# QUERY-BASED SUMMARIZATION USING REINFORCEMENT LEARNING AND TRANSFORMER MODEL

**ASIF MAHMUD**
**Bachelor of Science, Military Institute of Science and Technology, 2016**

A thesis submitted
in partial fulfillment of the requirements for the degree of

**MASTER OF SCIENCE**

in

**COMPUTER SCIENCE**

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

# QUERY-BASED SUMMARIZATION USING REINFORCEMENT LEARNING AND TRANSFORMER MODEL

ASIF MAHMUD

Date of Defence: December 20, 2019

# Dedication

I dedicate my thesis to the almighty for His blessings; and my parents who kept their full trust on me, and helped me with all the supports I needed throughout my whole life.

# Abstract

Query-based summarization problem is an interesting problem in the text summarization field. On the other hand, the reinforcement learning technique is popular for robotics, and becoming accessible for the text summarization problem in the last couple of years (Narayan et al., 2018). The lack of significant works using reinforcement learning to solve the query-based summarization problem inspired us to use this technique. While doing so, We also introduce a different approach for sentence ranking and clustering to avoid redundancy in summaries. We propose an unsupervised extractive summarization method, which provides state-of-the-art results on some metrics. We develop two abstractive multi-document summarization models using the reinforcement learning technique and the transformer model (Vaswani et al., 2017). We consider the importance of information coverage and diversity under a fixed sentence limit for our summarization models. We have done several experiments for our proposed models, which bring significant results across different evaluation metrics.

# Acknowledgments

“Bismillahir-Rahmanir-Rahim”

In the name of Allah, the almighty, most Gracious, most Compassionate...

Firstly I want to thank the almighty Allah for giving me the energy, ability and His endless blessings to complete this thesis.

I would like to convey my sincere gratitude to Professor **Dr. Yllias Chali**, for his tremendous support, invaluable advice and encouragement. He has shown me the correct path, and helped me to explore research challenges. The door to Prof. Chali’s office was always open whenever I ran into trouble or had any concerns about my research or writing this thesis.

I would also like to thank my M.Sc. supervisory committee members **Dr. John Zhang** and **Dr. John Anvik** for their time and effort.

I must thank the University of Lethbridge for financial support. I am also grateful to the **Natural Sciences and Engineering Research Council (NSERC)** of Canada discovery grant for providing me a TITAN X GPU machine to conduct my experiments. I am also thankful to my supervisor for the financial assistance I got from him.

I am forever thankful to my friends for their helps, friendship and encouragement, which is worth more than I can express here. I am in debt to those people who made my life in Lethbridge very comfortable and enjoyable.

Among the friends I got here, it would be a mistake if I do not express my gratitude to two people. First, Tanvir Ahmed Fuad, without whom I might have to go through severe difficulties throughout the last two years. Starting from the application process to completing this thesis, he helped me in each of the steps in such a way which can not be put into

words. Without him, completion of this thesis in time would have been an imagination. Secondly, I need to thank Iffatur Ridwan for her sisterly love towards me. She is the elder sister I never had, and maybe I do not deserve.

Last but not least, I would like to mention my father, who just only used a couple of shirts and one pair of shoes for the whole year; and my mother, who had to go through four to five hours bus journey each day to continue her job while raising me. Whatever I have done in my whole life, they are the reason behind all of my achievements, and I do not have any way to repay their efforts and sacrifices for me.

# Contents

**Contents** **vii**

**List of Tables** **x**

**List of Figures** **xi**

**1 Introduction** **1**
- 1.1 Contribution . . . 2
- 1.2 Thesis Overview . . . 2

**2 Background** **4**
- 2.1 Text Summarization: Overview of Recent Works . . . 4
  - 2.1.1 Extractive Summarization . . . 5
  - 2.1.2 Abstractive Summarization . . . 6
  - 2.1.3 Query Focused Summarization . . . 9
- 2.2 Summary Evaluation . . . 9
  - 2.2.1 ROUGE . . . 10
  - 2.2.2 METEOR . . . 13
- 2.3 Keyword Extraction . . . 14
  - 2.3.1 RAKE Algorithm . . . 14
- 2.4 Sentence Similarity . . . 17
  - 2.4.1 Jaccard Similarity . . . 17
  - 2.4.2 Cosine Similarity . . . 17
- 2.5 Word Embedding . . . 18
  - 2.5.1 One-Hot Vectors . . . 18
  - 2.5.2 Word2Vec . . . 19
  - 2.5.3 GloVe . . . 21
  - 2.5.4 FastText . . . 25
- 2.6 Sentence Embeddings . . . 25
  - 2.6.1 Word Mover Distance . . . 25
  - 2.6.2 Smooth Inverse Frequency (SIF) . . . 27
- 2.7 Neural Network . . . 28
  - 2.7.1 Convolutional Neural Network (CNN) . . . 28
  - 2.7.2 Recurrent Neural Network (RNN) . . . 29
  - 2.7.3 Long Short Term Memory (LSTM) . . . 30
  - 2.7.4 Gated Recurrent Unit (GRU) . . . 32
  - 2.7.5 Activation Functions . . . 32

2.8 Sequence-to-sequence (seq2seq) Model . . . . . . . . . . . . . . . . . . . . 34
2.9 Multi Sentence Compression . . . . . . . . . . . . . . . . . . . . . . . . . . 36
2.10 Reinforcement Learning (RL) . . . . . . . . . . . . . . . . . . . . . . . . . 37
2.11 Neural Machine Translation (NMT) . . . . . . . . . . . . . . . . . . . . . . 39
2.11.1 Encoder-Decoder Framework . . . . . . . . . . . . . . . . . . . . . 41
2.11.2 Training: Maximum Likelihood Estimation (MLE) . . . . . . . . . 43
2.11.3 Attention Mechanism . . . . . . . . . . . . . . . . . . . . . . . . . 43
2.11.4 Pointer Network . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
2.11.5 Greedy 1-Best Search . . . . . . . . . . . . . . . . . . . . . . . . . 47
2.11.6 Beam Search Algorithm . . . . . . . . . . . . . . . . . . . . . . . . 47
2.12 Transformer Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
2.12.1 Transformer Encoder . . . . . . . . . . . . . . . . . . . . . . . . . 49
2.12.2 Transformer Decoder . . . . . . . . . . . . . . . . . . . . . . . . . 50
2.12.3 Multi-Head Attention . . . . . . . . . . . . . . . . . . . . . . . . . 50
2.12.4 Positional Encoding . . . . . . . . . . . . . . . . . . . . . . . . . . 52
2.13 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53

**3 Proposed Models** **54**
3.1 Sentence Ranking and Clustering . . . . . . . . . . . . . . . . . . . . . . . 54
3.1.1 Text Ranking . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
3.1.2 Text Clustering . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
3.1.3 Cluster Selection . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
3.2 Unsupervised Extractive Summarization Model . . . . . . . . . . . . . . . 60
3.2.1 Preliminaries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
3.3 Abstractive Summarization Models . . . . . . . . . . . . . . . . . . . . . . 63
3.3.1 Abstractive Summarization using Transformer Model . . . . . . . . 64
3.3.2 Abstractive Summarization using Reinforcement Learning . . . . . 66
3.3.3 Abstractive Sentence Selection . . . . . . . . . . . . . . . . . . . . 68
3.3.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69

**4 Experiments and Results** **70**
4.1 Unsupervised Extractive Summarization . . . . . . . . . . . . . . . . . . . 70
4.1.1 Datasets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70
4.1.2 Baselines . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70
4.1.3 Evaluation Metrics . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
4.1.4 Experimental Results . . . . . . . . . . . . . . . . . . . . . . . . . 72
4.2 Abstractive Summarization Models . . . . . . . . . . . . . . . . . . . . . . 72
4.2.1 Datasets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72
4.2.2 Baselines . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72
4.2.3 Evaluation Metrics . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
4.2.4 Experimental Results . . . . . . . . . . . . . . . . . . . . . . . . . 74
4.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74

**5 Conclusion** **75**

**Bibliography** **76**

**Appendix A Sample System Generated Summaries** **86**

# List of Tables

2.1 Co-occurrence Matrix . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

2.2 Co-occurrence probabilities for target words ice and steam with selected context words from a 6 billion token corpus. (Pennington et al., 2014) . . . 22

3.1 Queries associated with the topic “algae bio-fuel” . . . . . . . . . . . . . . 58

4.1 Comparison of our proposed Unsupervised Extractive model with the baseline models. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71

4.2 Comparison of our proposed Abstractive Summarization Models with the baseline models. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73

A.1 Randomly selected outputs for our **Extractive** model. . . . . . . . . . . . . 86

A.2 Randomly selected outputs for our **Abstractive Reinforcement Learning** model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 86

A.3 Randomly selected outputs for our **Abstractive Transformer** models. . . . 87

# List of Figures

2.1 Visualization of word to word similarity of all non-stop words from both headlines is embedded into a Word2Vec space (Kusner et al., 2015). . . . . 19
2.2 N-gram neural language model (Nayeem, 2017). . . . . . . . . . . . . . . 20
2.3 Visualization of semantic relationships, e.g. male-female, verb tense and even country-capital relationships between words (Mikolov et al., 2013b). . 20
2.4 CBOW model (left) and Skip-gram model (right) from (Mikolov et al., 2013b). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
2.5 Concept of Word Mover Distance. (Saxena, 2018) . . . . . . . . . . . . . . 26
2.6 A Convolutional Neural Network. (Britz, 2015) . . . . . . . . . . . . . . . 28
2.7 Max Pooling. (Ujjwalkarn, 2016) . . . . . . . . . . . . . . . . . . . . . . 28
2.8 An unrolled recurrent neural network. (Olah, 2015) . . . . . . . . . . . . . 29
2.9 LSTM at time-step $t$ (Hochreiter and Schmidhuber, 1997) . . . . . . . . . . 31
2.10 GRU Gating Mechanism (Chung et al., 2014) . . . . . . . . . . . . . . . . 33
2.11 Basic structure of sequence to sequence model. (Kostadinov, 2019) . . . . . 35
2.12 Agent-Environment relational loop . . . . . . . . . . . . . . . . . . . . . 37
2.13 Reinforcement Learning Method's example. (Bajaj, 2018) . . . . . . . . . 38
2.14 Sequence to Sequence Learning with Neural Networks (Sutskever et al., 2014a) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
2.15 Attention Model (Bahdanau et al., 2014) . . . . . . . . . . . . . . . . . . 44
2.16 Pointer Network Architecture (Vinyals et al., 2015). . . . . . . . . . . . . . 46
2.17 Google's Neural Machine Translation (NMT) Model. (Wu et al., 2016a) . . 47
2.18 Transformer model architecture (Vaswani et al., 2017) . . . . . . . . . . . . 49
2.19 Transformer Encoder-Decoder Architecture. (Alammar, 2018) . . . . . . . 50
2.20 Multi-Head Attention Mechanism. (Vaswani et al., 2017) . . . . . . . . . . 51

3.1 Proposed method of sentence ranking and clustering . . . . . . . . . . . . . 55
3.2 Sentence Clustering Model . . . . . . . . . . . . . . . . . . . . . . . . . . 59
3.3 Proposed method of Unsupervised Extractive Summarization Model . . . . 61
3.4 Proposed method of Abstractive Summarization Model . . . . . . . . . . . 65
3.5 Extractive Agent. the convolutional sentence encoder calculates $r_j$ for every sentences. The RNN encoder (blue) calculates the context-aware representation $h_j$, and then the RNN decoder (green) selects sentences $j_t$ at time $t$. With $j_t$ selected, $h_{j_t}$ will be fed into the decoder at time $t+1$. (Chen and Bansal, 2018) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 67
3.6 Training step of the extractor agent, and the operation in between the extractor and abstractor agent. (Chen and Bansal, 2018) . . . . . . . . . . . . 68

# Chapter 1

# Introduction

"Text Summarization is the process of distilling the most important information from one or more texts to produce an abridged version for a particular task and user." (Section 23.3 of Jurafsky and Martin (2008))

We are living in a digital information world where we need to pass information as fast as possible at a minimum cost. Everything is at our hands with the assistance of the internet and digital media. The internet consists of huge amount of textual data with an exponential growth rate. A single query in the search engine generates thousands of results within a fraction of a second. Sometimes, it becomes difficult to choose the best result from the varieties of options and topics. These huge amounts of information might cause data redundancy and make it difficult to get concise information. Sometimes, going through large documents might cause a user to miss important information. From these particular concerns, the idea of developing automatic text summarization systems got focused, where the model has to generate a short summary from a single or a related set of documents. After a while, rather than getting the summary from the whole document, people started to feel the urge to get a precise summary based on a particular topic. From that thinking, the idea of query-based summarization gets the attention in the field of Natural Language Processing (NLP).

A quality summary includes most of the important information from the source document maintaining the readability and grammatical structure. For extractive summaries, sentences are directly chosen from the source text. So, sometimes it is quite impossible to express the actual idea, because we do not consider the hidden meaning of the sentences. On the other hand, in the process of abstractive summary generation new words and sentences are created, which might compromise the sentence structure and readability. In this thesis, we have proposed some approaches in the area of query-based text summarization focusing on solving these problems.

## 1.1 Contribution

In this thesis, we propose a few models. The summary of our contributions in this thesis are as follows:

- We design an unsupervised sentence ranking model, which is simple and effective.
- We propose an unsupervised extractive summarization model. To the best of our knowledge, this one has the best result as an unsupervised method on DUC 2005[1], DUC 2006[2] and DUC 2007[3] datasets based on ROUGE-2 evaluation metrics.
- We propose an abstractive summarization model using the Transformer model (Vaswani et al., 2017).
- We develop a reinforcement learning-based approach to solve the query-based abstractive summarization problem. To the best of our knowledge, our model is the first approach with reinforcement learning to solve the query-based abstractive summarization problem.

## 1.2 Thesis Overview

This thesis is organized as follows. In Chapter 2, we provide an overview of automatic text summarization. We also provide a brief introduction to the deep learning techniques, especially used in our text summarization methods. In Chapter 3, we present our model for

[1]https://duc.nist.gov/duc2005/
[2]https://duc.nist.gov/duc2006/
[3]https://duc.nist.gov/duc2007/

sentence ranking and text clustering, followed by the details of our unsupervised extractive summarization model. We also include our approaches for the abstractive summarization using a reinforcement learning-based model and a tensor2tensor[4] based Transformer model (Vaswani et al., 2017) at sentence level in this chapter. In Chapter 4, we include the experimental results of our summarization models. We conclude our thesis with an overview of our works and the future direction in Chapter 5.

[4]Tensor2tensor code library link: https://github.com/tensorflow/tensor2tensor

# Chapter 2

# Background

## 2.1 Text Summarization: Overview of Recent Works

The automated text summarization task intends to find the most relevant information in a document, and presents it in a short or condensed form. A good summary is one that retains the most vital contents from the source document or a collection of related documents while being non-redundant, coherent and grammatically readable (Yao et al., 2017).

Based on summary generation process, summarization can be categorized into two classes: Abstractive Summarization and Extractive Summarization. To generate an abstractive summary, extensive natural language generation is needed including but not limited to: paraphrasing, word deletion, sentence fusion etc (Chali et al., 2017). So, it is comparatively easy to generate extractive summaries, in which important sentences from the source text are selected without any alteration to generate a summary. Some of the commonly used abstractive summarization methods are sentence compression, lexical paraphrasing, and syntactic reorganization.

We can classify summarization into two classes based on the number of the source documents, namely single-document summarization and multi-document summarization. Because of the overlapping information between the documents of a particular topic, the multi-document summarization is more difficult to deal with than the single document summarization. However, the extractive techniques may generate a redundant summary or biased to any particular source document in the case of multi-document summarization, as all of the source documents usually contain similar information (Nayeem and Chali, 2017a).

We can also classify the summarization problem into generic summarization and query-based summarization. The generic summarization problem has only a set of source documents, and we need to generate the summary, which will represent the important information from the whole set of documents. For the query-based summarization problem, a set of query sentences is also provided along with the set of source documents. We need to generate the summary from the source documents, which will include the important information from the source documents related to the query. So, the generic summary provides the gist of the source documents, while the query-based summary represents the gist of the source documents based on the topic provided by the query.

In the following subsections, we will discuss about the recent works and techniques on generic summarization for both extractive and abstractive methods; then, we will continue for the query-based summarization techniques.

### 2.1.1 Extractive Summarization

Since the beginning of this century, a number of approaches have been taken for an automated extractive summary generation that is the combination of a collection of machine learning and graph-based technique. Computing sentence importance for text summarization, LexRank (Erkan and Radev, 2004) and TextRank (Mihalcea and Tarau, 2004) are graph-based methods. The RegSum system (Hong and Nenkova, 2014) makes use of a supervised model that predicts word importance. Instead of adding sentences greedily to form a summary, considering multi-document summarization as a sub-modular maximization problem was proven successful by Lin and Bilmes (2011). One of the most commonly used practices is to formulate the problem as integer linear programming (ILP). Therefore, concept-based ILP (Gillick and Favre, 2009; Nayeem and Chali, 2017a) was proposed where the aim was to maximize the summation of the weights of the concepts present in the summary.

Among the recent works on neural networks, an attention-based encoder-decoder was proposed by Cheng and Lapata (2016), and a simple recurrent network-based sequence classifier was used by Nallapati et al. (2017) to solve the problem of extractive summarization. However, they were limited to single document summarization, where sentences of summaries are ordered according to their position in the original document. Parveen and Strube (2015) and Parveen et al. (2015) proposed graph-based techniques to undertake coherence, which is also limited to single-document summarization. A multi-document summarization system was proposed by Wang et al. (2016), which combines both coherence and informativeness.

In the past few years, reinforcement-learning based summarization models have become popular in the field of natural language processing. Ryang and Abekawa (2012) proposed a framework to solve the multi-document summarization problem using reinforcement learning, which was the first attempt in this field. Later on Rioux et al. (2014) updated the model with different reward functions. Narayan et al. (2018) proposed an encoder-decoder based sentence ranking method using the ROUGE metric as the reward function.

### 2.1.2 Abstractive Summarization

Abstractive summarization is much more difficult in comparison to extractive summary, as it involves advanced techniques; namely: content organization, meaning representation, sentence fusion, sentence compression, and paraphrasing. In recent days, there has been a significant growing interest in document summarization. In this method, the source sentences are compressed into smaller sentences (Clarke and Lapata, 2006, 2008; Filippova, 2010) as a first step to generate an abstractive summary. The sentences that have been compressed from the original document only using the word deletion method are included in the techniques of compressive summarization. Sentence compression that is generated from more than one sentence is known as Multi-Sentence Compression (**MSC**). The majority of the previous MSC approaches depend on syntactic parsing to build the dependency tree for each related sentence in a cluster producing grammatical compressions (Filippova and Strube, 2008). It is mentioned that the syntactic parsers are not available for every language. That is why, an alternative word graph-based approach was proposed by Filippova (2010), which only requires a list of stopwords and a POS (Parts of Speech) tagger. Here, a directed word graph is built where nodes represent the words, and edges represent the adjacency between words in a sentence. Then the compressed sentences are generated by finding the k-shortest path in the word graph. Boudin and Morin (2013) refined Filippova's approach by re-ranking the fusion candidate paths according to the key phrases, which helps generate more informative sentences. However, in order to improve the informativity, grammaticality has been sacrificed in these works (Nayeem and Chali, 2017b; Nayeem, 2017).

Banerjee et al. (2015) used the sentence fusion approach of Filippova (2010), and combined it with Integer Linear Programming (ILP) sentence selection to develop an abstractive multi-document summarization system. Following Banerjee et al. (2015)'s work, several approaches were proposed recently with slight modifications. Shafiei Bavani et al. (2016) exploited Multiword Expressions (MWE) to generate more informative compressions. Tuan et al. (2017) included syntax factors along with Banerjee et al. (2015) for improving the performance of their model. Still, all the aforementioned systems attempt to generate compressions by making a copy of the source sentence words without any paraphrasing.

In recent days, end-to-end training with encoder-decoder neural networks has established great success for abstractive summarization (Bahdanau et al., 2014). To solve the sentence summarization task, these systems have followed encoder-decoder with attention (Bahdanau et al., 2014; Luong et al., 2015) models from the area of machine translation. Rush et al. (2015) were the first to use neural sequence-to-sequence learning to generate a headline from a single document. However, this research under the term sentence summarization (Rush et al., 2015), which only can generate a single sentence is somewhat misleading, and is termed as text summarization in some research works (Chopra et al., 2016; Nallapati et al., 2016; Ma et al., 2017; Nayeem et al., 2018; Zhou et al., 2017; Suzuki and Nagata, 2017). The previously mentioned models have several limitations: one of them is that the produced output is very short (about 75 characters). Similar to headline generation, the model also produces sentences without considering grammatical properties during generation. Moreover, there are also some attempts that use the CNN/DailyMail dataset (Hermann et al., 2015) as supervised training data for generating a multi-sentence summary from a single document (Li et al., 2017b; See et al., 2017; Paulus et al., 2017a; Narayan et al., 2018; Chali et al., 2017). The recently proposed abstractive summarization models generate compressed summaries by deleting the words from a single source document. The models do not involve any direct paraphrasing. Therefore, any new words are not generated that are different from the source document words (other than morphological variation), and this is also pointed out by the researcher's own experimental results. Some of the researchers make use of a neural network-based framework to take care of the summarization problem in a multi-document setting (Yasunaga et al., 2017; Li et al., 2017a). However, Yasunaga et al. (2017)'s work is only limited to extractive summarization while Li et al. (2017a)'s work is limited to the compressive summary generation which uses an ILP-based model where there have some great possibilities of the existence of redundant sentences in the summary side. Recently, Fuad et al. (2019) proposed an attention-based neural sentence fusion model to solve the multi-doc abstractive summarization problem, which achieves the state-of-the-art results in different evaluation metrics, and solves the redundancy problem to a great extent.

Paulus et al. (2017b) proposed a neural intra-attention model combined with reinforcement learning. This was the first significant effort in the abstractive text summarization problem using reinforcement learning. Liu et al. (2018) proposed a different approach of using the generative adversarial network to solve the abstractive summarization problem. Later on, Chen and Bansal (2018) proposed a method where they built an extractor agent to get the extractive summary from the source, and then used a sequence-to-sequence encoder-decoder model to convert that summary into a concise abstractive one. At the same time, Pasunuru and Bansal (2018) proposed a method of using the multi-reward function for reinforcement learning to solve the abstractive summarization problems which opens a great opportunity of solving the summarization problems using reinforcement learning.

### 2.1.3 Query Focused Summarization

The query-based summarization task was first introduced in the Document Understanding Conference (DUC) (Dang, 2005) in 2005, which was extended in 2006 and 2007. The task of query-based summarization provides us the relevant source documents paired with a set of queries. The expected output is a short summary generated from the source document while answering the question or focusing on the topic mentioned in the queries. Daumé III and Marcu (2006) proposed a Bayesian model to solve the query-based summarization model, which was one of the first recognized works in this field. Fisher and Roark (2006) proposed a supervised sentence ranking approach to solve the DUC 2006 task, and achieved a state-of-the-art result. Later Ouyang et al. (2011) made the initial attempt to introduce the regression model to solve the query-based summarization problems. Recently, Feigenblat et al. (2017) provided an unsupervised method named *Cross Entropy Summarizer* (CES), which achieved the state of the art ROUGE scores on DUC 2005-7 datasets. Until 2017, the topic-based DUC datasets were the only option to work with the query-based summarization problem, and all of those previous works were focused on extractive summarization. After that Nema et al. (2017) created a dataset from Debatepedia encyclopedia, which includes the pro and con arguments and quotes on different debate topics. They also proposed an attention-based abstractive summarization model to solve the problem with their dataset.

## 2.2 Summary Evaluation

We have used various automatic evaluation metrics to evaluate our models. Those are mentioned as follows:

### 2.2.1 ROUGE

To determine the accuracy of a machine-generated summary, we compare it with a reference or a number of reference summaries (generally human-annotated). Recall-Oriented Understudy for Gisting Evaluation (Lin, 2004) (ROUGE)[5] is an automatic tool which is used widely for this purpose. There are four different ROUGE metrics - namely ROUGE-N (1,2,3,4), ROUGE-L and ROUGE-S.

- **ROUGE-N:** A summary evaluation that measures uni-gram (one word), bi-gram (two words), tri-gram (three words) and higher-order n-gram overlap.
- **ROUGE-L:** A summary evaluation that measures the longest matching sequence of words using the Longest Common Sub-sequence (LCS).
- **ROUGE-S:** ROUGE-S is used to evaluate the summaries which allows arbitrary gaps between any pair of words. It is also known as skip-gram co-occurrence. For example, given any pair of words, we allow a maximum of two gaps in between them to measure the ROUGE-S score using skip-bigram. For instance, if the phrase is “I like to eat burgers,” the skip-bigrams would be “I like, I to, I eat, like to, like eat, like burgers, to eat, to burgers, eat burgers”.

For multi-document summarization, the most frequently used metrics among the above mentioned is **ROUGE-N**, where the number of overlapping n-grams is counted to evaluate between the system generated summary and the reference summaries. ROUGE-N can be defined as follows:

$$\text{ROUGE-N}(S_g, R_i) = \frac{\sum_{s_r \in R_i} \sum_{g_n \in s_r} Count_{match}(g_n)}{\sum_{s_r \in R_i} \sum_{g_n \in s_r} Count(g_n)} \tag{2.1}$$

where, $\mathbf{S_g}$ = system generated summary, $\mathbf{s_r}$ = reference summary sentence, $\mathbf{n}$ = length

[5]ROUGE package link: http://www.berouge.com

of the n-gram, $\mathbf{g_n}$ = n-grams, $\mathbf{Count}(\mathbf{g_n})$ = maximum number of n-grams in a system generated summary, and $\mathbf{Count_{match}}(\mathbf{g_n})$ is the maximum number of n-grams in the system generated summary that matches with the set of reference summaries (Lin, 2004).

While evaluating, if multiple reference summaries are used, a pairwise summary-level ROUGE-N score is computed between a candidate system generated summary $S_g$ and every human annotated reference $R_i$ from the reference set,

$$R = \{R_1, R_2, \ldots, R_n\} \tag{2.2}$$

The maximum among the summary-level ROUGE-N scores is the final ROUGE-N score. This can be described as follows:

$$\text{ROUGE-N}_{multi} = \underset{R_i \in R}{\arg\max}\ \text{ROUGE-N}(S_g, R_i) \tag{2.3}$$

Here, $n$ is the number of reference summaries.
For instance:
**System Summary (system generated):** People like the extra crispy french fries from McDonald's.
**Reference Summary (human annotated):** People like McDonald's extra crispy french fries.

The precision and recall can be computed using the overlap of words to get a good quantitative value. In terms of ROUGE, recall means how much of the reference summary is presented by the system generated summary. If the individual words are considered only, then the recall can be computed as:

$$\text{ROUGE-1 (recall)} = \frac{number\ of\ overlapping\ words}{total\ number\ of\ words\ in\ the\ reference\ summary} = \frac{7}{7} = 1.0 \tag{2.4}$$

It is easily identifiable that all the words in the human-annotated reference summary have been captured in the machine-generated system summary. But a machine-generated summary (i.e., system summary) can be exceedingly long if it captures all of the words from the human-annotated reference summary. In the system summary, most of the words might be useless, resulting in a summary with information that is redundant and repetitive. The precision can be defined as the amount of the system summary that was actually relevant or needed. For the same example, precision is measured as:

$$\text{ROUGE-1 (precision)} = \frac{number\ of\ overlapping\ words}{total\ number\ of\ words\ in\ system\ summary} = \frac{7}{9} = 0.78 \quad (2.5)$$

This means 7 out of the 9 words in the system summary were relevant. Let us assume, the following system summary instead of the previous example:

**System Summary 2 (machine generated):** People like McDonald's french fries because they are extra crispy and tasty.

The Precision is:

$$\text{ROUGE-1 (precision)} = \frac{7}{12} = 0.58 \quad (2.6)$$

It is observed that the precision score has now decreased. This is the result of a few redundant words appearing in the system summary. In the case of generating summaries that are concise in nature, precision is really crucial. Therefore, the best way to evaluate a summary is computing both the **Precision** and **Recall**. The system summaries are sometimes forced to be concise given some constraints (such as length limit constraint), and using just the recall should be sufficient since precision is a matter of less concern in this case. In this thesis, the limited length recall measure is only reported, and the performance has also been reported in terms of **ROUGE-SU4**, where **S** means skip-bi-gram (match two non-contiguous words with other words in between) allowing rephrasing and sentence reorganization. To evaluate abstractive summaries, ROUGE-SU4 can be called a good measure compare to the other ROUGE metrics. For other in-between words, **U4** has been used, which means a maximum of four uni-gram words is allowed within a skip-bi-gram.

### 2.2.2 METEOR

The METEOR metric (Banerjee and Lavie, 2005) was first proposed as an evaluation metric that evaluates at the sentence level more effectively. Before the computation of the METEOR score, at first, an alignment between the system generated sentence and the reference sentence is mapped. For this mapping, each uni-gram in the system generated sentence is given 0 or 1 compared to the uni-gram in the reference sentences. The alignment considers stems, synonyms and paraphrase matching in addition to exact matches. Based on the mapping, uni-gram precision and recall are computed to measure the METEOR score.

We can define the precision ($P$) and recall ($R$) as $m/g$ and $m/r$; where $g$ and $r$ are number of words in generated summary and reference summary, and $m$ is the number of matched uni-gram between the reference summary and the generated summary. Then we can calculate the $F_{mean}$ as follows:

$$F_{mean} = \frac{PR}{\alpha P + (1-\alpha)R} \tag{2.7}$$

where, $F_{mean}$ is the harmonic mean between precision and recall. We consider $\alpha$ as 0.1 here, so that, the weight of recall is 9 times as high as the weight of precision. We choose this weight, because recall is more important than the precision in summarization task.

We introduce a penalty function which can be defined as follows:

$$p = \omega(\frac{g}{m}) \tag{2.8}$$

here, $\omega$ is considered a value between 0 and 1 to scale the amount of penalty. Finally we calculate the METEOR score using the following formula:

$$\text{METEOR} = (1-p)F_{mean} \tag{2.9}$$

## 2.3 Keyword Extraction

Keyword extraction is the process of collecting the most important and relevant words or sub-sentences from a text. While working with text summarization tasks, keyword extraction is one of the most crucial and important tasks which can help the readers understand the text in a short time. In text summarization, we collect the gist of the source text by generating some sentences, which represents the meaning of the source text. If we can identify the keywords from a text, it kind of performs as a highlighter of the text where the reader can focus more on the important words or phrases, which would be helpful for them to understand the message of the text in a faster way.

A simple keyword extraction algorithm mainly works in three steps:

- **Candidate Listing:** At first, it identifies all the phrases, words and terms that can be considered as keywords.
- **Properties Identification:** After listing all the candidates, the second step is to calculate the properties of those candidates to confirm whether it is a keyword or not.
- **Sorting and Finalize Keywords:** Then, the algorithm gives a score to all the candidates based on their properties, and sort them according to their scores. While scoring, it needs to prioritize a candidate's length, impact on the sentence, and its probability to be a keyword.

### 2.3.1 RAKE Algorithm

Rose et al. (2010) proposed an automatic keyword extraction method named *Rapid Automatic Keyword Extraction* (RAKE), which got popularity among the researchers for the summarization task. We describe the details working process of RAKE algorithm here.

Assume we need to get the keywords from the following text from the abstract of this thesis:

*"reinforcement learning technique is popular for robotics, and become accessible for the text summarization problem in the last couple of years. The lack of significant works with reinforcement learning inspired us to solve the query-based summarization problem using this technique."*

Firstly, RAKE algorithm will convert this text into tokens. So we will have the following list of tokens:

"reinforcement", "learning", "technique", "is", "popular", "for", "robotics", "and", "become", "accessible", "for", "the", "text", "summarization", "problem", "in", "the", "last", "couple", "of", "years", "the", "lack", "of", "significant", "works", "with", "reinforcement", "learning", "inspired", "us", "to", "solve", "the", "query", "based", "summarization", "problem", "using", "this", "technique"

Then RAKE removes the stopwords from these tokens. After removing the stopwords, the resulting list of tokens will be like below:

"reinforcement", "learning", "technique", "popular", "robotics", "become", "accessible", "text", "summarization", "problem", "last", "couple", "years", "lack", "significant", "works", "reinforcement", "learning", "inspired", "solve", "query", "based", "summarization", "problem", "using", "technique"

Now, if we list down the candidate keywords, we will get the following list: "reinforcement learning technique", "popular", "robotics", "become accessible", "text summarization problem", "last couple", "years", "lack", "significant works", "reinforcement learning inspired", "solve", "query based summarization problem using", "technique"

Now RAKE algorithm will provide scores for each of the keywords based on their property. We will show the process for the candidate "reinforcement learning technique" here. The formula can be defined as follows:

$$totalScore(candidate) = \sum_{i=1}^{n} score(w_i) \tag{2.10}$$

$$score(w) = degree(w)/frequency(w) \tag{2.11}$$

where for a particular word *w*, degree function provides the number of words it associates with including itself. And frequency function provides the number of occurrences in the text. For example, the word "reinforcement" has frequency of two (2) and degree of four (4). In this way, we can calculate the *totalScore* for "reinforcement learning technique" as follows:

$$\begin{aligned}
score(reinforcement) &= degree(reinforcement)/frequency(reinforcement) \\
&= 4/2 \\
&= 2 \\
score(learning) &= degree(learning)/frequency(learning) \\
&= 6/2 \\
&= 3 \\
score(technique) &= degree(technique)/frequency(technique) \\
&= 3/2 \\
&= 1.5
\end{aligned}$$

$$\begin{aligned}
totalScore(reinforcement\ learning\ technique) &= 2 + 3 + 1.5 \\
&= 6.5
\end{aligned}$$

The algorithm also considers "reinforcement", "learning", "technique" and their bi-grams as candidate keywords and gives them relevant scores. Then RAKE algorithm sorts the keywords in descending order and provides us the list with corresponding scores.

## 2.4 Sentence Similarity

Sentences can be similar in many aspects; for example, sentences can have similar structures, the same topics, or the same ideas. There are several Natural Language Processing (NLP) tasks which require dealing with similar sentences. For example, question-answer related sites like Quora[6] or StackOverflow[7] may need to determine whether a similar or same question has been asked before. Based on the requirement, there are many ways to calculate the similarity between two sentences. In this thesis, two-sentence similarity approaches are discussed.

### 2.4.1 Jaccard Similarity

Jaccard similarity is computed using the Jaccard Index (Real and Vargas, 1996) between two sentences. Sometimes, Jaccard Index is also referred as “Intersection over Union”. To compute the Jaccard Similarity between two sentences, both of the sentences are represented as sets where the Jaccard similarity of two sets A and B can be found using:

$$jaccard\_similarity(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|} \tag{2.12}$$

Here, $A \bigcap B$ represents the number of common words between the set A and B; and $A \bigcup B$ represents total number of distinct words in the set A and B.

### 2.4.2 Cosine Similarity

Cosine similarity (Mihalcea et al., 2006) is usually measured between two non-zero sentence vectors. If a sentence only contains the stopwords, it might create a zero vector. Cosine similarity ignores these vectors while computing the similarity score. Cosine Similarity is basically the cosine distance between two vectors. To compute cosine similarity, the sentences first need to be converted into a vector representation, and this process can be

[6]https://www.quora.com/
[7]https://stackoverflow.com/

performed in various ways. There is a detailed discussion about the vector representations in Section 2.5. After the conversion of sentences into vectors, the similarity between two vectors $s_i$ and $s_j$ is computed using:

$$cosine\_similarity(s_i, s_j) = \frac{s_i \cdot s_j}{\|s_i\| \, \|s_j\|} \tag{2.13}$$

## 2.5 Word Embedding

Word embedding is the process of using a vector representation for a word. It is a popular method which is used in many NLP applications; for example document classification, question answering and text summarization. The term “Word Embedding” was first introduced by Bengio et al. (2003), where a word embedding model was proposed by training a neural language model. In this section, we discuss some of the popular word embedding techniques.

### 2.5.1 One-Hot Vectors

Before building different NLP models, the similarity between two words, sentences, or even paragraphs has to be calculated. Through a vector space model, the one-hot vector is a representation of all the words. The vector representation has the corresponding entry in the vector for each word as 1 (present), and all other entries as 0 (absent). The size of the dictionary or vocabulary will be the length of the one-hot vectors. Cosine similarity on one-hot vectors is not capable of capturing semantic information when documents say the same thing in entirely different words. Let us consider these two following news examples:

- Obama speaks to the media in Illinois.
- The President greets the press in Chicago.

These two sentences do not have any word in common except for the stopwords such as *the* and *in*, which is not crucial for measuring semantic similarity. According to the

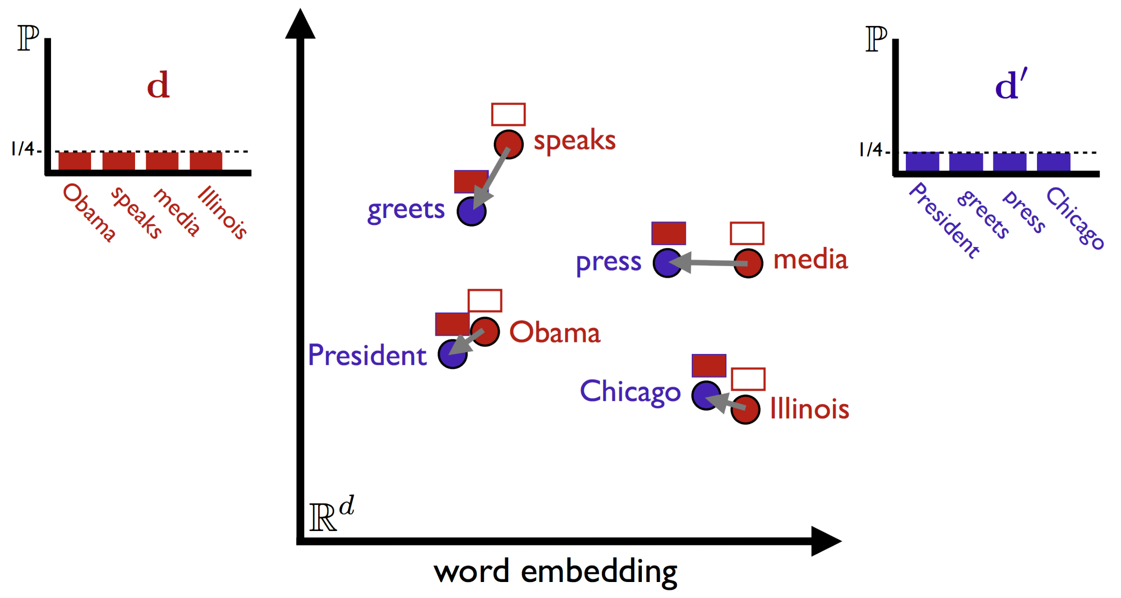

Figure 2.1: Visualization of word to word similarity of all non-stop words from both headlines is embedded into a Word2Vec space (Kusner et al., 2015).

one-hot vectors representation, their cosine distance would be maximal. To measure their semantic similarity accurately, further information is needed, which can be learned using large amounts of data through machine learning models (Kusner et al., 2015). Figure 2.1 visualizes the word to word similarity of the example.

### 2.5.2 Word2Vec

In distributional semantics, vector space models have been used since the 1990s to estimate continuous representations of words. Latent Dirichlet Allocation (LDA) (Blei et al., 2003) and Latent Semantic Analysis (LSA) (Landauer et al., 1998) are two examples. Word2Vec was proposed by Bengio et al. (2003), which was also the introduction of the idea of word embeddings. The language models build the joint probability $P(w_1,\ldots,w_T)$ of a sentence, where $w_i$ represents the $i^{th}$ word in the sentence. In the language model, grammatical and meaningful sentences are assigned with higher probabilities, and the lower probabilities are assigned to meaningless sentences. For example, let us assume that we are searching for something on the Internet using Figure 2.2; if we write “How long is a”

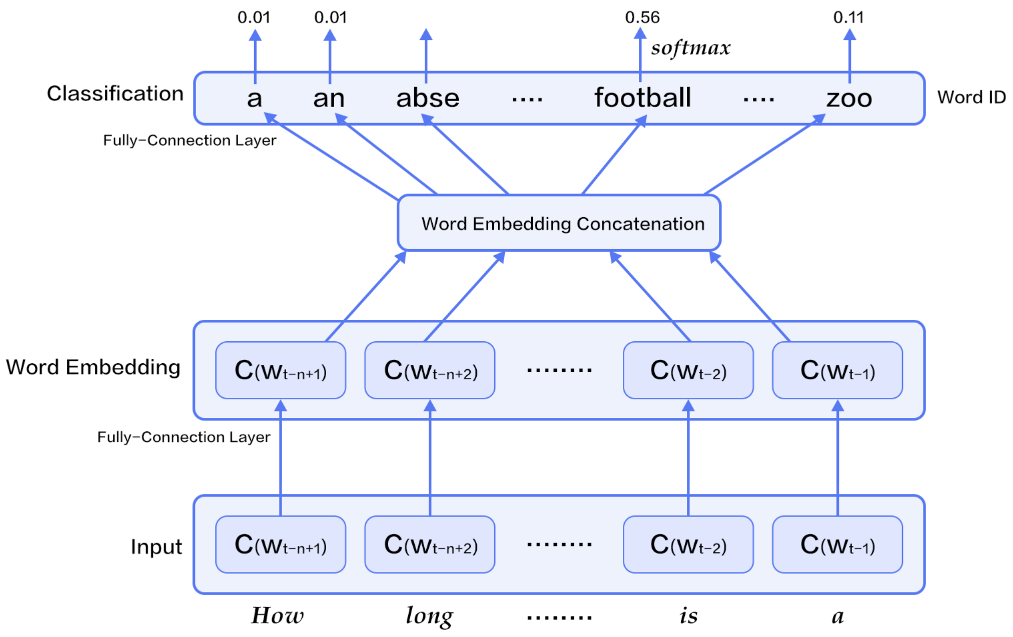

Figure 2.2: N-gram neural language model (Nayeem, 2017).

the search engines would suggest the next words "football field." This happens because, according to the language model, the probability of "How long is a football field" among all words in the target vocabulary is very high.

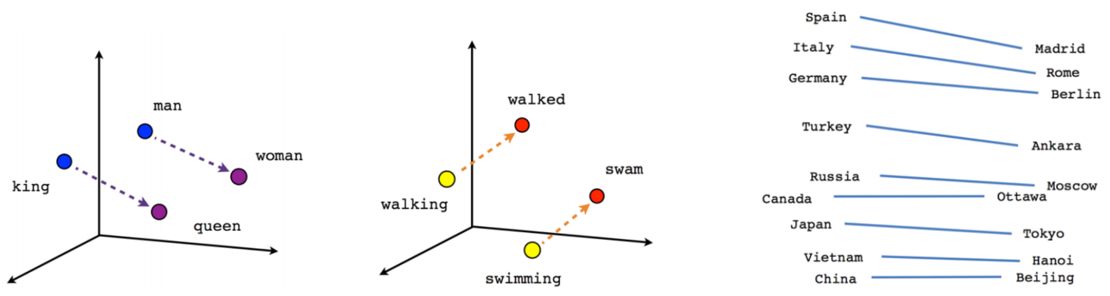

Figure 2.3: Visualization of semantic relationships, e.g. male-female, verb tense and even country-capital relationships between words (Mikolov et al., 2013b).

To create high-dimensional (50 to 300) representations of words in an unsupervised manner from a large amount of text, Word2Vec (Mikolov et al., 2013b,a) is one of the most popular models to learn word embeddings. As illustrated in Figure 2.3, Word2Vec embeds words in a continuous vector space in such a way that semantically similar words are placed as nearby points to each other. Recently, it has been shown that the word vectors are able to capture many linguistic regularities. For example, vector arithmetic operations [vector (“Paris”) - vector (“France”) + vector (“Italy”) ] implement a vector that is very close to vector (“Rome”), and [vector (“king”) - vector (“man”) + vector (“woman”) ] is close to vector(“queen”). Mikolov et al. (2013b) defined two architectures for learning word embeddings, the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model.

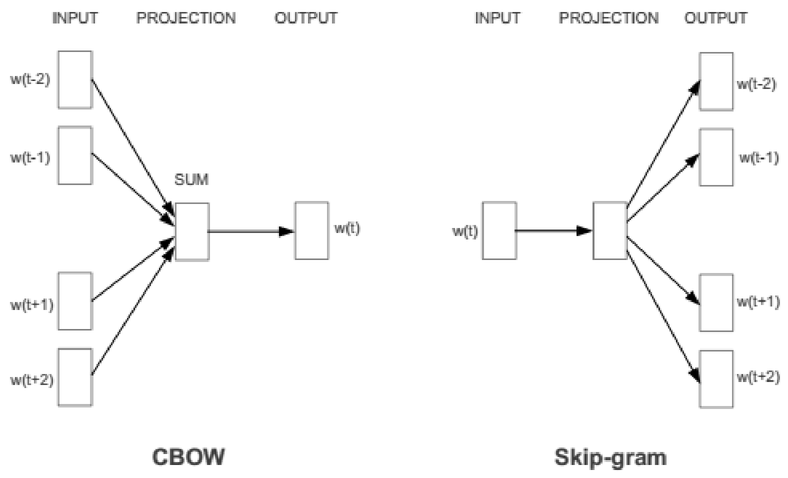

Figure 2.4: CBOW model (left) and Skip-gram model (right) from (Mikolov et al., 2013b).

**Continuous Bag-of-Words model (CBOW):** The CBOW model can predict the current word ,considering the previous N words and the next N words after it. The language models can only predict based on the past words, which gives the CBOW model an advantage over those language models. The structure of the model is shown in Figure 2.4 (left) where N=2.

**Skip-gram model:** Instead of using the surrounding words, skip-gram uses the centre word to predict the surrounding words as can be seen in Figure 2.4 (right).

### 2.5.3 GloVe

Unlike Word2Vec, GloVe (Pennington et al., 2014) takes advantage of two primary families of word vectors namely: global matrix factorization methods (e.g. LSA (Landauer et al., 1998)) and local context window-based methods (e.g. skip-gram (Mikolov et al., 2013b)). Again, GloVe is a count based model whereas Word2Vec is a prediction based model. GloVe first builds a co-occurrence matrix for the whole dataset. Then it is factorized to yield matrices for word vectors and context vectors.

Assume we have a corpus of W words. Then GloVe will generate a co-occurrence matrix M of $W * W$ dimension. Here the $i^{th}$ row and $j^{th}$ column represents the number of time words i and j have co-occurred in the sentences in that corpus. If we consider a single sentence- “The university awarded the students”, the co-occurrence matrix will look like this:

Table 2.1: Co-occurrence Matrix

| | the | university | awarded | students |
|---|---|---|---|---|
| the | 0 | 1 | 1 | 1 |
| university | 1 | 0 | 1 | 0 |
| awarded | 1 | 1 | 0 | 0 |
| students | 1 | 0 | 0 | 0 |

Then GloVe will determine the semantic similarity between words. Let us assume we have two reference words “ice” and “steam” and we want to measure the similarity of a different word with these two words.

Table 2.2: Co-occurrence probabilities for target words ice and steam with selected context words from a 6 billion token corpus. (Pennington et al., 2014)

| Probability and Ratio | k=solid | k=gas | k=water | k=fashion |
|---|---|---|---|---|
| $P(k\|ice)$ | $1.9 \times 10^{-4}$ | $6.6 \times 10^{-5}$ | $3.0 \times 10^{-3}$ | $1.7 \times 10^{-5}$ |
| $P(k\|steam)$ | $2.2 \times 10^{-5}$ | $7.8 \times 10^{-4}$ | $2.2 \times 10^{-3}$ | $1.8 \times 10^{-5}$ |
| $P(k\|ice)/P(k\|steam)$ | 8.9 | $8.5 \times 10^{-2}$ | 1.36 | 0.96 |

where $P(k|i)$ is defined as probability of getting the word $k$ with the word $i$, together in a sentence. We calculate this value by dividing the number of times those two words appeared together from the Co-occurrence matrix ($M_{ik}$) by the total number of times the word $i$ appears in the corpus ($M_i = \sum_{j=1}^{W} M_{ij}$). In Table 2.2, we present the probability of a set of third word ($k$) with respect to the words “ice” and “steam”. From the Table 2.2 we can observe that, if only one of the given words (i.e., ice and steam) is similar to the third word (when k=solid and k=gas) the value $P_{ik}/P_{jk}$ becomes either very high (k=solid) or very small (k=gas). If both of the words are similar to the third word or none of them are similar to the third word, the value $P_{ik}/P_{jk}$ becomes close to 1. We will define the probability function using the following formula:

$$F(w_i, w_j, w_k) = \frac{P_{ik}}{P_{ik}} \tag{2.14}$$

where $w_i$, $w_j$ and $w_k$ are word vectors. One of the goals of GloVe is to create vectors with meaningful dimensions using simple arithmetic. Firstly, we want the function F to present the ratio $\frac{P_{ik}}{P_{ik}}$ in the word vector space. Since vector spaces are inherently linear structures, we can consider the function F as the difference of two target words. For example, we can say that $w_{Madrid} - w_{Spain} + w_{Canada} = w_{Ottawa}$. So, we can modify the Equation 2.14 to,

$$F(w_i - w_j, w_k) = \frac{P_{ik}}{P_{ik}} \tag{2.15}$$

Now, we can see that the left-hand side of the equation is vector, and the right-hand side is scalar. To avoid this issue we take the dot product of the arguments,

$$F\Big(dot(w_i - w_j, w_k)\Big) = F\Big((w_i - w_j)^T w_k\Big) = \frac{P_{ik}}{P_{ik}} \tag{2.16}$$

Now we use two steps to determine F and simplify the equation:

- By taking the log of the probability ratios, we can convert the ratio into a subtraction between probabilities.
- By adding a bias term for each word, we can capture the fact that some words just occur more often than others.

Now, if we assume F has a homomorphism property between the additive $(R, +)$ and multiplicative groups $(R_{>0}, \times)$,

$$F\left((w_i - w_j)^T w_k\right) = \frac{F({w_i}^T, w_k)}{F({w_j}^T, w_k)} \tag{2.17}$$

From which we can solve the Equation 2.14 for a single entry as,

$$F({w_i}^T, w_k) = P_{ik} = \frac{M_{ik}}{M_i} \tag{2.18}$$

If we assume the function $F = exp$, we can rewrite the Equation 2.18 as,

$${w_i}^T w_k = \log(P_{ik}) = \log(M_{ik}) - \log(M_i) \tag{2.19}$$

This equation would exhibit the exchange symmetry if not for the $\log(M_i)$ on the right-hand side. However, this term is independent of $k$, so it can be absorbed into a bias $b_i$ for $w_i$. Finally, adding an additional bias $b_k$ for $w_k$ restores the symmetry.

$${w_i}^T w_k + b_i + b_k = \log(M_{ik}) \tag{2.20}$$

This is the core equation for GloVe embedding (Pennington et al., 2014). But, there is a simple problem in this equation. It considers all the co-occurrences with same importance. However, not all co-occurrences contains the same quality of information. So, instead of using 1 in the co-occurrence matrix, the authors created the following weighting function to improve the performance of function F.

$$weight(x) = \min\left(1, (x/x_{max})^{3/4}\right) \tag{2.21}$$

So, applying the weight function, the authors updated the Equation 2.20 as,

$$\sum_{ij=1}^{W} \Big(weight(M_{ij})\Big)\Big(w_i{}^T w_k + b_i + b_k - \log(M_{ij})\Big)^2 \tag{2.22}$$

### 2.5.4 FastText

Compared to other word embedding methods, FastText (Mikolov et al., 2018) is a new approach which can generate competitive results. Instead of a flat structure, FastText uses a hierarchical classifier, which is organized into a tree of different categories. Therefore, the depth in the tree of very frequent categories is smaller than the infrequent ones, which leads to further computational efficiency. A text is represented using FastText by a low dimensional vector which is obtained by summing vectors corresponding to the words which appear in the text. A low dimensional vector remains associated with each word of the vocabulary in FastText. This hidden representation is shared among all classifiers for different categories, and allows information about words learned for one category to be used by other categories. These kinds of representations which ignore word order are called a bag of words.

## 2.6 Sentence Embeddings

Sentence embeddings perform the same type of operations like word embeddings on the sentence level. It converts the sentences into some vectors of real numbers. Recently, many supervised and unsupervised sentence embeddings have been established. We explain some of them:

### 2.6.1 Word Mover Distance

Kusner et al. (2015) proposed the Word Mover Distance (WMD) embedding to measure the distance between two texts or two sentences. It measures the distance between two sentences by getting the cumulative sum of the minimum distance for each pair of words of the two sentences. The main advantage of WMD over the other similarity functions like

Cosine Similarity or Jaccard Similarity is that, WMD is capable of handling the synonyms as the same vector. Besides, WMD can leverage the ability of advanced word embeddings to overcome the basic limitations of distance measurements.

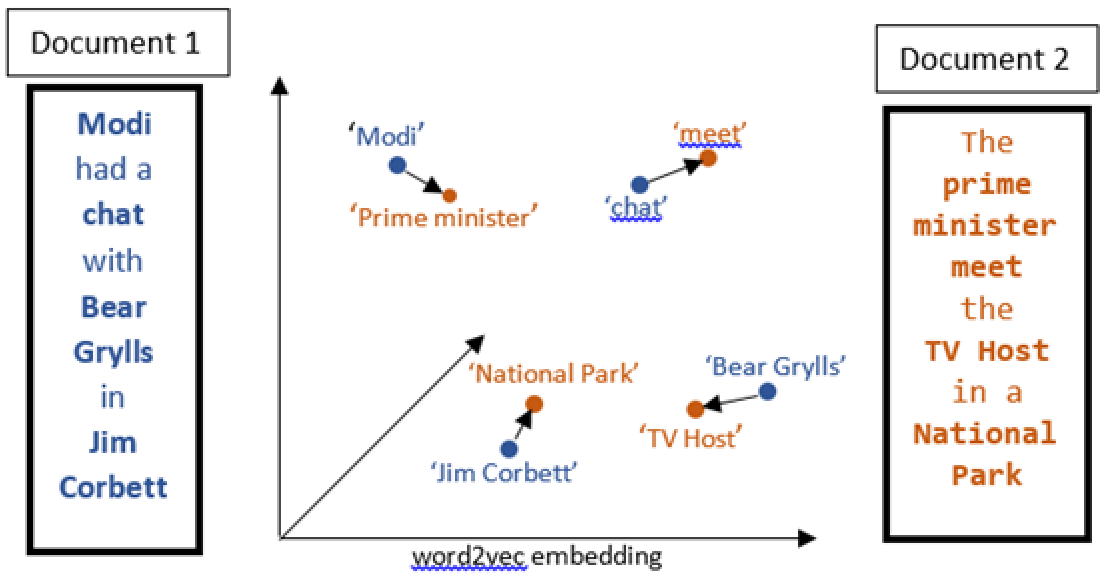

Figure 2.5: Concept of Word Mover Distance. (Saxena, 2018)

An example is shown in Figure 2.5. Let us assume we have two sentences:

Sentence1= Modi had a chat with Bear Grylls in Jim Corbett.

Sentence2= The prime minister meets the TV host in a national park.

Here, we can see, even though the two sentences represent the same topic, they do not have any common words except the stopwords. WMD can measure the distance in such situation where we do not have any common words. It can work with the assumption of having similar vectors for the same kind of words, which makes it more effective than the other similarity measures. It can be explained with another example: vector (Madrid) — vector (Spain) + vector (Canada) is almost same as vector (Ottawa), because the tuples of the vectors represent the country and its capital.

### 2.6.2 Smooth Inverse Frequency (SIF)

Smooth inverse frequency embeddings were originally conceived by Arora et al. (2017). The authors present a probabilistic motivation for the inverse frequency weighted continuous bag-of-words model. Taking the average of the word embeddings in a sentence tends to give more weight to words, which is quite unnecessary. SIF solves this issue in two ways:

**Measuring Weights**: SIF considers the weighted average of the word embeddings of a sentence; which can be represented by the following formula:

$$\frac{\alpha}{\alpha + f(w)} v_w \tag{2.23}$$

where $\alpha$ is a constant that is usually considered as 0.001, $f(w)$ is the estimated frequency of each word of the reference sentence in the corpus, and $v_w$ is the embedded word vector. We have used GloVe word embedding to get the word vectors. We use the model provided by the authors to predict the frequency. Assume we have a corpus of $W$ words. Then we can calculate $f(w)$ for any word $w$ using the following equation

$$f(w) = \frac{count(w)}{W} \tag{2.24}$$

Then for all the words for a sentence $S$, we concatenate the vectors, and create the sentence embedding matrix for that sentence.

**Remove the Common Components**: After that, the main components of the resulting embeddings are being computed for a collection of sentences. It then subtracts their projections on their first principal component from this embeddings. This process removes the variation related to the frequency and syntax that is irrelevant to the context semantically.

SIF ignores the stopwords such as the prepositions and articles, and preserves the information that represents the most semantic meaning of the sentences.

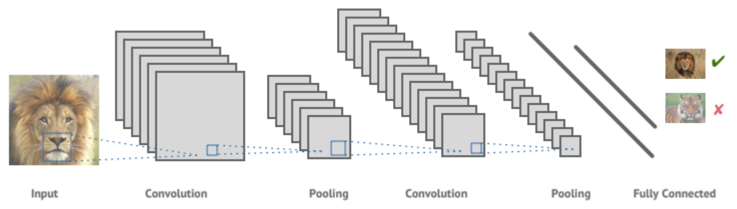

Figure 2.6: A Convolutional Neural Network. (Britz, 2015)

## 2.7 Neural Network

### 2.7.1 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) was first introduced by LeCun et al. (1998). It uses different layers based on convoluted filters to be applied on different local features (Kim, 2014). In recent years, CNN has become popular in different machine learning problems, i.e., pattern recognition, image classifications, sentence classifications. CNN is basically the combination of different layers of convolutions with activation functions like ReLU or tanh applied to the results. We apply convolutions over the input layers to generate the output. There might be thousands of filters in each layer of a CNN structure based on its applications, and the filters of the output is connected with the input of the next layer. The basic structure of a CNN is shown in Figure 2.6.

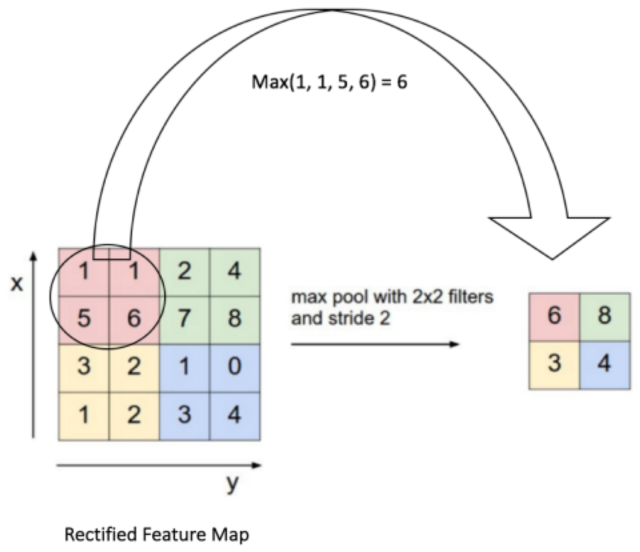

Figure 2.7: Max Pooling. (Ujjwalkarn, 2016)

**Max Pooling**

With the convolution, spatial pooling is also included with a convolutional layer. It does two things, firstly reduces the dimension to help the network with the operations, and secondly keeps all the important information. Pooling can be different types; for example Max Pooling, Average Pooling, Min Pooling etc. In case of max pooling (Figure 2.7), we define a spatial pooling in such a way that it will take the largest element from the rectified feature map within that window. Max pooling has shown better results compared to the others.

### 2.7.2 Recurrent Neural Network (RNN)

In a traditional neural network, it is assumed that all the inputs and outputs are not dependant on each other, which is not a good idea for many tasks. To predict the next word in a sentence, we need to know which words came before it. Recurrent neural networks are generally good for data if the previous inputs and the current input in a sequence have a relation between them. As Natural Language Processing (NLP) is a classical problem on sequential data, the RNNs have shown great success in many NLP tasks in the last few years such as topic modeling, syntax parsing, image captioning, dialog generation, machine translation, summarization and question answering etc.

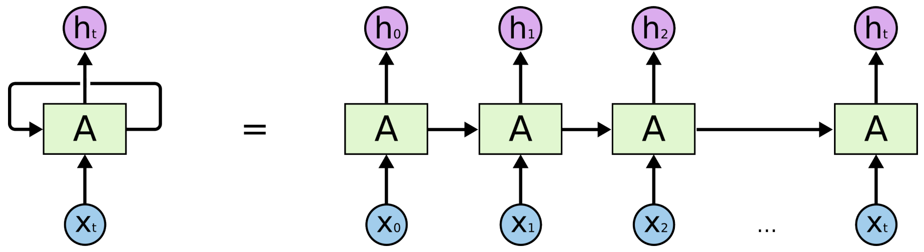

Figure 2.8: An unrolled recurrent neural network. (Olah, 2015)

As shown in Figure 2.8, by unfolding a RNN at the $t^{th}$ time-step, the network takes two inputs: the $t^{th}$ input vector $x_t$ (Normally, the embedded input word goes through an RNN as $e(x_t)$ at every time-step) and the hidden state from the last time-step $h_{t-1}$. From these vectors, it computes the hidden state of the current time-step $h_t$. This process is repeated until all inputs are processed in sequence. Considering the RNN as function $f$, the formulation is:

$$h_t = f(x_t, h_{t-1}) \tag{2.25}$$

### 2.7.3 Long Short Term Memory (LSTM)

One of the key properties of RNNs is their ability to connect past information to the present situation. Sometimes, it is only required to look at recent information to describe the current situation. For instance, if we consider a language model that tries to predict the last word based on the previous ones in the sentence "What is the duration of a football ***match***." It can be easily predicted that the next word should be ***match***. Here the gap between the relevant information (football) and the information that is required (match) is small. In this type of cases, RNNs can learn to use past information. In contrast, if we try to predict the last word of the sentence, "I grew up in Bangladesh, so I can speak fluently in ***Bengali***." Here the recent information indicates that the next word should be a language. But if we want to be certain about the language, we need the information about the country. For this case ***Bangladesh*** is needed, which is further back from the last word. Unfortunately, with the increase in the gap, RNNs cannot gather enough information to connect the words.

LSTM networks (Hochreiter and Schmidhuber, 1997) – are a specialized type of RNN which can avoid the long-distance dependencies problem (Bengio et al., 1994). They have been widely used on a large variety of NLP problems in recent times and worked exceptionally well.

In comparison to the structure of an RNN, an LSTM includes a memory cell $c$, an input gate $i$, a forget gate $f$, and an output gate $o$. These gates and memory cells can avoid the long term dependencies problem. We can formulate the LSTM as a function $f$, as follows:

$$h_t = f(x_t, h_{t-1}) \tag{2.26}$$

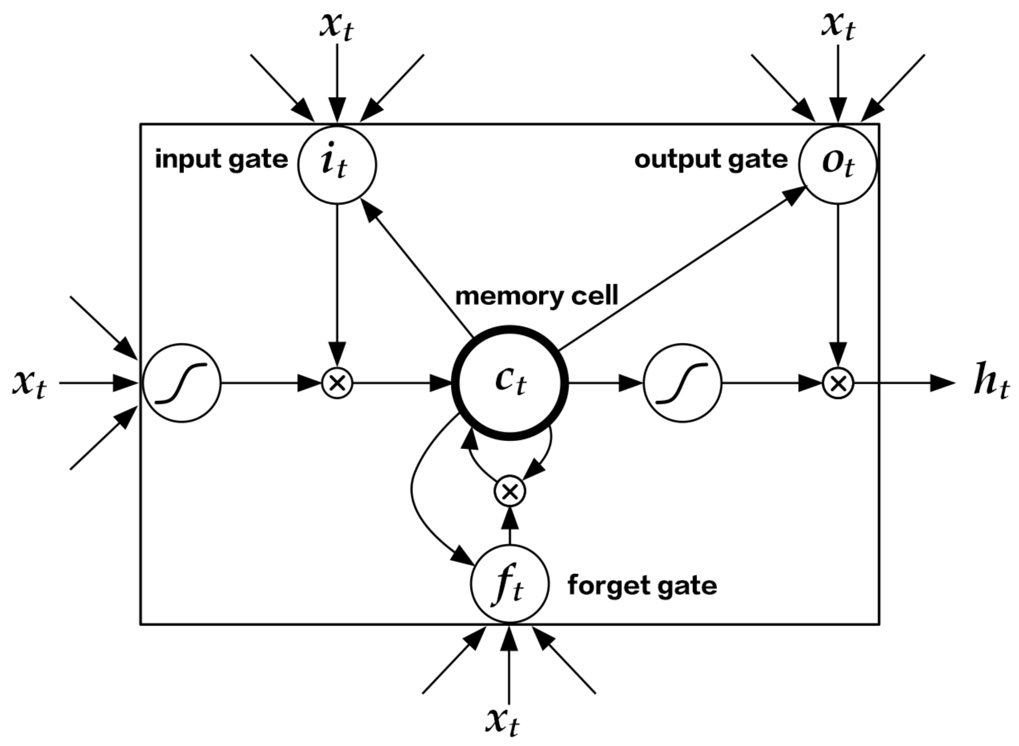

Figure 2.9: LSTM at time-step $t$ (Hochreiter and Schmidhuber, 1997)

Where, $f$ contains the following formulations (Hochreiter and Schmidhuber, 1997),

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{2.27}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{2.28}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.29}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{2.30}$$

$$h_t = o_t \odot \tanh(c_t) \tag{2.31}$$

In the above equations, $i_t, f_t, c_t, o_t$ stand for the input gate, forget gate, memory cell and output gate respectively. $W$ and $b$ denote model parameters, where $tanh$ is the hyperbolic tangent, and $\odot$ denotes the element-wise product operation as shown in Figure 2.9.

### 2.7.4 Gated Recurrent Unit (GRU)

GRU (Cho et al., 2014b) is related to an LSTM, but both use a different gating mechanism to prevent long-distance dependencies problems. GRUs are relatively new, have a less complex structure, train faster, are computationally more efficient and perform better than an LSTM on less training data (Chung et al., 2014). GRU also controls the flow of information like the LSTM unit, but without using a memory unit, and combines the forget and input gates into a single "update gate". GRU just exposes the full hidden content without any control (Cho et al., 2014b). A GRU layer is quite similar to an LSTM layer; the following equations are for a single GRU layer (Cho et al., 2014b):

$$z = \sigma(x_t U^z + s_{t-1} W^z) \tag{2.32}$$

$$r = \sigma(x_t U^r + s_{t-1} W^r) \tag{2.33}$$

$$h = tanh(x_t U^h + (s_{t-1} \bigodot r) W^h) \tag{2.34}$$

$$s_t = (1 - z) \bigodot h + z \bigodot s_{t-1} \tag{2.35}$$

In the above equations, a GRU has two gates, one reset gate $r$, and an update gate $z$. The reset gate defines the process of combining the new input with the past memory, and the update gate determines how much of the past memory to keep as shown in Figure 2.10. For all recurrent units the general formulation is,

$$h_t = f(x_t, h_{t-1}) \tag{2.36}$$

### 2.7.5 Activation Functions

In neural network, the activation function of a node defines the possible output from that node for a single input or a set of inputs. Based on the applications we use different

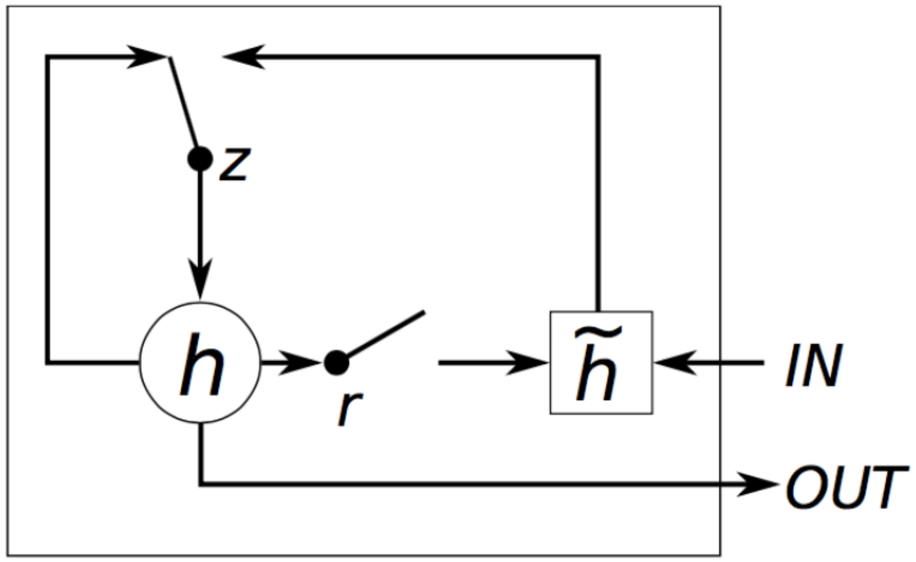

Figure 2.10: GRU Gating Mechanism (Chung et al., 2014)

activation functions; we describe some of the activation functions related to this thesis here:

- **Sigmoid Function:** The sigmoid function (Han and Moraga, 1995) limits the output of the node to a range of 0 and 1. So, it is mostly used for predicting the probaility as the output. The equation for the sigmoid function can be defined as below:

$$sigmoid(x) = \frac{e^x}{1+e^x} \tag{2.37}$$

- **Tanh Function:** Tanh function (Fan, 2000) limits the output of the nodes to a range between -1 to 1. Which means the negative values are given negative values, and zero inputs are mapped near zero in the tanh graph. The formula for tanh function is defined as follows:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.38}$$

- **SoftMax Function:** The Softmax function (Bridle, 1990) normalizes the input values into some vectors of values, and then provides a probability distribution whose total sum is equal to 1. The range of the output values is between 0 and 1. If we have a set of values $X = [x_1, x_2, \ldots, x_n]$, then for any $x_i$, the softmax function can be defined using the following formula:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.39}$$

- **Rectified Linear Unit (ReLU):** ReLU (Nair and Hinton, 2010) is one of the most used activation functions in deep learning as it can be used with almost all of the neural networks. ReLU is half rectified from bottom, which implies that it provides output as zero when the node has a value less than or equal to zero. The range of the ReLU function is from 0 to infinity ($\infty$). The formula can be defined as:

$$ReLU(x) = \max(0, x) \tag{2.40}$$

## 2.8 Sequence-to-sequence (seq2seq) Model

The sequence-to-sequence model (Sutskever et al., 2014b) was developed by Google in 2014. Sequence-to-sequence learning is used to convert a sequence from one domain to a different domain. For the text summarization problem, this model is mainly used to establish relation between two fixed-length texts where their lengths may differ. For example, translating a five words sentence from English to another language might give us six or seven words in that language. Regular LSTM networks cannot work with this type of example with accuracy. From that thinking, the idea of developing the sequence-to-sequence model was initiated.

We can represent the basic structure of this model as Figure 2.11:

The model has three sections: encoder, intermediate (encoder) vector, and decoder.

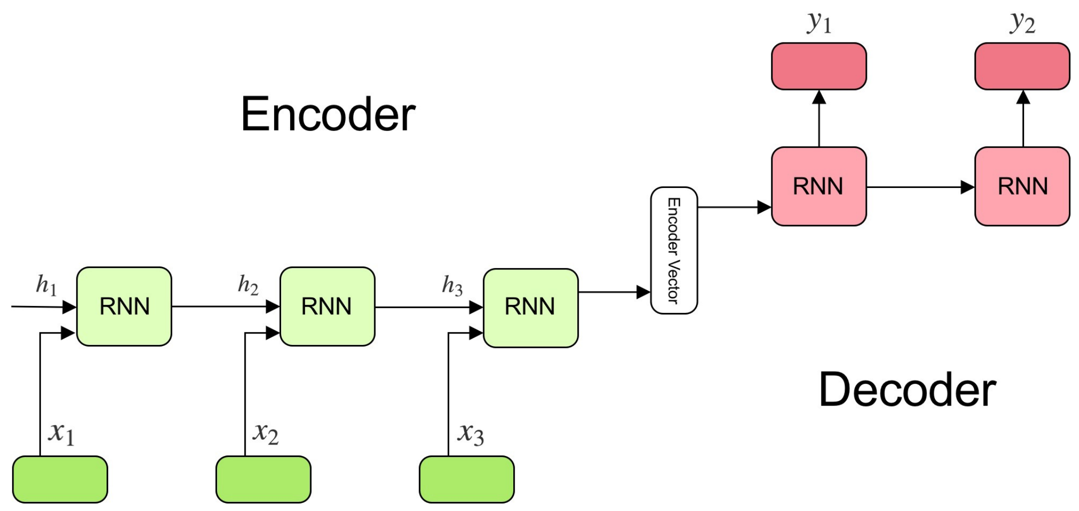

Figure 2.11: Basic structure of sequence to sequence model. (Kostadinov, 2019)

**Encoder:** The encoder is a collection of several recurrent units (LSTM or GRU for better performance) where each of them accepts a single element of the input sequence, collects information for that element, and propagates forward. For the question-answering problem, at first, the stopwords are removed from the question, and then the input sequence is built with all the remaining words. In that case, we can represent a word as $x_i$, where $i$ is the sequential order of the particular word. For any time $t$, the hidden state $h_t$ for the word $x_i$ can be calculated using the following formula:

$$h_t = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t) \tag{2.41}$$

This simple formula represents the result of a simple RNN. Here, the appropriate weights to the previous hidden state $h_{t-1}$ and the input vector $x_t$ are applied only.

**Encoder Vector:** This is the last hidden state produced by the encoder. It can be calculated using Equation 2.41. It is also the initial hidden state for the decoder of this model. This vector summarizes the information from the input elements to assist the decoder in predicting the outputs.

**Decoder:** A collection of several recurrent units where each of them predicts an output $y_t$ at time $t$. Each of the recurrent units accepts a hidden state from the previous unit, and produces both the output and its own hidden state. In question-answering problem, the output sequence can be defined as a list of words in the answer. Here, the words can be represented by $y_i$, where $i$ is the sequential order of the particular word. The hidden states $h_i$ can be computed using the following formula:

$$h_t = f(W^{(hh)} h_{t-1}) \tag{2.42}$$

The output $y_t$ on time $t$ can be computed by the following equation:

$$y_t = softmax(W^s h_t) \tag{2.43}$$

here, the softmax creates a probability vector that can help to generate the output. If we have $p$ numbers as $[n_1, n_2, \ldots, n_p]$ the softmax function can be defined as below:

$$softmax(n_i) = \frac{e^{n_i}}{\sum_{j=1}^{p} e^{n_j}} \tag{2.44}$$

The result will be always in range of [0,1].

The main quality of this model is the ability to map sequences of different lengths to each other. It is clear that the inputs and outputs are not related to each other. So in most of the cases, their lengths become different. This creates a new range of solutions in the field of summarization task.

## 2.9 Multi Sentence Compression

The multi-sentence compression (**MSC**) technique is a possible convenient solution to the summarization task. This method usually takes a collection of related sentences as input, and generates a single output sentence by merging the source sentences that relate to the same information. In this process, it tries to retain the most important information, and also maintains the grammatical structure of the generated sentence. MSC (popularly known as sentence fusion by Barzilay and McKeown (2005)) is a text-to-text generation process where a novel sentence is generated as a summary of a set of related sentences. On the other hand, lexical paraphrasing aims at replacing some selected words from the original text with some new words while preserving the meaning of the source documents. A good lexical substitution for a target word needs to have semantic similarity with the target word, and should be compatible with the given context (Melamud et al., 2015). For example, the sentence "Jack composed these verses in 1995" could be lexically paraphrased into, "Jack wrote these lines in 1995" without changing the actual meaning of the source sentences.

## 2.10 Reinforcement Learning (RL)

Recently reinforcement learning has become popular in the field of Natural Language Processing. The basic concept of reinforcement learning came from human actions, where the process is to take the best action from previous experiences. The main difference between supervised learning and reinforcement learning is, reinforcement learning works as a chain of decision making. We make a decision based on the inputs. The decision of our current step will affect our next step as we consider the past experiences while making a decision. But in supervised learning, whatever decision we make, it does not affect the decision making in the following steps as we do not consider the past experiences in supervised learning, rather it depends only on the current step.

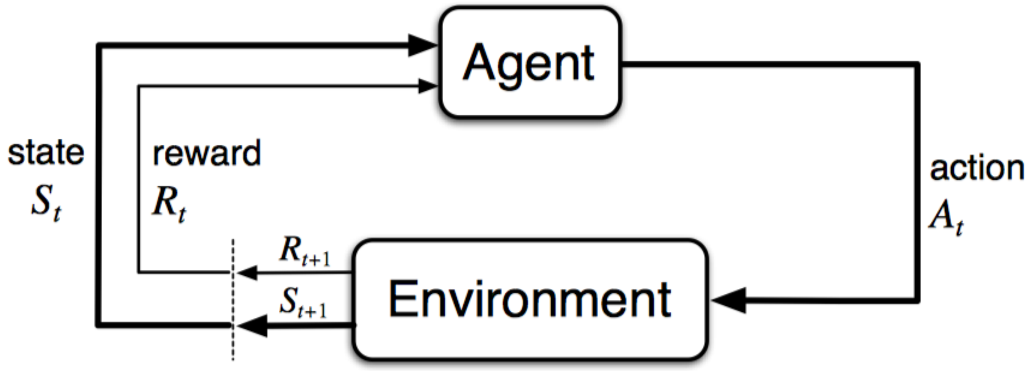

Figure 2.12: Agent-Environment relational loop

**Agent:** The basic structure of a reinforcement learning model includes an agent which will interact with the environment in different steps, and will receive a reward based on it's action. This transition lead the agent towards a new state where it will use its previous learning to take the decision. A canonical agent-environment feedback loop is shown in Figure 2.12:

**Reward:** The accuracy of a RL model highly depends on the assumption of the reward score, which can be defined as the ultimate goal of the agent. A reward is provided by the environment based on the actions of the agent. When an agent takes some decisions, the environment provides it with some results or feedback, which could be both positive and negative. In one word, we call this feedback as the reward. Based on the positive or negative reward, the agent decides which path it has to take to achieve the ultimate goal, to solve the problem while getting the maximum reward from the environment.

**Example:** In Figure 2.13 we demonstrate a sample Reinforcement Learning problem. In

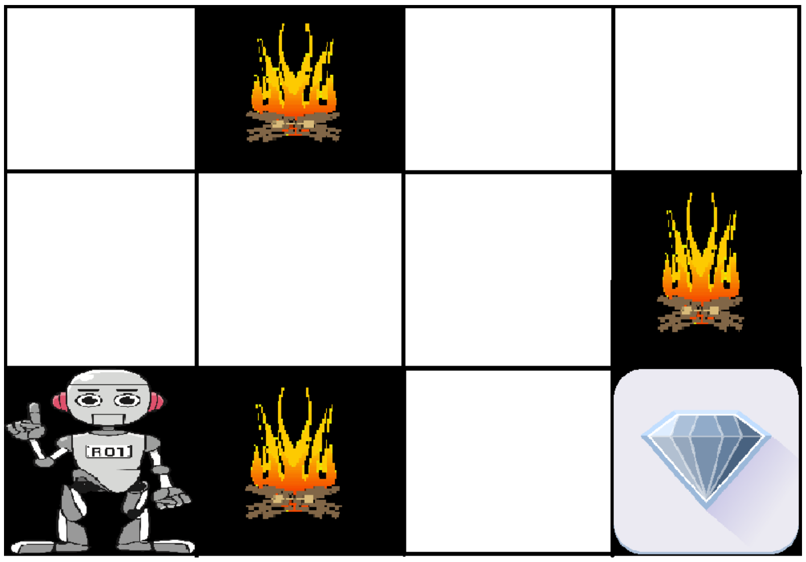

Figure 2.13: Reinforcement Learning Method's example. (Bajaj, 2018)

this problem, we have a robot that is our agent, and the diamond is the ultimate goal which the agent wants to achieve with many hurdles in between. The agent has to find the most optimum way to get the diamond while avoiding the fires. The robot tries each possible combination of steps to reach the diamond, and learn the process. With each step, it gets feedback as a reward score. If it gets a positive reward score, that means it has taken the right step; and if it gets a negative reward score, it implies that it has taken a wrong step. The absolute reward is calculated when it gets the diamond, which ends its task. Based on the reward in each step the robot tries to learn which step is the optimum option for it to get to the ultimate goal, and it makes decision in the next steps based on it's previous learning.

## 2.11 Neural Machine Translation (NMT)

The conversion process from one language to another language is called Machine Translation (MT). The input language to the MT system is referred to as the source language, while the output language is the target language. In summary, MT is the task of converting a sequence of words from the source language to a sequence of words of the target language, preserving the meaning of the source sentence. It is one of the most significant and well-known research topics in the field of Natural Language Processing (Neubig, 2017).

Statistical Machine Translation (SMT) techniques have been used (Brown et al., 1993) for early automatic MT systems. But these statistical machine translation models pose many limitations. Pre-processing techniques of SMT rely heavily upon processes like word segmentation, word alignment, tokenization, syntactic parsing and rule-extraction. However, the problem is that all possible linguistic variations cannot be covered, and all global features cannot be used by human-designed features. The recent development of deep learning presents new and better solutions compared to previous approaches to these problems of Statistical Machine Translation mentioned previously. Neural Machine Translation (NMT) (Sutskever et al., 2014a) does not need any pre-designed features. The goal of NMT is to design a fully trainable model, where in order to maximize its performance, every component is tuned based on a large-scale training data.

Let us consider a sequence of words as the rawest representation of a sentence. Then it can be said that a fully trainable NMT model $\mathcal{M}$ starts from a raw representation of a

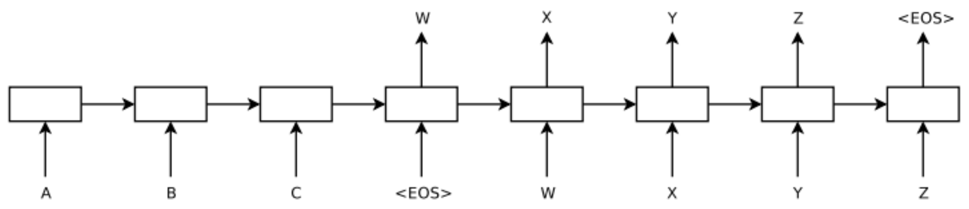

Figure 2.14: Sequence to Sequence Learning with Neural Networks (Sutskever et al., 2014a)

source sentence, and finishes by generating a raw representation of a target sentence. In a vocabulary with a fixed number of words, each word is represented by its integer index. For instance, in the vocabulary $V$ of English words, which is sorted according to their frequency of appearance in a training corpus, the very first frequent word is represented as an integer 1. Let, $X = (x_1, x_2, \ldots, x_N)$ be a source sentence, and $Y = (y_1, y_2, \ldots, y_M)$ be a target sentence where $N$ and $M$ are not necessarily the same number of words (Sutskever et al., 2014a). The NMT model $\mathcal{M}$ attempts to find an output sequence $Y$ which is able to maximize the conditional probability given an input sequence $X$:

$$\underset{\mathbf{Y} \in V}{arg\ max}\ P(\mathbf{Y}|\mathbf{X}) \tag{2.45}$$

The sequence-to-sequence network has gained popularity with Natural Language Processing researchers to solve the problem of NMT (Sutskever et al., 2014a; Bahdanau et al., 2014).

For example, according to Figure 2.14, we have “ABC” as the input and “WXYZ” as the output. The two sequences have different lengths. So the question is, how does sequence-to-sequence solve that problem of different sequence lengths? The solution is to develop two different models, which consist of two separate recurrent neural networks named **Encoder** and **Decoder**, respectively.

### 2.11.1 Encoder-Decoder Framework

The Encoder-Decoder framework (Cho et al., 2014b) has found a solution for the mapping of one sequence to a sequence with different lengths. The encoder turns a source sequence of words into a fixed-size feature vector, which is then to be decoded by a decoder as a target sequence by maximizing the predictive probability. The encoder and the decoder are both typically implemented using a simple RNN, LSTM, or GRU.

#### Encoder

An encoder can encode the sequence of a sentence in three steps:

1. Considering one-hot vector representation of a word where each word $x_i$ in the source $x = \{x_1, x_2, \ldots, x_N\}$ is represented as a vector $w_i \in \{0,1\}^{|V|}, i = 1, 2, \ldots, N$ where $w_i$ has same number of dimension as the vocabulary $|V|$, and has an element of one corresponding to the location of the word in the dictionary and zero elsewhere.

2. There are a couple of limitations with the one-hot vector representation. Firstly, The dimension of each word vector is enormous. Also, capturing semantic relationships between words in a source sentence is complicated. Hence, it is advantageous to convert the one-hot vector into a low-dimensional semantic space as a dense vector with fixed dimensions. For instance, $s_i = Cw_i$ for the $i^{th}$ word, with $C \varepsilon R^{K \times |V|}$ as the projection matrix, and $K$ is the dimensionality of the word embedding vector, and $|V|$ is the size of the fixed vocabulary.

3. The source sequence of words is then encoded using RNN:

$$h_i = \varnothing_\theta (h_{i-1}, s_i) \tag{2.46}$$

where, $h_0$ is a zero vector, $\varnothing_\theta$ is a non-linear activation function (e.g. sigmoid, ReLU, tanh), and $\mathbf{h} = \{h_1, \ldots, h_N\}$ is the sequential encoding of the first $N$ words from the input source sequence. After the last word's continuous vector $s_N$ is projected, the RNN's internal state $h_n$ represents the summary of the whole source sentence.

### Decoder

The decoder aims at maximizing the probability of the next possible correct word in the target language sequence. We can build a decoder by following steps:

1. For any time-step $i$, given a summary vector (or encoding vector) $c$ of the sequence for the source sentences, the $i$-th word $u_i$, the hidden state $z_i$, the next hidden state $z_{i+1}$ are computed as:

$$z_{i+1} = \phi_{\theta'}(c, u_i, z_i) \tag{2.47}$$

   where $\phi_{\theta'}$ is a non-linear activation function, and $c = qh$ is the context vector of the source sentence sequence, $c$ can be described as $c = h_T.u_i$, which denotes the $i^{th}$ word from the target language sequence, and $u_0$ denotes the beginning of the target language sequence, which indicates the beginning of the decoding. Lastly, $z_0$ is an all-zero vector, and $z_i$ is the RNN hidden state at time-step $i$.

2. Calculating the probability $p_{i+1}$ for the $(i+1)$-th word in the target language sequence is described as:

$$p(u_{i+1}|u_{<i+1}, x) = softmax(W_s z_{i+1} + b_z) \tag{2.48}$$

   where, $W_s z_{i+1} + b_z$ scores each possible words in the vocabulary $|V|$, and then the scores are normalized using softmax, which converts the scores into probability $p_{i+1}$ for the $i+1$-th word in the whole target sequence.

3. The cost is computed according to $p_{i+1}$ and $u_{i+1}$.

4. Repeat the steps 1-3 until all the words have been processed, which usually terminates by a $< eos >$ token.

### 2.11.2 Training: Maximum Likelihood Estimation (MLE)

Following the development of the neural translation model, sometimes it needs to be trained using parallel data (source sentences and target sentences). The previously described encoder-decoder model uses Maximum Likelihood Estimation (MLE) (White, 1982), which is a standard statistical technique for training. Let us consider a parallel corpus $D$, in which each sample in the corpus is a pair $(X^n, Y^n)$ of the source and target sentences. Each of these sentences is a sequence of integer indices based on the vocabulary set $V$, which represents the sequence of one-hot vectors. If we consider any pair of words from the sentences, the NMT model can compute the conditional log-probability of $Y^n$ given $X^n$: $\log P(Y^n|X^n,\theta)$, where, $\theta$ is the training parameter, and the log-likelihood of the whole training corpus can be described as,

$$L_t(\theta) = \sum_{(x,y)\in D} log\, P(\mathbf{Y}|\mathbf{X};\theta) \tag{2.49}$$

$$P(\mathbf{Y}|\mathbf{X};\theta) = \prod_{t=1} P(y_t|y_{1:t-1},\mathbf{X}) \tag{2.50}$$

The machine translation process is actually the process of converting a source sentence from one language to another sentence in the target language without changing the meaning. For this process, it uses a pre-trained model. In the decoding step, there are different strategies like greedy search and beam search to generate the next word in the output sequence.

### 2.11.3 Attention Mechanism

It is seemingly unreasonable to encode all the information of a sentence with a fixed dimensional vector representation regardless of the length of the sentence. In theory, algorithms like LSTMs are supposed to be able to deal with this. But in practice long-range dependency issues still create problems due to the vanishing gradient problem (Hochreiter, 1998).

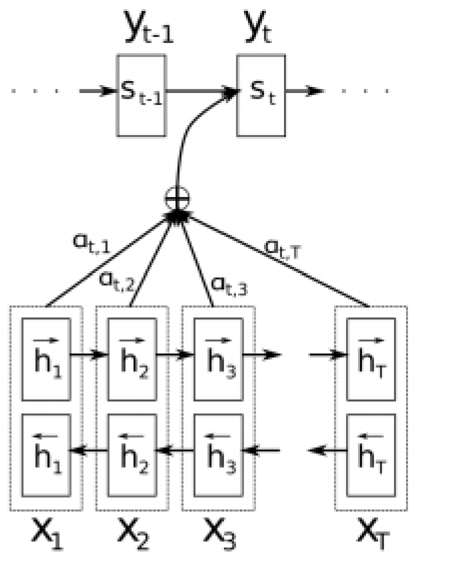

Figure 2.15: Attention Model (Bahdanau et al., 2014)

While processing a source input sentence, the model usually pays more attention or concentration to the parts in the source sentence which has more relevance to the output translation, which is currently in the decoding stage. But in the source sentence, the focus changes in the process of the translation. With a fixed dimensional vector, all the words from the source sentence are treated as equals. This is not reasonable in any circumstances. That is why Bahdanau et al. (2014) proposed attention mechanism in NMT (see Figure 2.15), which can decode based on different parts of the context sequence to tackle the difficulty of feature learning for long sentences (Wu et al., 2016b). With an attention mechanism, it is not necessary to encode the full source input sentence into a fixed-length vector anymore. Instead, the model allows the decoder to "attend" (focus on) the different parts of the source sentence at each time-step of the output generation process. In the case of a decoder with attention, the $z_{i+1}$ is computed as:

$$z_{i+1} = \phi_{\theta'}(c_i, u_i, z_i) \tag{2.51}$$

During each time-step in the decoder, instead of using a fixed context, a distinct context vector $c_i$ is used for processing word $y_i$. In short, this context vector $c_i$ is the weighted sum of the RNN hidden states ($h_j$) of the encoder. The weight $a_{ij}$ which denotes the strength of attention of the $i^{th}$ word in the target language sentence to the $j^{th}$ word in the source sentence.

$$c_i = \sum_{j=1}^{N} a_{ij} h_j \tag{2.52}$$

$$a_i = [a_{i1}, a_{i2}, \ldots, a_{iN}] \tag{2.53}$$

$$a_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{N} exp(e_{ik})} \tag{2.54}$$

$$e_{ij} = align(z_i, h_j) \tag{2.55}$$

where, *align* is an alignment model that measures how well the inputs around the position $j$ and the output at position $i$ match. The score is based on the RNN hidden state $z_{i-1}$ and the $j^{th}$ annotation $h_j$ of the input sentence (Bahdanau et al., 2014). In the conventional alignment model **hard alignment** is used, which means each word in the target language explicitly corresponds to one or more words from the source language sentence. On the other hand, if any word in the source input sentence has relation to any word in the target language output **soft alignment** is used. The output of the alignment model is a real number which represents the strength of the attention. The decision to use a hard or soft alignment model depends entirely on the problem.

### 2.11.4 Pointer Network

Pointer network (Vinyals et al., 2015) is a combination of sequence-to-sequence model with the attention mechanism. It is highly used in the summarization task. The difference of pointer network with a simple attention based translation model is, it does not directly translate one sentence into another, rather it yields a succession pointers to the word vectors of the input sentence. A simple workflow of a pointer network is shown in Figure 2.16.

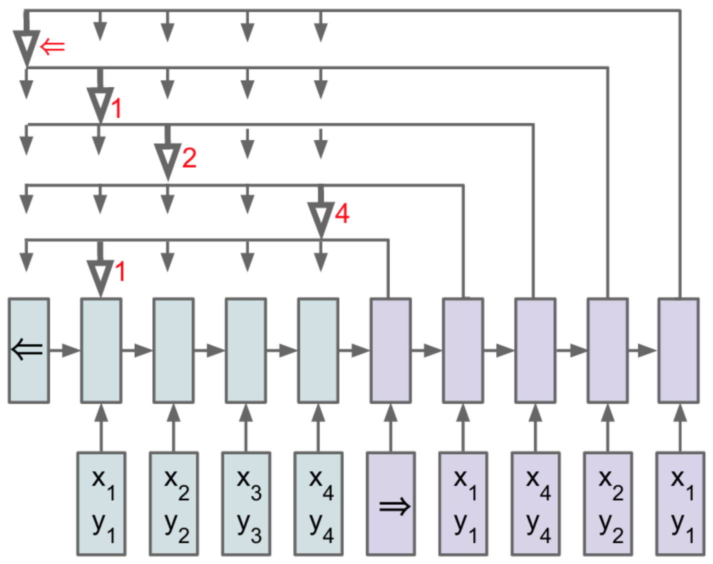

Figure 2.16: Pointer Network Architecture (Vinyals et al., 2015).

Here, an encoding RNN converts the input sequence to a code (blue) that is fed to the generating network (purple). At each step, the generating network produces a vector that modulates a content-based attention mechanism over inputs. The output of the attention mechanism is a softmax distribution with dictionary size equal to the length of the input. If we have a sequence of integer indices $C = (c_1, \ldots, c_m)$ and a sequence of input vectors $P = (p_1, \ldots, p_n)$, a pointer network will calculate the probability of index $c_i$ using the following formula:

$$u_i = v^T tanh(W_1 + W_2 d_i) \tag{2.56}$$

$$P(c_i | c_1, \ldots, c_{i-1}, P) = softmax(u_i) \tag{2.57}$$

here, $j \in (1, \ldots, n)$ and the softmax function normalizes the vector $u_i$ of length $n$ to an output distribution of different inputs. Here $v$,$W_1$ and $W_2$ are different learnable output parameters of the model. $v^T$ is the transpose of the input matrix.

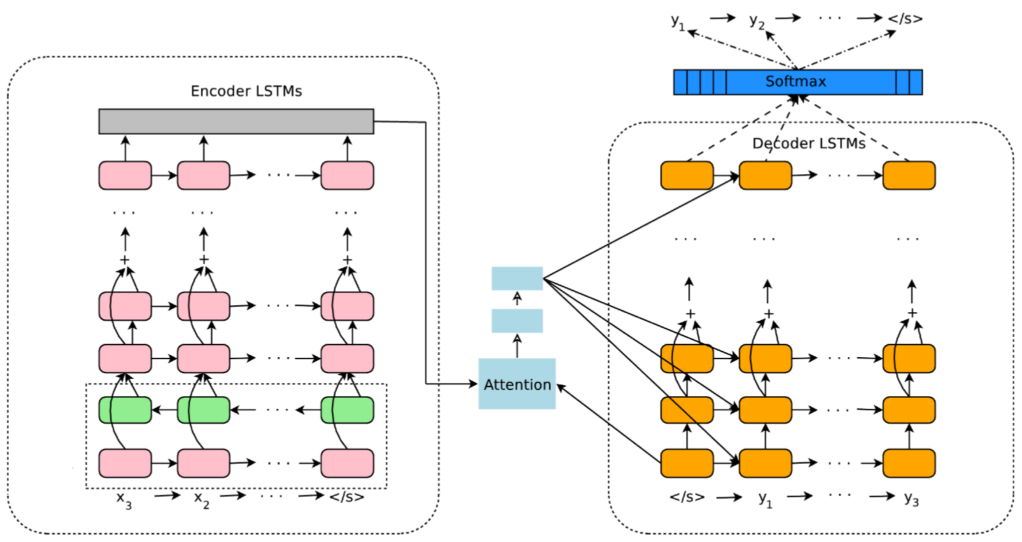
Encoder LSTMs
Decoder LSTMs
Softmax
Attention
$x_3$ → $x_2$ → ··· → </s>
</s> → $y_1$ → ··· → $y_3$
$y_1$ → $y_2$ → ··· → </s>

Figure 2.17: Google's Neural Machine Translation (NMT) Model. (Wu et al., 2016a)

### 2.11.5 Greedy 1-Best Search

Greedy 1-Best Search is very useful in machine translation, if we simply require the best output according to the model. It follows the method of a simple BFS or DFS algorithm. The difference of Greedy 1-best search with regular BFS or DFS algorithm is: BFS and DFS algorithms consider each possible option, and go through all of the options in the list. On the contrary, the greedy 1-best search calculates the probability of achieving the goal $p_t$ at every time-step, then selects the word which gives the highest probability (1-best), and uses it to predict the next word in the sequence (Neubig, 2017). But, sometimes it cannot guarantee to provide the output with the highest probability, where using the $n$-best words in each step could be a possible solution.

### 2.11.6 Beam Search Algorithm

Beam Search (Bennell and Song, 2010) is mostly used when the possible solution might be too large for a NLP application. It is a heuristic search algorithm which explores a graph by expanding it to find the most probable node in a limited set. It is mostly useful when we are short of memory where it can provide a suitable solution.

Beam search uses a breadth first search (BFS) (Beamer et al., 2013) algorithm to construct a search tree, and sorts the nodes according to a heuristic cost at each level of the tree. The main difference between beam search and greedy search is, unlike greedy search it uses the $b$ best words in each step where $b$ is the width of the beam size (sometimes called beam search size). Therefore, in the next level, $b$ best nodes with highest scores are expanded. Through this process the space and time requirements are significantly reduced. However, beam search does not guarantee a global optimum solution. If the end-of-sentence token $< eos >$ is generated while decoding, that means either the search process has stopped or the maximum length of the sentence has reached.

Figure 2.17 is an example of the Google's recent machine translation framework (Wu et al., 2016a) that uses almost all of the techniques described. We can divide the model into three parts; the left side is the encoder network, the right side is the decoder network, and in between these two is the attention module. We have a total of eight layers for each of the encoder and decoder networks. All of the encoder layers are uni-directional except the bottom layer. The bottom layer is bi-directional where the pink nodes gather information from left to right, and the green nodes gather information from right to left. While training the model, the bottom bi-directional encoder layers are computed in parallel at first. Once the computing is finished, on separate GPU the uni-directional encoder layers starts computing. Here, the softmax layer is also partitioned and placed on multiple GPUs.

## 2.12 Transformer Model

Similar to most NMT models, the Transformer model (Vaswani et al., 2017) is also based on the popular encoder-decoder structure. The difference between the transformer model (Vaswani et al., 2017) and any other NMT model is being entirely based on attention mechanisms and dot-products, which contain fully connected layers for both the encoder and the decoder sides. Though the model follows the actual architecture for a standard encoder-decoder model, the most commonly used recurrent layers in encoder-decoder architectures are replaced by the multi-head self-attention. In short, it can be said that this model is computationally cheaper than any other NMT models. The basic structure of the transformer model (Vaswani et al., 2017) is given in Figure 2.18.

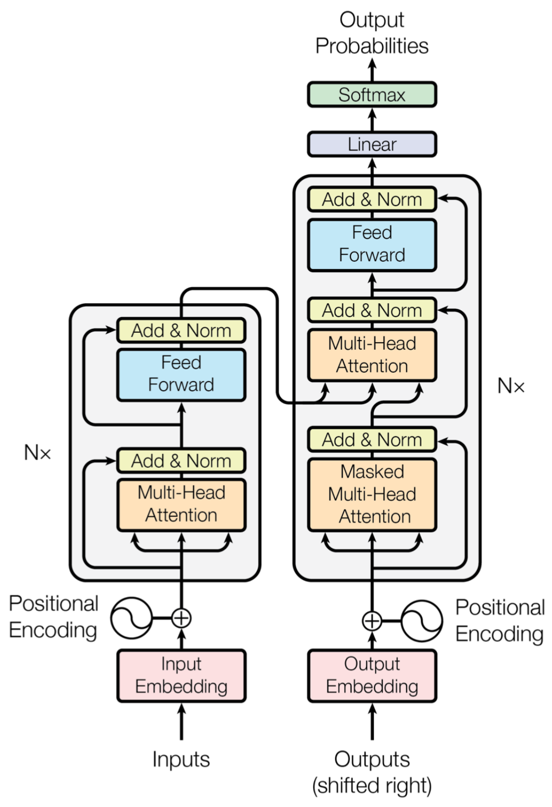

Figure 2.18: Transformer model architecture (Vaswani et al., 2017)

### 2.12.1 Transformer Encoder

A Transformer's encoder is created using six identical layers, where each layer consists of two sub-layers (Figure 2.19). The first layer is a multi-head self-attention mechanism, and the second one is a simple layer which is a position-wise fully connected feed-forward network. A residual connection is employed around each of the two sub-layers, followed by a layer normalization. The output of each sub-layer is $LayerNorm(x + Sublayer(x))$, where $Sublayer(x)$ represents the function that is implemented by the sub-layer itself. As the authors used the dimension of the model as 512; all sub-layers in this architecture along with the embedding layers produce outputs of dimension $d_{model} = 512$ to ensure the proper operations of those residual connections.

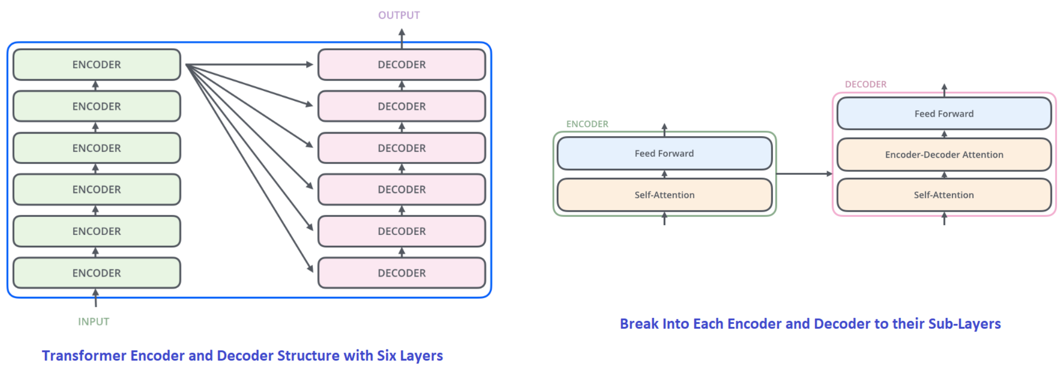
OUTPUT
ENCODER
DECODER
INPUT
Transformer Encoder and Decoder Structure with Six Layers
ENCODER
Feed Forward
Self-Attention
DECODER
Feed Forward
Encoder-Decoder Attention
Self-Attention
Break Into Each Encoder and Decoder to their Sub-Layers

Figure 2.19: Transformer Encoder-Decoder Architecture. (Alammar, 2018)

### 2.12.2 Transformer Decoder

As the encoder discussed above, the decoder also consists of six identical layers (Figure 2.19). Additional to the two sub-layers from the encoder, the decoder adds a third sub-layer executing a multi-head attention mechanism over the output of the encoder side. Like the encoder, residual connections are employed around each of the sub-layers followed by layer normalization.

### 2.12.3 Multi-Head Attention

The Transformer's (Vaswani et al., 2017) attention mechanism computes the relevance of a set of values (information) based on what it is currently processing (queries and keys). In recent days, multi-head attention (Figure 2.20) based models are becoming popular among NLP researchers. In multi-head attention mechanism, the Transformer (Vaswani et al., 2017) runs through the same attention mechanism on different projected version of queries, keys and values parallelly. There are eight attention layers in this mechanism which are called as "heads". Instead of performing a single attention function, it is more effective to linearly project the queries, keys and values $h$ times with different, learned linear projections to $d_q$, $d_k$ and $d_v$ dimensions, respectively with $d_{model}$ dimensional keys, values and queries (Vaswani et al., 2017).

Using multi-head attention has some advantages. It helps the model to understand the

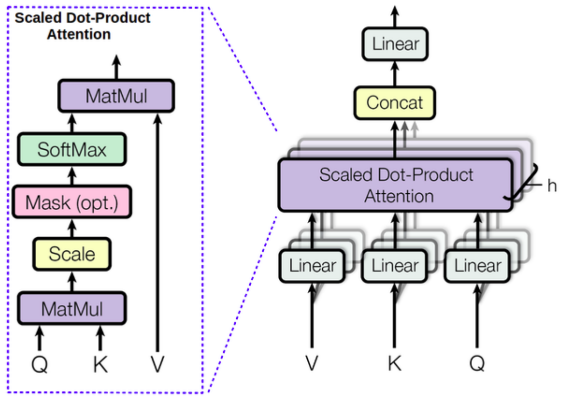

Figure 2.20: Multi-Head Attention Mechanism. (Vaswani et al., 2017)

relationship between the words in different positions. For example, if we want to translate a sentence: “The woman yelled at the cat because she was drunk”, we need to know which word “she” refers to. We can easily understand that “she” refers to “the woman”, but for a machine it has to consider both “the woman” and “the cat”. It can measure that using the “Positional Encoding”. Also, multiple attention mechanisms encode the words in different ways, which expand the range of the model’s learning ability.

The Transformer (Vaswani et al., 2017) computes the attention function on a set of queries, keys and values packed together into a matrix Q, K and V respectively. The steps of multi-head attention mechanism are as follows:

- It considers each of the words from the input sequence, and embeds them individually.
- It splits the embedding into eight heads, and calculates the attention using the following equation (Vaswani et al., 2017):

$$Z_i = head_i = Attention(QW_i{}^Q, KW_i{}^K, VW_i{}^V) \tag{2.58}$$

  Here $W_i{}^Q, W_i{}^K, W_i{}^V$ are different weight matrices for the queries, keys and values respectively.

- Later, it concatenates the resulting matrices $Z_i$, and multiplies it with $W^o$ to produce the output ($Z$) of that layer. The equation is as follows (Vaswani et al., 2017):

$$Z = MultiHead(Q, K, V) = Concat(Z_1, \ldots, Z_h)W^o \tag{2.59}$$

For multi-head attention mechanism, $h = 8$ parallel attention layers are used. The dimension of each head is considered as $d_q = d_v = d_k = d_{model}/h = 512/8 = 64$. So, the total dimension remains the same as the single-head attention mechanism which ensures the similar computational cost.

### 2.12.4 Positional Encoding

The transformer model (Vaswani et al., 2017) does not contain any recurrence or convolution like the recurrent networks. In order to ensure the use of the order of the sequence by the model some additional information is required about the relative or absolute position of the tokens in the sequence. For that purpose "Positional Encoding" is added to the input embeddings at the end of the encoder and decoder stacks. Without positional encoding, the output of the multi-head attention network could be same for the sentences, "I like burgers more than pizzas" and "I like pizzas more than burgers" as it cannot differentiate the positions of the words "burgers" and "pizzas".

The positional encoding holds the same dimension $d_{model}$ as the sentence embeddings. So, it can encode the relative/absolute positions of the words of each sentence as embedded vectors, and then add them with the sentence embeddings. The positional encodings use the following equations (Vaswani et al., 2017):

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{2.60}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{2.61}$$

Where $pos$ represents the position and $i$ is the dimension. Basically, each dimension of the positional encoding is a sinusoidal wave with a frequency from $2\pi$ to $10000.2\pi$. This allows the model to easily learn to attend to relative positions. $PE_{(pos,k)}$ can be represented as a linear function of $PE_{(pos)}$ when we use a fixed frequency $k$ for the equation. So the relative position between different embeddings can be easily inferred.

## 2.13 Summary

In this chapter, we have discussed the basic idea and the related works on different types of the summarization task. We also include all the basic and relevant tools for summarization task and the systems we have used in our thesis. In the following chapters, we will discuss our proposed models and experiments along with the results.

# Chapter 3

# Proposed Models

Query-based summarization has become a popular problem in Natural Language Processing (NLP) field in recent years (Nema et al., 2017). In this problem, we have a tuple - query, source and summary. The query and the source text are the inputs, and the summary is the sample output based on a particular query. A query is a set of words and can be represented as $Q = q_1, q_2, q_3......q_m$ and the source is a set of sentences which can be represented as $S = s_1, s_2, s_3.......s_n$. To solve this problem, we propose some approaches, which are presented in this chapter. First, we describe our Sentence Ranking and Clustering model, and then using the outcome from this model we continue with our Query Based Unsupervised Extractive Summarization Model (QBUEM), Abstractive Summarization using Transformer Model (QBATM), and Reinforcement Learning (QBARLM).

## 3.1 Sentence Ranking and Clustering

In this work, two initial steps are taken. At first, we rank the source sentences according to their similarity with the query to preserve the information coverage; and then we distribute the high scored sentences into different clusters to maintain the diversity. Our proposed method for sentence ranking and clustering is shown in Figure 3.1. For the multi-document summarization part, all the source documents are merged into a single document, and then the sentence ranking and clustering process are applied. In this chapter, the sentence ranking and clustering methods are explained.

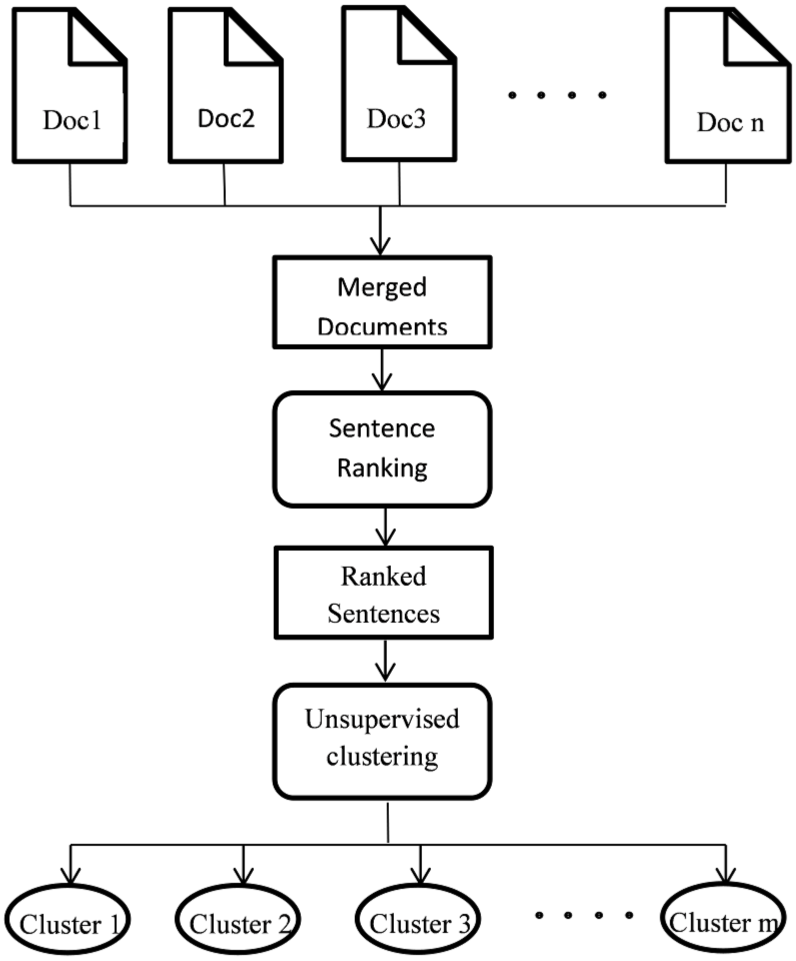

Figure 3.1: Proposed method of sentence ranking and clustering

### 3.1.1 Text Ranking

A good query-based summary is expected to reflect the important information from the source documents as much as possible, and the information should be related to the given query (Wang et al., 2008). In this problem, we need to consider two things while ranking the sentences: the relevance with the query and the important information coverage. For our experiment, each sentence is given a score based on these two points and ranked accordingly. The score for each source sentences can be defined as the following:

$$S = \alpha X + \beta Y \tag{3.1}$$

Where $\alpha$ and $\beta$ are two proportional constants considered as 0.9 and 0.1 and $X$, $Y$ are two scores generated based on its relevance with the query and the information importance. We consider the values for $\alpha$ and $\beta$ to prioritize the relevancy score with query. These two scores are explained as follows:

**Relevance with Query**

In the query-based summarization system, the most important part is to maintain the relevancy of the summary with the query. To ensure that, we assign a value X to every source sentence. X is calculated based on the similarity of that sentence with the query. X can be defined using the following formula:

$$X = \max(\frac{1}{WMD(query, sentence)}, SIF(query, sentence)) \tag{3.2}$$

where WMD is the word mover distance score, and SIF is the Smooth Inverse Frequency score. To compute these score we use the following formulas:

$$WMD(query, sentence) = \frac{Q_w \cdot S_w}{||Q_w||\,||S_w||} \tag{3.3}$$

Here $Q_w$ and $S_w$ are the vectors generated by Word Mover Distance sentence embedding.

$$SIF(query, sentence) = \frac{Q_s \cdot S_s}{||Q_s||\,||S_s||} \tag{3.4}$$

Here $Q_s$ and $S_s$ are the vectors generated by Smooth Inverse Frequency sentence embedding.

The details about these sentence embeddings are presented in Chapter 2. We have used the author provided implementation[8] to calculate the WMD and SIF.

### Important Information Coverage

It is essential to keep the important sentences from the source text in the summary. To tackle this challenge, we have used the Rake algorithm to extract the keywords from all the source sentences. The Rake algorithm identifies the keywords, and gives them a score based on their importance. Then, we have assigned every source sentence a value $Y$ based on the existing keywords in that sentence, which can be defined using the following formula:

$$Y = \sum_{i=1}^{n} score_i \tag{3.5}$$

where $n$ is the number of keywords in the sentence, and *score* is the score for each keyword generated using the RAKE algorithm.

### Pre-Trained Word Embeddings

The word embeddings are low dimensional vector representations of words such as word2vec (Mikolov et al., 2013b) and GloVe (Pennington et al., 2014), which have become popular in various natural language processing tasks. Recently, Bojanowski et al. (2017) proposed a simple method named FastText, which takes the sub-word information into account to learn the word representations. To calculate the similarity score in this thesis, GloVe is used for WMD and SIF. All of the embeddings are explained in Chapter 2.

### Datasets

In this work, both query-based single document dataset and topic-based multi-document datasets are used for testing and evaluation. Those datasets are presented in the following

[8]https://github.com/nlptown/nlp-notebooks/blob/master/Simple%20Sentence%20Similarity.ipynb

Table 3.1: Queries associated with the topic “algae bio-fuel”

| **Topics** | **Queries** |
|---|---|
| Emissions | Is algae biofuel good for combating global warming? |
| Economics | Is algae biofuel economically viable? |
| Land-use | Does algae biofuel take up too much land? |
| Ecosystem | Is algae biofuel generally good for ecosystem? |
| Water-use | Does algae biofuel use too much water? |

sections:

**Single Document Dataset** The main challenge about the query-based summarization problem is getting a suitable dataset. As there is no existing standard dataset for query-based abstractive summarization, we have used the Debatepedia dataset (Nema et al., 2017). Debatepedia is an encyclopedia of pro and con arguments and quotes on critical debate topics. There are 53 overlapping categories such as Politics, Law, Crime, Health, etc. which contains 663 debates in the corpus. The overlapping category represents the idea that a given topic can represent the characteristics of multiple categories. For example, the topic “Eye for an Eye philosophy” belongs to both “Law” and “Morality” categories. The average documents per query are four, and the average number of queries per debate is five. For example, Table 3.1 shows the queries associated with the topic “Algae Bio-fuel”. The dataset also provides the documents and an abstractive summary for each of the queries.

**Multi-Doc Datasets** There is no existing multi-document dataset for query-based summarization. So for this task, we have used the topic based datasets provided from the Document Understanding Conference (DUC 2005, DUC 2006, DUC 2007) (Dang, 2005) to test and evaluate all of our models. These datasets include around fifty topics, and each topic has at least 35 relevant documents associated with them. In this task, the topics are considered as the query, and the associated documents with the topic are considered as the source.

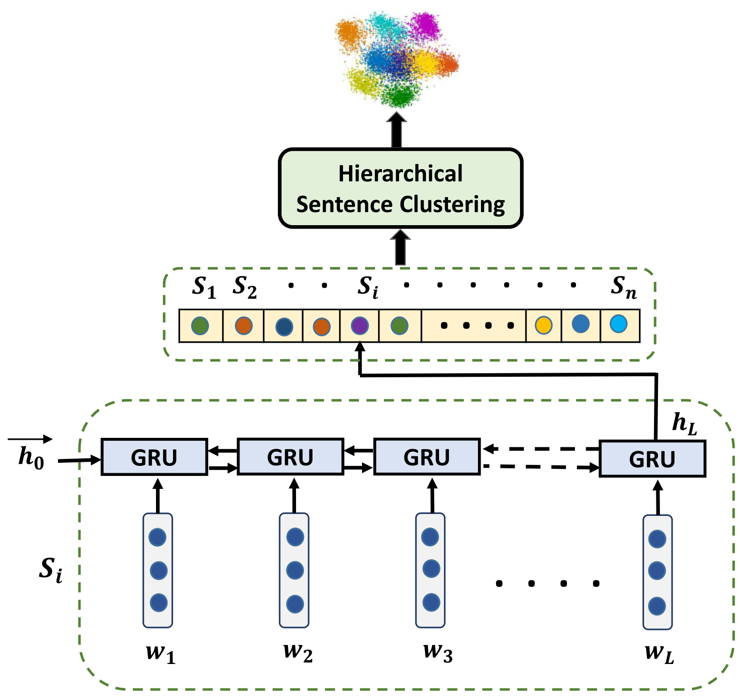

Figure 3.2: Sentence Clustering Model

### 3.1.2 Text Clustering

Due to the sparseness of text representation, text clustering has become a challenging task (Aggarwal and Zhai, 2012). As most of the words only occur once in a text, the Term Frequency-Inverse Document Frequency (TF-IDF) measure can not provide good results. We have used the clustering model presented by Fuad et al. (2019) to avoid this problem.

#### Model

We can represent a sentence as a sequence of words $w$, $\mathbf{S} = (w_1, w_2, ...., w_L)$ of length, $L$. We encode a sentence using bi-directional GRUs (Cho et al., 2014a) which read input symbols in forward ($\overrightarrow{h_t} = \mathbf{GRU}(\overrightarrow{h_{t-1}}, e(w_t)))$ and backward ($\overleftarrow{h_t} = \mathbf{GRU}(\overleftarrow{h_{t+1}}, e(w_t)))$ directions shown in Figure 3.2. At time $t$, A GRU learns the hidden annotations $h_t$ using Equation 3.6 where $\oplus$ indicates concatenation.

$$h_t = \overrightarrow{h_t} \oplus \overleftarrow{h_t} \tag{3.6}$$

$$S_i = x_i = h_L \tag{3.7}$$

Where, the $h_t \in \mathbb{R}^n$ encodes all content at time $t$ and $e(w_t) \in \mathbb{R}^m$ is the $m$-dimensional embedding of the current word $w_t$ using pre-trained word vectors. The output sentence embedding $x_i$ for the sentence $S_i$ is the last hidden state. We then use a hierarchical clustering algorithm with complete linkage criteria. Which means, if two clusters have any pair of sentences that can pass the threshold value, we will merge the those two clusters into a single cluster. In the process, we use cosine similarity as the distance between the sentence embeddings obtained from Equation 3.7. We set a similarity threshold ($\tau = 0.5$) to stop the clustering process by using a hold out dataset SICK[9] of SemEval-2014 to get optimal performance.

### 3.1.3 Cluster Selection

We apply our clustering method explained in Section 3.1.2 on the ranked sentences generated from the process explained in Section 3.1.1. When all the clusters are generated, we filter them based on the number of the sentence in each cluster. We ignore the clusters which have less than three sentences, and proceed with the remaining clusters. These clusters are considered for our models explained in the next sections.

## 3.2 Unsupervised Extractive Summarization Model

Extractive summarization models generate a summary by choosing the most important sentences from the source documents, and then concatenating them into a single paragraph. One of the problems with the automated supervised summarization method is using sparse input representation, which might not observe enough data in the training process (Gupta and Lehal, 2010). To avoid this issue, we have used an unsupervised sentence se-

[9]http://clic.cimec.unitn.it/composes/sick.html

lection method to construct our extractive summarization model. Several authors have proposed some unsupervised extractive summarization models (Schluter and Søgaard, 2015; Yousefi-Azar and Hamey, 2017) in the last few years, and provided good results in the case of generic summarization problems. Feigenblat et al. (2017) proposed an unsupervised method using a cross-entropy method over the DUC 2005-2007 datasets. But as per our knowledge, there is no existing unsupervised approach for the Debatepedia dataset. That makes our method as the first approach to the query-based summarization problem where an unsupervised sentence selection method are used on the Debatepedia dataset.

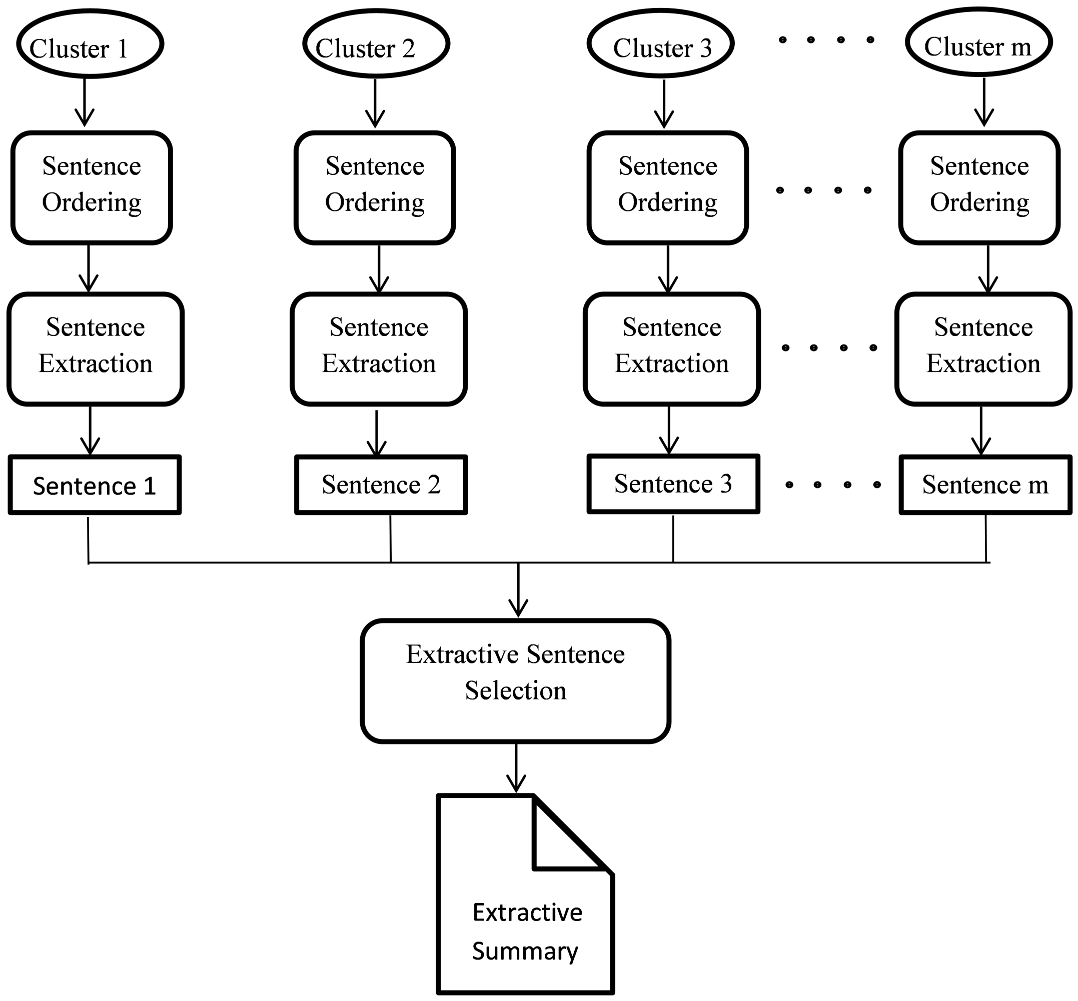

Figure 3.3: Proposed method of Unsupervised Extractive Summarization Model

In this section, we describe our unsupervised extractive summarization model. The flow diagram of our extractive summarization model is shown in Figure 3.3. For this task, we have used the clusters generated from the previous experiment explained in Section 3.1. We choose a single sentence from each cluster, and consider them for our extractive summary. The challenging part of this task is to order these sentences in such a way, which will give us maximum readability, and provide us a good summary. All of these steps are explained as follows.

### 3.2.1 Preliminaries

In this section, we have included the details of our extractive method, where we have selected sentences from the clusters, and ranked them as our system requires.

**Sentence Ordering for Clusters and Extraction** Each of our cluster is a collection of sentences related to each other meaning-wise. So, it is important to extract the most relevant sentences from each cluster which will represent the perfect meaning, and is highly relevant to the query. To do so, we give a normalized score to each of these sentences based on their similarity with other sentences in their respective clusters. We can formulate the intra-cluster score for each sentence as follows:

$$\Delta_k = \sum_{i=1}^{n} cosine\_similarity(sentence_k, sentence_i) \tag{3.8}$$

where we calculate the sum of the cosine similarity scores to get the intra-cluster score. This score represents their relevancy with other sentences in a particular cluster. So, the sentence having the highest score is likely to have the ability to represent the meaning of most of the sentences in the cluster. We calculate the intra-cluster score for all of the sentences in a particular cluster. Then from each cluster we consider a single sentence which has the maximum intra-cluster score. We consider those sentences as the summary from each cluster, and continue to the next step.

**Extracted Sentence Ordering** Our next challenge is to order the extracted sentences in such a way which provides us with the maximum readability. For this purpose, we consider the relevant position of each extracted sentences in the source documents, and assigned those sentences a score of Ω, which can be formulated as follows:

$$\Omega_i = P_i/N \tag{3.9}$$

where, $P_i$ denotes the position of a sentence ($i$) in the source document, and $N$ is the number of sentences in the source document. For multi-document datasets, we consider the position in the corresponding document as the score $P_i$, and the number of sentences in that particular document as $N$.

Based on the score, we rearrange the sentences in ascending order, and use them for the next step.

**Sentence Limit for the Summaries** One of the crucial parts for generating a summary is to determine a fixed length of the summary. Among the extracted sentences from the clusters, we consider the first three sentences for the Depatepedia dataset, and for DUC2005, DUC 2006 and DUC 2007, we take the first seven sentences to match the length of the reference summaries.

## 3.3 Abstractive Summarization Models

In this section, we describe our abstractive summarization methods, and compare them with different baseline systems. We have implemented two abstractive models using various techniques; for the first one, we have used the tensor2tensor (Vaswani et al., 2018) model which is a part of the Transformer Model (Vaswani et al., 2017), and for the second one, we have used the reinforcement learning (RL). We use these models to get a single sentence from each of the clusters generated in Section 3.1. Both of these models are described separately in the following sections. Then we explain our sentence selection method, and we include our experimental results afterward. The workflow for our abstractive model is shown in Figure 3.4.

### 3.3.1 Abstractive Summarization using Transformer Model

For this model, we have used the Transformer model (Vaswani et al., 2017), which has brought significant improvement over different applications. The Transformer (Vaswani et al., 2017) is built by maintaining the overall architecture of a standard encoder-decoder model. It does not include the complex recurrent or convolutional layers most commonly used in encoder-decoder architectures. Rather it uses the multi-headed self-attention with positional encoder. The natural ability of a multi-head attention mechanism to jointly attend to similar phrases from different positions of a sequence makes this an appropriate choice for our model. We use the implementation provided by the authors. The detailed description of the structure of this model is provided in Section 2.11 and 2.12.

**Abstractive Sentence Generation** Given a set of source sentences of a related topic $\mathbf{S} = (S_1, S_2, \ldots, S_N)$, our model needs to predict its abstractive multi-sentence compression target, $\mathbf{T} = (t_1, t_2, \ldots, t_M)$ where $N > 1$ and $M < |S_1| + |S_2| + \cdots + |S_N|$. To train our model for this task, we have created a dataset from the CNN/DailyMail dataset (Hermann et al., 2015) having more than 600,000 samples of source documents (S) of almost the same topics and the target sentence (T). The CNN/DailyMail dataset contains more than 300,000 documents, each having multiple highlighted sentences which represent the content for the article. We take each highlighted sentence, and get the similarity score with each of the sentences in the document. We only pick the sentences having the similarity score more than 0.50 for a particular highlight. We consider the threshold value as 0.50, as it gave us comparatively the best performance from our model. Then we consider them as the source sentences, and the highlight as the target sentence.

For training, we split the tokens into a 32000 word-piece vocabulary as the author of

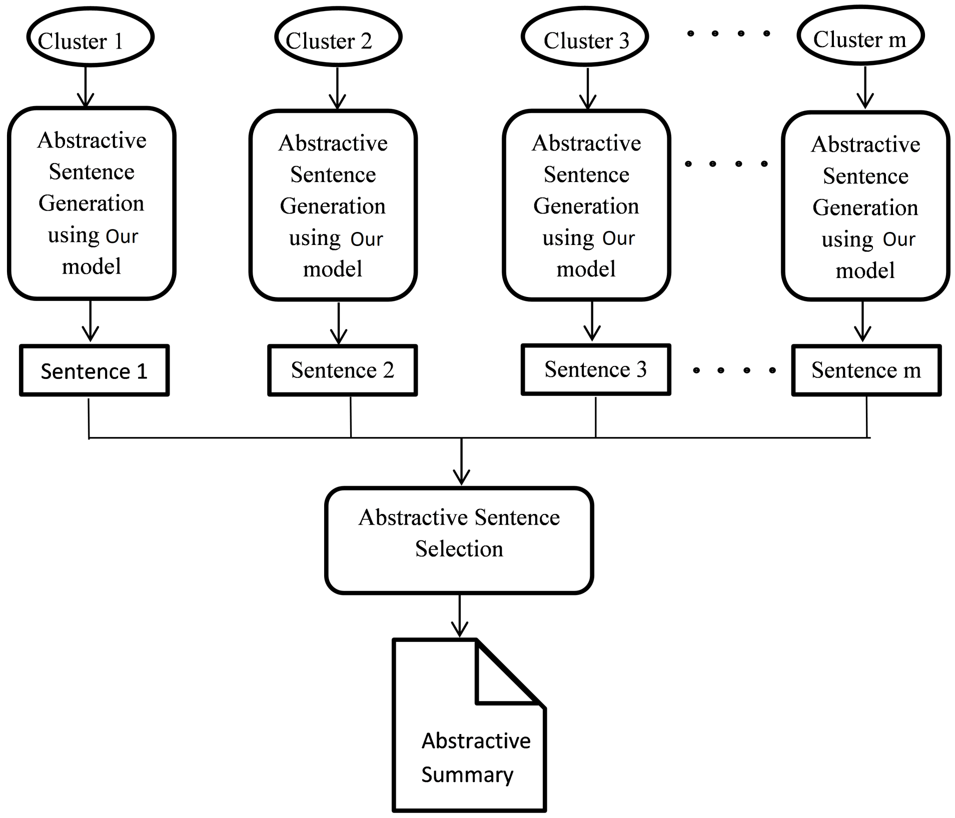

Figure 3.4: Proposed method of Abstractive Summarization Model

the WMT 2014 English-French task. We have used the tensor2tensor (Vaswani et al., 2018) neural translation model to train our model with a TITAN GTX GPU machine. We ran the training for 500,000 epoch which took almost eight days. We have used the Adam Optimizer (Kingma and Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\varepsilon = 10^{-9}$. We followed the equation from Vaswani et al. (2017) to vary the learning rate ($lrate$) during the training, which is as follows:

$$lrate = d_{model}^{-0.5} \cdot min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \tag{3.10}$$

**Adam Optimizer** is an adaptive learning rate algorithm which is highly used to train the neural networks. It computes the learning rate in each step based on different parameters, and optimizes the learning rate for the next steps. The parameters are given below:

- $\beta_1$ - is used to decay the running average of the gradient (0.9 in our model).
- $\beta_2$ - is used to decay the running average of the square of gradient (0.98 in our model).
- $\varepsilon$ - is used to prevent the division by zero error ($10^{-9}$ in our model).

We have used the $warmup\_steps = 4000$. This equation corresponds to increasing the learning rate for the first 4000 steps, and decreasing it proportionally to the inverse square root of the step number.

After the model is trained, we have used the clusters generated from Section 3.1. As described before, all of the clusters consist of at least three sentences which are highly related to each other. We run our trained model on those clusters to get a single abstractive sentence from each of them as shown in Figure 3.4.

### 3.3.2 Abstractive Summarization using Reinforcement Learning

In this section, we describe our reinforcement learning-based abstractive summarization model. We have used the idea of using the policy-based reinforcement learning (RL) technique to combine both extractive and abstractive architecture to get a hybrid structure (Chen and Bansal, 2018). This structure is formulated in such a way that it would give us a single abstractive sentence from each of our clusters. In this method, we have used an extractive RL agent to select the extractive sentences from the clusters, and an abstractive network to convert the sentences into their abstractive form by rewriting them. We have combined a sentence level metric reward with a policy gradient method to build a connection between our extractive and abstractive models. Further details of our models are explained as follows:

**Extractive Agent** We have implemented a hierarchical neural model to learn the representations of each sentence in a document. Then we have used a selection network to extract the sentences.

First, we compute the sentence representation for each of the sentences in the clusters using the temporal convolutional model introduced by Kim (2014). We have generated a distributed vector representation for each individual words in the sentence using a pre-trained word embedding matrix. Then we collect the sequence of word vectors from each sentence, and provide them to a 1-D single-layer Convolutional Neural Network (CNN) with various window sizes in the range of three to five (3, 4, 5) followed by ReLU non-linear activation and max pooling. The purpose of this is to capture the temporal dependencies of nearby words. Then we concatenate the outputs from the activation layers for all the different filter window size to get the convolutional representations of each sentence. To understand the context of the sentences and identify the semantic dependencies between them, we use a bidirectional LSTM-RNN (Hochreiter and Schmidhuber, 1997) convolutional output.

For sentence extraction, we have used an LSTM-RNN to train a “pointer network” (Vinyals et al., 2015) in such a way so that it can recurrently extract the sentences from the source text. This model performs the classification of all sentences of the document in every step of the extraction process. The process is shown in Figure 3.5.

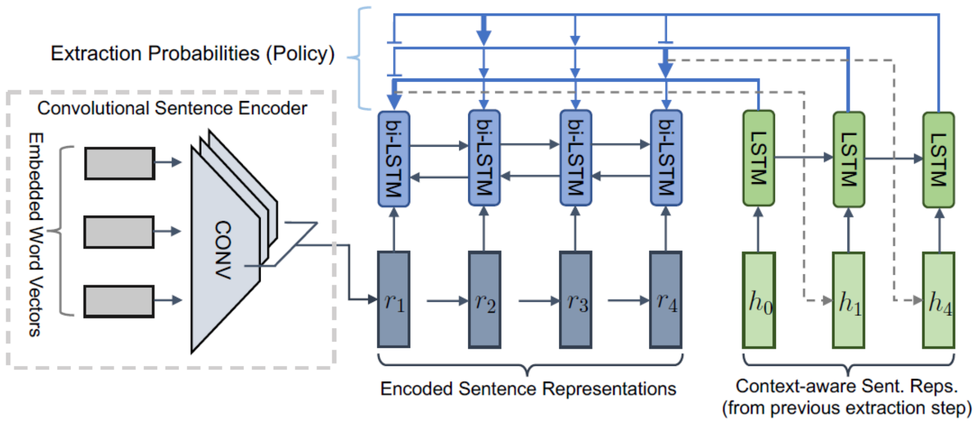
Extraction Probabilities (Policy)
Convolutional Sentence Encoder
Embedded Word Vectors
CONV
bi-LSTM
bi-LSTM
bi-LSTM
bi-LSTM
LSTM
LSTM
LSTM
$r_1$
$r_2$
$r_3$
$r_4$
$h_0$
$h_1$
$h_4$
Encoded Sentence Representations
Context-aware Sent. Reps.
(from previous extraction step)

Figure 3.5: Extractive Agent. the convolutional sentence encoder calculates $r_j$ for every sentences. The RNN encoder (blue) calculates the context-aware representation $h_j$, and then the RNN decoder (green) selects sentences $j_t$ at time $t$. With $j_t$ selected, $h_{jt}$ will be fed into the decoder at time $t+1$. (Chen and Bansal, 2018)

**Abstractive Agent** To develop our abstractive agent, we use the model provided by Chen and Bansal (2018) which performs two modifications on the extracted sentences to provide the summary in a concise way: firstly, compressing the sentences and then paraphrasing them. The standard encoder-aligner-decoder (Bahdanau et al., 2014; Luong et al., 2015) with the bi-linear multiplicative attention function (Luong et al., 2015) are used to develop this model. The pointer generator copy mechanism (See et al., 2017) is also used to help the model to copy some out of vocabulary words. A copy probability is calculated by learnable parameters, and then used to compute the weighted sum of the probability of source vocabulary and the predefined vocabulary. The combined structure is shown in Figure 3.6

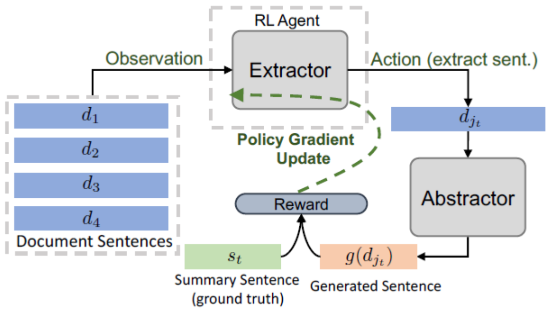

Figure 3.6: Training step of the extractor agent, and the operation in between the extractor and abstractor agent. (Chen and Bansal, 2018)

We have used our own dataset created from CNN/DailyMail for training our model.

### 3.3.3 Abstractive Sentence Selection

Our first challenge is to order the sentences in such a way that it provides us the maximum readability, and also limit the sentences to get a concise summary. For this purpose, we calculate the intra-cluster score for all the sentences in a cluster. Then we get the average intra-cluster sentence score for each cluster, and rank the clusters accordingly. Then, we generate one abstractive sentence from each cluster using our different abstractive methods, and rank those sentences according to the rank of their respective clusters. Our next challenge is to determine a fixed length of the summaries. Among the abstractive sentences from the clusters, we consider the first three sentences for the Depatepedia dataset, and for DUC 2005, DUC 2006 and DUC 2007, we take the first seven sentences to match the length of the reference summaries.

### 3.3.4 Summary

In this chapter, we have described our different proposed models. In our next chapter the experimental results of our models are discussed.

# Chapter 4

# Experiments and Results

In this chapter, we present our different summarization models and compare them with different baseline models. The details of our datasets, baseline models and evaluation metrics are also included here.

## 4.1 Unsupervised Extractive Summarization

In this section, we describe the preliminaries and the experimental results for our Unsupervised Extractive Summarization model.

### 4.1.1 Datasets

In this work, both query based single document dataset Debatepedia (Nema et al., 2017) and query based multi document datasets DUC 2005-2007 are used. Details of these datasets are provided in Chapter 3.

### 4.1.2 Baselines

To the best of our knowledge, there is no existing extractive summarization method for the Debatepedia dataset. So, we could not compare our extractive model with any existing models. For the DUC datasets, Zhong et al. (2015) proposed a query-oriented deep extraction (QODE) with a new deep architecture and an unsupervised deep learning algorithm, which was the first attempt of using deep learning techniques in the query-based multi-document summarization task. Feigenblat et al. (2017) proposed a Cross-Entropy Summarizer (CES), which provided a possible solution for the length constraint problem.

This method achieved the state-of-the-art ROUGE scores on DUC 2005-2007 datasets. We compare our model with these two baseline models.

### 4.1.3 Evaluation Metrics

This model is evaluated using ROUGE[10] (Lin, 2004). However, ROUGE scores are unfairly biased towards lexical overlap at the surface level. Taking this into account, this system is also evaluated with a recently proposed metric, ROUGE-SU4. Limited length recall performance are used for both metrics, as the system generated summary is concise through some constraint such as sentence limit. Therefore, only the recall is considered since precision is of less concern in this scenario.

Table 4.1: Comparison of our proposed Unsupervised Extractive model with the baseline models.

| **Debatepedia** | | | | |
|---|---|---|---|---|
| **Models** | **ROUGE-1** | **ROUGE-2** | **ROUGE-L** | **ROUGE-SU4** |
| QBUEM (ours) | 34.67 | 6.54 | 28.05 | 12.37 |

| **DUC 2005** | | | | |
|---|---|---|---|---|
| **Models** | **ROUGE-1** | **ROUGE-2** | **ROUGE-L** | **ROUGE-SU4** |
| Zhong et al. (2015) (QODE) | 37.51 | 07.75 | 30.57 | 13.41 |
| Feigenblat et al. (2017) (CES) | **40.33** | 7.94 | **32.76** | 13.89 |
| QBUEM (ours) | 38.30 | **9.62** | 31.36 | **14.41** |

| **DUC 2006** | | | | |
|---|---|---|---|---|
| **Models** | **ROUGE-1** | **ROUGE-2** | **ROUGE-L** | **ROUGE-SU4** |
| Zhong et al. (2015) (QODE) | 40.15 | 9.28 | 31.04 | 14.79 |
| Feigenblat et al. (2017) (CES) | **43.00** | 9.69 | **32.98** | **16.63** |
| QBUEM (ours) | 38.76 | **11.31** | 32.61 | 15.83 |

| **DUC 2007** | | | | |
|---|---|---|---|---|
| **Models** | **ROUGE-1** | **ROUGE-2** | **ROUGE-L** | **ROUGE-SU4** |
| Zhong et al. (2015) (QODE) | 42.95 | 11.63 | 34.23 | 16.85 |
| Feigenblat et al. (2017) (CES) | **45.43** | 12.02 | **35.12** | **17.50** |
| QBUEM (ours) | 41.25 | **12.11** | 34.58 | 16.92 |

[10]ROUGE-1.5.5 with options: -n 2 -m -u -c 95 -x -r 1000 -f A -p 0.5 -t 0

### 4.1.4 Experimental Results

We report the performance of our Query Based Unsupervised Extractive Method (QBUEM) compared to other baseline systems in Table 4.1. For the DUC datasets, our model improves the ROUGE-2 score, and matched almost the same level as the baseline model for ROUGE-L and ROUGE-SU4 metric. The ROUGE metrics measure the exact tokens; for example, ROUGE-1 compares between single tokens, on the other hand, ROUGE-2 compares with bi-grams. Although, we did not achieve better ROUGE-1 score than the baseline models, but for extractive summarization ROUGE-2 score represents a better quality summary as it takes the bi-grams into account while ROUGE-1 just considers uni-grams. We present example of outputs generated by our model in Appendix A.

## 4.2 Abstractive Summarization Models

In this section, the detailed experimental results of our abstractive models are presented.

### 4.2.1 Datasets

To train our model we have created a customized dataset from CNN-Dailymail (Hermann et al., 2015) dataset which is mentioned in Chapter 3. For evaluation, both of the query based single document dataset Debatepedia (Nema et al., 2017) and the query based multi document datasets DUC 2005-2007 are used. We describe the datasets in detail in Chapter 3.

### 4.2.2 Baselines

As the baseline, we compare our model with the abstractive model proposed by Nema et al. (2017) for the Debatepedia dataset. For the DUC datasets, we could not find suitable abstractive models with good results to compare our models with. So we compare our abstractive models result with the query-oriented deep extraction (QODE) model proposed by (Zhong et al., 2015), and the Cross-Entropy Summarizer (CES) model proposed by (Feigenblat et al., 2017).

### 4.2.3 Evaluation Metrics

We evaluate our Query Based Abstractive Summarization using Transformer Model (QBATM) and Query Based Abstractive summarization using Reinforcement Learning Model (QBARLM) using the ROUGE metric along with the METEOR metric. Denkowski and Lavie (2014) used a combination of both precision and recall in the METEOR metric. The alignment is based on WordNet synonyms, stemmed tokens and look-up table paraphrases in addition to the exact token matching.

Table 4.2: Comparison of our proposed Abstractive Summarization Models with the baseline models.

| **Debatepedia** | | | | | |
|---|---|---|---|---|---|
| **Models** | **ROUGE-1** | **ROUGE-2** | **ROUGE-L** | **ROUGE-SU4** | **METEOR** |
| Nema et al. (2017) | **41.26** | **18.75** | **40.43** | **15.93** | **9.12** |
| QBATM (ours) | 26.91 | 6.54 | 20.51 | 12.43 | 5.92 |
| QBARLM (ours) | 40.77 | 7.98 | 33.69 | 15.21 | 6.38 |

| **DUC 2005** | | | | | |
|---|---|---|---|---|---|
| **Models** | **ROUGE-1** | **ROUGE-2** | **ROUGE-L** | **ROUGE-SU4** | **METEOR** |
| Zhong et al. (2015) (QODE) | 37.51 | 07.75 | 30.57 | 13.41 | 10.53 |
| Feigenblat et al. (2017) (CES) | **40.33** | 7.94 | **32.76** | **13.89** | 11.68 |
| QBATM (ours) | 28.96 | 5.69 | 20.79 | 10.60 | 10.93 |
| QBARLM (ours) | 36.30 | **8.50** | 29.29 | 13.11 | **13.29** |

| **DUC 2006** | | | | | |
|---|---|---|---|---|---|
| **Models** | **ROUGE-1** | **ROUGE-2** | **ROUGE-L** | **ROUGE-SU4** | **METEOR** |
| Zhong et al. (2015) (QODE) | 40.15 | 9.28 | 32.86 | 14.79 | 11.45 |
| Feigenblat et al. (2017) (CES) | **43.00** | 9.69 | **34.76** | **16.63** | 12.70 |
| QBATM (ours) | 32.38 | 8.74 | 27.40 | 10.91 | 12.21 |
| QBARLM (ours) | 37.07 | **10.43** | 30.72 | 14.79 | **13.93** |

| **DUC 2007** | | | | | |
|---|---|---|---|---|---|
| **Models** | **ROUGE-1** | **ROUGE-2** | **ROUGE-L** | **ROUGE-SU4** | **METEOR** |
| Zhong et al. (2015) (QODE) | 42.95 | 11.63 | 33.83 | 16.85 | 11.34 |
| Feigenblat et al. (2017) (CES) | **45.43** | 12.02 | **34.88** | **17.50** | 14.63 |
| QBATM (ours) | 33.56 | 10.30 | 31.28 | 15.86 | 13.49 |
| QBARLM (ours) | 40.01 | **12.11** | 34.57 | 16.93 | **15.25** |

### 4.2.4 Experimental Results

We have reported the performance of our Query Based Abstractive Reinforcement Learning Model (QBARLM) and Query Based Abstractive Transformer Model (QBATM) compared to other baseline systems in Table 4.2. We have reported different ROUGE metrics and the METEOR[11] score as well. As the baseline models for DUC 2005-2007 are extractive summarization models, they are expected to have better ROUGE scores than our models, because, the ROUGE metric matches the exact tokens between the reference and generated summary. On the other hand, the METEOR metric represents the abstractiveness because it considers the stemmed tokens and paraphrases which is the important property of an abstractive model. In spite of that, our model has achieved better ROUGE-2 result as well as the METEOR scores which represents both the accuracy and abstractivenes of our model. We present examples of our system generated summary in Appendix A.

## 4.3 Summary

In this chapter, we have presented the experimental results of our models and the comparison with the baseline models in term of different evaluation metric. In the next chapter our overall work and future direction are discussed.

[11]The code for the METEOR metric is taken from https://github.com/Maluuba/nlg-eval

# Chapter 5

# Conclusion

We have developed an unsupervised method for solving extractive summarization along with our own sentence ranking and hierarchical clustering approach. We also provide a novel approach to solve the query-based summarization problem using reinforcement learning. We have also proposed a method to solve the query-based summarization problem using the transformer model (Vaswani et al., 2017). For this task, we have created a dataset on our own to train our model. Our approaches are applied to several datasets, and compared with several proposed methods. Our models achieve competitive results compared to the baseline models for multi-doc datasets. The combined operation of our techniques also achieve a state-of-the-art result based on ROUGE-2 and METEOR evaluation metrics.

Though the results we obtain have already shown the effectiveness of our approaches, it could be further improved in a number of ways:

- Our extractive summarization model can be implemented using reinforcement learning. We can do it only using the extractor agent of our model.
- Using multiple reward function for our Reinforcement Learning model is possible as well
- One of the problems we face is the lack of proper query-based datasets. The creation of a proper query-based summarization dataset is possible.

## Bibliography

Charu C. Aggarwal and ChengXiang Zhai. 2012. *A Survey of Text Clustering Algorithms*, Springer US, Boston, MA, pages 77–128. https://doi.org/10.1007/978-1-4614-3223-4$_4$.

Jay Alammar. 2018. The Illustrated Transformer. `https://jalammar.github.io/illustrated-transformer/`. [Online; accessed 27-June-2018].

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings .

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* .

Prateek Bajaj. 2018. Reinforcement learning. `https://www.geeksforgeeks.org/what-is-reinforcement-learning/`.

Satanjeev Banerjee and Alon Lavie. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Association for Computational Linguistics, pages 65–72. http://www.aclweb.org/anthology/W05-0909.

Siddhartha Banerjee, Prasenjit Mitra, and Kazunari Sugiyama. 2015. Multi-document abstractive summarization using ilp based multi-sentence compression. In *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, IJCAI'15, pages 1208–1214. http://dl.acm.org/citation.cfm?id=2832415.2832417.

Regina Barzilay and Kathleen R. McKeown. 2005. Sentence fusion for multidocument news summarization. *Comput. Linguist.* 31(3):297–328. https://doi.org/10.1162/089120105774321091.

Scott Beamer, Krste Asanović, and David Patterson. 2013. Direction-optimizing breadth-first search. *Scientific Programming* 21(3-4):137–148.

Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.* 5(2):157–166. https://doi.org/10.1109/72.279181.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.* 3:1137–1155. http://dl.acm.org/citation.cfm?id=944919.944966.

Julia A Bennell and Xiang Song. 2010. A beam search implementation for the irregular shape packing problem. *Journal of Heuristics* 16(2):167–188.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *J. Mach. Learn. Res.* 3:993–1022. http://dl.acm.org/citation.cfm?id=944919.944937.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5:135–146.

Florian Boudin and Emmanuel Morin. 2013. Keyphrase extraction for n-best reranking in multi-sentence compression. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Atlanta, Georgia, pages 298–305. http://www.aclweb.org/anthology/N13-1030.

John S Bridle. 1990. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, Springer, pages 227–236.

Denny Britz. 2015. Understanding Convolutional Neural Networks for NLP. `http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/`. [Online; accessed 7-November-2015].

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Comput. Linguist.* 19(2):263–311. http://dl.acm.org/citation.cfm?id=972470.972474.

Yllias Chali, Moin Tanvee, and Mir Tafseer Nayeem. 2017. Towards abstractive multi-document summarization using submodular function-based framework, sentence compression and merging. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing, IJCNLP 2017, Taipei, Taiwan, November 27 - December 1, 2017, Volume 2: Short Papers*. pages 418–424. https://aclanthology.info/papers/I17-2071/i17-2071.

Yen-Chun Chen and Mohit Bansal. 2018. Fast abstractive summarization with reinforce-selected sentence rewriting. *arXiv preprint arXiv:1805.11080* .

Jianpeng Cheng and Mirella Lapata. 2016. Neural summarization by extracting sentences and words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, pages 484–494.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014a. On the properties of neural machine translation: Encoder–decoder approaches pages 103–111. http://www.aclweb.org/anthology/W14-4012.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014b. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, pages 1724–1734. http://www.aclweb.org/anthology/D14-1179.

Sumit Chopra, Michael Auli, and Alexander M. Rush. 2016. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, San Diego, California, pages 93–98. http://www.aclweb.org/anthology/N16-1012.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR* abs/1412.3555. http://arxiv.org/abs/1412.3555.

James Clarke and Mirella Lapata. 2006. Models for sentence compression: A comparison across domains, training requirements and evaluation measures. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Stroudsburg, PA, USA, ACL-44, pages 377–384.

James Clarke and Mirella Lapata. 2008. Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research* 31:399–429.

Hoa Trang Dang. 2005. Overview of duc 2005. In *Proceedings of the document understanding conference*. volume 2005, pages 1–12.

Hal Daumé III and Daniel Marcu. 2006. Bayesian query-focused summarization. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 305–312.

Michael Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*. Association for Computational Linguistics, Baltimore, Maryland, USA, pages 376–380. http://www.aclweb.org/anthology/W14-3348.

Günes Erkan and Dragomir R. Radev. 2004. Lexrank: Graph-based lexical centrality as salience in text summarization. *J. Artif. Int. Res.* 22(1):457–479.

Engui Fan. 2000. Extended tanh-function method and its applications to nonlinear equations. *Physics Letters A* 277(4-5):212–218.

Guy Feigenblat, Haggai Roitman, Odellia Boni, and David Konopnicki. 2017. Unsupervised query-focused multi-document summarization using the cross entropy method. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, pages 961–964.

Katja Filippova. 2010. Multi-sentence compression: Finding shortest paths in word graphs. In *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics, Stroudsburg, PA, USA, COLING '10, pages 322–330. http://dl.acm.org/citation.cfm?id=1873781.1873818.

Katja Filippova and Michael Strube. 2008. Sentence fusion via dependency graph compression. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Stroudsburg, PA, USA, EMNLP '08, pages 177–185. http://dl.acm.org/citation.cfm?id=1613715.1613741.

Seeger Fisher and Brian Roark. 2006. Query-focused summarization by supervised sentence ranking and skewed word distributions. In *Proceedings of the Document Understanding Conference, DUC-2006, New York, USA*. Citeseer.

Tanvir Ahmed Fuad, Mir Tafseer Nayeem, Asif Mahmud, and Yllias Chali. 2019. Neural sentence fusion for diversity driven abstractive multi-document summarization. *Computer Speech & Language* 58:216–230.

Dan Gillick and Benoit Favre. 2009. A scalable global model for summarization. In *Proceedings of the Workshop on Integer Linear Programming for Natural Langauge Processing*. Association for Computational Linguistics, Stroudsburg, PA, USA, ILP '09, pages 10–18. http://dl.acm.org/citation.cfm?id=1611638.1611640.

Vishal Gupta and Gurpreet Singh Lehal. 2010. A survey of text summarization extractive techniques. *Journal of emerging technologies in web intelligence* 2(3):258–268.

Jun Han and Claudio Moraga. 1995. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*. Springer, pages 195–201.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. MIT Press, Cambridge, MA, USA, NIPS'15, pages 1693–1701. http://dl.acm.org/citation.cfm?id=2969239.2969428.

Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6(02):107–116.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Kai Hong and Ani Nenkova. 2014. Improving the estimation of word importance for news multi-document summarization. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, Gothenburg, Sweden, pages 712–721.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* .

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Simeon Kostadinov. 2019. Understanding Encoder-Decoder Sequence to Sequence Model. https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346. [Online; accessed 4-February-2019].

Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. 2015. From word embeddings to document distances. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. JMLR.org, ICML'15, pages 957–966. http://dl.acm.org/citation.cfm?id=3045118.3045221.

Thomas K Landauer, Peter W Foltz, and Darrell Laham. 1998. An introduction to latent semantic analysis. *Discourse processes* 25(2-3):259–284.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Piji Li, Wai Lam, Lidong Bing, Weiwei Guo, and Hang Li. 2017a. Cascaded attention based unsupervised information distillation for compressive summarization. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Copenhagen, Denmark, pages 2081–2090. https://www.aclweb.org/anthology/D17-1221.

Piji Li, Wai Lam, Lidong Bing, and Zihao Wang. 2017b. Deep recurrent generative decoder for abstractive text summarization. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Copenhagen, Denmark, pages 2091–2100. https://www.aclweb.org/anthology/D17-1222.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In Stan Szpakowicz Marie-Francine Moens, editor, *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*. Association for Computational Linguistics, Barcelona, Spain, pages 74–81.

Hui Lin and Jeff Bilmes. 2011. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*. Association for Computational Linguistics, Stroudsburg, PA, USA, HLT '11, pages 510–520.

Linqing Liu, Yao Lu, Min Yang, Qiang Qu, Jia Zhu, and Hongyan Li. 2018. Generative adversarial network for abstractive text summarization. In *Thirty-second AAAI conference on artificial intelligence*.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Lisbon, Portugal, pages 1412–1421.

Shuming Ma, Xu Sun, Jingjing Xu, Houfeng Wang, Wenjie Li, and Qi Su. 2017. Improving semantic relevance for sequence-to-sequence learning of chinese social media text summarization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, Vancouver, Canada, pages 635–640. http://aclweb.org/anthology/P17-2100.

Oren Melamud, Omer Levy, and Ido Dagan. 2015. A simple word embedding model for lexical substitution. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*. Association for Computational Linguistics, Denver, Colorado, pages 1–7. http://www.aclweb.org/anthology/W15-1501.

Rada Mihalcea, Courtney Corley, Carlo Strapparava, et al. 2006. Corpus-based and knowledge-based measures of text semantic similarity. In *Aaai*. volume 6, pages 775–780.

Rada Mihalcea and Paul Tarau. 2004. Textrank: Bringing order into texts. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*. Association for Computational Linguistics, Barcelona, Spain, pages 404–411. http://www.aclweb.org/anthology/W04-3252.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *CoRR* abs/1301.3781. http://dblp.uni-trier.de/db/journals/corr/corr1301.htmlabs-1301-3781.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*. Curran Associates Inc., USA, NIPS'13, pages 3111–3119. http://dl.acm.org/citation.cfm?id=2999792.2999959.

Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. pages 807–814.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2017. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*. pages 3075–3081.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Ça glar Gulçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. *CoNLL 2016* page 280.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Ranking Sentences for Extractive Summarization with Reinforcement Learning. In *Proceedings of the NAACL 2018 - Conference of the North American Chapter of the Association for Computational Linguistics*.

Mir Tafseer Nayeem. 2017. *Methods of sentence extraction, abstraction and ordering for automatic text summarization*. Master's thesis, Lethbridge, Alta.: Universtiy of Lethbridge, Department of Mathematics and Computer Science.

Mir Tafseer Nayeem and Yllias Chali. 2017a. Extract with order for coherent multi-document summarization. In *Proceedings of TextGraphs@ACL 2017: the 11th Workshop on Graph-based Methods for Natural Language Processing, Vancouver, Canada, August 3, 2017*. pages 51–56. https://aclanthology.info/papers/W17-2407/w17-2407.

Mir Tafseer Nayeem and Yllias Chali. 2017b. Paraphrastic fusion for abstractive multi-sentence compression generation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*. pages 2223–2226. https://doi.org/10.1145/3132847.3133106.

Mir Tafseer Nayeem, Tanvir Ahmed Fuad, and Yllias Chali. 2018. Abstractive unsupervised multi-document summarization using paraphrastic sentence fusion. In *Proceedings of the 27th International Conference on Computational Linguistics*. Association for Computational Linguistics, pages 1191–1204. http://aclweb.org/anthology/C18-1102.

Preksha Nema, Mitesh Khapra, Anirban Laha, and Balaraman Ravindran. 2017. Diversity driven attention model for query-based abstractive summarization. *arXiv preprint arXiv:1704.08300* .

Graham Neubig. 2017. Neural machine translation and sequence-to-sequence models: A tutorial. *CoRR* abs/1703.01619. http://arxiv.org/abs/1703.01619.

Christopher Olah. 2015. Understanding LSTM Networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. [Online; accessed 27-August-2015].

You Ouyang, Wenjie Li, Sujian Li, and Qin Lu. 2011. Applying regression models to query-focused multi-document summarization. *Information Processing & Management* 47(2):227–237.

Daraksha Parveen, Hans-Martin Ramsl, and Michael Strube. 2015. Topical coherence for graph-based extractive summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 1949–1954.

Daraksha Parveen and Michael Strube. 2015. Integrating importance, non-redundancy and coherence in graph-based extractive summarization. In *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, IJCAI'15, pages 1298–1304.

Ramakanth Pasunuru and Mohit Bansal. 2018. Multi-reward reinforced summarization with saliency and entailment. *arXiv preprint arXiv:1804.06451* .

Romain Paulus, Caiming Xiong, and Richard Socher. 2017a. A deep reinforced model for abstractive summarization. *CoRR* abs/1705.04304.

Romain Paulus, Caiming Xiong, and Richard Socher. 2017b. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304* .

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, pages 1532–1543. http://www.aclweb.org/anthology/D14-1162.

Raimundo Real and Juan M Vargas. 1996. The probabilistic basis of jaccard's index of similarity. *Systematic biology* 45(3):380–385.

Cody Rioux, Sadid A Hasan, and Yllias Chali. 2014. Fear the reaper: A system for automatic multi-document summarization with reinforcement learning. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. pages 681–690.

Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. 2010. Automatic keyword extraction from individual documents. *Text Mining* pages 1–20.

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

Seonggi Ryang and Takeshi Abekawa. 2012. Framework of automatic text summarization using reinforcement learning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, pages 256–265.

Nihit Saxena. 2018. Word Mover's Distance for Text Similarity. `https://towardsdatascience.com/word-movers-distance-for-text-similarity-7492aeca71b0`. [Online; accessed 26-August-2018].

Natalie Schluter and Anders Søgaard. 2015. Unsupervised extractive summarization via coverage maximization with syntactic and semantic concepts. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. pages 840–844.

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368* .

Elaheh Shafiei Bavani, Mohammad Ebrahimi, Raymond K Wong, and Fang Chen. 2016. An efficient approach for multi-sentence compression. In *Asian Conference on Machine Learning*. pages 414–429.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014a. Sequence to sequence learning with neural networks. *CoRR* abs/1409.3215. http://arxiv.org/abs/1409.3215.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014b. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. pages 3104–3112.

Jun Suzuki and Masaaki Nagata. 2017. Cutting-off redundant repeating generations for neural abstractive summarization. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, Valencia, Spain, pages 291–297. http://www.aclweb.org/anthology/E17-2047.

Dung Tran Tuan, Nam Van Chi, and Minh-Quoc Nghiem. 2017. *Multi-sentence Compression Using Word Graph and Integer Linear Programming*, Springer International Publishing, Cham, pages 367–377. https://doi.org/10.1007/978-3-319-56660-$3_3$2.

Ujjwalkarn. 2016. An Intuitive Explanation of Convolutional Neural Networks. `https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/`. [Online; accessed 11-August-2016].

Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N. Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. 2018. Tensor2tensor for neural machine translation. *CoRR* abs/1803.07416. http://arxiv.org/abs/1803.07416.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., pages 5998–6008. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*. pages 2692–2700.

Dingding Wang, Tao Li, Shenghuo Zhu, and Chris Ding. 2008. Multi-document summarization via sentence-level semantic analysis and symmetric matrix factorization. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, pages 307–314.

Xun Wang, Masaaki Nishino, Tsutomu Hirao, Katsuhito Sudoh, and Masaaki Nagata. 2016. Exploring text links for coherent multi-document summarization. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. The COLING 2016 Organizing Committee, Osaka, Japan, pages 213–223.

Halbert White. 1982. Maximum likelihood estimation of misspecified models. *Econometrica: Journal of the Econometric Society* pages 1–25.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016a. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR* abs/1609.08144. http://arxiv.org/abs/1609.08144.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016b. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR* abs/1609.08144. http://arxiv.org/abs/1609.08144.

Jin-ge Yao, Xiaojun Wan, and Jianguo Xiao. 2017. Recent advances in document summarization. *Knowledge and Information Systems* 53(2):297–336.

Michihiro Yasunaga, Rui Zhang, Kshitijh Meelu, Ayush Pareek, Krishnan Srinivasan, and Dragomir Radev. 2017. Graph-based neural multi-document summarization. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*. Association for Computational Linguistics, Vancouver, Canada, pages 452–462. http://aclweb.org/anthology/K17-1045.

Mahmood Yousefi-Azar and Len Hamey. 2017. Text summarization using unsupervised deep learning. *Expert Systems with Applications* 68:93–105.

Sheng-hua Zhong, Yan Liu, Bin Li, and Jing Long. 2015. Query-oriented unsupervised multi-document summarization via deep learning model. *Expert systems with applications* 42(21):8146–8155.

Qingyu Zhou, Nan Yang, Furu Wei, and Ming Zhou. 2017. Selective encoding for abstractive sentence summarization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, pages 1095–1104. http://aclweb.org/anthology/P17-1101.

# Appendix A

# Sample System Generated Summaries

Table A.1: Randomly selected outputs for our **Extractive** model.

| **Unsupervised Extractive Model** | |
|---|---|
| **Query** | How has Starbucks Coffee attempted to expand and diversify through joint ventures, acquisitions, or subsidiaries? |
| **Reference** | The New York Times Co. and Starbucks Coffee Co. said Tuesday that they had agreed to the sale of The New York Times newspapers in Starbucks stores for three years in exchange for The Times' advertising promotion of Starbucks. The agreement calls for The Times to be sold at all American Starbucks stores. Some stores may also offer local newspapers and other national newspapers. The Times will use its advertising resources to promote the products and retail outlets of Starbucks, which is based in Seattle. Starbucks will also consider acquisitions and investments in other companies that provide some of the products to be sold on the Web site. There will also be a link between The New York Times' circulation Web site (www.homedelivery.nytimes.com) and the Starbucks Web site (www.starbucks.com). |
| **System Output** | Starbucks will also consider acquisitions and investments in other companies that provide some of the products to be sold on the Web site. The Times will use its advertising resources to promote the products and retail outlets of Starbucks, which is based in Seattle. It also wants to expand overseas. The company has big plans to continue leveraging that loyalty to its brand built so carefully at the retail coffee shops. In addition, Starbucks will promote Kozmo and train its sales staff to answer questions about the Internet company. And though e-commerce companies are loath to open up stores for this reason alone, they are clearly in the market for help from companies that can help streamline returns. The agreement calls for The Times to be sold at all American Starbucks stores. |

Table A.2: Randomly selected outputs for our **Abstractive Reinforcement Learning** model

| **Reinforcement Learning Model** | |
|---|---|
| **Query** | Describe the drug Interferon, its uses, effectiveness, patient tolerance and side effects. |
| **Reference** | Interferon is a protein isolated from human cells. Researchers in China have succeeded in developing recombinant a-1b Interferon from healthy leukocyte cells. Interferon beta-1a, known by the brand name Avonex is a genetically engineered replica of a naturally occurring human protein. Interferon is expensive with a year's supply in some applications running as high as \$18,000. For many years, Interferon was the only approved drug for the treatment of hepatitis C. The traditional drug, interferon, approved in Canada in 1991, has helped in 40 percent of the cases, but the benefits are generally not long lasting as less than 15 percent of the patients achieve complete remission after the treatment. A more effective treatment for hepatitis C was to use the drug ribavarin in combination with Interferon. Interferon alpha-2b is used to treat viral diseases, including Ebola, and is also used to treat some cancer. It is used to slow down the debilitating neurological disease, multiple sclerosis. Interferon has also saved many children with life threatening growths, called hemangiomas by its ability to knock out a tumor's ability to recruit blood vessels from nearby tissues. Interferon is known to produce flu-like side effects. For some people, interferon treatment can be nearly as devastating as the disease it's meant to cure. |
| **System Output** | A drug called interferon approved in Canada in 1991 helps in 40 percent of cases, but the benefits are short-lived. The only approved treatment is interferon alpha, which can cause flu-like side effects. Researchers say not all hepatitis c patients need rebetron, which can have severe side effects including flulike symptoms, anemia, depression and suicidal behavior. The US food and drug administration approved use of another drug , ribavirin, the treatment is a combination of interferon and ribavirin. Some doctors and patient groups have criticized schering-plough for refusing to "unbundle" the two drugs. The other drug in the kit is schering-plough's brand of alpha interferon. Schering-plough has been the sole source of ribavirin. |

Table A.3: Randomly selected outputs for our **Abstractive Transformer** models.

| **Transformer Model** | |
|---|---|
| **Query** | Describe steps taken and worldwide reaction prior to introduction of the Euro on January 1, 1999. Include predictions and expectations reported in the press. |
| **Reference** | Prior to the scheduled introduction of the euro, polls indicated most EU citizens were in favor and believe it will be successful. However, only 20% felt well informed about upcoming changes. Despite media skepticism, the European Commission maintained it would be introduced on schedule January 1, 1999, initially for cashless trading. The founding euro countries were Belgium, Germany, Spain, France, Ireland, Italy, Luxembourg, the Netherlands, Austria, Portugal and Finland. Euro design development began in 1996. That year, Britain's mortgage lenders prepared for euro mortgages. In 1997, the code EUR was ascribed to the euro; Zambia expected to benefit from the currency; and Germany's Bundesbank claimed investors would benefit. Italy was to make provisions for Vatican City and San Marino, which used Italian lire, to use euros. France was to do the same for Monaco, St. Pierre and Miquelon, which used French francs. Signs of the euro's acceptance included: two international banks quoting prices in the euro; China welcoming its initiation; Eurostat offering euro-related information online; Bulgaria moving to link its currency with the euro; and Thailand considering using it in foreign reserves. However, the European Central Bank announced it wouldn't encourage it as reserve currency. By late 1998, Romania's Central Bank included the euro in hard currency exchange rates and performed euro services. The Reserve Bank of India permitted Indian banks to do euro transactions. France, Finland, Belgium and Spain had begun production of euros. Germany was preparing to do so. Euros would become legal tender on January 1, 2002. |
| **System Output** | The euro will be launched on January 1, 1999, new code has been issued to allow progress to the European single currency, EU member nations agreed to the EU single currency. France has already struck one billion coins of the European single currency euro. Euro bank notes and coins will become legal tender on January 1, 2002. European union 's single-currency plan will take effect in 11 of its member states on January 1, 1999. German economics minister Guenter Rexrodt says German companies are not prepared for the shift to the single European currency, the euro, which is due to be launched on, only 20 percent of German companies are prepared for the euro. Private investors are set to benefit from the 1999 introduction of the single European currency, investors are also set to benefit from reduced transaction costs. |