( O(n) \) decisions for a given document (\( n \) being the length of the document). In contrast, a model which predicts a score per candidate summary can make \( O(2^n) \) decisions. (Note that this classification should not be confused with the concept of...
Sentence Extractors and Summary Extractors introduced in Section 1.5. More details in Section 4.1). The Sentence Ranking models can again be divided further on finer details, based on how they decide the score for each sentence in the document. An arrangement of related works in this field is shown in Figure 2.1. We will discuss each of these subdivisions in the following sections. Note that the explanations here are mainly concerned with the design of the models rather than the implementation details.

2.5.1 Sequential Decision Making

In this type of formulation, the decisions for including a sentence into the extracted summary are made sequentially. There is a temporal dependence between the decisions, and it can be said that the decisions are made in timesteps. The decision for the selection of a sentence depends on the decisions that were made in previous timesteps. This sequential-decision-making can again be of two types: Document order and Free order. In Document-order-based formulations, the decisions are made for each sentence in the order they occur in the document. Typically the decisions involve predicting yes/no kind of decision for each sentence. In Free-order-based formulations, the sentence selection resembles formulations used in text generation. In text-generation, a token is selected from all the tokens to be added to the prediction in each timestep. Similarly, here, a sentence is selected in each timestep to be added to the summary. These two formulations are discussed in detail, with works implementing them below.

2.5.1.1 Document Order

![Figure 2.2](image)

Figure 2.2 Showing typical framework for sequential selection of sentences based on document order. This diagram shows the decision making process given document and sentence representations. Example is shown for a document with 5 sentences.

Most works in this formulation use the Encoder-Decoder framework, where the representation of the document is first projected into an intermediate representation. This representation is usually considered as the condensed Document representation or the Abstract Summary representation. This is then used to initialize the decoder state, and the sentences are decoded sequentially. At each timestep, an Yes/No decision is made for each sentence. These decisions can then be trained with different formulations.
Cheng et al. [12] use a classification based approach. In their work, the Yes/No decision in Figure 2.2 is represented using a single value $p$, which represents the probability of selecting the corresponding sentence. Additionally, their function for predicting the $p$ value for each sentence is modelled on the Encoder states (using Attention mechanisms [4]) and the $p$ value of the previous sentence. To generate the extractive summary, they sort the sentences according to their $p$ values and select the top 3 as their predicted summary.

To generate the gold-standard labels needed to train their model, they considered different features for each sentence. Some of those features include uni-gram and bi-gram overlap of each sentence with the abstract summary, number of named entities, etc. For more details, we refer the reader to [12].

With the same Decoder based format shown in Figure 2.2, Narayan et al., in their work Refresh [49] formulate Extractive Summarisation as Reinforcement Learning (RL) [60] problem. Specifically, they use the policy-gradient algorithm REINFORCE [65] to train their model. In their formulation, the Yes/No decisions are represented by two values $p$ and $q$, respectively. This is to represent the actions available to the agent at each timestep, a common practice in RL. In simple terms, the probability of selecting a particular summary ($\hat{y}$) is the event of selecting Yes for the sentences occurring in that particular summary and No for other sentences. Let us denote those corresponding values for each sentence $s_i$ as $\hat{y}_i$. The product of all of these values represents the probability of selecting a particular summary. This probability is then weighed using the Reward $r(\hat{y})$ the agent obtained by selecting that summary. For training their model in this way without dealing with the Combinatorial Explosion problem, they pre-select the summaries that the model will be trained on (The typical practice in RL frameworks is to let the model explore and learn based on its own predictions and decisions [46]). The gradients of the loss function used to train their model are computed as shown in equation (2.1).

$$\nabla L(\theta) = -r(\hat{y}) \sum_{i=1}^{n} \nabla \log[p(\hat{y}_i|s_i, D, \theta)]$$

Finally, the predicted summary of their model is selected by sorting all of the sentences according to their $p$ values and selecting the top 3 sentences. We discuss this work in a little more detail in Section 4.3 of this document, where we discuss adopting more complex Reinforcement Learning formulations to Extractive Summarization.

Al-Sahabi et al. [11] also use a similar architecture with a unique twist. Their probability $p_i$ for inclusion of each sentence $s_i$ into the summary is dependent on the partial summary representation up-to that point. The partial summary representation $o_i$ for sentence $s_i$ is computed as shown in equation (2.2). Then their final probability score $p_i$ for the $i^{th}$ sentence is dependent on Document representation $d$, sentence representation $s_i$, and partial summary representation $o_i$.

$$o_i = \sum_{t=1}^{i-1} p_t * s_t$$

The work presented in Chapter 3 of this document also falls under this category.
2.5.1.2 Free order

In this formulation, instead of making a Yes/No decision in each timestep, a sentence is selected from among all the sentences in the document to be included in the summary. For comparison, a model based on document order will require $n$ steps, while a model based on random order will require only 3-4 steps (the length of the summary being selected).

An example of this formulation is the "Score and Select" model by Zhou et al. [73]. In their work, the sentence representations $\{s_i \mid s_i \epsilon D\}$, are encoded first using Bi-LSTM to create Document Level context vectors $e_i$. The decoder is initialized separately using a null vector. The hidden state $h_t$ of the decoder and the encoder states are used to select one sentence greedily at each timestep. This is shown in Figure 2.3. In the Figure 2.3 the sentences selected at $t = 0$ and $t = 1$ are $s_5$ and $s_1$ respectively. Scores for each of the sentences are computed as a function of $h_t$, and the Encoder states $e_i$. The sentence with the highest score is selected in each timestep.

The objective using which their model selected a sentence at each timestep was based on a greedy function. This greedy function tries to approximate the amount of ROUGE improvement that adding each sentence will bring to the partial summary. Their training data was also generated similarly. For a given document, their data consisted of multiple partially extracted summaries and the ROUGE improvement observed by adding each of the sentences in the document to each of those partial summaries. These improvements were normalized before training to account for differences between different documents.

Another work using the Free order formulation is by Yao et al [69]. They model the same problem as Deep Q learning (DQN) [46]. While their network training is based on DQN formulation, the way their network works is very similar to 'Score and Select'. The DQN formulations, along with this work,
is discussed more in detail in Section 4.3. The work presented in Section 4.3 of this document belongs to this category.

### 2.5.2 Parallel Decision Making

![Figure 2.4](image)

**Figure 2.4** In this formulation, a representation for each sentence $s_i$ is obtained for each sentence, and a corresponding score $p_i$ is predicted. The computations involved for all of these sentences are symmetric in nature. Optionally, a document representation can also be used in scoring sentences.

This is the most popular framework for this task. Many works [38, 39, 48, 21, 29] fall under this category. The typical formulation involved is to get a representation $s_i$ for each of the sentences and computing a score for them based on this representation. Sometimes, a document representation $d$ is also used in generating scores [48]. And the most common training objective for this kind of architecture is classification. The summary is then considered to be the top 3 sentences with the highest scores.

For training the model using classification objectives, gold labels for which sentences to select are required. Many works construct this using a greedy formulation first proposed by Nallapati et al. [48]. The method involves greedily adding sentences to the partial summary, which has the maximum ROUGE improvement. Examples of some works which use this labeling are [48, 38, 39, 29]. Liu et al. [38] also experimented with globally optimal summaries for a given document and found that using such labels gives slightly better results compared to the greedy method.

Jia et al. [29] take an interesting approach to classification by using non-binary labels for learning binary classification. They first order the summaries in the document according to their ROUGE scores, then assign normalized weights for each sentence based on the rank of the best summary each of them are a part of. These weights, along with binary labels for the best summary, are used together for training. They use Cross-Entropy for both sets of labels.

Dong and Shen et al. [21] formulate their training objective as a Contextual Bandit problem. In their framework, the document is considered as the context, and each action is represented by each unique ordered set of sentences. Contextual Bandit is a subset of the Markov Decision Process with episodes of length 1. They train their network using REINFORCE [65] algorithm, and their objective is similar to Refresh [49]. Their main contribution is their sampling method used in their training. During training,
they repeatedly sampled sentences using a $\epsilon$-greedy policy to select sentences to be included in the summary until a maximum predefined length.

### 2.5.3 Summary Ranking

In this type of formulation, a score is generated for each candidate summary rather than each sentence.

MatchSum [72] is one work that uses Summary Ranking. They used Siamese-Bert [55] architecture for the task. The score for each candidate summary $c_i$ is computed as the cosine similarity with the document representation $d$. To deal with the exponential number of possible candidate summaries, they used another state-of-the-art Summarisation model to shortlist their candidates. Specifically, they used BertSumExt [39] model for selecting five sentences for a given document and used these sentences to construct all possible candidate summaries of length 2 and 3, which lead to a total of 20 candidate summaries. Their model was trained and evaluated only on these 20 candidates for each document. Their training objective was based on Margin Ranking Loss, where the margin between two candidates $c_i$ and $c_j$ is proportional to the difference in the ranks between the candidates. The rank for each candidate was computed based on the ROUGE scores of the candidates with the abstract summary.

MatchSum completely ignores the concept of a sentence for getting both candidate and document representations. They appended all the sentences in the document (or candidate summary) together to get the document (or candidate summary) representations. Even [SEP] tokens weren’t used between sentences, a common practice to mention sentence endings in transformer-based language models [38].

### 2.5.4 Adopting Transformer Language Models for Summarisation

![Figure 2.5](image)

**Figure 2.5** Architectures of the original BERT model (left) and BERTSUM (right). Image taken from [39]

Transformer-based language models like Bert [19] were trained and used for sentence-level data. They are not designed for handling document-level information. Liu et al. [38] proposed a modification to Bert to encode document-level data. This modification is shown in Figure 2.5 In the Figure 2.5, the
sequence on top is the input document. This input is made of tokens from all sentences in the document, and each sentence preceded by a [CLS] token and followed by a [SEP] token. This document input is summed with three different kinds of embeddings for each token. The summed result is the input to the transformer model. The output tokens corresponding to each of the [CLS] tokens are considered to represent the sentence it precedes, and these are considered as the sentence representations $s_i$ for the $i^{th}$ sentence in the document. These sentence representations can then be used with any of the formulations given above.

2.6 Post processing

This section describes different post processing methods that could be used to improve the quality of the predicted summary.

2.6.1 Trigram Blocking

Trigram Blocking is one of the most common and most simple tricks used to improve the quality of the predicted summary. This method is only applicable to sentence ranking methods, where each sentence is assigned a score. The sentences are sorted into a list according to their predicted scores. The sentences with the highest score are then removed from this list and added to the predicted summary, provided there is no trigram overlap between words of sentences already in the predicted summary the sentence being added. If there is an overlap, the sentence is deleted. This is repeated until we have a summary of predefined length or we run out of sentences to add from the document.

2.6.2 Modifying sentences after Extraction

Some works use a two-stage approach. These stages are 1) sentences are first extracted from the given document to form the predicted summary 2) these sentences are modified either by paraphrasing or deletions to improve the quality of the summary. Usually, this is done because sentences in the human-written summaries are generally more concise and smaller than those in the document. These kinds of works can be divided into two types:

- **Extract-then-Rewrite** In this framework, sentences extracted are paraphrased. This is usually represented by two different networks, one for extraction and one for paraphrasing. These models can be considered to be Abstractive Summarizers more than Extractive. These two networks are often trained end-to-end using different formulations like Reinforcement Learning [11] or Latent variables [71] to allow for gradient flow between these two networks.

- **Extract-then-Compress** In this framework, extracted sentences are compressed by deletion. Some [67] use phrase-deletion based formulations similar to the formulations for Sentence Sum-
marisation introduced in Section 1.2.1. Some works [33, 43] use models similar to abstractive summarisation for compression.
Chapter 3

Implicitly learning document structures using Extractive Summarisation

Text documents are made of paragraphs, which in turn are made of sentences. But a paragraph is not simply a bunch of sentences put together randomly, and similarly, a document is not a bunch of random paragraphs. There are implicit relationships between the way sentences are structured and arranged. And it is due to this structure that when sentences are put together sequentially, they become meaningful and coherent. And the same can be said about the arrangement of paragraphs with respect to the document. This underlying structure, which most people don’t really notice and is usually taken for granted, is what makes a piece of textual information meaningful. This implicit document structure is studied under the Rhetorical Structure Theory (RST). For more details, see Section 1.3.1.

As seen in Section 1.3.1 these rhetorical relations (or document structure) can aid in summarisation. But if document structures can be used to aid in summarisation, then the opposite should also be true. Since document understanding heavily relies on this implicit structure, and summarization is a combination of Document-Understanding + Document-Compression, formulating the problem of summarisation appropriately should help us take advantage of this implicit structure. Moreover, if we use algorithms capable of learning and representing such structures, we should be able to some proof that we have indeed learned such structures from summarisation. That is the goal of the work presented in this chapter.

3.1 Related Work

Many works manipulate discourse structures in attempts to summarise documents, which span from ILP based formulations to using neural networks. Daume et al. [18] propose a compression method for documents that directly model the probability of summary given RST-DT using noisy channel models. Hirao et al. [27] rearrange RST-DT into a dependency tree and prune the tree for getting the summary by modeling it as an ILP problem. [31] also model summarization as tree compression. Cohan et al. [16] summarise scientific documents using a small subset of RST relations specific to the domain, like 'Hypothesis', 'method', 'results', 'implication', etc. However, none of these works mix a neural
approach with discourse-based approach for summarization. Works like [40] do use neural-discourse approach, but it was published much later than the period in which the work described in this chapter was done.

3.2 Tree Neural Networks

Tree Neural Networks are Recurrent Neural Networks that are capable of encoding tree-structured data. Unlike typical RNN architectures like LSTM, GRU, etc., which take one input per timestep, Tree Networks take two or more inputs and combine them into one representation. This step is repeatedly done until we have one final representation.

To realize our goal of learning implicit discourse structures using summarisation, we needed architectures capable of learning to construct trees. Many works like [61] and [74] proposed encoding tree data using recurrent neural networks, but they cannot learn to construct trees by themselves. Choi et al. [13] was the first work to propose an architecture that can learn task-specific tree structures. In our experiments, we use their network with appropriate modifications, which are detailed in later sections. They make their tree capable of learning by using Gumbell-Softmax estimator [28]. At each step, they construct all possible parents and use Gumbell-Softmax to select one parent greedily. Specifically, they greedily select which two nodes to combine at each timestep. For more details, we refer the reader to the paper [13] for details.

3.3 The Joint task: Learning to Summarise and implicitly identify Discourse Structures

3.3.1 Model Architecture

We construct the sentence representations using a CNN network, similar to [12] and [32], and add position embeddings to them. We encode these representations using the Tree Neural Network proposed by [13] to get a document representation. This document representation is used to initialize the decoder, which takes \( i^{th} \) sentence as input in the \( i^{th} \) timestep and outputs a vector corresponding to each sentence. This vector is then passed through an MLP to get the final score. The architecture for the same can be seen in figure 3.1. This architecture can be classified under document order models introduced in section 2.5.1.1.

3.3.2 Modifications to the Tree Network

To detail the modifications made to the Tree Network, we will first briefly look at how the tree network works. At any timestep, given adjacent vectors \( h_l \) and \( h_r \), the operations described in equations
Figure 3.1 The architecture diagram of the Summarisation model for learning implicit discourse structures. The tree is learnt dynamically by prioritising summarisation, which should aid us in implicitly modelling discourse structures present in the document.

Equations (3.1), (3.2), and (3.3) are performed to get parent representation $h_p$. Equation (3.1) is used to compute equivalents to gates in LSTM, (3.2) is used to compute cell memory, (3.3) gives the final output of this computation. These equations combined represent the Tree-LSTM.

\[
\begin{bmatrix}
i \\
f_l \\
f_r \\
o \\
g
\end{bmatrix} = \begin{bmatrix}
\sigma \\
\sigma \\
\sigma \\
\sigma \\
tanh
\end{bmatrix} \begin{bmatrix}
W_{comp} \\
h_l \\
h_r
\end{bmatrix} + b_{comp} \tag{3.1}
\]

\[c_p = f_l \odot c_l + f_r \odot c_r + i \odot g \tag{3.2}\]

\[h_p = o \odot tanh(c_p) \tag{3.3}\]

At every timestep, $n$ nodes are converted into $n - 1$ nodes (one pair of $h_l$ and $h_r$ are combined to make a parent $h_p$). The selection of which two nodes to combine is a function of $h_p$. Specifically, parents for all pairs are computed, and the function in (3.4) is used to score the parent vectors, $q$ is quality vector learned during training. The parent with the highest score takes the place of its children,
while all other vectors are simply copied as shown in (3.5). This recursive process is repeated as many times as the number of nodes.

\[
v_i = \frac{\exp(q_i h_t^{i+1})}{\sum_{j=1}^{M_{t+1}} \exp(q_i h_j^{t+1})} \tag{3.4}
\]

\[
h_{j+1}^{t+1} = \begin{cases} 
h_j^t & j < i \\
TreeLSTM(h_j^t, h_{j+1}^t) & j = i \\
h_{j+1}^t & j > i 
\end{cases} \tag{3.5}
\]

The modifications we make are to the parent scoring function are shown in (3.4). Since our decision to combine nodes depends on the relation between adjacent nodes at every timestep, we try to see if we can make this a function of the children themselves to try to model the quality of relations between them. We experiment with the following modifications: Using a bi-linear function of the children nodes (3.6), and using a Multi-Layer-Perceptron (MLP) as shown in (3.7).

\[
v_i = \frac{\exp((h_i^t)^T W (h_{i+1}^t))}{\sum_{j=1}^{M_{t+1}} \exp((h_j^t)^T W (h_{j+1}^t))} \tag{3.6}
\]

\[
v_i = \frac{\exp(MLP([h_i^t; h_{i+1}^t; h_{i+1}^{t+1}]))}{\sum_{j=1}^{M_{t+1}} \exp(MLP([h_j^t; h_{j+1}^t; h_{j+1}^{t+1}]))} \tag{3.7}
\]

### 3.4 Label generation and Training

Our model was a classification model, that predicts a probability \(p_i\) for selecting each sentence \(s_i\). To train this model, we need gold labels, which we have to prepare indirectly, as seen in section 2.1. For each document, we set top \(k\) sentences with the highest individual rouge scores with a label of 1 and the rest with 0. These labels were then used to train the model using cross-entropy loss. The top 3 sentences with the highest prediction probabilities are selected as the predicted summary of a given document during validation.

### 3.5 Results

The results are shown in table 3.1. Our models, NDS (Neural Discourse Summarizers) variants, perform very close to Cheng et al. [12] and Nallapati et al. [47] in Rouge 1, but lags in Rouge 2 and Rouge L. Of the different variants of NDS models, the bilinear performs better. Note that the bi-linear variant (equation (3.7)) can be made computationally more efficient since this scoring function does not involve parent representation computation for all nodes at each timestep.
### Table 3.1

<table>
<thead>
<tr>
<th>Model</th>
<th>Rouge 1</th>
<th>Rouge 2</th>
<th>Rouge L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lead-3</td>
<td>39.6</td>
<td>17.7</td>
<td>36.2</td>
</tr>
<tr>
<td>Cheng et al.</td>
<td>35.5</td>
<td>14.7</td>
<td>32.2</td>
</tr>
<tr>
<td>Nallapati et al.</td>
<td>35.4</td>
<td>13.3</td>
<td>32.6</td>
</tr>
<tr>
<td>NDS (original)</td>
<td>35.2</td>
<td>11.1</td>
<td>28.3</td>
</tr>
<tr>
<td>NDS (bilinear)</td>
<td>35.3</td>
<td>11.2</td>
<td>28.3</td>
</tr>
<tr>
<td>NDS (MLP)</td>
<td>35.1</td>
<td>10.9</td>
<td>27.6</td>
</tr>
</tbody>
</table>

The results of the NDS model and related works on the CNN/DailyMail dataset. NDS (bilinear) and NDS (MLP) represent the modified variants shown in equations (3.6) and (3.7) respectively. Note: Works that were not published at the time of this work are not included.

### 3.6 Issues due to Lead bias

In most cases, the trees learned tended to be primarily left-leaning trees, especially near the top. Basically, the tree network learned to converge to encoding the document in reverse. This technique of encoding the input in reverse has been shown to improve numbers in various tasks ranging from translation [58] to summarisation [49]. However, the reason this happens in this particular task could be due to lead bias. In fact, Lead-3 is a strong baseline in Summarisation, and many works struggle to beat it. In the next section, we see a step towards mitigating this issue.

### 3.7 Improving quality of learned document structures

To improve the learned document structures, we tried to see if we can include paragraph information into training somehow so that the trees constructed learn to encode the discourse relations between sentences. We initially tried to model this as a joint-learning problem with summarization by using teacher-forcing on parent scores shown in (3.4), but this proved to be difficult due to the nature of the algorithm. In the end, we decided to make this a multi-task learning where we learn to classify paragraphs and summarize documents. The model was alternated between both tasks during training. While this reduced the left-leaning tendency to a certain extent, it did not provide an improvement in summarisation results.

Note that if a tree is left-leaning, it will have 100% prediction accuracy on the last paragraph and 0% on the remaining. To not be limited by this issue, we took triplets of sentences, a pair of adjacent sentences, and a random sentence and asked the TreeLSTM model to score all the parent representations constructed out of them. We ensured that the correct pair of sentences are always together in the correct order, while the random sentence can either be the first or last sentence in the triplet. The model fine-tuned with paragraph information uses a similar format to train. On a held-out dataset, the results for the accuracy of paragraph prediction were computed are given in table 3.2. Notice that the results are better than random by a good amount, even without paragraph-based training.
3.8 Evaluating the quality of implicitly learned Discourse Structures

<table>
<thead>
<tr>
<th>Variant</th>
<th>Paragraph prediction accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>random</td>
<td>50.2%</td>
</tr>
<tr>
<td>NDS (without paragraph training)</td>
<td>62.1%</td>
</tr>
<tr>
<td>NDS (with paragraph training)</td>
<td>69.3%</td>
</tr>
</tbody>
</table>

Table 3.2 Results for the paragraph prediction accuracy. The variant of the NDS used uses the original TreeLSTM model with no modifications (NDS(original)). random is the result when predicting the result randomly.

As was shown by Choi et al. [13] in their own work, since the tree construction learning is unsupervised, the learned structure of the tree might not always resemble the tree structures that most theories use / that we are familiar with. One common reason could be that all human-made theories use the principle of Occum’s razor to a certain extent [7], but the things a network learns need not follow such constraints. Choi et al. show this in the context of sentence parsing. In our case, we should expect similar problems. Moreover, since the discourse structure of documents is much more loosely defined than sentence structure, it will not be easy to evaluate our model on the actual RST dataset. To overcome this, we simplify the evaluation to see if the network is capable of identifying paragraphs. Since paragraphs are self-contained structural units, we see if each sub-tree encodes a paragraph fully before encoding a sentence from neighboring paragraphs.

3.9 Conclusion

In this chapter, we looked at how an approach can be formulated to implicitly learn discourse structures with Extractive Summarisation by leveraging Document Understanding. We also showed some proof that Summarisation models are indeed capable of learning document structures to some extend.
Chapter 4

Neural Approaches to Solving Combinatorial Explosion problem in Extractive Summarisation

4.1 Motivation: Optimality for ROUGE in Extractive Summarisation

The problem of optimizing directly for Rouge [36] has been studied deeply in two works. Narayan et al.'s work, Refresh [49] shows the inability of Cross-Entropy to optimize for Rouge directly and propose an RL-based solution. However, their solution also faces similar issues due to their problem formulation and model design choices. Zhong et al. [72] show this problem in much more general terms. This section provides the motivation for the attempts described later in this chapter. Although some works described in this chapter were done before the publication of MatchSum, their analysis clearly articulates the motivation behind them.

4.1.1 Sentence-Level vs Summary-Level Extractors

In section 1.5, we briefly introduced the concept of Sentence Extractors and Summary Extractors, and in section 2.5, we mentioned that Sentence Ranking models should not be confused with sentence-level Extractors. Here, we will look at what they mean in detail.

Sentence level and Summary level extractors are abstractions of the ends of a continuous spectrum. In simple terms, this spectrum can be said to denote the complexity of the methods used for extractive summarization. For simplicity, assume the Sentence-Level extractor is the left end, and the Summary-Level extractor is the right end. As we go more and more towards the left, the decisions made for a sentence depend less and less on other sentences in a given document. And at the extreme left, there is no inter-dependence between the decisions made for each sentence. But as we go more towards the right, the decisions made for each sentence depend more and more on the decisions for other sentences. Furthermore, at the rightmost end, we have the perfect Summary level models, which do not make predictions for sentences but rather only for candidate summaries (which are a subset of sentences in the document). So, while MatchSum [72] can be considered a perfect Summary-Level extractor, no
The story of Anthony Stokes was supposed to have a happy ending. Instead it ended Tuesday, police say, with the teen heart transplant recipient. In 2013, the teen’s family told media that an Atlanta hospital rejected Anthony for a transplant. At the time, Mark Bell was acting as a Stokes family spokesman. Bell told CNN that a doctor told the family that Anthony’s low grades were the reason why he wasn’t a good candidate. "I guess he didn’t think Anthony was going to be a productive citizen." About a week after Stokes’ story made headlines, Children’s Healthcare of Atlanta. On Tuesday, Stokes carjacked someone at a mall, kicked in the door of the victim's home, drove away in a black SUV, she said. Police spotted the car and ran its plates which showed it had been stolen. Police chased the vehicle. Stokes lost control of the car, hit a pedestrian and then a pole, Holland said. Stokes died at a hospital, Holland said.

Table 4.1 An example for showing sentence level scores of individual sentences with the human-written summary for a given document. Only showing first 16 sentences out of 29.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Sentence</th>
<th>Score</th>
<th>Rank</th>
<th>Sentence</th>
<th>Score</th>
<th>Rank</th>
<th>Sentence</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(CNN)</td>
<td>0.4152</td>
<td>7</td>
<td>(2, 12, 16)</td>
<td>0.4020</td>
<td>13</td>
<td>(1, 2, 17)</td>
<td>0.3946</td>
</tr>
<tr>
<td>2</td>
<td>(1, 2, 16)</td>
<td>0.4129</td>
<td>8</td>
<td>(1, 2, 10)</td>
<td>0.4003</td>
<td>14</td>
<td>(1, 2, 12)</td>
<td>0.3933</td>
</tr>
<tr>
<td>3</td>
<td>(2, 13, 16)</td>
<td>0.4120</td>
<td>9</td>
<td>(3, 9, 13)</td>
<td>0.4002</td>
<td>15</td>
<td>(2, 8, 10)</td>
<td>0.3932</td>
</tr>
<tr>
<td>4</td>
<td>(2, 16, 17)</td>
<td>0.4078</td>
<td>10</td>
<td>(2, 10, 13)</td>
<td>0.3997</td>
<td>16</td>
<td>(2, 16, 21)</td>
<td>0.3932</td>
</tr>
<tr>
<td>5</td>
<td>(2, 8, 16)</td>
<td>0.4053</td>
<td>11</td>
<td>(2, 3, 16)</td>
<td>0.3992</td>
<td>17</td>
<td>(2, 16, 28)</td>
<td>0.3927</td>
</tr>
<tr>
<td>6</td>
<td>(2, 16)</td>
<td>0.4039</td>
<td>12</td>
<td>(2, 10, 17)</td>
<td>0.3958</td>
<td>18</td>
<td>(2, 4, 16)</td>
<td>0.3926</td>
</tr>
</tbody>
</table>

Table 4.2 Showing top 18 candidate summaries of length 2 and 3 for the document shown in Table 4.1. Notice that the 3rd best sentence occurs for the first time only in 9th best candidate and that the best summary is made of sentences with ranks 1, 13, 10.

The work discussed in this document is a perfect Sentence-Level extractor. Instead, depending on their architecture, their expressibility power, etc., they fall somewhere between both ends.

Consider the example shown in Table 4.1. If we were to apply a perfect Sentence-Level extractor model to this document, we would ideally select the sentences with the top 3 highest sentence scores. These would be 2nd, 9th and 3rd sentences. This candidate summary is the 44th best summary for this document, even when considering only candidate summaries of lengths 2 and 3. This immediately shows how bad the performance of Sentence Extractors would be for this document. Not only that, as shown in Table 4.2 the 2nd best sentence only occurs twice in the top 18 candidates, and the 3rd best sentence only occurs once. This shows that selecting either of these sentences is worse compared to...
selecting the 16th sentence ranked at 10th place, which occurs in all of the top 7 candidate summaries. This also shows that nothing can be said about the ROUGE scores [36] of candidate summaries based on the scores of the sentences that it consists of.

The reason why Sentence Ranking models (section 2.5) are not Sentence-Level extractors is because, in all of the Sentence Ranking models, the decisions made are inter-dependent. For example, this inter-dependence can be modelled during the Contextual-sentence-representation construction (in Parallel decision making methods (section 2.5.2)) or during decoding stages (in Sequential decision making methods (section 2.5.1)).

This is where the concept of Pearl Summaries comes in. For example, for the document in Table 4.1 Candidate summary (1, 2, 3) has a higher sentence level average score compared to the best candidate summary (2, 10, 16), yet the combined score of the best candidate is higher than the combined score of candidate (1, 2, 3). Hence, the best candidate summary for this document is a Pearl Summary. In CNN/DailyMail dataset [26], in over 81% of the documents, the best summary is a pearl summary. This calls for the need for more models more closer to the right end of the spectrum (Summary-Extractors).

This can also explain why simple methods like Trigram-Blocking work miracles on this dataset [38]. Applying Trigram-Blocking to a model’s outputs makes decisions similar to how they are made in document-order methods (section 2.5.1.1). Most works that employ this method are parallel-decision-making methods [38, 39, 29].

In this section, we will try to explore some solutions which are very much towards the right end of the spectrum (Summary Extractors). Specifically:

• **Section 4.2** talks about an approach that first constricts the number of candidate sentences for selection into summary and then to ranking the candidate summaries constructed out of them in a joint fashion.

• **Section 4.3** talks about adapting complex Reinforcement Learning algorithms like Deep Q learning and Policy Gradients to Extractive Summarisation. It also explains the challenges that arise in doing so and provides solutions for the same.

• **Section 4.4** talks about improving ranking capabilities of a Summary Ranking Network, MatchSum using deep-neural-network-based sorter loss functions from SoDeep [22].
4.2 Hierarchical Summarisation Framework: Jointly Extracting sentences and Scoring Summaries

4.2.1 Introduction

In the CNN/DailyMail dataset, the average length of a document is around 25 sentences. The average length of a summary is 3.4 sentences. In practice, the optimal length of predicted summaries is 2 or 3 since the sentences in the document are much longer than the sentences in the human-written summaries. The complexity of the problem as the length (number of sentences) of the document increases is shown in table 4.3. The average length of a document in the dataset is 25. Notice how the number of candidates doubles every five sentences when the document length is around 20. The problem gets even worse in longer documents.

<table>
<thead>
<tr>
<th>Document Length</th>
<th>Number of possible candidate summaries</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>66</td>
</tr>
<tr>
<td>10</td>
<td>165</td>
</tr>
<tr>
<td>20</td>
<td>1140</td>
</tr>
<tr>
<td>25</td>
<td>2300</td>
</tr>
<tr>
<td>30</td>
<td>4495</td>
</tr>
</tbody>
</table>

Table 4.3 Showcasing the computational complexity of the Summarisation problem. Notice that an increase in only 5 sentences from 20 to 25 almost doubles the amount of possible candidate summaries.

The reason this complexity is an issue is due to the nature of ROUGE [36]. It is simply not possible to make any conclusions about the ROUGE scores of combinations of sentences from the scores of individual sentences. This was shown in the previous section (section 4.1). Hence, one of the only ways to optimize for ROUGE directly is by learning to rank candidate summaries. But this is computationally very expensive and often infeasible for longer documents.

One simple solution to this problem is to prune the number of sentences in the document from which candidate summaries are constructed. On average, 19% of the sentences in a document have a zero ROUGE score with the gold summary. Adding these sentences to the summary is always detrimental since they don’t bring any value. Our goal is to have a model to prune useless sentences and construct and rank candidate summaries from the remaining sentences. This framework is shown in figure 4.1.
4.2.2 Architecture

An overview of the model architecture is shown in Figure 4.2. This architecture represents the pipelined model introduced in Figure 4.1. The way this pipeline is realised in this architecture is explained below.

![Architecture Diagram]

**Figure 4.2** Architecture for jointly learning to extract sentences and score summaries.

### 4.2.2.1 Sentence Representation

Glove [52] was used to initialize word embeddings. The word embeddings of each sentence were then passed on to a Bi-Lstm layer to get both forward and backward encoded context vectors for each word. For each word, the forward and backward vectors were combined to get one single vector. This
is shown in equation (4.1), where \( w_i \) represents the \( i^{th} \) word in the sentence, \( h^f_i \) and \( h^b_i \) represent the forward and backward context encoded vectors \( cw_i \) corresponding to \( w_i \).

\[
\begin{align*}
h^f_i &= LstmCell(h^f_{i-1}, w_i) \\
h^b_i &= LstmCell(h^b_{i+1}, w_i) \\
cw_i &= [h^f_i, h^b_i]
\end{align*}
\tag{4.1}
\]

These context encoded vectors \( (cw_i) \) were then passed to an attention layer, which combined them into a single sentence vector \( s \). This is shown in equation (4.2). This process is repeated for each of the sentences in the document.

\[
\begin{align*}
q_i &= v^T \ast cw_i \\
a_i &= e^{q_i} / \left( \sum_{j=1}^{n} e^{q_j} \right) \\
s &= \sum_{j=1}^{n} a_i \ast cw_i
\end{align*}
\tag{4.2}
\]

4.2.2.2 Document Representation

The sentence vectors were encoded into a single document vector using the LSTM network. The sentences were encoded in reverse since news articles tend to have more important information in the beginning / first paragraph. This approach is similar to the document representation approaches by \[49, 12\], and was first introduced by \[59\]. The hidden state vector obtained at the final timestep of the LSTM network is considered as the document representation \( d \), as shown in equation (4.3). This is shown as the ‘abstract document representation’ in Figure 4.2. This document representation should mainly encode content relevant to the summary.

\[
d = LSTM([s_n, ..., s_1])
\tag{4.3}
\]

4.2.2.3 Sentence Extractor

The document representation is used to score the sentences individually. A bilinear layer was used, which took two vectors as input, document representation \( d \) and \( i^{th} \) sentence representation \( s_i \) as input. This value is passed through sigmoid activation to get a score \( p_i \) for each sentence, as shown in equation (4.4). The sentences are then sorted according to their score, and top \( k \) sentences are selected for constructing candidate summaries. Empirically, \( k = 10 \) performed better than other \( k \) values.
\[ p_i = Sigmoid(d^T * W_a * s_i) \]  
(4.4)

4.2.2.4 Summary Ranker

The sentences extracted by the Sentence Extractor in the previous step are combined to create all possible length two and length three summaries. Let us call the \( i^{th} \) such candidate summary \( cs_i \), and the representation of the \( j^{th} \) sentence that make up the candidate summary \( cs_i \) as \( s^i_j \). These \( s^i_j \)'s are the sentence representation obtained in equation (4.2), and are arranged in the order they occur in the document.

Each of these candidate summaries is encoded using a Summary Encoder. This Summary Encoder is implemented using an LSTM network. The final hidden state vector of the Summary Encoder is considered as the Candidate Summary Representation. This is shown in equation (4.5).

\[ cs_i = LSTM([s^i_1, ..., s^i_k]) \]  
(4.5)

Each of the candidate summary representation \( cs_i \) is then concatenated with the document representation \( d \) and passed through a Multi-layered-perceptron to get a score \( score_{cs_i} \). The candidate with the highest score is considered to be the summary of the document.

\[ score_{cs_i} = MLP([d; cs_i]) \]  
(4.6)

4.2.3 Training

Because the scores \( score_{cs_i} \) obtained by Summary Ranker (4.6) is not dependent directly on the score \( p_i \) generated by Sentence Extractors (4.4), it is difficult to train the Sentence Extractor and Summary Ranking parts of the network together. The step where we select top \( k \) sentences based on \( p_i \) values is non-differentiable, so the network cannot be trained end-to-end. This requires us to use some innovative methods to train our model.

It is also difficult to account for all the possible sentences that could make up the top \( k \) sentences as training happens. And since the possible candidate summaries depend directly on these \( k \) sentences, generating ROUGE scores to train for all possible combinations is simply not possible.

To solve this, we use teacher training, similar to the idea used in Seq2Seq models for Natural Language Generation [59]. We train the Sentence Extractor and Summary Ranker separately like they are two different heads of the same network. At each iteration, the Sentence Extraction network is trained using sentence labels which are 1 for \( k \) pre-selected sentences and 0 for the rest. The Summary Ranker is trained using all possible candidate summaries constructed from the \( k \) pre-selected sentences. This loss computation for method is shown in figure 4.3. This loss is then used to train the model using Gradient
Figure 4.3 This graph shows how the different parts of the network are trained. Notice that Sentence Extractor and Summary Ranker do not interact during training.

Descent. Since the Sentence Extractor usually has a very high F1 score in correctly selecting sentences (0.71), this should not have a huge impact on performance or train-test input distribution difference.

4.2.4 Data creation and Labelling

The gold labels for Sentence Extractor are selected greedily using ROUGE scores of sentences. The sentences with zero overlap with human-written summary always have a label of 0. Among the remaining sentences, sentences are ranked according to their ROUGE score, and the top k sentences get a label of 1, while the remaining get 0. Note that if the number of sentences is less than k after removing sentences with zero overlaps, all of them are assigned a label of 1. These labels are then used to train the Sentence Extractor model using Binary-Cross Entropy loss.

For Summary Ranker, all possible candidate summaries are constructed from the gold extracted sentences, and a ROUGE score for each of them with the human-written summary is obtained. This part of the network is then trained using Regression to predict the ROUGE scores. We found that normalizing the ROUGE score and the output of the model to have zero mean and by standard deviation
helped improve results. The mean and std normalization are detached from the gradient computation graph and are not part of the final loss.

4.2.5 Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Rouge 1</th>
<th>Rouge 2</th>
<th>Rouge L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cheng and Lapata [2016]</td>
<td>35.5</td>
<td>14.7</td>
<td>32.2</td>
</tr>
<tr>
<td>SummaRunner</td>
<td>39.6</td>
<td>16.2</td>
<td>35.3</td>
</tr>
<tr>
<td>Refresh</td>
<td>40.0</td>
<td>18.2</td>
<td>36.6</td>
</tr>
<tr>
<td>Ours</td>
<td>38.3</td>
<td>14.7</td>
<td>35</td>
</tr>
</tbody>
</table>

Table 4.4 Results on the CNN/DailyMail dataset

The results of the model are shown in table 4.4. Our model performs poorly compared to most of the models shown. Related Work which wasn’t published at the time of this work are not shown here.

In this section, we saw a pipelined approach to Extractive Summarisation which has the potential to deal with the combinatorial explosion problem (section 1.5) that we have to face when we use a Summary Ranking approach. We also saw a unique training formulation for training different parts of the pipeline together using teacher training. While our model’s architecture is inspired by the most optimal design for Summary Extractors, it is still not able to match the performance of other models.

4.3 Adapting Summarisation to Episodic Reinforcement Learning Framework

Some works have framed the problem of Extractive Summarization in an episodic framework [73, 69]. In this framework, a sentence is selected in each timestep from all the sentences in the document. The sentence selected at each timestep is dependent on the sentences selected in previous timesteps and the document. This formulation is the same as the Free order formulation introduced in section 2.5.1.2. But since the ROUGE scores of the summaries are non-differentiable and non-divisible (section 4.1), it often makes it difficult to train models using this framework, and hence the lesser scale of adaptation. And one important paradigm which deals with non-differentiable rewards is reinforcement learning.

There are also other reasons why Extractive Summarisation could benefit from being framed as an RL problem. One of the main strengths of RL algorithms is their ability to learn by exploring complex environments. And this exploration can be used for dealing with the combinatorial explosion problem in this task and also directly optimize for ROUGE [49].
But because ROUGE is non-divisible for a given candidate summary, it is difficult to say how rewards should be assigned to the model in each timestep. One work that tries to solve this problem is Yao et al. [69]. But their models don’t produce good results. In this section, we will introduce Reinforcement Learning concepts, see how they can be used in Summarisation, limitations of the work by Yao et al. [69] and Zhou et al. [73] (which use the same free order formulation but a regression-based training objective) and also show how we can improve upon them.

4.3.1 The Episodic RL Framework

![Figure 4.4 Typical setting of an Episodic RL problem](image)

In the episodic setting, there is an agent $K$ and an environment $E$, and the environment is in state $S_t$ at time $t$. In every timestep, the agent has to choose an action $a_t$, where $\{a_t \in A\}$, where $A$ denotes the set of all actions available to the agent. A pictorial representation for the same is shown in 4.4. The objective of the task is to choose actions such that the cumulative reward when reaching a terminal state (a state from which the agent can no longer take action) is reached. In this work, we are only interested in timestep-based terminal state: the model is said to have reached the terminal state after taking a pre-defined finite number of actions.

Some notations and definitions required are:

- **Policy** $\pi$ denotes how an agent makes decisions. $\pi(s, a)$ denotes the probabilities choosing action $a$ in a given state $s$.

- $Q(s_t, a_t)$ denotes the Q value of taking a particular action from a given state. This represents the expected cumulative sum of rewards that can be obtained from state $s_t$ from time $t$ by choosing action $a_t$ at time $t$.

- $V(s_t)$ denotes the value function of a given state $s_t$. Value function denotes the cumulative sum of rewards that can be obtained from state $s_t$ from time $t$. Note that value function can be written using policy $\pi$ and $Q(s, a)$ values.
Different works involve different adaptations of reinforcement learning for extractive summarization. Banditsum [21] adapts the formulation of Contextual Bandit for Extractive Summarization. In a Contextual Bandit, a context is sampled from all contexts and shown to the agent in each trial. The agent, in each such trial, selects an action based on the context that was sampled. The agent is expected to learn which set of actions give good rewards based on the context shown. Contextual Bandits can be considered a subset of the Markov Decision Process (MDP), where each episode is length one. Each document is considered a context in their work, and each subset of sentences a different action.

Similarly, Narayan et al. also adapt RL for extractive summarization in their work Refresh [49]. They formulate the problem as a policy learning problem, where they try to optimize the probabilities $p(y_i|s_i, D, \theta)$ for each sentence $s_i$ for the sentences that occur in high-quality summaries.

If BanditSum can be considered a MDP of length 1, our formulation can be considered a MDP of length 3. In our formulation, the environment is a single document, the state $s_t$ an agent is in at timestep $t$ is denoted by the partial summary $ps_t$ selected until that timestep. From this setting, we approach the problem using two different formulations. One is Deep Q Networks, similar to Yao et al. [69], another is using REINFORCE [65] algorithm. These are explained in the following sections.

One thing to keep in mind is that, in Summarisation, no matter how the problem is framed, the environment is always deterministic and not stochastic. Choosing the same action from the same state will always give the same reward, and reach the same destination state.

### 4.3.1.1 Deep Q networks

Deep Q networks [46] are neural networks which learn to predict the value of $Q(s_t, a_t)$, given $s_t$ and $a_t$. These fall under the class of function approximators [60]. Function approximators are an approximate estimate of either the $Q$ function or the value function $V$, which are helpful in cases where there are either a large number of states or in the case of contiguous states. In many cases, the state representations themselves can be used to model $Q$ values by generalizing for different states.

DQNs are often solved by using Value Iteration method, where a value of a state $s_t$ is learnt using the immediate reward $r_t$ and value of the next state $s_{t+1}$ obtained by taking action $a_t$ at $s_t$ at time $t$ as shown in (4.7). The $max$ in (4.7) can also be replaced with Expection over actions, especially in the case of non-deterministic environments.

$$Q(s_t, a_t) = R(s_t, a_t) + \gamma \times max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \quad (4.7)$$

The network is trained using the rewards obtained $R(s_t, a_t)$ at each timestep. The loss is designed around the difference between the $Q$ values of the current state $s_t$ and the next state $s_{t+1}$ as shown in (4.8). A common loss function used in this setting is HuberLoss, but common regression losses like MSE can also be used.

37
4.3.1.2 Policy Gradients

Policy gradient methods are a type of reinforcement learning technique that relies upon optimizing parametrized policies with respect to the expected return (long-term cumulative reward) using gradient descent. In essence, the objective is to maximize the expected reward obtained when following a policy as shown in equation (4.9), $\theta$ represents the parameters, $J(\theta)$ represents the expected reward when following policy from parameters $\theta$ and $r(\tau)$ represents the cumulative reward obtained when following a trajectory $\tau$. A trajectory represents the probability of a sequence of decisions and states visited by an agent in an environment. The parameters are updated as shown in (4.10), where $\alpha$ denotes the learning rate, which when expanded is as shown in (4.11). $a_t$ and $s_t$ denote the state and action at time $t$. For more details, we refer the reader to [60, 34].

$$J(\theta) = \mathbb{E}[r(\tau)]$$  \hspace{1cm} (4.9)

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta)$$  \hspace{1cm} (4.10)

$$\nabla \mathbb{E}[r(\tau)] = \mathbb{E}_\pi [r(\tau)(\sum_{t=1}^{T}\nabla \log \pi_\theta(a_t|s_t))]$$  \hspace{1cm} (4.11)

Policy gradient algorithms like REINFORCE [65] have an objective to maximize the expected reward when following a policy. REFRESH [49] also uses the policy gradient algorithm but their formulation is slightly different. In their case, the agent goes through each sentence from the document as they occur, and the action the agent takes at each timestep is whether to include the sentence in the summary or not. In equation (4.11), $a_t$ denotes yes/no decision, while $s_t$ denotes decisions upto sentence $s_t$. This imposes an architectural limit on the model, since the model has to decide for some sentences, before even looking at others. Our formulation is slightly different, as in our cases, the actions are represented by the sentences themselves, and in equation (4.11), $a_t$ denotes a sentence, and $s_t$ denotes the partial summary selected until $t$ steps.

A common improvement to the equation (4.11) is to use the cumulative sum of rewards obtained from each state, which is used to weigh the decision taken at timestep $t$ as shown in equation (4.12). $G_t$ is the cumulative sum of rewards obtained from time $t$, as shown in (4.13), where $\gamma$ is the discounted factor that is used to discount future reward. An intuitive reason why equation (4.12) is better than equation (4.11) can be clearly seen in case of positive rewards. The decisions made in earlier timesteps are always higher in equation (4.12) compared to later timesteps, since $G_t$ decreases with increase in $t$. 

$$HuberLoss(R(s_t, a_t), \gamma * max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$  \hspace{1cm} (4.8)
\[ \nabla \mathbb{E}[r(\tau)] = \mathbb{E}_{\pi_{\theta}} \left[ \sum_{t=1}^{T} G_t \nabla \log \pi_{\theta}(a_t|s_t) \right] \]  
(4.12)

\[ G_t = \sum_{i=t}^{T} \gamma^{t-i} * r(i) \]  
(4.13)

While equation (4.12) has shown to be better in performance compared to (4.11) [34], since it weighs each decision differently, it is not possible to apply this version directly to Extractive Summarisation. This is due to the nature of the Rouge scores, which are indivisible in nature, as explained in the next section.

### 4.3.2 Limitations of ROUGE in RL framework

Rouge scores are not additive. A sentence with a higher individual Rouge score does not imply that it is a part of a summary with a higher Rouge score. For example, consider a document \( d \) and sentences \( \{s_i|s_i \in d\} \). Let \( S_k \) denote a subset of sentences of document \( d \) and \( R(S_k) \) denote the ROUGE score of \( S_k \) with respect to the document \( d \). Then, nothing can be said about the scores of summaries from the scores of sentences as shown in (4.14). This is also the direct implication of pearl summaries studied by Zhong et al. [72].

\[ R(\{s_i\}) > R(\{s_j\}) \rightarrow R(\{s_i\} \cup S_k) > R(\{s_j\} \cup S_k) \]  
(4.14)

Rouge scores are only summary level and not sentence level. If used directly, this means that the reward is only known at the end of the trajectory, and there is no preferential order for selecting sentences. But such a design can be improved by having sentence-level rewards for the following reasons:

- **Can help in making better decisions early on.** Selecting better sentences early is better than selecting bad sentences early. Here a better sentence could mean either a sentence that occurs in lots of good summaries or a sentence that occurs in the best summary etc. Since the sentences selected later in an episodic framework is dependent on the sentences selected earlier, selecting good ones earlier is more important.

- **Helps in resolving ambiguity.** If there are no sentence-level rewards, it is difficult to say which sentence out the sentences in the best summary that we should select at any given timestep. Works like [73] use Rouge improvement to decide which sentence should be selected when, but as already shown by [72] and also (4.14), that is suboptimal.
4.3.3 Adapting ROUGE to Reinforcement Learning Framework

Since ROUGE is not directly usable for summarisation, we look for some indirect means. In this section, we will first look at a new reward formulation $R^*$, which gives the reward obtained based on the sentences included till time $t$. Then we will look at how this fits into Deep Q Learning and Policy Gradient algorithms.

### 4.3.3.1 New Reward Design $R^*$ based on ROUGE

<table>
<thead>
<tr>
<th>Summary</th>
<th>R( )</th>
<th>Selection Order</th>
<th>Cumulative Reward for the summary selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>${s_1, s_2, s_3}$</td>
<td>40</td>
<td>s.1, s.3, s.2</td>
<td>$R^<em>(s_1) + R^</em>(s_1, s_2) + R^*(s_1, s_2, s_3) = 42 + 42 + 40 = 124$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s.3, s.2, s.1</td>
<td>$R^<em>(s_3) + R^</em>(s_3, s_2) + R^*(s_1, s_2, s_3) = 40 + 40 + 40 = 120$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s.2, s.3, s.1</td>
<td>$R^<em>(s_2) + R^</em>(s_2, s_3) + R^*(s_1, s_2, s_3) = 42 + 42 + 40 = 124$</td>
</tr>
<tr>
<td>${s_1, s_3, s_4}$</td>
<td>39</td>
<td>s.4, s.3, s.1</td>
<td>$R^<em>(s_1) + R^</em>(s_3, s_4) + R^*(s_1, s_3, s_4) = 42 + 39 + 39 = 120$</td>
</tr>
<tr>
<td>${s_1, s_2, s_4}$</td>
<td>42</td>
<td>s.2, s.1, s.4</td>
<td>$R^<em>(s_2) + R^</em>(s_1, s_2) + R^*(s_1, s_2, s_4) = 42 + 42 + 42 = 126$</td>
</tr>
<tr>
<td>${s_2, s_3, s_4}$</td>
<td>28</td>
<td>s.2, s.4, s.3</td>
<td>$R^<em>(s_2) + R^</em>(s_2, s_4) + R^*(s_2, s_3, s_4) = 42 + 42 + 28 = 112$</td>
</tr>
</tbody>
</table>

Table 4.5 This table shows an example for the way in which rewards are designed based on the ROUGE score. The first column denotes the summary being selected, the second column denotes the ROUGE score of the summary. The third column denotes the order in which sentences are selected in each timestep, The fourth column shows how individual rewards and cumulative rewards are computed.

For adapting ROUGE to episodic framework the ROUGE scores of summaries are manipulated as described below. Let $R^*(S_j)$ denote the new reward design which are obtained after manipulation as shown in equation (4.15).

$$R^*(S_j) = Max_{S_k} R(S_k)$$ where $S_j \subset S_k$ and $|S_k| = 3$ (4.15)

The reward at each timestep is defined as the ROUGE score of the best possible summary of length 3 that can be constructed from the set of sentences $S_t$ that are already selected till time $t$ as shown in (4.15). The cumulative reward is then the sum of $R^*(S_t)$ of the set of selected sentences at each timestep $t$. This is shown in the example in table 4.5 for a document $d$ with four sentences $\{s_i | i \in I, i \geq 1 \text{ and } i \leq 4\}$. The selection order column denotes the order in which the sentences were selected in each timestep.

From table 4.5 we can see how selecting $s_3$ first is penalised more compared to selecting $s_1$ or $s_2$. $s_3$ does not occur in as many high scoring summaries as $s_1$ and $s_2$, and hence, choosing it early can lead to many cascading bad choices compared to choosing $s_1$ and $s_3$. 
4.3.3.2 Using $R^*$ with Deep Q learning

In the DQN setting, we try to predict the $Q$ value of the subset of summaries selected till time $t$. We consider all possible $Q$ values of all actions given a state, which represents which sentence to include into the partial summary already selected, and select the sentence with the highest $Q$ value. Formally, given a subset of $t$ sentences already selected to be included in the summary as $S_t$, we choose the next sentence $s_i$ to be included into $S_t$, based on the $Q$ values. This is shown in equation (4.16).

$$Q(S_t, s_i) = R^*(S_t \cup \{s_i\}) + \max_{s_j} Q(S_t \cup \{s_i, s_j\})$$  \hspace{1cm} (4.16)

In practice, since the $\max$ operation is computationally expensive, we only train on preselected trajectories. This means we already know $s_j$ and only have to compute the $Q$ value for one sentence. The problem is then formulated as a regression problem, where the network is asked to correctly predict the difference between the two $Q$ values as shown in (4.17). Since this loss depends on difference between $Q$ values in two consecutive timesteps, the $Q$ value $Q(S_1)$ at $t = n$ is trained directly as shown in equation (4.18). Note that $S_1$ is simply null set, $S_t$ is the partial summary at $i^{th}$ timestep, and $s_i$ is the $i^{th}$ sentence to be included into the partial summary.

$$Loss_{t<n} = HuberLoss(R^*(S_t \cup \{s_i\}), Q(S_t, s_t) - Q(S_{t+1}, s_{t+1}))$$  \hspace{1cm} (4.17)

$$Loss_{t=n} = HuberLoss(R^*(S_n), Q(S_n, s_n))$$  \hspace{1cm} (4.18)

4.3.3.3 Using $R^*$ with Policy Gradients

In the policy gradient setting, we use the REINFORCE algorithm directly as shown in equation (4.12). We compute $G_t$ as shown in equation (4.19), where $S_t$ denotes the subset that was selected until time $t$. The probability $\pi(a_t|S_t)$ is replaced with $\pi(s_i|S_t)$ where $s_i \epsilon d$. The final loss function obtained is given in equation (4.20), where $S_t$ represents the partial summary selected till time $t$, $s_t$ denotes the sentence selected at time $t$. The $E_{\pi_{\theta}}$ part is removed since the rewards are deterministic and not probabilistic, the result does not change by adding it.

$$G_t = R^*(S_t) + R^*(S_{t+1}) + ...$$  \hspace{1cm} (4.19)

$$Loss = -(\Sigma_{t=1}^{T} G_t \nabla log\pi_{\theta}(s_t|S_t))$$  \hspace{1cm} (4.20)

For demonstrating how this would work in practice, we take an example from table 4.5. Consider the summary $(s_1, s_3, s_2)$ selected in the same order. i.e., at $t = 1$, $s_1$ was selected, at $t = 2$, $s_3$ was...
selected, and \( t = 3, s_2 \). Then the value \( G_1 \) would be \( 42 + 42 + 40 = 124 \), \( G_2 \) would be \( 42 + 40 \), and \( G_3 \) would be 40.

### 4.3.3.4 Differences with similar works

**Deep Q Learning** One other work which uses DQN formulation is by Yao et al [69]. Their problem formulation is similar to ours but their reward design is different, and is shown in equation (4.12).

\[
    r_t = \begin{cases} 
        R(pst_t) - R(pst_{t-1}) & \text{if } R(pst_t) - R(pst_{t-1}) > \text{threshold} \\
        -1 & \text{else}
    \end{cases} 
\]

(4.21)

Here \( R(pst_t) \) represents the ROUGE value of the partial summary \( pst_t \) selected until time \( t \). The main limitation with this method is that it does not differentiate between different orders of selection of sentences for a particular summary during training since the final cumulative reward for selecting a summary is the rouge score of that summary. Also, since their training is exploration-based, and they start with a random policy, it could be that their model had difficulty in convergence, as noted by Narayan et al. [49].

**Policy Gradients** There are two important works in the field which use Reinforce formulation introduced in (4.11). We already saw why the formulation we use (equation (4.12)) is better in section 4.3.1.2.

### 4.3.4 Model Architecture

For training, we use a slightly modified version of Transformer Encoder-Decoder architecture [64]. For Encoder we use Bert [19] with similar modifications to [38]. Sentences are parsed using Bert Tokenizer to get sentence tokens. We then append all the sentence tokens together, each sentence preceded by a [CLS] token and followed by [SEP] token, as shown in figure 4.5. Bert gives an output vector for each of the input tokens. The outputs corresponding to the [CLS] tokens are considered to be the representations of the sentence immediately following it. Other tokens at the final layer of Bert are ignored. Unlike typical Transformer Encoder-Decoder architecture, where all the output vectors of the encoder layer are fed to the decoder, only the outputs of [CLS] tokens are used by the decoder. For more details regarding the encoder, we refer the reader to BertSum [38].

For Summary Construction, we use a slightly modified version of Transformer Decoder, which selects a sentence at each time-step to be included in the summary. The architecture is shown in the figure 4.5. At each Time-step, the decoder considers the partial-summary \( S_t \) (set of sentence vectors already selected), the Sentence Vectors from the Encoder, to output a vector. This vector is then multiplied with all the sentence vectors to get a score for each sentence. These scores are used to select the sentence \( s_i \) to be included in the partial summary. For DQN based loss, these scores are directly fed into loss.
The architecture of the model for Episodic Summarisation. The encoder is similar to the way Bert was used in BertSum. The token embeddings in the Transformer decoder are replaced with Sentence Vectors. Other functions remain the same. Note that the diagram provided is for Policy Gradient based training. In the DQN based version, the softmax layer is removed and the output of the product is used directly.

Figure 4.5

function since the network is trained using Regression. These values represent the $Q(S_t, s_i)$ in equation (4.17). For REINFORCE loss, the softmax layer is used to compute probabilities for each sentence. These probabilities represent the probability $\pi(s_i|S_t)$ in equation (4.20). These loss functions are then used to train the model using Gradient Descent.

4.3.5 Choosing Trajectories

Since ROUGE is computationally complex to compute at run-time, and given the exponential number of possible summaries, it will not be suitable to start training with a random policy. So we shortlist the summaries beforehand, which will serve as trajectories during training. A similar approach was taken by Narayan et al. in their work REFRESH [49]. We consider the average of F1 scores of Rouge-1, Rouge-2, and Rouge-L to be the score of each summary. We also normalize this average rouge scores of all candidate summaries per document to be between -1 and 1.

We choose the top 15 summaries for each document, and from the next 85 top summaries, we randomly sample 15 summaries. These 30 summaries are used for training the model. During training, for each document in a mini-batch, a summary is sampled from these 30 summaries and is used for training.
We trained the model until convergence saving checkpoints every 1000 steps, with a batch size of 16. The best model is chosen based on the validation set results and is then evaluated on the Test set. These results on the test set are shown in table 4.6.

<table>
<thead>
<tr>
<th>Model</th>
<th>Rouge 1</th>
<th>Rouge 2</th>
<th>Rouge L</th>
</tr>
</thead>
<tbody>
<tr>
<td>SummaRunner</td>
<td>39.6</td>
<td>16.2</td>
<td>35.3</td>
</tr>
<tr>
<td>Refresh</td>
<td>40.0</td>
<td>18.2</td>
<td>36.6</td>
</tr>
<tr>
<td>BanditSum</td>
<td>41.5</td>
<td>18.7</td>
<td>37.6</td>
</tr>
<tr>
<td>Yao et al</td>
<td>39.4</td>
<td>16.1</td>
<td>35.6</td>
</tr>
<tr>
<td>BertSum</td>
<td>43.25</td>
<td>20.24</td>
<td>39.63</td>
</tr>
<tr>
<td>ES-DQN (ours)</td>
<td>42.7</td>
<td>19.8</td>
<td>38.5</td>
</tr>
<tr>
<td>ES-PG (ours)</td>
<td>42.1</td>
<td>19.2</td>
<td>37.9</td>
</tr>
</tbody>
</table>

*Table 4.6* Results on the CNN/DailyMail dataset. ES-DQN and ES-PG represent the models presented in this section, based on DQN and Policy Gradient respectively.

### 4.3.6 Results

The results for our model are shown in table 4.6. The models ES-DQN and ES-PG represent models trained with Deep Q Learning (section 4.3.3.2) and Policy Gradients (section 4.3.3.3) respectively. As we can see, our model performs better than all of the reinforcement learning works like REFRESH [49], BanditSum [21], and Yao et al. [69]. The results of our model are close to BertSum [38] without trigram-blocking, but with trigram-blocking enabled, our model lags behind BertSum.

In this subsection, we saw a novel method to adapt complex RL algorithms to Extractive Summarization by designing a new reward function to overcome the limitations of ROUGE.

### 4.4 DNN based loss function for learning to sort summaries

As already seen in previous sections, summarization is better modeled as summary ranking instead of sentence ranking, and this was the main idea behind the work by Zhong et al. [72]. They rank summaries using Margin Ranking Loss. In this section, we try to see if we can improve the sorting capabilities of the network by using approximating the sorting loss function using a deep neural network proposed by [22].

#### 4.4.1 SoDeep: A DNN based loss function for sorting

The SoDeep architecture is a deep neural network trained to sort numbers. The model is trained by using randomly generated data. It has shown improvements in various tasks [17, 37, 23], where
Figure 4.6 The architecture for using the SoDeep loss function. Here $\Theta_A$ represents the neural network used for ranking summaries. $y$ is the output scores of the network. These outputs are fed to another deep Neural Network $\Theta_B$ which predicts the ranks $r$ of the scores $y$. These ranks are then compared with the actual ranks $y^*$, which gives the loss in sorting. This loss is backpropagated through the entire network (both $\Theta_A$ and $\Theta_B$).

the objective functions are non-differentiable. Some examples of non-differentiable objective functions are Spearman's correlation $[20]$, Recall@k, Mean Average Precision etc. These objective functions are usually used when the task is similar to ranking/sorting to measure the quality of the outputs. Given that the objective function of Extractive Summarization is also non-differentiable, and the State-of-the-Art network is a sorting-based network, approximating the sorting loss using Margin Ranking Loss, we investigate if there can be benefits obtained by using SoDeep as the loss function.

Given a set of numbers, the SoDeep network predicts the integer rank of these numbers. Let the output of the Summarisation network be $y$, which is the scores for multiple candidate summaries. When fed to the sorter network, it predicts the ranks $r$ of the predicted scores. These ranks are then compared with the actual ranks of summaries $r^*$ using Mean Absolute Error (MAE) $e$. By backpropagating through the SoDeep network, we obtain the gradients of $y$ with respect to the MAE computed as $\partial e \partial y$. These intermediate gradients are equivalent to the $\partial \text{loss} / \partial \text{prediction}$, the gradient of the prediction with respect to the loss function. These gradients are then backpropagated through the sorting network. The SoDeep network can either be kept fixed or fine-tuned during training.

The SoDeep network has very high accuracy (close to 99.9%) in predicting ranks. So the cascading effect of the error in rank prediction affecting the Summary network should be negligible.

In our experiments, we used the LSTM based sorter proposed by the author. For implementation details, we refer the reader to the paper $[22]$ and their code $[1]$. 

45
Table 4.7 Results for training the MatchSum Network with SoDeep loss

<table>
<thead>
<tr>
<th>Variant</th>
<th>Rouge 1</th>
<th>Rouge 2</th>
<th>Rouge L</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatchSum (original)</td>
<td>44.2</td>
<td>20.6</td>
<td>40.4</td>
</tr>
<tr>
<td>MatchSum (SoDeep)</td>
<td>43.92</td>
<td>20.47</td>
<td>40.1</td>
</tr>
</tbody>
</table>

4.4.2 Results

For Summary Ranking, we use MatchSum [72] with no modifications to the network itself. For more details regarding architecture, training etc, we refer the reader to their work [72]. Our contribution is in upgrading the loss function using SoDeep, and adapting the formulations for the same. The results for the same are given in table 4.7.

4.5 Conclusion

In this section, we saw in detail the combinatorial explosion problem in the extractive summarisation task, and how we can design models using various problem formulations. Specifically, we saw

- The complexity of the problem of extractive summarisation and why it is not simply sentence ranking. We also explored the concept of Sentence level and Summary level extractors in detail and how it affects different works.

- How we can tackle the exponential increase of candidate summaries with the increase in number sentences using a pipelined model, and how we can train different parts of the pipeline using teacher forcing.

- How we can use complex Reinforcement Learning formulations to train Extractive Summarisation models. We also introduced a new reward formulation that was able to distinguish between different orders of summary construction when constructing it sentence by sentence. We also saw how and why our problem formulation is better compared to other similar works in the field.

- Some experiments in optimising sorting using deep neural network based loss functions for Summary ranking models.

1https://github.com/technicolor-research/sodeep
Chapter 5

Issues specific to training Neural Extractive Summarisation Models

Extractive Summarization has been modeled as the problem of selecting sentences that have the maximum ROUGE score. This involves selecting a subset of sentences that best represent the document as its summary from among other subsets. ROUGE scores are not additive, and hence simply selecting the best sentences alone is not enough. We also need to check the compatibility of these sentences with each other. This makes this problem of summarization different from a sentence ranking problem. (section 4.1)

One issue with training a network directly for selecting a subset from among subsets (as opposed to selecting sentences from among all sentences) is the combinatorial explosion problem. As the number of sentences increases, the number of possible candidate summaries increases exponentially. For example, a document of length 25 can have as many as 2300 possible candidate summaries, only considering those made of 2 and 3 sentences. (section 1.5 and section 4.2.1).

As already seen in chapter 2, many popular works try to model Extractive Summarization by avoiding this combinatorial explosion problem altogether. Many model the problem as Sequence to Sequence [38, 12], Sequence Labelling [48, 1], Reinforcement Learning [21, 49] and Representation Learning [72]. However, except for Representation Learning, all of them use sentence ranking to construct the predicted summary. Post-processing methods like Tri-gram blocking also help a lot whenever sentence ranking is involved.

One main issue with sentence ranking approaches is that they cannot optimize for ROUGE directly. Narayan et al., in their work Refresh [49] show this for models trained using classification objectives. They state that there is a discrepancy between objectives during training and testing time when the model is asked to minimize errors on binary labels. 1) The model is trained to rank summaries during train time, 2) but evaluated using rouge during test time, which is not at all similar to classification. The difference between points 1 and 2 is the discrepancy. But Refresh itself is not free from this limitation since it again uses sentence ranking to select its predicted summary (section 2.5.1.1).

Zhong et al. [72], on the other hand, optimize for rouge directly since they try to rank summaries according to their rouge scores. But then, they are facing the problem of combinatorial explosion head-
on. To avoid this problem, they used another previous state-of-the-art model, BertSum, [38] to shortlist candidate summaries to train their model on. (section 2.5.3)

In almost all cases, the number of possible summaries shown to the model during training is very less than the actual number of possible summaries. Moreover, since most of these summaries are very similar to each other (as shown in section 5.1), such shortsighted training can lead to many issues with the model. In this work, we explore how optimizing only for a small set of summaries can sabotage a model and verify the same with quantitative experiments on some recent works.

Specifically, our contribution in this chapter is two fold:

- We showcase the bias in two different kinds of seminal works in Extractive Summarisation quantitatively, and postulate reasons behind it and also show some evidence.
- We demonstrate the importance of training models using multiple summaries.

5.1 Dataset Overview

Here we take a look at different statistics of the CNN/DailyMail dataset, which will help us in making observations in later sections.

Figure 5.1 Here we show the average of ROUGE scores of summaries when sorted according to their rank on randomly selected 10,000 documents. Notice that the top 5 candidates are within a 0.02 margin of each other, which makes them qualitatively very hard to distinguish.

- The average number of unique sentences in top 10 summaries is around 8.4, while the average in top 20 summaries is 12.6
• Each document contains many possible candidate summaries. The distinction between the neighbouring summaries are not really meaningful, at least from ROUGE’s perspective, since all of them have very similar scores. We show this in Figure 5.1. We can see that the top 5 candidates are within a 0.02 margin of each other, which makes it qualitatively very hard to distinguish between them.

• On average, each document contains around 25 sentences, which brings the average number of possible candidate summaries to 2600.

5.2 Issues with existing models

In this section, we analyze two seminal works in the field of Extractive Summarisation. The two works that we consider are 1) BertSum [38] (representing works which generate one label per sentence), 2) MatchSum [72] (representing summary extractors, where the model summarises a document by ranking candidate summaries). We are mainly interested in the caveats that occur due to the limited scope of data shown to the models during training.

5.2.1 BertSum

Model Overview BertSum [38] is a Bert model [19] fine-tuned for Extractive Summarisation. It is a sentence ranking model, providing a score for each sentence, and the top 3 sentences (after trigram-blocking) are considered as summary. For details, we refer the reader to [38].

Issue 1. We notice that the candidate summaries generated by the model are the best at just after 1000 training steps, after which the quality of those candidates slowly deteriorates. We show this in Figure 5.2. This figure shows the average number of candidates which can be constructed from the top 5 sentences predicted by the model on the test set. Each unit on x-axis consists of 1000 training steps. At the end of every 1000 steps, we plot the model’s quality of candidates. Notice how the average number of candidates keeps dropping as training progresses. This is one issue that can greatly affect MatchSum [72] since they use the candidates generated by BertSum. Had they selected a savepoint that wasn’t the best in predicting the best summary and went with a model which had not even finished its warmup training steps (warmup steps = 10000), they might have had better candidates.

The gradual decrease in the quality of candidates can be due to either of the following reasons: 1) The model is learning the nuances of predicting the best summary, 2) The model is overfitting the training data. We examine what the model is doing next.

Issue 2. Since the model is trained using only one summary, and since there is not much difference between the top summaries (as noted in section 3), we suspect that the model should overfit the best summary on training data, while should not be able to distinguish as much between the top summaries
Figure 5.2 Here we show the number of actual top 20 candidate summaries that are among those that can be constructed from the top 5 sentences of BertSum on Test Set, as training progresses. We notice that the number is decreasing with increasing epochs. This could either mean 1) That the model is learning to predict the correct first summary with increased accuracy, while ignoring others, or 2) The model is overfitting the first summary, since that is the only summary it ever sees.

on test data. We show this in Figure 5.3 The difference in accuracy in predicting the first summary between train data and test data is almost 10%, 0.7% for the second summary, and almost negligible from the third.

We postulate that this discrepancy arises between the train and test data because the model is expected to do a lot with a limited architecture. By showing the model only the best summary during training, we expect the model to ideally also learn to select the best summary on the test set. However, since there is not much difference between the first best summary, the second, and the third, even qualitatively w.r.t to the evaluation metric, we cannot expect the model alone to learn it. This leads to overloading, where the model should not only determine which combination of sentences to select as a good summary but also which combinations to avoid, without us actually teaching the model how to do so. Due to this, we believe the model struggles during inference to single one out of them over others.

5.2.2 MatchSum

Model Overview MatchSum [72] is a Siamese-Bert Model [55]. The model is directly trained to rank summaries by constructing representation for each of them. To solve the problem of Combinatorial Explosion, MatchSum [72] uses only candidate summaries constructed from the top 5 sentences predicted by BertSum [38]. For more details, we refer the reader to [72]
Here we show the BertSum’s bias for the first summary during training, which is not as prevalent on test set. Notice how the slope is lower on the test set. This could be due to the model not being able to distinguish as much between 1st, 2nd and 3rd summaries, since they are similar, while they are over-fitting the training data.

**Issue 1.** From Section 3, we can see that the top 20 summaries of a document on average contain around 12 sentences, which is way higher than the 5 in the 20 candidate summaries used by MatchSum. Not only that, MatchSum has also never looked at a bad summary during training. And since all the candidates are very similar, we suspected that there is little reason for MatchSum to actually learn to match text using their semantics. We postulate that MatchSum is learning some other features that help distinguish better summaries from good ones since semantics alone might not be the best way to distinguish between them. We suspect that, due to the way the model is trained, this model should fail on candidates which have more variance.

We show this in table 5.1 where the model is asked to predict the ranks of 120 candidate summaries on the test set. We can immediately see that the model performs even worse than BertSum, whose predictions were used to train this model. This shows that the model has actually not learned to match the semantics of the content it encodes but rather even more nuanced features.

### 5.3 Importance of training with variance

This section demonstrates the limitations of training with limited scope (either using one/small set of summaries) using a simple experiment. We train a simple summary ranking model and compare it with the existing state-of-the-art models on how well they are able to rank summaries. We explain the details of the model briefly below.
5.3.1 Model details

We modify the architecture of BertSum [38] to allow for summary ranking. We use the same input representation as BertSum, and get the representation for each sentence. Let the representation for the $i^{th}$ be $s_i$. To get summary representations, we use an LSTM based Summary encoder, which encodes multiple sentence representations in the order they occur in the document. i.e., if the candidate summary is $(s_a, s_b, s_c)$, and $(a < b < c)$, then $s_a$, $s_b$, $s_c$ are the inputs to the summary encoder at 1$^{st}$, 2$^{nd}$ and 3$^{rd}$ timestep respectively. The final hidden state of the summary encoder is considered as the representation. We consider the representation of the document to be the summary encoder’s final state for all the sentences in the document. The cosine distance between the summary representation and the document representation is considered the score of the summary. We train our network as a ranking network using Margin-ranking-loss, similar to MatchSum [72].

We construct all possible summaries of lengths 2 and 3 for each document and sample 50 summaries for each document during training. Based on how we sample these summaries, we trained two models: 1) **Un-Randomized SR** for which we sample summaries at every step and 2) **Randomized SR** which uses a static set of summaries to train (candidate summaries are sampled before training starts). MatchSum uses a static list of summaries to train, and we can see the effects of such training impacting the ability to rank clearly in table 5.1.

5.3.2 Evaluation

To evaluate, we take the top 20 summaries predicted by each model and compare them with the actual top k summaries of the document for various values of k. Specifically, we compute the precision with which each model is able to predict the top 20 summaries correctly. We show this result in table 5.1. We also show the percentage of documents in the test set which had at least one summary out of the actual k summaries in their top 20 predictions. We score the summaries using the Perl script provided by Lin et al. [36].

5.4 Results and Observations

In Table 5.1 we show the qualitative results. We can see immediately that MatchSum [72] performs worse than BertSum [38], on whose candidates it was trained on. We suspect that this is due to the fact that MatchSum is trained only on very similar candidate summaries. Since the content of the candidates were similar, it is highly possible that MatchSum learned much more nuanced features to distinguish between them, which wasn’t semantic-based. If so, it can explain why MatchSum would perform badly when tested on candidates with a lot of variance in their content. For comparison, MatchSum’s 20
Table 5.1 This table shows the ability of the model to rank high quality summaries higher. In each column, the number at the top represents the average top k summaries per document was among the top 20 summaries selected by the model. The number in the bottom shows the percentage of documents which had at least one summary among the actual top k in its top 20 summaries.

<table>
<thead>
<tr>
<th></th>
<th>Top 1</th>
<th>Top 3</th>
<th>Top 5</th>
<th>Top 10</th>
<th>Top 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>BertSum</td>
<td>0.15</td>
<td>0.37</td>
<td>0.54</td>
<td>0.89</td>
<td>1.42</td>
</tr>
<tr>
<td></td>
<td>(15%)</td>
<td>(28%)</td>
<td>(36%)</td>
<td>(46%)</td>
<td>(57%)</td>
</tr>
<tr>
<td>MatchSum</td>
<td>0.07</td>
<td>0.17</td>
<td>0.27</td>
<td>0.49</td>
<td>0.86</td>
</tr>
<tr>
<td></td>
<td>(7%)</td>
<td>(14%)</td>
<td>(20%)</td>
<td>(29%)</td>
<td>(41%)</td>
</tr>
<tr>
<td>Un-Randomized SR</td>
<td>0.13</td>
<td>0.3</td>
<td>0.43</td>
<td>0.72</td>
<td>1.17</td>
</tr>
<tr>
<td></td>
<td>(13%)</td>
<td>(23%)</td>
<td>(29%)</td>
<td>(38%)</td>
<td>(49%)</td>
</tr>
<tr>
<td>Randomized SR</td>
<td><strong>0.18</strong></td>
<td><strong>0.44</strong></td>
<td><strong>0.65</strong></td>
<td><strong>1.07</strong></td>
<td><strong>1.71</strong></td>
</tr>
<tr>
<td></td>
<td>(18%)</td>
<td>(31%)</td>
<td>(38%)</td>
<td>(48%)</td>
<td>(60%)</td>
</tr>
</tbody>
</table>

candidates only contained on 5 sentences per document, while our 165 candidates contained on average around 16 sentences.

We also note the difference between the variants of our proposed model. We can see that the Un-Randomized model performs poorly compared to the Randomized model. The act of randomizing the candidates during training made it difficult for the model to overfit the training data, which is a problem we showed with BertSum in section 5.2.1.

5.5 Experimental Details

5.5.1 Models

For BertSum, we trained our own model following the implementation details from [38], and we produce similar results. For MatchSum, we use the model provided by [72] on their github repository [1]. We use Bert based models for all cases. All of our experiments were done using PyTorch [51] and Transformers Library [66].

5.5.2 Hyper-parameters

We experimented with multiple hyper-parameter setting for our models and found the below to work best on the validation set. Most of them are the default for training Bert. Our learning rate function, dropouts and regularisation are the same as [38]. We save the model every 1000 training steps, and use validation set to select the best model. Our effective batch size was 32 for all models, and whenever we weren’t able to fit 32 in a batch, we used gradient accumulation every 2 steps with batch size of 16,

---

1[https://github.com/maszhongming/MatchSum]
effectively same as batchsize 32. For all models (except MatchSum), we trained using 4 Nvidia GTX 1080 Ti and 120 GB of RAM.

5.6 Discussion

Due to the nature of ROUGE in extractive summarization, we say that the selection of a particular sentence should depend on the selection of other sentences. However, formulating an objective based on this makes the problem circular. For example, consider a document $d$ of only two sentences: we are saying that the selection of sentence $s_2$ should depend on the selection of sentence $s_1$, while also saying that selection of sentence $s_1$ should depend on the selection of sentence $s_2$. This is shown in equation (5.1).

\[
p(s_1) = \text{func}(s_1, p(s_2)) \\
p(s_2) = \text{func}(s_2, p(s_1))
\]  

(5.1)

But we have no polynomial solution to this set of equations and hence cannot model this directly using a Neural Network. But we do see many works trying to work around this, some by modelling $p(s_t)$ on all probabilities of sentences before $s_t$, as shown in section 2.5.1.1. This approximate formulation is shown in equation (5.2).

\[
p(s_t) = \text{func}(s_t, p(s_i) | i < t, d, \theta)
\]  

(5.2)

Some works also completely ignore the dependencies altogether (section 2.5.2) or make it a multi-step process (section 2.5.3).

The inability of modeling equation (5.1) and using workarounds could be one of the leading causes of the problems discussed in this chapter.

However, the summary ranking objective bypasses the need for working around (5.1) altogether since it directly optimizes at the summary level. While we did see some issues with MatchSum, a summary ranking method, in section 5.2.2, the issue should be mainly attributed to poor training and not to the problem formulation itself. This might mean that the potential in summary ranking could be yet untapped.

5.7 Conclusion

In this chapter, we saw inefficiencies and discrepancies in two different classes of models due to differences in training and testing objectives. We showed that models trained with classification tend to overfit the labels on training data but cannot generalize for the same on test data. We also saw how reliant
the MatchSum is on the candidate summary generator and how it cannot actually rank summaries unless they are very similar. This could be attributed to the limited training MatchSum received, where it didn’t even see a bad summary. We also saw some glimpse of what we can do to generalize summarisation models using simple experiments.
Chapter 6

Conclusions and Future Work

In this work, we looked at solving Extractive Summarisation from a wide range of angles. We first saw the theoretical frameworks for the Sentence and Document structures and how they can aid in Summarisation (chapter 1). We then saw why the problem of Extractive Summarization is not as simple as it seems, even though we are simply copying content from the source to construct the representative summary (section 2.1) and the Combinatorial Explosion problem that arises with more expressive and complicated models (section 1.5). Then we delved into the different approaches to solving the Extractive Summarisation problem. In Chapter 3 we formulated and tested a hypothesis: can a model learning to Summarize implicitly learn to identify structures that are prevalent in the Document. We proposed constructing summarization models using Tree-LSTM based document-encoders that are capable of learning dynamic structures [13] in the data from end-to-end training of the model. In theory, this Tree-LSTM could learn to represent the same structures that are observed in RST discourse trees. But in practice, the structures learned by the models rarely represent those that are present in theories proposed by humans. One reason could be that almost all such theories adopt the principles of Occum’s Razor [7], in one way or the other to simplify things. But while we weren’t able to show that our models actually learned discourse relations, we were able to show some proof that our model was able to learn document structure. We showed this by testing our model on predicting which sentences occur next to each other in a paragraph. Our results showed that our model had indeed learned some document structure that wasn’t shown to the model (Section 3.8). Choi et al. also showed similar indirect results for sentence structures [13].

We then looked at different approaches for dealing with the exponential number of possibilities of extractable summaries. First, we looked at a pipelined framework for Extractive Summarization, which divided the task into two steps: Sentence Extraction and Summary Ranking. This two-stage architecture is used to solve the problem that arises when we want to rank candidate summaries: How do we deal with the exponential amount of possible candidate summaries? To answer this, we use the Sentence Extraction step to shortlist the number of sentences that can be used for constructing candidate summaries for ranking. This reduces our search space for summaries from over two thousand
to one hundred somethings. We then construct all possible summaries of lengths 2 and 3, which the Summary Ranker then ranks. We also proposed a unique training methodology based on teacher forcing for training the two parts of the network since the design involving the connection between sentence extractor and summary ranker is non-differentiable.

In Chapter 4, we looked at adopting multiple complex Reinforcement Learning Frameworks (Policy-Gradients and Deep-Q-Learning) to this task. We presented a unique reward formulation that helped us in differentiating between different sequences of sentence selection. Since, in general, bad decisions made early can have cascading effects on subsequent steps, making good decisions early is very important. We showed how our reward formulation satisfies this principle, which other related works with similar formulations have failed to do so. We then saw how this reward formulation could be used to adopt DQN and more complex Policy Gradient formulations based on REINFORCE [65] compared to other works which use a much simpler formulation of the same REINFORCE algorithm to train their models [49, 21]. Then we looked at using dynamic-neural-network-based loss functions for summary ranking, using the SoDeep network [22].

Finally, we looked at the effects of shortsighted training on some State-of-the-art models. We saw the discrepancies that arose between what the model was trained for and what the model was actually doing due to the way training objectives were designed. We explored such issues on two prominent works in this field: BertSum [38], and MatchSum [72]. We saw how BertSum could not differentiate between the top summaries and how it overfits the one summary used to train the model. We also saw some detrimental effects it could have had on MatchSum’s results. MatchSum has never looked at a bad summary during training and has only learned to rank similar summaries which have very high overlap in their content. Due to this limited training, the performance of the model is affected very badly when it is asked to rank candidate summaries with huge variance. This shows the huge dependence MatchSum has on the candidate summary shortlisting step. One of the main reasons for such discrepancies lies in the design of the training objectives for the task, which does not directly represent the Extractive Summarisation objective. We then demonstrate how big the effect of overfitting actually is using a simple model and provide a glimpse at the kind of solutions required to overcome this discrepancy between the training and test objectives.

In this work, we mainly used Language models, which are good at representing sentence-level information. For example, Bert has learned to identify sentence structures implicitly, do coreference resolution, etc. [15]. Not only that, transformer-based language models like Bert can be used in zero/one-shot learning for many tasks. However, these do not have the same expressibility power at the level of documents. We do have models like GPT-3 [9], which can be said to excel at representing and understanding documents. It has generated a lot of buzz in the last two years. But given the size of that model and the
cost involved in training and maintaining such models, only a few giant companies can successfully use them. Research on much smaller, cost-effective, and computationally less intensive models will have the most impact on the equality of AI for everyone.

Reinforcement Learning in Extractive Summarization is also still in primitive stages. In this work, we saw adopting Value Function approximation and Policy Gradients to the task separately, in Chapter 4. But there are more advanced algorithms like A2C, A3C [45], DPG [50], DDPG [35], D4PG [5] etc which can still be explored and adopted for the same problem.

One other important direction this work can be extended is in addressing the issues that are present in the Extractive Summarisation models, some of which were shown in Chapter 5. Improvements can come in the form of redefining the evaluation metrics, studying how differences in the problem formulation involved in training the models vs. the actual objective can lead to discrepancies. One of the most important reasons why such issues occur (and are difficult to detect) is that we have very little knowledge about what the networks we train actually learn. We do not have the reasoning ability to look at the millions of computations involved in those models and say what exactly the model is doing. Some foundations for what is required for such understanding were laid by Zhang et al. in their work “Understanding deep learning (still) requires rethinking generalization” [70]. Here they question what we think is true about Deep Learning, what the term “generalization” actually means, and show that it does not mean what we think. This lack of understanding DNNs has also led to the development of a whole branch of studies under the topic “Adversarial attacks on Deep learning models”, which mainly involve minor modifications to input to completely lead the DNN models astray [25], questioning the numbers and metrics that we use to determine how good the models that we use actually are in different scenarios.

Overall, in this work, we explored the problem of Extractive Summarisation from many different perspectives. We hope that the work in this field will find the experiments detailed in this work useful.
Related Publications

• Addressing issues in training Extractive Summarisation Models. *Ramkishore Saravanan, Nikhil Pinnaparaju, Vasudeva Varma*. Accepted at 9th International Conference on Pattern Recognition and Machine Intelligence (PReMI’21)
Bibliography


