QUERY FOCUSED ABSTRACTIVE SUMMARIZATION USING NEURAL NETWORKS

CHUDAMANI ARYAL
Bachelor of Science in Computer Science, University of Texas at Arlington, 2012

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

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

QUERY FOCUSED ABSTRACTIVE SUMMARIZATION USING NEURAL NETWORKS

CHUDAMANI ARYAL

Date of Defence: April 8, 2019

Dr. Yllias Chali		
Supervisor	Professor	Ph.D.
Dr. Wendy Osborn		
Thesis Examination Committee Member	Associate Professor	Ph.D.
Dr. John Zhang		
Thesis Examination Committee Member	Associate Professor	Ph.D.
Dr. Howard Cheng		
Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

This thesis work is dedicated to my dearest brother, **Sagar Aryal**, who is the only reason why I am here. Without his deep faith and unselfish support, I would not have become who I am today. He stood by me when things looked bleak and led me through the valley of darkness with light of hope and support.

This thesis work is also dedicated to my beloved wife, **Nirmala Pandey**, who has been a constant source of support and encouragement during the challenges of graduate school and life. She made sure that each and every second of these two years was fully dedicated only for study, research and thesis work.

I am truly thankful to God for having you both in my life.

Abstract

Query-focused abstractive document summarization (QFADS) is a process of shortening a document into a summary while keeping the context of query in mind. We implemented a model consisting of a novel selective mechanism for QFADS. A selective mechanism was used for improving the representation of a long input (passage) sequence. We conducted experiments on the Debatepedia dataset, a recently developed dataset for query-focused abstractive summarization task, which showed that our model outperforms the state-of-the art model in all ROUGE scores. Also, we proposed three models all of which consist of a coarse-to-fine approach and a novel selective mechanism for query-focused abstractive multi document summarization (QFAMDS). The coarse-to-fine approach was used to reduce the length of the passage input from multiple documents. We conducted experiments on the MS MARCO dataset, a recently developed large scale dataset by Microsoft for reading comprehension, and have reported our scores using various evaluation metrics.

Acknowledgments

I would first like to express my most sincere gratitude to my supervisor, **Prof. Yllias Chali**, for believing in me and giving me a wonderful opportunity to study and research at this reputed university. His support, patience, motivation, and immense knowledge have tremendously helped me complete my Masters Degree in smooth and timely manner. He has been like a father figure to me for these two years. I would not be able to come this far without his help. His guidance helped me at all the time of my study, research and thesis work. I could not have imagined having a better supervisor and mentor than him. I thank you very much, Prof. Chali, from the bottom core of my heart.

I would also like to thank my M.Sc. supervisory committee members **Dr. Wendy Osborn** and **Dr. John Zhang** for their time and effort spent on thesis committee meetings. Dr. Wendy helped me boost my morale when I was going through tough times. Dr Zhang helped me focus and narrow down the problems by carefully evaluating every possible scenario.

I would also like to thank the University of Lethbridge for the financial support during my Masters Degree. I am also thankful to Natural Sciences and Engineering Research Council (NSERC) of Canada for providing our team with a TITAN X GPU machine to carry out our experiments.

I would also like to thank my fellow researchers for their valuable insights and stimulating discussions regarding my thesis topic. Special thanks to **Elozino Ofualagba Egonmwan** for her words of wisdom.

Last but not the least; I would like to thank my wife, my brother, and my parents for providing me with constant support and encouragement throughout my Masters Degree.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

A document summarization is a process of summarizing a document by shortening it to a condensed summary where the summary retains the key information about the document. Manually writing summaries for a huge number of text documents is a very labour intensive task. Also, writing summaries requires a domain knowledge for the document being summarized. With the massive overload of information, there has been a great need to manage and retrieve important information in an efficient way. This problem led to the rise of automatic summarization. Extractive and abstractive methods are two kinds of strategy used in summarizing documents. Extractive methods generate the document summary by obtaining important sentences from the original document. They have the advantage of good grammaticality and the disadvantage of redundancy and incoherence between sentences. Abstractive methods generate the document summary by forming sentences on its own with the help of natural language generation techniques. They have the advantage of conciseness and coherence between sentences and the disadvantage of bad grammaticality.

Query-focused Abstractive Summarization is the process of summarizing a document into a condensed summary based on the context of the query in an abstractive manner. For example, the query “How was the food on Christmas?” on “Christmas” documents would generate a abstractive summary based on food and not cover the events that happened on Christmas day. Generating a query-focused summary from multiple documents is more practical than generating a general summary of a single document as readers want specific information related to their questions from multiple sources.

Until recently, automatic summarization was dominated by traditional summarization techniques and unsupervised information retrieval models. In 2014, Kågebäck et al. (2014) demonstrated that the neural-based continuous vector models have better capabilities than the traditional summarization techniques. This promise for the automatic summarization marked the beginning of the widespread use of neural network-based summarization models.

1.2 Contributions of this Thesis

- We designed a novel selective mechanism for improving the representation of a long input sequence. This selective mechanism helps remove redundant information from the encoded passage and is used in all our proposed models.

- We implemented a model consisting of our novel selective mechanism for query-focused abstractive document summarization using neural networks. Our model uses a sequence-to-sequence network with an Encoder-Decoder Framework with Attention Mechanism architecture to solve the problem of query-focused abstractive document summarization where the query and the document are the input sequences and the summary is the output sequence.

- We implemented three models all of which consisted of our coarse-to-fine approach and our novel selective mechanism for query-focused abstractive multi-document summarization using neural networks. All of our models also use the same sequence-to-sequence network structure of the single document summarization model. The coarse-to-fine approach is the only difference between multi document and single document model.

1.3 Overview of the Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 provides a brief survey of automatic text summarization along with a short introduction of the deep learning techniques used in text summarization. Chapter 3 presents our proposed query-focused abstractive summarization model for the single document setting. Chapter 4 presents our three proposed query-focused abstractive summarization models for the multiple document setting. Chapter 5 provides conclusions and future work.

Chapter 2

Background

2.1 Automatic Text Summarization Using Neural Networks: A survey

2.1.1 Extractive Summarization

Many research works have been done on extractive summarization methods as these methods are comparatively easier than abstractive summarization. Important early research works on extractive summarization methods include Edmundson (1969), Carbonell and Goldstein (1998), and McDonald (2007). Recently with their success, neural network based frameworks are used to tackle the extractive summarization problem. Cheng and Lapata (2016) proposed a neural summarization model by extracting sentences and words. Nallapati et al. (2017) proposed a recurrent neural network (RNN)-based sequence model for the extractive summarization of documents. Zhang et al. (2016) proposed a convolutional neural network based summarization framework which learns sentence features and performs sentence ranking jointly. Ma et al. (2016) proposed a document-level reconstruction framework named DocRebuild, which reconstructs the documents with summary sentences through a neural document model and selects summary sentences to minimize the reconstruction error. Cao et al. (2016) devised a summarization system called AttSum. This system jointly performs query relevance ranking and sentence saliency ranking. Ren et al. (2018) proposed a deep neural network model called sentence relation-based summarization where they studied the use of sentence relations, such as contextual sentence relations (CSR), title sentence relations (TSR), and query sentence relations (QSR) to improve the performance of extractive summarization.

2.1.2 Abstractive Summarization

Considerably, few research works have been done with abstractive summarization methods as these methods are difficult to implement. However, good progress has been made for abstractive summarization using neural network models. Rush et al. (2015) proposed a neural attention model for abstractive sentence summarization based on a local attention model. Gu et al. (2016) proposed a copying mechanism in sequence-to-sequence learning to improve the grammar quality of the generated summaries. Chen et al. (2016) proposed distraction-based neural networks for a document summarization, which avoids the redundancies in the generated summaries. Nallapati et al. (2016) proposed a Recurrent Neural Network (RNN) model which improves training efficiency, captures keywords, handles rare/unseen words, and captures document hierarchy structure. Li et al. (2017) proposed a deep recurrent generative decoder for abstractive text summarization which learns the latent structure information in the reference summary by capturing the summary structure information to improve the generated summary quality. Nayeem et al. (2018) proposed a paraphrastic sentence fusion model which jointly performs sentence fusion and paraphrasing using a skip-gram word embedding model at the sentence level. Niu et al. (2017) proposed a chunk graph and recurrent neural network language model based summarization model where the chunk graph organizes all the information in a sentence cluster and the language model generates the abstractive, readable, and informative summaries. Nema et al. (2017) proposed a diversity driven attention model for query based abstractive summarization, which alleviates the problem of repeating phrases in the system summaries.

However, most of the papers still do not address all the keys factors of summarization like saliency, fluency, coherence and novelty, and do not capture summary structure information all at once. Also, the performance of the state-of-art abstractive summarization model are still comparable to the performance of the state-of-art extractive summarization model; however, the abstractive summarization has the better upper bound than the extractive summarization.

2.2 Evaluation of Summarization Systems

Evaluation is critical for the measurement of developed summarization systems. As what makes a good summary is highly subjective, the evaluation criteria used for measuring the summarization systems still remains unclear. The most common evaluation technique is to compare system-generated summaries (system summaries) with human-created reference summaries. This allows the use of quantitative measures such as precision and recall. Recall-Oriented Understudy for Gisting Evaluation (ROUGE) and Bilingual Evaluation Understudy (BLEU) are the two tools that we have used to evaluate the performance of our models.

2.2.1 Recall-Oriented Understudy for Gisting Evaluation (ROUGE)

Recall-Oriented Understudy for Gisting Evaluation (ROUGE) (Lin, 2004) is an automatic tool to determine the quality of a machine generated summary by comparing it against a reference or a set of reference summaries. There are 4 different ROUGE metrics - namely ROUGE-N (1,2,3,4), ROUGE-L, ROUGE-W, and ROUGE-SU.

- **ROUGE-N** computes the percentage of n-gram overlap between the system and reference summaries. ROUGE-1 refers to the overlap of unigrams between the system summary and reference summary. ROUGE-2 refers to the overlap of bigrams between the system and reference summaries.

- **ROUGE-L** computes the sum of the longest in sequence matches of each reference sentence to the system summary. It considers the sentence-level word orders and automatically identifies the longest in-sequence word overlapping without a pre-defined n-gram length.

- **ROUGE-W** assigns different weights to consecutive in-sequence matches in the Longest Common Sub-sequence (LCS).

- **ROUGE-SU** measures the percentage of overlapping skip-bigrams and unigrams.

A Skip-bigram consists of two words from the sentence with arbitrary gaps in their sentence order. As an example, for the phrase “cat in the hat” the skip-bigrams would be “cat in, cat the, cat hat, in the, in hat, the hat”. Applying skip-bigrams without any constraint on the distance between the words usually produce spurious bigram matchings. Therefore, ROUGE-SU is usually used with a limited maximum skip distance, such as ROUGE-SU4 with maximum skip distance of 4.

Among these above mentioned measures, ROUGE-N is used the most for multi-document summarization research. ROUGE-N can be defined as follows:

$$\textbf{ROUGE-N} = \frac{\sum_{\mathbf{S}\in\mathbf{R}}\sum_{\mathbf{g}_n\in\mathbf{S}}\textbf{Count}_{\textbf{match}}(\mathbf{g}_n)}{\sum_{\mathbf{S}\in\mathbf{R}}\sum_{\mathbf{g}_n\in\mathbf{S}}\textbf{Count}(\mathbf{g}_n)}$$

where n is the length of the n-gram, $Count_{match}(g_n)$ and g_n are the maximum number of n-grams co-occurring in a candidate summary and a set of reference summaries. When multiple reference summaries are used for evaluation, a pairwise summary-level ROUGE-N between a candidate machine generated summary S and every human produced reference r_i from the reference set $R = \{r_1, r_2, \ldots, r_n\}$ is computed. The final ROUGE-N score is then obtained by taking the maximum of the summary-level ROUGE-N scores as follows:

$$\textbf{ROUGE-N}_{\textbf{multi}} = \textbf{argmax}_i\ (\textbf{ROUGE-N}(\mathbf{r}_i, \mathbf{s}))$$

Recall in the context of ROUGE means how much of the reference summary is the system summary recovering or capturing. The recall can be computed as:

$$\textbf{Recall} = \frac{\textbf{num of overlapping words}}{\textbf{total words in reference summary}}$$

Precision in the context of ROUGE mean how much of the system summary was in fact relevant or needed. The precision is measured as:

$$\textbf{Precision} = \frac{\textbf{num of overlapping words}}{\textbf{total words in system summary}}$$

F-measure is a harmonic mean of precision and recall. It is measured as:

$$\textbf{F-Measure = 2*}\frac{\textbf{Precision*Recall}}{\textbf{Precision + Recall}}$$

For example, let us say we have the following system and reference summaries:

System Summary: the the cat was found under the bed

Reference Summary: the cat was under the bed

Then,

$$\begin{aligned}
&\text{ROUGE-1 Recall} = \frac{6}{6} = 1.0\\
&\text{ROUGE-1 Precision} = \frac{7}{7} = 1.0\\
&\text{ROUGE-1 F-Measure} = 2 * \frac{1.0 * 1.0}{1.0 + 1.0} = 1.0
\end{aligned}$$

Also,

$$\begin{aligned}
&\text{ROUGE-L Recall} = \frac{3}{6} = 0.5\\
&\text{ROUGE-L Precision} = \frac{3}{7} = 0.43\\
&\text{ROUGE-L F-Measure} = 2 * \frac{0.5 * 0.43}{0.5 + 0.43} = 0.46
\end{aligned}$$

In this thesis, we report ROUGE scores for all our proposed models.

2.2.2 Bilingual Evaluation Understudy (BLEU)

Bilingual Evaluation Understudy (BLEU) (Papineni et al., 2002) is also an automatic tool to determine the quality of a machine generated summaries by comparing them against a reference or a set of reference summaries. BLEU score is more precision oriented than

ROUGE score. A combined BLEU score calculates the precision for n-grams overlap (of size 1 to 4) between a system generated output and a reference. Also, it adds the brevity penalty when a summary generation is too short. Usually, this score is computed over multiple sentences (i.e., entire corpus), not just single sentences. The formula for the combined BLEU score is as follows:

$$\mathbf{BLEU} = \mathbf{min}(\mathbf{1}, \frac{\textbf{generated summary length}}{\textbf{reference length}})(\prod_{i=1}^{4}\mathbf{precision}_i)^{\frac{1}{4}}$$

The modified precision of each n-gram is calculated by finding the clipped count of n-gram on reference and dividing that by the total count of n-gram on system summary. Clipped count is the maximum the number of words present in the reference summary. The formula for modified precision is as follows:

$$\mathbf{Precision}_{\textbf{n-gram}} = \frac{\sum_{\textbf{n-gram}\in\textbf{System Summary}} \mathbf{Count}_{\textbf{clipped}}(\textbf{n-gram})}{\sum_{\textbf{n-gram}\in\textbf{System Summary}} \mathbf{Count}(\textbf{n-gram})}$$

For example, let us say we have the following system and reference summaries:

System Summary A: Israeli officials responsibility of airport security

Reference Summary: Israeli officials are responsible for airport security

System Summary B: airport security Israeli official are responsible

Here, “airport” is an example of 1-gram match for System A and “Israeli officials” is an example of 2-gram match for System A. There are no 3-gram and 4-gram matches for System A.

However, “Israeli officials are” is an example of 3-gram match for System B and “Israeli officials are responsible” is an example of 4-gram match for System B.

The brevity penalty for both Systems is $\frac{6}{7}$ as both Systems have an output length of 6 and a reference length of 7.

Hence, the combined BLEU scores for each Systems are as follows:

$$\text{System A: BLEU} = min(1, \frac{6}{7}) * (\frac{4}{6} * \frac{2}{5} * \frac{0}{4} * \frac{0}{3})^{\frac{1}{4}} = 0.0$$

$$\text{System B: BLEU} = min(1, \frac{6}{7}) * (\frac{6}{6} * \frac{4}{5} * \frac{2}{4} * \frac{1}{3})^{\frac{1}{4}} = 0.52$$

In this thesis, we report BLEU scores for all our proposed query-focused abstractive multi-document summarization models.

Now using the same example from ROUGE, let us compare these two evaluation metrics. Using ROUGE formula, we calculated that ROUGE-1 Precision was 1.0. Now, let us calculate the BLEU-1 precision using the modified precision formula.

$$\text{BLEU-1 Precision} = \frac{2(the) + 1(cat) + 1(was) + 1(under) + 1(bed)}{7} = \frac{6}{7} = 0.667$$

Here, we can see that the BLEU score accounts for the extra *the* in the system summary as the maximum numbers of *the* word present in reference summary (i.e., 2) is taken to consideration while calculating the modified precision.

2.3 Machine Learning (ML)

A document summarization is a sub domain of Natural Language Processing (NLP) that deals with extracting or abstracting summaries from huge chunks of texts. There are two main types of techniques used for summarization: NLP-based techniques and deep machine learning-based techniques. In this thesis, inspired by the popularity of neural networks, we use deep learning-based techniques for document summarization. Machine learning (ML) is the study of algorithms and mathematical models that computer systems use to progressively improve their performance on a specific task such as summarization, translation, and music generation. ML gives computers the capability to automatically learn from data without exactly telling the machine what to learn. Machine learning can be roughly separated into three categories:

- Supervised learning: The machine learning program is given both the input data and the corresponding labels. This means that the learning data has to be labelled by a human being beforehand if it is not already labelled. Classification and regression are examples of supervised learning algorithm.

- Unsupervised learning: No labels are provided to the learning algorithm. The algorithm has to determine the clustering of the input data. Clustering and dimensionality reduction are two examples of unsupervised learning algorithms.

- Reinforcement learning: A computer program dynamically interacts with its environment. This means that the program receives positive and/or negative feedback to improve it performance. Game playing and control problems such as elevator scheduling are examples of reinforcement learning algorithm.

In this thesis, our query based summarization task is supervised learning. Although there are various approaches within machine learning to solve the problem of query based summarization, we are focusing mainly on deep neural network based frameworks.

2.4 Deep Neural Networks (DNN)

The idea of an artificial neural networks (ANN) was inspired by the functionality of a human brain where the brain is a network of neurons. In an ANN, each neuron is composed of an input unit, a processing unit, and an output unit. The network is composed of numbers of simple, highly interconnected neurons. The neurons process information by its dynamic state response to the inputs. The most common form of an ANN is a feed-forward neural network (FFNN). The first layer of an FFNN is an input layer and the last layer of an FFNN is an output layer. All the layers that are in between the input layer and the output layer are called hidden layers. An FFNN can have one or more hidden layers. If the network has more than one hidden layer, it is called a deep neural network. Although it is more computationally expensive than a shallow neural network (i.e., network with one hidden layer), a deep neural network shows greater promise on learning a task. In this thesis, we use a deep neural network for the task of query-focused abstractive summarization.

2.5 Programming Aspect

2.5.1 Python

The Python programming language is used for the implementation of our query based abstractive summarization strategy. By far, Python remains the most popular language for use in machine learning. This language was built for readability, versatility and ease of use. It can handle large datasets and has a large community support. Python provides libraries and packages for many tasks from data pre-processing to difficult machine learning algorithms. Due to its minimal syntax, it is also easy and quick to experiment with new ideas and code prototypes in this language.

2.5.2 TensorFlow

Introduced by Abadi et al. (2016), the TensorFlow library is used to implement our query based summarization strategy. It is a Python-friendly open source library for numerical computation that makes machine learning faster and easier. TensorFlow can train and run deep neural networks for sequence-to-sequence models in NLP. The single biggest benefit TensorFlow provides for machine learning development is abstraction. Instead of dealing with the specific details of implementing algorithms, the developer can focus on the overall logic of the application. TensorFlow takes care of the details behind the scenes.

2.6 Word Representation

Word Representation means representing each part of a word in a way that the computer can understand. There are many different ways of representing words in a document. Word embedding is a vectorized form to represent the words. It is an extremely popular word representation used in many natural language processing applications, such as document classification, text summarization and question answering. The word embedding method was first introduced by Bengio et al. (2003).

2.6.1 One Hot Vector

One hot vector is a word representation in which the sequence of the vector consist of zeros and a single one where zero represent the absence of the word in the dictionary and one represent the presence of the word in the dictionary. The size of the vector is equal to the number of words in the dictionary. For example, let us say there are four words in a dictionary: refrigerator, oven, microwave, and kitchen. Each word from the dictionary can be represented to one hot vector representation as follows:

$$\text{refrigerator} = [1000]$$

$$\text{oven} = [0100]$$

$$\text{microwave} = [0010]$$

$$\text{kitchen} = [0001]$$

Although this is very simple to implement, it fails to capture meaningful representation between two words or sentences. For example, let us say there are two words, oven and microwave. According to their one hot vector representation, these two are different words. However, these two words can have a similar meaning. As a result, one hot encoding is not a common way for representing words.

2.6.2 Word2Vec Word Embeddings

The Word2Vec model is an efficient solution to the problem of one hot vector. Developed by Mikolov et al. (2013), Word2Vec is one of the most popular techniques to learn word embeddings using shallow neural network. Word2Vec uses the surrounding words to represent the target words with a neural network whose hidden layer encodes the word representation. A large amount of text is used to learn the word embeddings in an unsupervised manner. The dimension of Word2Vec word embeddings ranges from 50 to 300 where each dimension denotes a specific feature. As a result, semantically similar words are placed as nearby points to each other because features of similar words would also have similar feature values. Figure 2.1[1] shows a word-to-word similarity embedded using the Word2Vec model. This figure shows an n-dimension Word2Vec space embedded into a 2-d Word2Vec space using a dimensionality reduction technique (F.R.S., 1901). Mostly, the pre-trained models are used to convert the words into the vector representation.

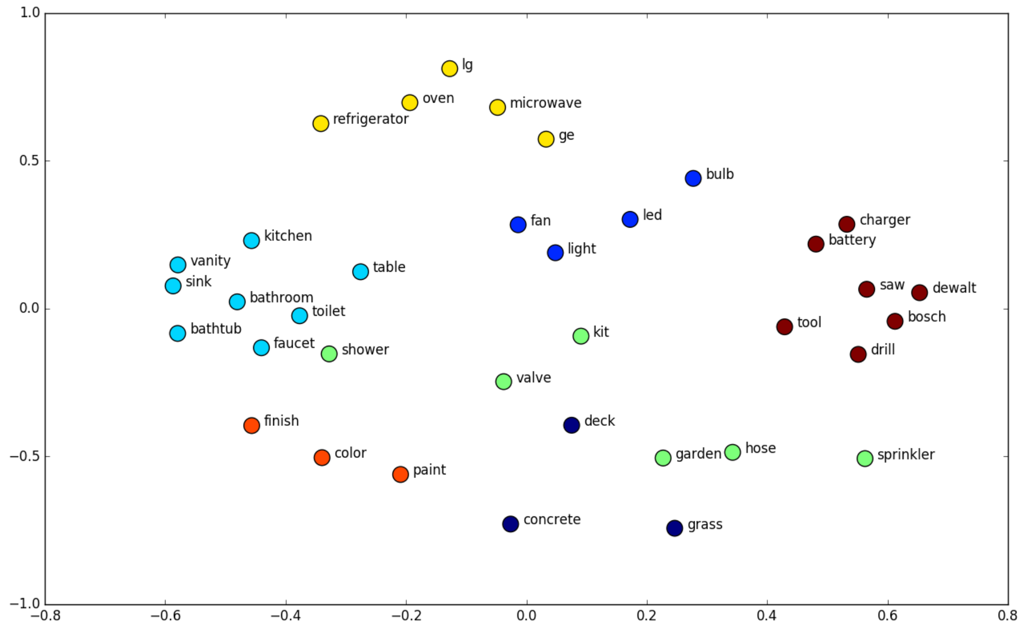

Figure 2.1: 2D word embedding space, where similar words are found in similar locations.

There are two types of Word2Vec models: Skip-gram and Continuous Bag of Words (CBOW).

Skip-gram: For skip-gram, the input is the target word and the outputs are the words surrounding the target words. For example, in the sentence "I have a good friend", the input would be "a", and the output is "I", "have", "good", and "friend". The input would be "a"

[1]Source: https://www.shanelynn.ie/get-busy-with-word-embeddings-introduction/

because we want to train the embedding for this word. Also, let us assume, the window size is five for this example. All the input and output words are converted into one-hot vectors using the method described earlier and they all have same dimension as the dictionary size is constant. The network contains one hidden layer whose dimension is equal to the embedding size, which is smaller than the one hot vector size of input/output words. At the end of the output layer, a softmax activation function is applied so that each element of the output vector describes how likely a specific word will appear in that particular feature. Figure 2.2[2] visualizes the network structure.

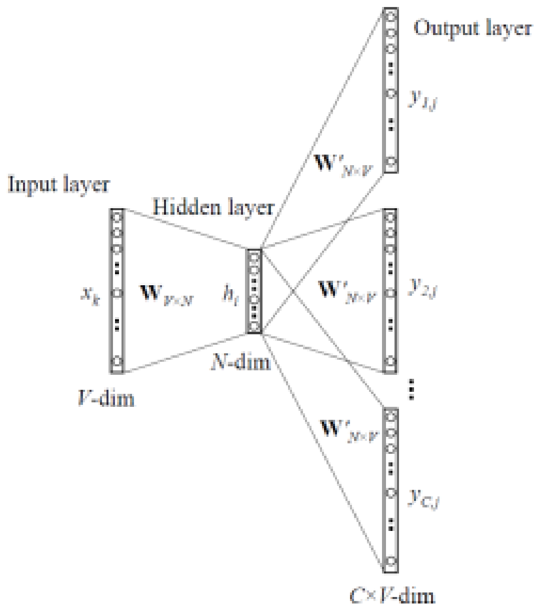

Figure 2.2: Skip-gram Word2Vec Model.

The vectors extracted using this method are more meaningful in terms of describing the relationship between words. Figure 2.1 states that the word *oven* and *microwave* are in fact similar. Based on the previous example, the one hot vector approach was not able to extract this important semantic relationship as it considered the two words as two different words. Also, the word2vec vectors obtained by subtracting two related words sometimes express a meaningful concept such as gender or verb tense, as shown in Figure 2.3[3].

[2]Source: https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
[3]Source: https://towardsdatascience.com/word-embedding-with-word2vec-and-fasttext-a209c1d3e12c

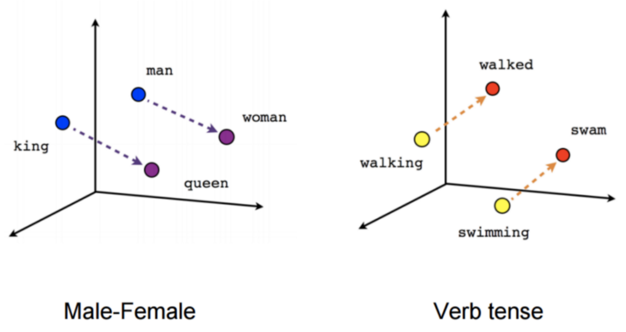

Figure 2.3: Visualization of meaningful concepts such as gender or verb tense.

CBOW: Continuous Bag of Words (CBOW) model finds which word is most likely to appear, given a context. As a result, CBOW is very similar to the skip-gram model except that the input and the output data are exchanged. In the case of Skip-gram, it feeds in one one-hot encoded vector as input, while CBOW feeds the whole context sentence into the neural network. For a Skip-gram, the outputs are the words surrounding the target words, whereas, for CBOW the output is the target word.

For example, let us assume we only have two sentences using a word *a*, “He is a nice man” and “She is a wise woman”. To compute the word representation for the word *a*, we need to feed in all sentences containing *a*, two sentences in our example, “He is nice man”, and “She is wise woman” into the neural network and take the average of the value in the hidden layer. The hidden layer of a CBOW is similar to a Skip-gram model. Figure 2.4[4] visualizes the network structure.

2.6.3 FastText Word Embedding

FastText is an extension to Word2Vec proposed by Bojanowski et al. (2017). Instead of feeding individual words into the Neural Network, FastText breaks words into several n-grams (sub-words). For instance, the tri-grams for the word *vector* is *vec*, *ect*, *cto*, and *tor* (ignoring the starting and ending of boundaries of words). The word embedding vector for *vector* will be the sum of all these n-grams. After training the Neural Network, we will

[4]Source: http://www.claudiobellei.com/2018/01/06/backprop-word2vec/

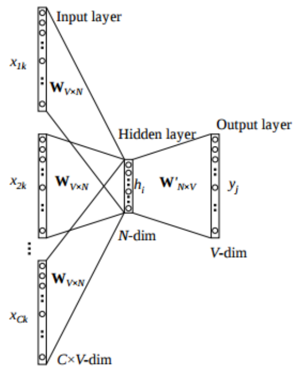
Input layer
x_{1k}
$\mathbf{W}_{V \times N}$
Hidden layer
Output layer
x_{2k}
$\mathbf{W}_{V \times N}$
h_i
$\mathbf{W}'_{N \times V}$
y_j
N-dim
V-dim
$\mathbf{W}_{V \times N}$
x_{Ck}
C×V-dim

Figure 2.4: CBOW Word2Vec Model

have word embeddings for all the n-grams given the training dataset. Rare words can now be properly represented since it is highly likely that some of their n-grams also appears in other words.

2.7 Sentence Embedding

Similar to the word embedding, a sentence embedding is a technique to map the sentences into vectors of real numbers. There are currently many competing schemes for learning sentence embeddings. While simple baselines like averaging word embeddings consistently give strong results, a few novel unsupervised and supervised approaches, as well as multi-task learning schemes, have emerged in late 2017-early 2018 and have led to interesting improvements.

2.7.1 Universal Sentence Encoder (USE)

Google's Universal Sentence Encoder (USE) model (Cer et al., 2018) is recently proposed promising model. The USE model's encoder uses a transformer-network that is trained on a variety of data sources and tasks with the aim of being useful for a wide variety of natural language understanding tasks. The USE model is trained with a deep averaging network (DAN) encoder. The model is trained and optimized for greater-than-word length text, such as sentences, phrases or short paragraphs. A pre-trained version has been made available for TensorFlow. In this thesis, we use this pre-trained model for sentence embeddings. The input is variable length English text sentence and the output is a 512 dimensional vector.

2.8 Word Mover's Distance (WMD)

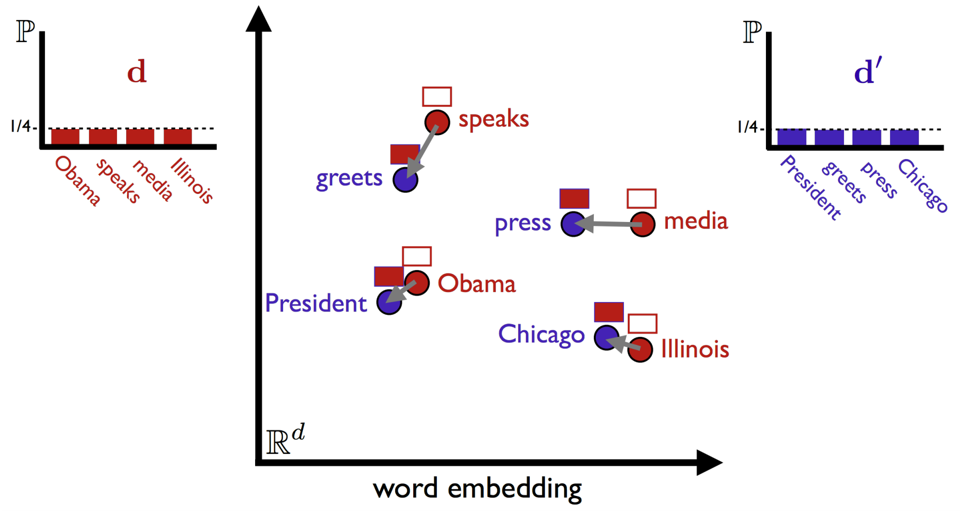

Figure 2.5: WMD: An example diagram from the paper by Kusner et al. (2015)

Introduced by Kusner et al. (2015), Word Mover's Distance (WMD) is proposed for a distance measurement between two texts. In our proposed model, the two texts are two sentences. It measures the distance between two texts as the cumulative sum of the minimum distance each word in one text must move in vector space to the closest word in the other text. To be precise, it uses normalized Bag-of-Words and Word Embeddings to calculate the distance between texts. This distance function has an advantage over other distance functions like Euclidean Distance, Cosine Distance and Jaccard Similarity as WMD was designed to overcome the synonym problem in other distance functions. Also, it leverages Word Embedding's power to overcome those basic distance measurement limitations by using word embeddings to calculate the similarities.

To understand better, for example, let us say we have two sentences.

Sentence 1: Obama speaks to the media in Illinois

Sentence 2: The president greets the press in Chicago

Here, except for the stop words, there are no common words between the two sentences, but both of them are talking about the same topic. WMD uses word embeddings to calculate the distance even though there is no common word. WMD works under the assumption that similar words should have similar vectors.

First, the WMD function retrieves vectors from any pre-trained word embeddings models. Our choice of word vectors model was the FastText model. After this it uses a normalized bag-of-words (nBOW) to represent the weight or importance. It assumes that higher frequency implies that it is more important. Once we have collected the set of vectors and weights of the two sentences, we use the Earth Mover's Distance (EMD) solver to obtain the distance. Introduced by Rubner et al. (2000), given a set of heaps of dirt and a set of holes, where both the heaps and the holes have a point as a location (a vector) and a certain mass or capacity (a weight), EMD determines the least amount of work needed to move all the dirt into the holes or to fill all the holes completely (depending on whether there is more total mass or total capacity).

As a result, the WMD function pairs together the most closest words between sentence 1 and sentence 2 based on the minimum transportation cost to transport every word from sentence 1 to sentence 2. It is illustrated in Figure 2.5 for the above example. Stopwords are removed from the example to reduce the computation cost.

For our proposed models, the WMD is calculated using a $wmd()$ function in Gensim's KeyedVectors module (Řehůřek and Sojka, 2010).

2.9 Neural Networks (NN)

A Neural Network (NN) is composed of many simple neurons. Although these networks are inspired by the neurobiological model, these networks are more closely related to the mathematical and statistical models. A very common kind of a neural network is a feed forward neural network.

2.9.1 Feed Forward Neural Network (FFNN)

A feed forward neural network (FFNN) is an artificial neural network where the information moves in only a forward direction, from input nodes, through hidden nodes to output nodes. There are no cycles or loops in the network. It is the first proposed and a simplest form of a neural network. McCulloch and Pitts (1988) created a computational model for a neural networks based on mathematics and algorithms called threshold logic. This model ignited the spark for the neural network research which led to the application of neural networks to artificial intelligence.

The simplest kind of neural network in the field of artificial intelligence is a single-layer perceptron network. In this network, the inputs are fed directly to the outputs via a series of weights. The weighted sum of the inputs is calculated for each node. If the value is above a certain threshold the neuron fires and takes the activated value; otherwise the value is below a certain threshold and the neuron misses and takes the deactivated value.

Different than a single-layer perception network, a multi-layer perceptron (MLP) network consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to all the neurons of the subsequent layers.

2.10 Recurrent Neural Network (RNN)

A recurrent neural network (RNN) is a type of advanced artificial neural network (ANN) that involves directed cycles in memory as opposed to an FFNN which has no cycles. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. In the case of an RNN, however, the output of the current element is dependent on an RNN's previous computation. For example, if we wanted to predict the next word in a sentence, it would be easier to predict the word if we knew which words came before it. An RNN keeps tracks of those words in a form of memory which captures the information about what has been calculated so far. A traditional neural network will not be able to do that as it does not have access to previous computations. As a result, it is easier to guess the word using an RNN instead of a traditional neural network. The purpose of this RNN is to make use of the previous sequential information in that sentence. In theory, an RNN can make use of previous information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps. We will discuss more about this in a later section. A typical RNN is shown in Figure 2.6[5]:

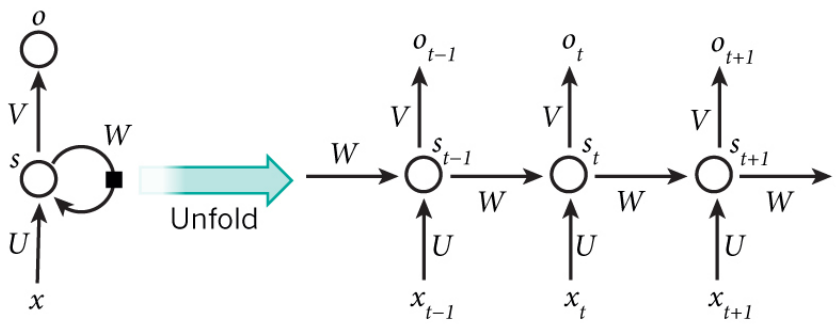

Figure 2.6: Unrolled Into Full Recurrent Neural Network

Figure 2.6 shows an RNN being unrolled into a full network. Unrolling is to write out the network for the complete sequence. For example, if the sequence is a sentence of five words, the network would be unrolled into a five-layer neural network, one layer for each word where output layer of previous layer in included in the computation of current layer. The standard RNN contains following computations:

$$\mathbf{s}_t = \mathbf{f}(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{s}_{t-1})$$

$$\mathbf{o}_t = \mathbf{softmax}(\mathbf{V}\mathbf{s}_t)$$

Here, U, V, and W are learnable parameters. s_t is the hidden state at time step t. It is the so-called "memory" of the network. The hidden state s_t is calculated based on the previous hidden state, s_{t-1} and x_t, the input at the current time step t. The function f usually is a non-linear function such as tanh or ReLU. o_t is the output at step t. For example, if we

[5]Source: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. $softmax$ represents a softmax function which takes an un-normalized vector, and normalizes it into a probability distribution.

There are a few important things to note here:

- The hidden state s_t represents the "memory" of the network. s_t captures information about what happened over all the previous time steps. This is unlike a traditional neural network. The output at step o_t is calculated solely based on the memory at time t. As briefly mentioned above, it is more complicated in practice because s_t typically can only capture information from previous few steps.
- Unlike a traditional deep neural network, which uses different parameters at each layer, a RNN shares the same parameters (U,V,W above) across all steps. This reflects the fact that we are performing the same task at each step, just with different inputs. This greatly reduces the total number of parameters we need to learn.

Due to its difference to traditional neural networks, RNNs have shown great success in many NLP tasks. The most commonly used type of RNNs are Long Short Term Memory (LSTMs) and Gated Recurrent Units (GRUs), which are much better at capturing long-term dependencies than standard RNNs. LSTMs and GRUs have a different way of computing the hidden state. They will be discussed more in depth in the following section. Some example applications of RNNs in NLP are: Language Modeling and Text Generation, Machine Translation, Speech Recognition and Image Description Generation.

2.10.1 Long Short-Term Memory (LSTM)

Introduced by Hochreiter and Schmidhuber (1997), long short term memory (LSTM) networks are a special kind of RNNs which are capable of avoiding the long-distance dependencies problem (Bengio et al., 1994). They work exceptionally well, and are widely used on a large variety of NLP problems recently.

Although standard RNNs are good at connecting previous information to the present situation, they can not connect previous information to the present situation if the gap between the previous information and the current situation is too long. Standard RNNs become unable to learn to connect the information. However, an LSTM is good at learning information having longer gaps. For example, if we had to guess the next word for the following sentence: “I do some research work at the university. I am a ”. A standard RNN would know that the next word would be a noun as the last word is *a*. However, it would not be able to narrow down which noun the next word would be as it would not be able to learn to connect too far to the word *research*. However, an LSTM would be able to learn to connect the information to the present situation and would guess *researcher* as the next word, which is the correct guess.

LSTMs do not have a fundamentally different architecture from standard RNNs, but they use a different function to compute the hidden state. 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 have the ability to avoid the long term dependencies problem. We can formulate the LSTM denoted as a function $LSTM$, as follows:

$$\mathbf{s}_t = \mathbf{LSTM}(\mathbf{x}_t, \mathbf{s}_{t-1})$$

The hidden state s_t of the LSTM contains h_t and c_t where c_t is the current cell memory of the LSTM and h_t is the hidden output of the LSTM. The above $LSTM$ function contains the formulations from Hochreiter and Schmidhuber (1997),

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i) \tag{2.1}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f) \tag{2.2}$$

$$\tilde{\mathbf{c}_t} = \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c) \tag{2.3}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}_t} \tag{2.4}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o) \tag{2.5}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{2.6}$$

Here, x_t represents the input vector, while $i_t, f_t, \tilde{c_t}, c_t, o_t, h_t$ represent the input gate, forget gate, cell candidate, memory cell, output gate, and output vector respectively. W and b are learnable model parameters. σ is the sigmoid function. *tanh* is the hyperbolic tangent function, and $\odot$ denotes an element-wise product operation as shown in Figure 2.7[6].

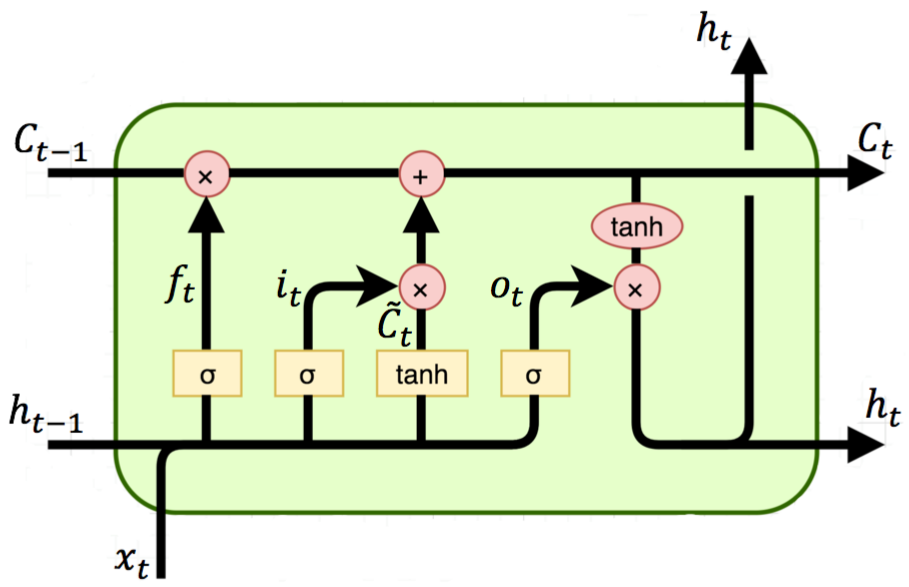

Figure 2.7: Long Short-Term Memory

[6]Source: https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57

2.10.2 Gated Recurrent Unit (GRU)

Introduced by Cho et al. (2014), a Gated Recurrent Unit (GRU) also aims to solve the long-distance dependencies problem associated with a standard RNN by using a different gating mechanism than an LSTM. The GRU and LSTM are considered almost equivalent because both are designed similarly and, in many cases, produce equally excellent results. An GRU controls the flow of information like the LSTM unit, but without having to use a memory unit. Also, it combines the forget and input gates into a single "update gate". GRUs are relatively new, have a less complex structure, train faster, are computationally more efficient and can perform better than LSTM on less training data. We can formulate the GRU denoted as a function *GRU*, as follows:

$$\mathbf{s}_t = \mathbf{GRU}(\mathbf{x}_t, \mathbf{s}_{t-1})$$

The above *GRU* function contains the formulations from Cho et al. (2014),

$$\begin{aligned}
\mathbf{z}_t &= \sigma_g(\mathbf{U}_z\mathbf{x}_t + \mathbf{W}_z\mathbf{s}_{t-1} + \mathbf{b}_z) \\
\mathbf{r}_t &= \sigma_g(\mathbf{U}_r\mathbf{x}_t + \mathbf{W}_r\mathbf{s}_{t-1} + \mathbf{b}_r) \\
\mathbf{s}_t &= z_t \odot \mathbf{s}_{t-1} + (\mathbf{1} - \mathbf{z}_t) \odot \sigma_h(\mathbf{U}_s\mathbf{x}_t + \mathbf{W}_s(\mathbf{r}_t \odot \mathbf{s}_{t-1}) + \mathbf{b}_s)
\end{aligned}$$

Here, x_t and s_t denotes input vector and hidden state, respectively. z_t and r_t denotes the update gate and the reset gate. The U, W, B denotes model parameters. σ_g and σ_h denotes sigmoid function and hyperbolic tangent. The reset gate determines how to combine the new input with the previous memory, and the update gate defines how much of the previous memory to keep as shown in Figure 2.8[7].

[7]Source: https://www.kaggle.com/honeysingh/intro-to-recurrent-neural-networks-lstm-gru

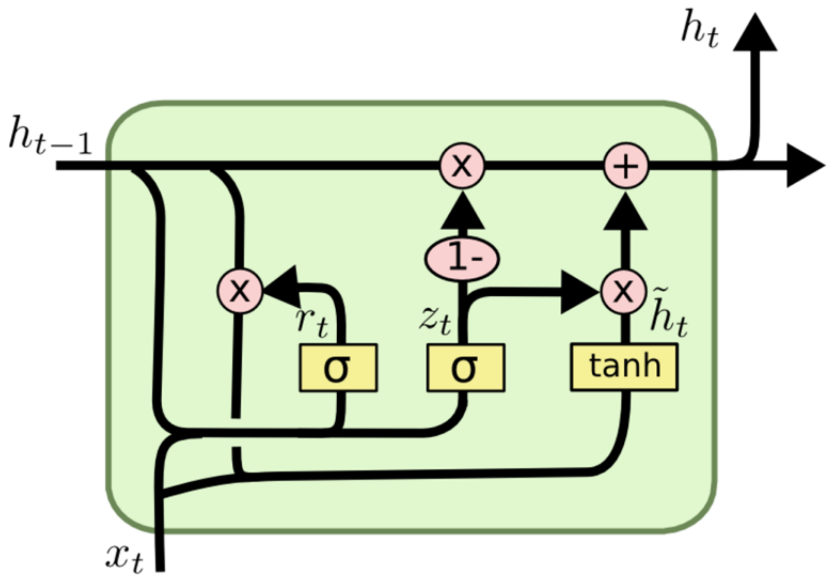

Figure 2.8: Gated Recurrent Unit

2.10.3 Bidirectional Recurrent Neural Network (Bi-RNN)

Over the years researchers have developed more sophisticated types of RNNs to deal with some of the shortcomings of the standard RNN model. Bidirectional RNNs (Bi-RNNs) are based on the idea that the output at time may not only depend on the previous elements in the sequence, but also in future elements in the sequence. For example, to predict a missing word in a sequence we look at both the left and the right context. A bidirectional RNN is two RNNs (forward and backward RNN) stacked on top of each other. Here, the forward RNN encodes the source sequence in its original order $(x_1, x_2, \ldots, x_T)$ from left-to-right and generates a sequence of hidden states $(\overrightarrow{s_1}, \overrightarrow{s_2}, \ldots, \overrightarrow{s_T})$. The backward RNN encodes the source sequence in reverse order, from right-to-left $(x_T, x_{T-1}, \ldots, x_1)$ and generates $(\overleftarrow{s_1}, \overleftarrow{s_2}, \ldots, \overleftarrow{s_T})$. The final hidden state is then computed based on the combined hidden state of both RNNs i.e., $s_i = \left[\overrightarrow{s_i^T}, \overleftarrow{s_i^T}\right]^T$. Figure 2.9[8] shows the conceptual diagram of an Bi-RNN. This figure also shows the difference between a simple RNN and an Bi-RNN. These Bi-RNN can be a simple RNN, LSTM or GRU.

[8]Source: http://colah.github.io/posts/2015-09-NN-Types-FP/

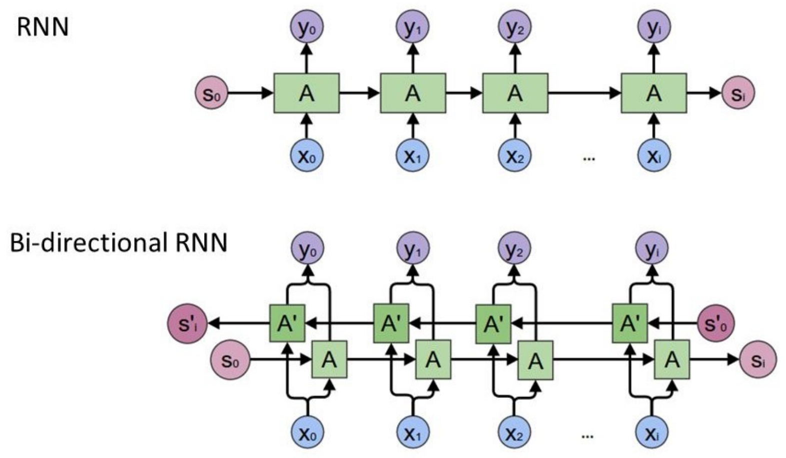

Figure 2.9: An RNN vs. A Bidirectional Recurrent Neural Network

2.10.4 Deep Bidirectional Recurrent Neural Network (D-Bi-RNN)

Deep Bidirectional RNNs (D-Bi-RNNs) are similar to Bidirectional RNNs; however they have multiple layers per time step. For example, let us say we have a 3-layer deep Bi-RNN. Then at every time step t, the output of Layer 1 becomes the input for Layer 2, and the output of Layer 2 becomes the input for Layer 3. In practice this gives us a higher learning capacity; however, we need more training data to show its improved learning capacity. Figure 2.10[9] shows the conceptual diagram of an D-Bi-RNN. An D-Bi-RNN can use an RNN, LSTM or GRU. The calculation at time step t for a 3-layer deep Bi-RNN can be formulated as follows:

$$\mathbf{s}_{1,t} = \mathbf{RNN_1}(\mathbf{x}_t, \mathbf{s}_{1,t-1})$$

$$\mathbf{s}_{2,t} = \mathbf{RNN_2}(\mathbf{s}_{1,t}, \mathbf{s}_{2,t-1})$$

$$\mathbf{s}_{3,t} = \mathbf{RNN_3}(\mathbf{s}_{2,t}, \mathbf{s}_{3,t-1})$$

where $s_{n,t}$ is the hidden state for the nth layer at time step t of an RNN. RNN_n is the RNN function for the nth layer. This formula can be generalized to calculate a n-layer deep

[9]Source: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Bi-RNN.

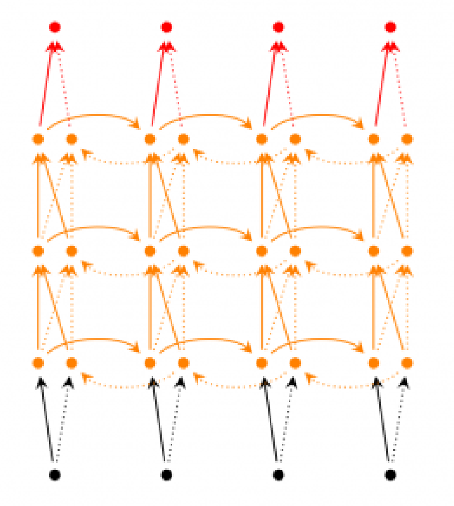

Figure 2.10: A Deep Bidirectional Recurrent Neural Network

2.11 Convolutional Neural Network (CNN)

Proposed by LeCun et al. (1999), a convolutional neural network (CNN) is one of the variants of neural networks used heavily in the field of Computer Vision. It derives its name from the convolution operation which is performed on the hidden layers of the network. The hidden layers of an CNN typically consist of convolutional layers, pooling layers, fully connected layers, and normalization layers. Instead of just using the normal activation function such as a Rectified Linear Unit (ReLU), the convolution and pooling operations are used in addition to the activation function. Weight sharing is one of its main features. As a result, it reduces the number of weights significantly. Also, CNNs are very good feature extractors. They can extract useful attributes from an already trained CNN with its trained weights. The following are the major building blocks of a basic CNN:

Convolutional Layer: A convolutional layer applies a convolution operation on the input layer, passing the results to next layer. A convolution operation is computing a dot product between the filter weight and a small region in the input. This will change the output dimensions depending on the filter size used and number of filters used. Figure 2.11[10] shows the convolution operation. Here, red box in matrix I denotes the small region for convolution, K denotes the weights for the filter and * denotes the dot product. Therefore, I*K denotes the convoluted feature and the 4 shown in green is the dot product of red box and blue box. The matrix for the convoluted feature is filled by sliding the convolution operation from left to right and top to bottom. The sliding operation is determined by a parameter called *stride* in an CNN.

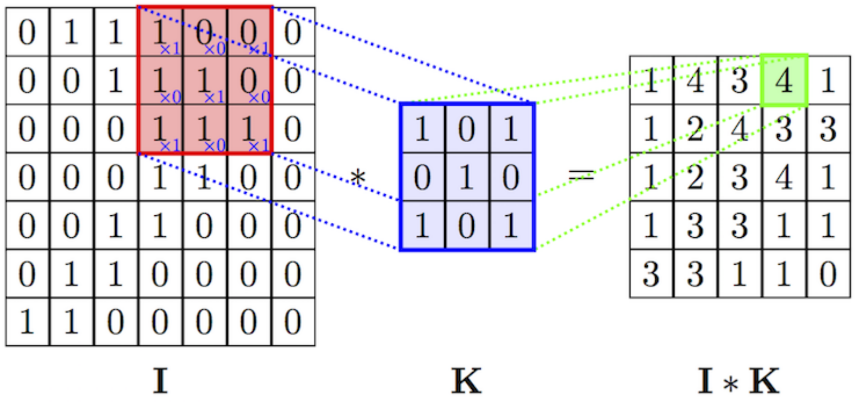

Figure 2.11: Convolution Operation

Activation Function Layer: This layer applies one of the activation functions. Rectifying Linear Unit (ReLU) is the most common activation function in the case of an CNN. The ReLU layer applies the ReLU activation element-wise. It does not change the dimensions of the previous layer. It is a mathematical function, which returns a positive value or 0 in place of previous negative values. Other examples of activation functions for an CNN are: Leaky ReLU, Scaled Exponential Linear Unit (SELU), tanh, sigmoid, and step activation functions.

Pooling Layer: The Pooling layer performs a down-sampling operation along the width and results in the reduction of the dimensions. The sole purpose of pooling is to reduce the number of spatial dimensions. There are various types of pooling in which the most common is max pooling (i.e., taking the maximum element from the window). Figure

[10]Source: http://www.deeplearningessentials.science/convolutionalNetwork/

2.12[11] shows an example of max pooling.

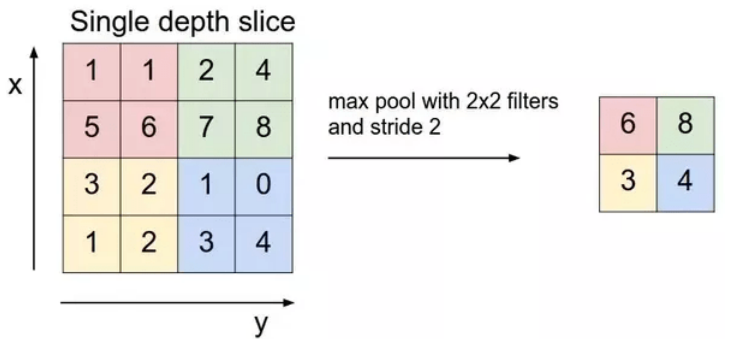

Figure 2.12: Example of max pooling

Stride: Stride indicates by how much we move our window. For example, when we have a stride of one we move across and down a single pixel. With higher stride values, we move a larger number of pixels at a time and hence produce smaller output volumes.

Padding: Padding is used to preserve the boundary information, since without padding they are only traversed once.

Flattening Layer: This layer converts the 3-dimensions (height,width,depth) into a single long vector inorder to feed it to next layer which is usually a fully connected layer or Dense layer. It connects every neuron in one layer to every neuron in the next layer.

Fully Connected Layer: Fully connected layers or dense layers are the same hidden layers consisting of a defined number of neurons connected with elements of adjacent layers that we discussed in Section 2.9.1.

Output Layer: The output layer is also the same as a fully connected layer but the number of neurons depends on our task. For example, a task to classify five sentiments will have five neurons on the output layer.

Figure 2.13[12] shows the complete process of a convolutional neural network. We pass an input image to the first convolutional layer. The convoluted output is obtained as an activation map. The filters applied in the convolution layer extract relevant features from

[11]Source: http://cs231n.github.io/convolutional-networks/
[12]Source: https://towardsdatascience.com/how-to-teach-a-computer-to-see-with-convolutional-neural-networks-96c120827cd1

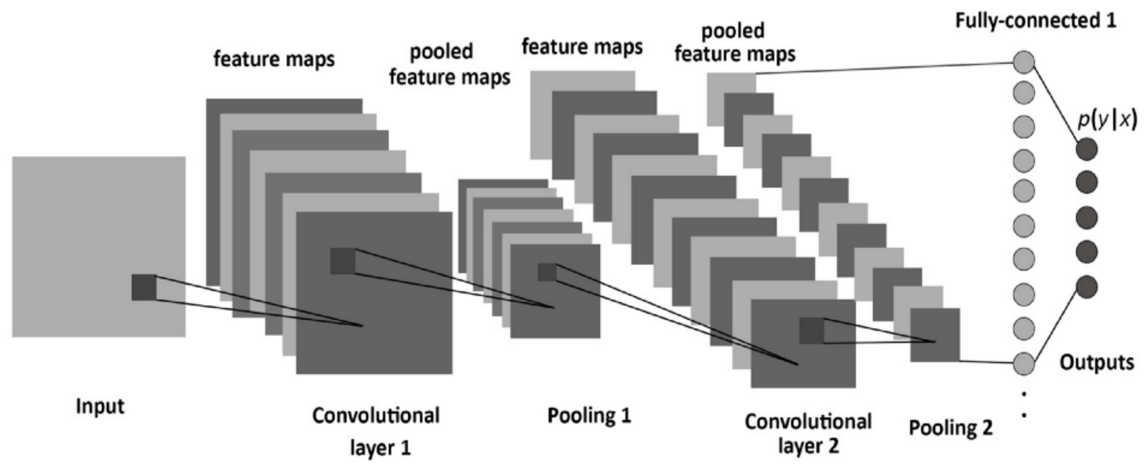

Figure 2.13: Convolutional Neural Network: Putting Everything Together

the input image to pass further. Each filter gives a different feature to help with the correct class prediction. In case we need to retain the size of the image, we use the same padding (zero padding). Otherwise valid padding is used since it helps to reduce the number of features. Pooling layers are then added to further reduce the number of parameters. Several convolution and pooling layers are added before the prediction is made. A convolutional layer helps in extracting features. As we go deeper in the network more specific features are extracted as compared to a shallow network where the features extracted are more generic. The output layer in an CNN as mentioned previously is a fully connected layer, where the input from the other layers is flattened and sent so as to transform the output into the number of classes as desired by the network.

2.12 Model Improvement Techniques

In this section we will discuss some improvement techniques that were used in our proposed model.

2.12.1 Residual Network (ResNet)

A Residual Network (ResNet) is a Neural Network architecture which solves the problem of vanishing gradients by skipping over a layer during gradient computation. It provides a shortcut at each layer during back propagation when it is difficult to compute the gradient. Usually, the activation function at the traditional network's layer is defined as: $y = f(x)$. Here, f(x) can be a convolution, matrix multiplication, or batch normalization. When calculating the gradient during back propagation through time, the gradient always must pass through $f(x)$. This computation can be problematic when these functions are non-linear. ResNet solves this problem by creating a skip connection and by avoiding the gradient computation of "f(x)". Residual connection is formulated as below:

$$\mathbf{y = f(x) + x}$$

Here, "+ x" creates the shortcut to the activation function for the layer. This shortcut allows the gradient to pass backwards directly by skipping over the "f(x)" part. In theory, by stacking these layers, the gradient could skip over all the layers and reach the bottom without being diminished. As a result, this architecture could be used to create a deeper Network without having the issue of vanishing gradient.

2.12.2 Highway Network

The Highway Network is the extension of the Residual Network. For the Residual Network, the activation function consists of the non-linear function and the skip connection in a uniform way; whereas for the Highway Network, the activation function consists of the weighted average of the non-linear function and the skip connection. The weights are learned by the model. These weights determines to what extent each layer should have a skip connection or a nonlinear connection. A Highway Network is formulated as below:

$$\mathbf{y = W\ f(x) + (1 - W)\ x}$$

Here, $f(x)$ and x are similar to the Residual Network. However, this time in Highway Network each non-linear connection and skip connection are governed by the weight W.

This weight is learned by the model itself and determines to what extent each connection should be used during gradient computation.

2.12.3 Linear Layer

The linear layer is a fully connected feed forward layer without having any activation function. A fully connected feed forward network is one of the types of a Feed Forward Network. We already talked about Feed Forward Networks in Section 2.9.1. In a Fully Connected Feed Forward Neural Network (FCFFNN), each unit of one layer are connected to all units of the previous and forward layers. FCFFNN is one of the most basic and simplest neural networks. FCFFNNs are usually governed by an activation function for each unit; however in the linear layer, there is no activation function. The output of each processing unit is just the weighted sum of its inputs. This layer is used when the contribution of each input to another layer is not known and these inputs are passed to the linear layer so the model learns the weights or contributions for each input. Also, we use a linear layer when we know or assume that there exists a linear relationship between one layer and another layer.

2.12.4 Dropout

One of the most common problems while training a model is overfitting as this reduces the model's performance on unseen data. Regularization is a technique which makes slight modifications to the learning algorithm such that the model generalizes better, and thus improves the model's performance on the unseen data. Introduced by Srivastava et al. (2014), dropout is the most sought after regularization technique which produces very good results. Dropout performs better than a normal neural network model. Dropout is usually preferred when we have a large neural network structure in order to introduce more randomness. Hence, we have used the dropout technique in our proposed model.

The term "dropout" refers to dropping out units (i.e., neurons) in a neural network. Chosen randomly, these selected units are not considered during a particular forward or backward pass of the training phase. The probability of choosing how many nodes are dropped is one of the hyperparameters of the model. To understand dropout better, say we have a fully connected neural network architecture where each neuron of one layer are connected to all neurons of another layer. At every layer, a dropout function randomly selects some nodes and removes them along with all of their incoming and outgoing connections. As a result, each iteration has a different set of nodes and this results in a different set of outputs.

2.12.5 Batch Normalization

Introduced by Ioffe and Szegedy (2015), batch normalization is a technique used for improving the performance and stability of a neural network. Batch normalization leads to faster training and can help avoid large weights in the network. It is a technique to provide any layer in a neural network with inputs and outputs that are zero mean variance.

Usually we only normalize our input layer with the hope of scaling our input data and just ignore to normalize the hidden layer. We assumed that only normalizing the input layer is sufficient for our purpose. However, this is not the case. For example, in our neural network, if one of the weights of the hidden layer drastically increases to a big number, then this will result in the exploding of the weights of the subsequent layers. This might trigger the exploding of the whole network. This exploding can be prevented with the help of batch normalization where the outputs of each activation layer are normalized. As the normalization is done per batch basis, the name of batch normalization is given to it. The batch normalization can be formulated as below:

$$\mathbf{z} = \frac{\mathbf{x} - \mathbf{m}}{\mathbf{s}}$$

$$\mathbf{z} = \mathbf{z} * \mathbf{g} + \mathbf{b}$$

Here, the first equation normalizes output from the activation function. The second equation passes the normalized activation output through a linear equation where g and b are trainable parameters. x is the activation output, m is the mean, and s is the standard deviation, and z is the normalized activation output which is the input for the next layer. For example, say there is a batch normalization between hidden layer 1 and hidden layer 2. Then the output of hidden layer 1 is the input to batch normalization (represented by x in above equation). Then the output of the batch normalization will be the input to hidden layer 2 (represented as z in above equation).

2.12.6 Bucketing Mechanism

The bucketing mechanism is implemented in our model to improve the training speed. As bucketing helps reduce the amount of padding, this leads to faster training. To understand it better, let us say we have two buckets [100, 20, 20] and [200, 40, 400]. The first bucket takes the passage length of 100, the query length of 20, and the summary length of 20 and the second one takes the passage length of 200, the query length of 40, and the summary length of 40. Now let us assume we have four data of length (50, 10, 10), (100, 20, 20), (150, 30, 30) and (200, 40, 40). Assuming the batch of 2, and not using bucketing mechanism, if we pick two data, then those two data could be random. For simplicity, let us pick the data of length (50, 10, 10) and (200, 40, 40). Now to pass both the data as a batch, the data of length (50, 10, 10) would have to be padded 150 times for the passage, 30 times for the query and 30 times for the summary. For the second batch with data of length (100, 20, 20) and (150, 30, 30), the data of length (100, 20, 20) would be padded 50 times for the passage, and 10 times for the summary. All together, the total padding would be (150+30+30+50+10+10) = 280 times. Now, if we implement a bucketing mechanism, then data of length (50, 10, 10) and (100, 20, 200) would go under the bucket [100, 20, 20] and the data of length (150, 30, 30) and (200, 40, 40) would go under the bucket [200, 40, 40]. The padding for the first bucket would be 50 times for the passage, 10 times for the query, and 10 times for the summary. The padding for the second bucket would be 50 times for the passage, 10 times for the query, 10 times for the summary. All together, the total padding would be (50+10+10+50+10+10) = 120 times. As a result, with this example we can sense that the bucketing mechanism reduces the unnecessary padding at the end of the sequence. This would lead to less computation and hence lead to faster training of the model. In a way, the bucketing mechanism helps partially sort the data based on the data length. Selection of similar length data for the batch reduces the amount of padding.

2.12.7 Beam Search

Since the summarization task is considered an NP-hard problem, it is computationally expensive to get an exact solution to the decoding the output sequence. We normally use some heuristic algorithm to solve this task. The most basic one is a Greedy Search algorithm. This algorithm is useful in machine translation and many seq2seq models. In a greedy search, we simply calculate the probability distribution of the word at every time step, and select the word that gives us the highest probability, and use it as the next word in our sequence. However, a greedy search is not guaranteed to find the output translation with the highest probability due to the local optimum. One possible solution to this problem is to consider the *n*-best words at each time step of the decoder. A Beam Search is a heuristic search algorithm that explores a graph by expanding the most probable node in a limited set (usually called beam search size). It is often used when the solution space is significantly very large for the applications such as machine translation, speech recognition and natural language generation. It is extremely useful where there is not enough memory available to consider all the possible solutions.

A Beam search builds a search tree using a breadth first search algorithm and sorts the nodes according to the sum of the log probability of the generated words at each level of the tree. A Beam search is similar to a greedy search, but instead of considering only the 1-best word, we consider the *b* best words at each time step of the decoding, where *b* is the beam search size. Thus, the *b* best nodes with highest scores are expanded in the next level. This reduces the space and time requirements significantly. However, as it is a heuristic algorithm, there is no guarantee of a global optimum solution in case of beam search. However, a beam search is often used in decoding process as this algorithm yields better performance than a greedy search. Our proposed model also uses the beam search algorithm during the decoding at the inference or test phase.

2.13 Sequence to Sequence (Seq2Seq) Network

A Sequence to Sequence (Seq2Seq) network, or seq2seq network, is a model consisting of two RNNs (or sometimes an CNN and an RNN) called the encoder and decoder respectively. The encoder reads a variable length input sequence and outputs a single fixed size context vector, and the decoder reads that fixed length context vector to produce a variable length output sequence. The encoder-decoder model provides a pattern for using RNNs or CNNs to address challenging sequence-to-sequence prediction problems, such as machine translation and document summarization. As a result, the Encoder-Decoder architecture with RNNs has become an effective and standard approach for both Neural Machine Translation (NMT) and Sequence-to-Sequence (Seq2Seq) prediction in general.

The key benefits of the network are the ability to train a single end-to-end model directly on input and output sentences and the ability to handle variable length input and output sequences of text. Also, seq2seq network does not rely on human designed features which was one of the major requirements to previous traditional approaches like statistical machine translation. This eliminates the need to heavily pre-process the data.

2.13.1 Encoder-Decoder Framework

The details about the encoder-decoder framework will be discussed in Chapter 3. But in general, the task of an encoder network is to understand the input sequence, and create a smaller dimensional representation of it. This representation is then forwarded to a decoder network which generates a sequence of its own that represents the output. The encoder of a seq2seq network is an RNN that outputs some value for every word from the input sentence. For every input word the encoder outputs a vector and a hidden state, and uses the hidden state for the next input word. The decoder is another RNN that takes the encoder output vector(s) and outputs a sequence of words to create the translation (in case of machine translation) or the summary (in case of document summarization). In the simplest seq2seq decoder we use only last output of the encoder. This last output is sometimes called the context vector as it encodes context from the entire sequence. This context vector is used as the initial hidden state of the decoder. At every step of decoding, the decoder is given an input token and hidden state.

2.13.2 Attention Mechanism

If only the context vector is passed between the encoder and decoder, that single vector carries the burden of encoding the entire sentence. Attention allows the decoder network to "focus" on a different part of the encoder's outputs for every step of the decoder's own outputs. First we calculate a set of attention weights. These will be multiplied by the encoder output vectors to create a weighted combination. The result contains information about that specific part of the input sequence to focus, and thus helps the decoder choose the right output words. Calculating the attention weights is done with another feed-forward layer, using the decoder's hidden state and encoder's hidden outputs as inputs.

2.14 Summary

In this chapter, we presented recent related works in neural summarization research and discussed the necessary background information. All information from this chapter is necessary to understand this thesis work as our proposed models heavily depend on these concepts. From the next chapters, we will start introducing our proposed query-focused summarization models.

Chapter 3

Query Focused Abstractive Document Summarization Using Neural Network

3.1 Introduction

Query-focused abstractive document summarization (QFADS) is a process of summarizing a single document into a condensed summary that focuses on the context of the query where the summary sentences are formed on its own with the help of natural language generation techniques. According to our previous explanation of a seq2seq network, QFADS can be regarded as a sequence mapping task where the source text (i.e., query combined with a document) should be mapped to the target summary. The dataset for QFADS contains (query, document, summary) triplet tuples. Therefore, a sequence-to-sequence learning can be applied to QFADS when the model consists of two encoders (one each for the query and the document) and a decoder. Attention mechanisms have been broadly used in seq2seq models where the decoder focuses on the specific parts of the document to extract information which are considered important for the decoding process. The same attention mechanism can be used for query-focused abstractive summarization as well. Many proposed attention-based seq2seq methods for abstractive summarization have outperformed the traditional statistical methods.

One of the current major problems in the research field of query-focused document summarization is the lack of research on this task. Although attention based seq2seq models have gained popularity on generic abstractive summarization, not enough attention is given to the query-focused abstractive text summarization where the aim is to generate the summary of a document given the context of a query. Although query-focused summarization is more practical than the generic summarization, there has not been enough research conducted in this field. In addition, there does not exist a proper standard dataset which can be used for the task of query-focused summarization.

Also, recent studies have shown that there are various shortcomings to the attention mechanism. Although the purpose of the attention mechanism is to show the alignment relationship between the input sequence and the output sequence, no clear alignment relationship exists between an input sequence and an output sequence (Zhou et al., 2017). The alignment is the match between the target output and the specific input word. High alignment score shows that the decoder was focusing heavily on that input word while generating the target output. Our thesis work is based on the hypothesis that the encoder is too confused and noisy with only simple bidirectional RNN (Bi-RNN) unit. The Bi-RNN unit was initially proposed for sentence summarization; however, current research is using the Bi-RNN to encode the whole document. As a result, the confused encoder output contains noise for the attention. This eventually leads to unclear alignment between the input sequence and the output sequence. Hence, the current attention-based seq2seq model for query-focused and generic abstractive summarization suffer from grammatical errors, repetitions, and poor representations of important ideas from the source document.

To solve the above problems, we propose a model consisting of a novel selective mechanism for the QFADS task. In text summarization, the document texts are usually very long and noisy. The document often contains unnecessary or redundant information. By the time an RNN is encoding the end text of the document, the start encoding information has already vanished due to the inability of the RNN to represent long texts. This results in the bad representations of the long texts, and therefore making the encoder outputs noisy and confused. Although it takes twice the encoding time, an Bi-RNN is an alternative to deal with the problem. However, an Bi-RNN does not represent the middle of a text well when the texts are too long. One way to solve this problem is by selectively encoding a long text. Many words in a long text are unimportant and can be discarded. Using this idea, we propose a novel selective mechanism to better represent long texts. This selective mechanism can reduce the unnecessary information and enhance the important information to represent a long text in a better way. We use a convolutional gated network for the purpose of selective mechanism on a long document text. An CNN architecture is chosen in our mechanism as they are great feature extractors as mentioned in Section 2.11. Also due to its parameter sharing ability, CNNs are great at learning global features of the corpus. CNNs also learn important local features from the document due to its convolution operation. The final selective encoding is done on a document with the consideration of these global and local features. As the queries are relatively short, the selective encoding is only done on documents. We conduct experiments on Debatepedia (Nema et al., 2017) dataset, a recently developed dataset for query-focused abstractive summarization, which shows that our model outperforms the state-of-the art model in ROUGE-1, ROUGE-2, ROUGE-L scores.

3.2 Task Description

Given a query $q = q_1, q_2, \ldots, q_k$ containing k words, a passage $s = s_1, s_2, \ldots, s_n$ containing n sentences, and a sentence $w = w_1, w_2, \ldots, w_l$ containing l words, the task is to generate a context based summary $y = y_1, y_2, \ldots, y_m$ containing m words. This task of a QFADS can be achieved by finding the probability $y*$ such that it maximizes the probability $p(y|q,s)$. Using Bayes theorem, it can be further decomposed into following:

$$\mathbf{y}^* = \mathbf{argmax}_y \prod_{t=1}^{m} \mathbf{p}(\mathbf{y}_t|\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}, \mathbf{q}, \mathbf{s})$$

The above equation for $y*$ can be modeled using the neural network framework.

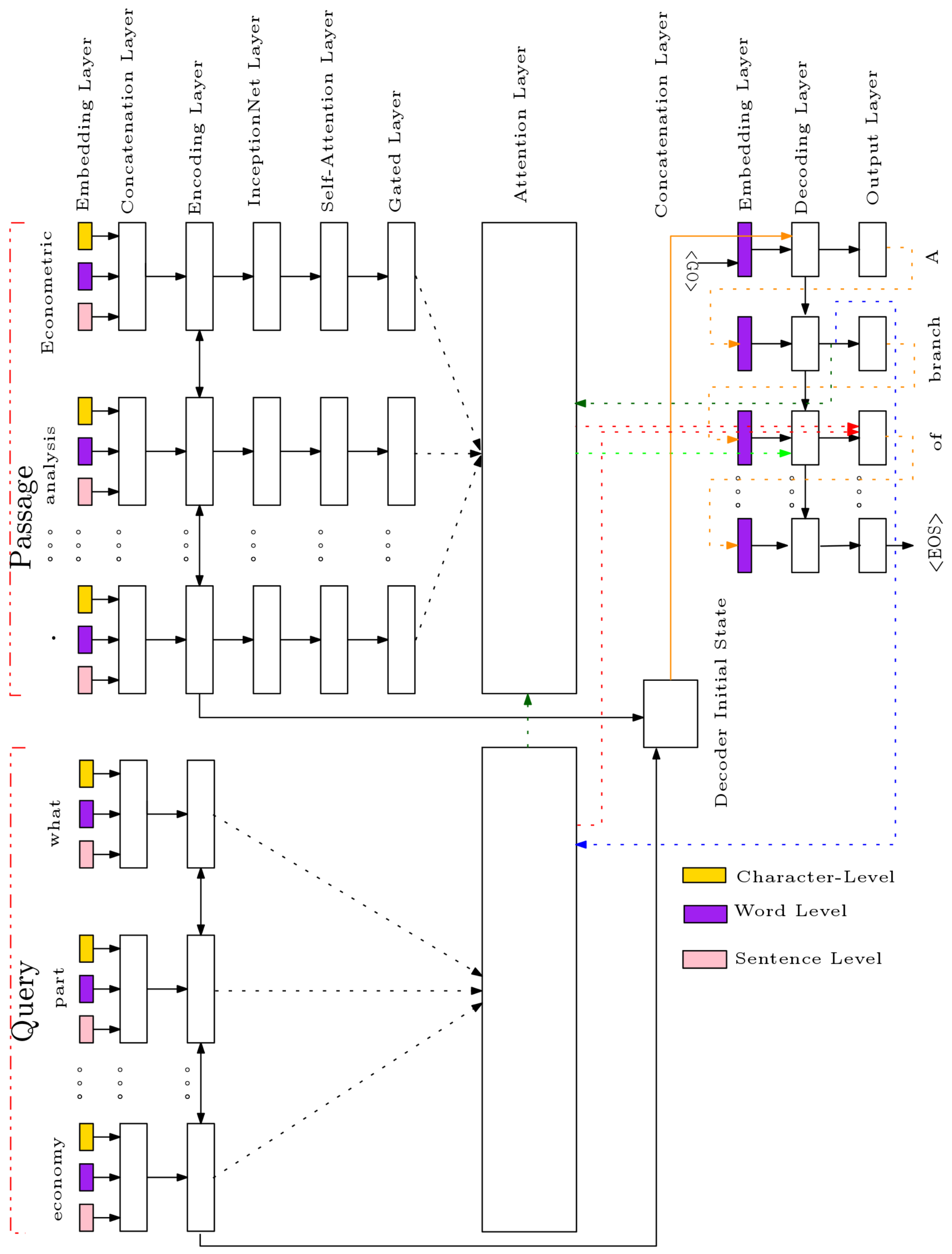

Figure 3.1: Our Proposed Model For Query Focused Abstractive Document Summarization

3.3 Proposed Model: A Complete Diagram

Figure 3.1 shows the complete diagram of our proposed model for the QFADS. The lines with the arrow show the flow of the data from one layer to another layer. The solid lines signify that the data remains same at every time step of decoding process whereas the dotted lines signify that the data may change at every time step of the decoding process. For example, the solid line from the self-attention layer to the gated layer indicates that the data will be same throughout all time steps of the decoding process, but the dotted line between the passage gated layer and the passage attention layer indicate the weights of the attention may change at every time step of the decoding process.

For both the query and the passage of the document:

- The embedding layer consists of three different embeddings: character-level, word-level, and sentence-level embedding and they are denoted by a gold, purple, and pink colored box, respectively.
- The concatenation layer consists of a linear layer and two highway layers to combine all the embeddings into a single embedding.
- The encoding layer encodes the concatenated embeddings using two LSTMs (one each for query and passage). The double-sided arrow on the encoding layer indicates that the LSTMs for both the passage and the query are bidirectional.

Only the passage of the document goes through next three layers (InceptionNet Layer, Self-Attention Layer, and Gated Layer) as a part of our novel selective mechanism.

As it would be too confusing to show all the connections of the attention mechanism for the passage and the query at every time step of the decoding process, we are only showing the connections of the attention mechanism for time step 2 of the decoding process assuming the time step started from zero. The inputs to the query attention are the outputs of the query D-Bi-LSTM encoder and the hidden decoder state at time step 1 (shown in dotted blue). The query attention is used to calculate the query context. The query context is used to calculate the passage attention and to predict the next output word. As a result, the query context is passed to the passage attention layer (shown in dotted dark green) and the current output layer (shown in dotted red) as input. The inputs to the passage attention are the outputs of gated layer, the query context, and the hidden decoder state at time step 1 (shown in dotted dark green). The passage attention is used to calculate the passage context. The passage context is used to calculate the current hidden decoder state and to predict the next output word. As a result, the passage context is passed to the current decoding layer (shown in dotted light green) and the current output layer (shown in dotted red) as input.

The decoder initial state is the input at time step 0 (shown in solid dark orange). For time step 2, the passage context (shown in dotted light green), the embedding of the predicted output word at time step 1 (shown in dotted dark orange), and the previous hidden decoder state at time step 1 (shown in solid black) become the inputs for the decoding layer. Then the hidden output of the decoder layer (shown in solid black), the passage context (shown in dotted red), and the query context (shown in dotted red) become the inputs for the output layer. The output of the output layer is the predicted word at time step 2. For time step 2, in our diagram, the output of the output layer is the word "of" as shown in Figure 3.1.

3.4 Proposed Model: An Overview

Our model uses a sequence-to-sequence network to solve the problem of QFADS. The Encoder-Decoder Framework with Attention Mechanism (EDA) architecture with RNNs has become an effective and standard approach for Sequence-to-Sequence (Seq2Seq) prediction in general. We are also using this EDA architecture to solve the task. However, our EDA architecture is different as compared to the traditional EDA architecture. First of all, our model uses CNNs along with RNNs to predict the output sequence. Also, our model has multiple layers before the encoding layer and in between the encoding and the attention layer. Usually the traditional EDA architecture has layers in following order: input embedding layer, encoding layer, attention layer, output embedding layer, decoding layer, and finally output layer; however, our model has layers in following order: input embedding layer, concatenation layer, encoding layer, InceptionNet layer, self-attention layer, gated layer, attention layer, concatenation layer, output embedding layer, decoding layer, and finally output layer. Therefore, we add four new layers to the EDA framework; concatenation layer, InceptionNet layer, self-attention layer, and gated layer. Also, our model uses a different embedding layer. Usually the traditional EDA framework only has the word embedding layer which is forwarded to the encoding layer; however, our model's embedding layer has the character-level, word-level, and sentence-level embeddings which are concatenated using the concatenation layer and forwarded to the encoding layer. Multi-level embedding is done for both the query and the passage of the document. Also, most of the traditional seq2seq networks were used for the task of generic abstractive document summarization; however, we are using this framework for QFADS. The rest of this section shows the flow of the data of our modified sequence-to-sequence network:

- First, the words are embedded into vector spaces, one using a character level embedding layer and another using a word level embedding layer. Then, the sentences are embedded into another vector space using a sentence level embedding layer. Since our proposed model is query-focused summarization, this is done for both the query and the passage of the document.
- Then, three query embeddings are concatenated into a single query embedding using a concatenation operation, a linear layer and two highway layers. Similarly, three passage embeddings are concatenated into a single passage embedding. As a result, we will have one concatenated embedding for the passage of the document and have another concatenated embedding for the query.
- Once all the embeddings are concatenated, the query embedding is fed into the query encoder to produce hidden query encoder states. Similarly, the passage embedding is fed into the passage encoder to produce hidden passage encoder states.

- Then, the hidden passage encoder outputs are passed through the InceptionNet layer. we use InceptionNet layer because the hidden passage encoder outputs are too noisy and confused due to the inability of the LSTM to handle long lengths of passage text in a document. Local and global features are learned for the document and the corpus, respectively. Unlike the hidden passage encoder outputs, the hidden query encoder outputs are not passed through the InceptionNet layer as the query of the document is relatively short and an LSTM can handle short lengths of query text in a document. As a result, the hidden query encoder outputs are also not passed through self-attention layer and gated layer.

- Once all the features of the document and the corpus are learned through the InceptionNet layer, the hidden feature passage encoder outputs are passed to the self-attention layer to further strengthen the local and global information learning.

- Then, this hidden strengthened featured passage encoder outputs are passed through the gated layer which removes the unnecessary words based on the information of local and global features. The combined effect of the InceptionNet, self-attention and gated layers results in a comparatively shorter but important text which the LSTM can handle comfortably. One downside to this text shortening process may be that the syntax information of the sentence might be forfeited as several words in a sentence are eliminated. However, we hope that the feature extraction layer learns this syntax information as well since the feature extraction layer learns various local and global features.

- Then gated passage encoder outputs and the hidden query encoder outputs are calculated, they are passed to the attention layer in order to calculate the passage attention and the query attention. For the query attention, the hidden query encoder outputs and the previous hidden decoder state are used to calculate the query attention energy score. Then, this query attention energy score is used to calculate the query context.

For the passage attention, the gated passage encoder outputs, the query context, and the previous hidden decoder state are used to calculate passage attention energy score. Then this passage attention energy score is used to calculate the passage context. The query context and passage context may be different at each time step of the decoding process.

- Then, the final hidden states of both the query encoder and the passage encoder are passed through the final state concatenation layer to get the hidden decoder initial state. This hidden decoder initial state is used at time step 0 of the decoding process as an input to the decoding layer.
- Then, during inference, the previously predicted word is embedded into vector spaces using the output embedding layer. This output embedding layer only uses a word level embedding layer unlike the query and the passage. During the training, the target input is embedded into vector spaces.
- Then, the passage context is passed to the decoding layer in order to generate the current hidden decoder state. This decoding layer also takes the previous hidden decoder state and the embedded vector as input.
- Then, the current hidden decoder output is fed into the output layer along with the passage context and the query context to predict the next output word. This predicted output word is fed into the next time step's decoding layer as an input during inference and is compared against the target output during training.

3.5 Proposed Model: Layer Descriptions

In this section, we will describe each layer which are included in our proposed model for the QFADS task. The layers described are in accordance to the data flow mentioned in Section 3.4.

3.5.1 Input Embedding Layer

The input embedding layer is the first layer of the model. As told earlier in Section 2.6, the embedding layer is responsible for converting the words from human language to high-dimensional vector space which the machine/computer is able to understand. This layer is different than the embedding layer of a traditional EDA framework. The traditional EDA framework's embedding layer consists of single word level embedding layer whereas our embedding layer consists of three sub-layers: character level embedding layer, word level embedding layer, and sentence level embedding layer. The reason behind having these three sub-layers is to include the information on the overall text document structure in the proposed model. As a document text consists of characters, words, and sentences, we chose to embed the texts of the document on the character-level, word-level and sentence-level respectively. By having these layers, it reflects the true nature of the document text structure. Paragraph-level and document-level embedding can be performed as a part of future work as the document consists of paragraphs and complete document along with characters, words, and sentences. The remaining of the section describes each embedding sub-layer.

Character Level Embedding Layer

The Character level embedding layer is responsible for mapping each word to a high-dimensional vector space using a character-level CNN. Let $w1, \ldots, w_D$ and $q1, \ldots, q_Q$ represent the words in the passage and query of the document, respectively. Using the technique from Kim (2014), we obtain the character level embedding of these words (i.e., both the query words and the passage words) using an CNN. The characters in these words are embedded into vectors (of size 300 dimension for this proposed model), which can be considered as two dimensional (2D) inputs to the CNN, and whose size is the input channel size of the CNN. The outputs of the CNN are fixed-size vectors for each word. As a result, any length of word is converted into a fixed-sized vector representation. In our proposed model, the output size of the CNN is the same as the embedding size of the word embedding layer (i.e., 300). Advanced regularization techniques like dropout and batch normalization are used to improve the model performance. The character level embedding layer for the query and the passage can be formulated as follows:

CLQWE = Conv2D (query inputs character embeddings)

CLPWE = Conv2D (passage inputs character embeddings)

Here, $CLQWE$ and $CLPWE$ represent Character-Level Query Word Embeddings and Character-Level Passage Word Embeddings, respectively. The query inputs character embeddings are obtained via the lookup of character embedding matrix for each characters in query words. The passage inputs character embeddings are obtained via the lookup of character embedding matrix for each characters in passage words. *Conv2D* computes a 2-D convolution given 4-D input.

Word Level Embedding Layer

The Word level embedding layer also maps each word to a high-dimensional vector space. Word embeddings provide a dense representation of words and their relative meanings as compared to a one hot vector. They are an improvement over sparse representations like one hot encoding or the bag of words model. Word embeddings can be learned from text data and reused among projects. They can also be learned as part of fitting a neural network on text data. We use pre-trained word vectors called FastText Word Embedding (Bojanowski et al., 2017), to obtain the fixed word embedding of each word. The pre-trained word embedding used in this model can be further fine tuned as well; however, our word level word embeddings remained constant throughout the training and inference period. A Word level embedding layer was utilized for both the query and the passage of the document using the same pre-trained word embedding matrix. The word embedding of each word (both the query and the passage) is obtained via the lookup of that particular word on same word embedding matrix. The word level embedding layer for the query and the passage can be formulated as follows:

WLQWE = Lookup (query word inputs, matrix)

WLPWE = Lookup (passage word inputs, matrix)

Here, $WLQWE$ and $WLPWE$ represent Word-Level Query Word Embeddings and Word-Level Passage Word Embeddings, respectively. The query word inputs and the passage word inputs represents the input sequence of query words and passage words, respectively. $Lookup$ represents the lookup operation mentioned above, and $matrix$ represents the word embedding matrix mentioned above.

Sentence Level Embedding Layer

The Sentence level embedding layer also maps each sentence to a high-dimensional vector space. Sentence embeddings provide a dense representation of sentences and their relative meanings. We used Google's pre-trained Universal Sentence Encoder (USE) model (Cer et al., 2018) to obtain the fixed sentence embedding of each sentence. The embedding size of the sentence embedding is 512 dimension. The sentence embedding remained constant throughout the training and inference period. The sentence level embedding layer for the query and the passage can be formulated as follows:

SLQSE = USE (query sentence inputs)

SLPSE = USE (passage sentence inputs)

Here, *SLQSE* and *SLPSE* represent Sentence-Level Query Sentence Embeddings and Sentence-Level Passage Sentence Embeddings, respectively. The query sentence inputs are the input sequence of the query sentence and the passage sentence inputs are the input sequence of the passage sentence. *USE* represents the Google's pre-trained USE model which is used to obtain the sentence embedding for each sentence in the query and the passage.

3.5.2 Input Concatenation Layer

The input concatenation layer is a utility layer that concatenates three embeddings into a single embedding. As only a single embedding for each word is accepted as the input to the encoder, it is required to combine all three embeddings into a single embedding for each word for both the query and the passage of the document. The model automatically learns how much information should be taken from each embedding as all three embeddings are passed through the linear and highway layer and the weighted average of all three embeddings is passed to the encoding layer.

First, using a linear layer the sentence embeddings are converted into the 300-dimension vectors as both the character-level and the word-level word embeddings are of that size. Once all the embeddings have the same dimension, they are concatenated using another linear layer. Then, those concatenated embeddings are passed through a highway layer. We are using a highway layer as the part of concatenation process because the highway layer is superior to the basic linear layer. This whole process is done for both the query embeddings and the passage embeddings. The input concatenation layer for the query and the passage can be formulated as follows:

QE = ConcatLayer (CLQWE, WLQWE, SLQSE)

PE = ConcatLayer (CLPWE, WLPWE, SLPSE)

Here, *CLQWE*, *CLPWE*, *WLQWE*, *WLPWE*, *SLQSE* and *SLPSE* are defined in previous layers. *QE* and *PE* represent the final concatenated Query Embeddings and Passage Embeddings, respectively. *ConcatLayer* represents the concatenation operation along with the linear and highway layers.

3.5.3 Encoding Layer

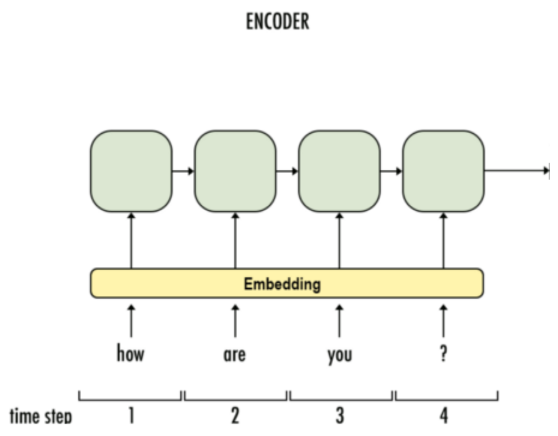

Figure 3.2: A general encoder

The encoding layer is the third layer of our proposed model. This layer is responsible for encoding the input sequence (both the query and the passage of the document) a via deep bidirectional LSTM. The encoding layer consists of two encoders: Query Encoder and Passage Encoder. The query encoder encodes the query representation whereas the passage encoder encodes the passage representation. Figure 3.2[13] illustrates a general encoder which shows the process of encoding words to sequential hidden encoder states.

Query Encoder

The query encoder in our proposed model is a deep bidirectional LSTM as an LSTM has the same performance as an GRU along with the advantage of customizability. Given a query $q = q_1, q_2, \ldots, q_k$ containing k words, the query encoder encodes the source query. The query encoder takes the concatenated query embeddings and the previous query encoder state as the input. The output of this query encoder is hidden encoder query state. This can be described mathematically as follows:

[13]Source: https://towardsdatascience.com/sequence-to-sequence-model-introduction-and-concepts-44d9b41cd42d

$$\mathbf{s}_i^q = \mathbf{DeepBiLSTM}(\mathbf{s}_{i-1}^q, \mathbf{e}(\mathbf{q}_i))$$

Here, s_i^q represents hidden encoder state at time step i. As mentioned before in Section 2.10.1, s_i^q contains h_i^q and c_i^q. $h^q = \{h_1^q, \ldots, h_k^q\}$ is the sequential hidden query outputs of the first k words from the input query sequence. $DeepBiLSTM$ is a deep bidirectional LSTM function discussed in Section 2.10.4. $e(q_i)$ represents the concatenated embedding of query word at position i. Once encoded, the sequential hidden query outputs are passed to the attention layer to calculate query attention.

Passage Encoder

The passage encoder in our proposed model is also a deep bidirectional LSTM. Given a passage $p = p_1, p_2, \ldots, p_l$ containing l words, the passage encoder encodes the source passage. The passage encoder takes the concatenated passage embeddings and the previous passage encoder states as the input. The output of this passage encoder is hidden encoder passage state. This can be described mathematically as follows:

$$\mathbf{s}_i^p = \mathbf{DeepBiLSTM}(\mathbf{s}_{i-1}^p, \mathbf{e}(\mathbf{p}_i))$$

Here, the notations are similar to query encoder except for letter q (for query) which is replaced by letter p (for passage). The hidden passage encoder outputs are then forwarded to the InceptionNet layer.

3.5.4 InceptionNet Layer

The InceptionNet layer is the fourth layer of our proposed model and the first layer to our novel selective mechanism. This layer is new to the traditional EDA framework. The InceptionNet layer is responsible for learning the local and global features of the document and the corpus. This layer is applied only to a passage of the document as the length of passage text can be quite long. During the introduction of this chapter, we discussed the problem related to long text. Our discussed solution was to selectively encode the long text. This idea of selectively encoding the text was introduced by Lin et al. (2018). Our model is inspired by their model; however, our proposed model is different than their model. First of all, their model was used for the task of generic abstractive document summarization; however, our model is used for the task of QFADS. Along with passage (document) encoder and passage (document) attention, our model also includes query encoder and query attention. Also, their model uses an Inception module to learn the local and global features whereas our model uses an Inception Network. An Inception Network chains together multiple Inception modules to form a single network. The size of each Inception Model is varied to create a filtering effect. Furthermore, their model uses an attention mechanism as a post-step to the decoding layer whereas our model uses an attention mechanism as a pre-step to the decoding layer. Also, inputs to the passage (document) attention mechanism are different as the query contexts are included in the passage (document) attention along with the addition of separate query attention to the model's attention mechanism.

Introduced by Szegedy et al. (2015), the main idea of the Inception module or network is to carefully design a model that would allow the model to increase the depth (the number of network levels) and width (the number of units on each level) of the network while keeping the computational complexity constant. One of the easiest ways to improve the performance of the neural network is to increase the depth and the width of the network. However, a uniform increase in the depth and the width of the network leads to a drastic increase in the computational cost of the network. For example, if two convolutional layers are chained, any uniform increase in the number of their filters results in a quadratic increase of the computation cost. Since our computation resources are finite, we need some way to keep the computational complexity constant despite the increase in the depth and width of the network. The Inception module solves this problem by introducing sparsity and replacing the fully connected layers by the sparse ones. This idea was based on the work by Arora et al. (2014) which states that the optimal network topology can be constructed layer after layer by analyzing the previous layer and putting together neurons with highly correlated outputs, given the probability distribution of the dataset can be represented by a large, very sparse deep neural network. This idea also fits nicely with well known Hebbian (Shaw, 1986) principle which states that the neurons that are fired together, are also wired together.

The Inception module is the basic building block for our InceptionNet layer. The InceptionNet layer consists of five Inception modules, five residual connections (one connection for each Inception module), one concatenation operation followed by a linear layer, and one batch normalization operation. The rest of the section describes in details about the Inception module and complete InceptionNet layer.

Inception Module

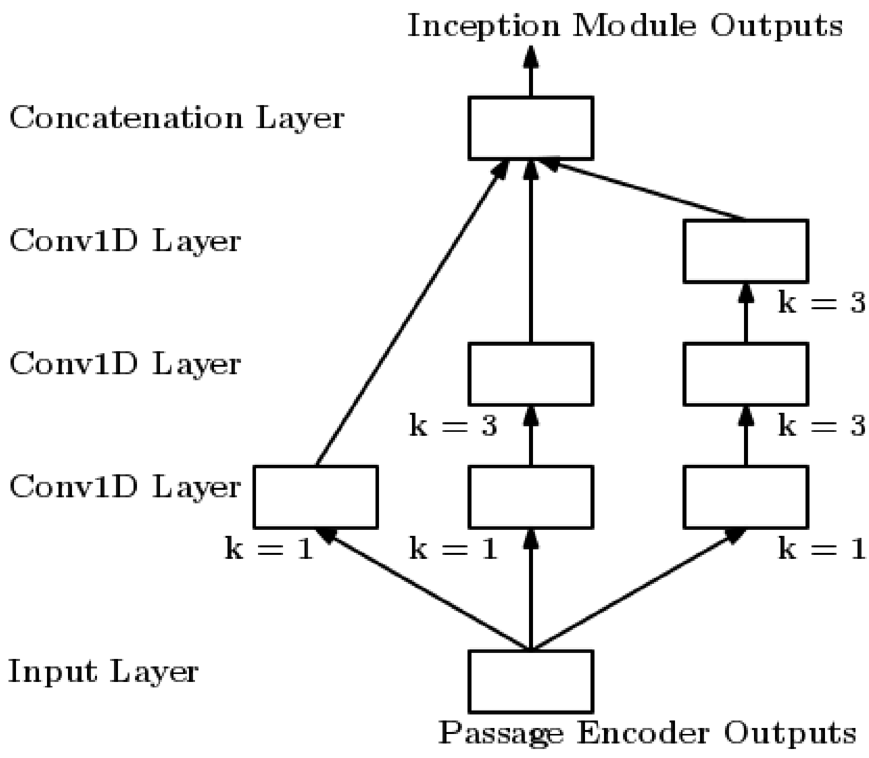

Figure 3.3: Modified Inception Module inspired by Szegedy et al. (2015)

The Inception module is the heart of the InceptionNet layer as it is the most important unit of the InceptionNet layer. As mentioned earlier, our Inception module is inspired by the Inception Model (Szegedy et al., 2015). Our Inception module consists of several 1-D convolution layers which convolves all the passage encoder outputs. Each convolution unit extract local features related to the input passage. Similar to image, the language also contains local correlations, such as the internal correlation of phrase structure. The convolutional units extract these common features in the passage and indicate the correlations among the input passage sequences. Also, the parameters sharing the convolutional kernels allow the model to extract certain types of global features, specifically n-gram features.

However, the key thing about this Inception module is not the use of the CNN architecture but the structure of this module. The structure of the Inception module can be seen in Figure 3.3. The idea of the original Inception model was to use all 1x1 convolution, 3x3 convolution, 5x5 convolution, and max pooling operation and let the model learn which operation to use instead of guessing which operation to use. However, the increase in the number of operations to the network lead the drastic increase in the computational cost of the network. To keep the computational cost of the network constant despite the increase in the number of operations, they introduced the concept of sparsity. Basically, except for the 1x1 convolution operation, they created a bottleneck layer by adding 1x1 convolution before 3x3 convolution and 5x5 convolution and by adding 1x1 convolution operation after the max pooling operation. This reduced the computational cost to one-tenth of its previous computational cost. This result inspired us to design our Inception module is a similar way. However, our structure is modified as compared to the original Inception model. Instead of using 5x5 convolution, we used two 3x3 convolutions to avoid a huge kernel size. Also, we have not used the max pooling operation at all. As shown in Figure 3.3, our Inception module can be formulated as below:

IMFO = Conv1D (Passage Encoder Outputs)

IMSO = Conv1D (Passage Encoder Outputs)

IMSO = Conv1D (IMSO)

IMTO = Conv1D (Passage Encoder Outputs)

IMTO = Conv1D (IMTO)

IMTO = Conv1D (IMTO)

IMO = Linear(Concat (IMFO, IMSO, IMTO))

Here, IMO represents the final Inception module outputs and $IMFO$, $IMSO$, $IMTO$ represent the convolved outputs for three different operations (as compared to original Inception module which consists a 1x1 convolution, a 3x3 convolution, a 5x5 convolution, and a max pooling). *Passage Encoder Outputs* represents the hidden outputs from the passage encoder, denoted as h^p in Section 3.5.3. They are the inputs for each initial 1x1 convolution operation. $Conv1D$ represents the 1-D convolution operation on 3-D inputs and the kernel size of each 1-D convolution are shown in Figure 3.3 represented as k. Also, each $Conv1D$ unit includes dropout and batch normalization operation along with convolution operation. *Concat* represents the concatenation of the convolved outputs for three different operations into one final inception module outputs. The concatenation is followed by a linear layer (represented as *Linear*) to match the output size.

InceptionNet Layer: Details

The InceptionNet layer chains together five Inception modules to form a single network which is used by the model to learn the local and global features of the document and the corpus. Even though a single Inception module would be sufficient to learn the features, we chained multiple modules together to give the increased depth to the network of our

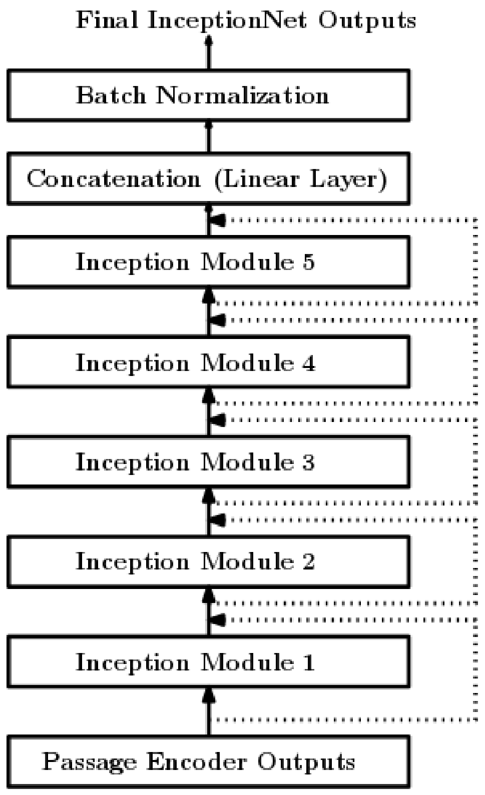

Figure 3.4: InceptionNet Layer

proposed model as one of the many ways to improve the performance of the model is by increasing the depth of the network. As a result, using a InceptionNet layer instead of only using an Inception module leads to better learning of the features. Also the width (i.e., the size of the Inception module) of each Inception module is varied. The regular size of the Inception module is same as the size of LSTM (i.e., 400). The size of Inception Module 2 is reduced to half and the size of Inception 4 is doubled. The variation in size of the Inception module is done to create a filtering effect to the features. Size variation is a popular technique in neural network to create filtering effect. As a result, along with extracting features, the InceptionNet layer filters the features. Furthermore, five residual connections are added (one for each Inception module) to the InceptionNet layer. Residual connections help the network go even deeper. They also help ease the training of the deeper neural network. As shown in Figure 3.4, the complete process of the InceptionNet layer can be formulated as below:

IMO1 = InceptionModule (Passage Encoder Outputs)

IMO1 = ResidualConnection (Passage Encoder Outputs, IMO1)

IMO2 = InceptionModule (IMO1)

IMO2 = ResidualConnection (IMO1, IMO2)

IMO3 = InceptionModule (IMO2)

IMO3 = ResidualConnection (IMO2, IMO3)

IMO4 = InceptionModule (IMO3)

IMO4 = ResidualConnection (IMO3, IMO4)

IMO5 = InceptionModule (IMO4)

IMO5 = ResidualConnection (IMO4, IMO5)

INO = Linear(Concat (IMO1, IMO2, IMO3, IMO4, IMO5))

INO = BatchNorm (INO)

Here, $IMO1$, $IMO2$, $IMO3$, $IMO4$, and $IMO5$ represents the outputs of each Inception module. INO represents the final output of the InceptionNet layer. $InceptionModule$ represents the steps that are taken in each Inception module which are described above subsection. $ResidualConnection$ represents the addition of residual connection for each Inception module. It is represented by a dotted line in Figure 3.4.

3.5.5 Self-Attention Layer

The self-attention layer is the fifth layer of our proposed model and the second layer to our novel selective mechanism. This layer is applied only to the passage of the document as the length of the passage text is quite long. This layer is also new to the traditional EDA framework. For the traditional EDA framework, the energy score of regular attention

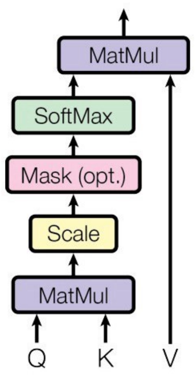

Figure 3.5: Scaled Dot Product Attention by Vaswani et al. (2017)

mechanism is calculated based on both the hidden encoder outputs and the previous hidden decoder state ; however, the energy score for the self-attention mechanism is calculated based only on a single hidden outputs or states.

Introduced by Vaswani et al. (2017), self-attention has less computational complexity per layer as compared to the recurrent and convolutional networks. Also, it reduces the number of sequential operations required. As a result, self-attention can be used to parallelize the computation which would lead to faster training of the model. Furthermore, self-attention is good at learning long range dependencies in a sequence transduction tasks. Also, based on the observations of the authors, self-attention learns the behavior related to the syntactic and semantic structure of the sentences. Hence, the use of self-attention in our proposed model is to strengthen the learning of features as the syntactic and semantic structure of the sentences are among the many features to be learned. As it can be used for improving the learning of long range dependencies, self-attention is also used for improving the encoding of the passage of the document. The authors call their self-attention as "Scaled Dot-Product Attention" and we will keep the same name. Scaled Dot-Product Attention can be formulated as below:

$$\textbf{Attention(Q,K,V)} = \textbf{softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{\mathbf{d}_k}}\right)\mathbf{V}$$

Here, Q and V both are the dense representation of the final output of the InceptionNet layer. $K = W_{att}V$ where W_{att} is a learnable matrix. Also, $softmax$ represents a softmax function which takes an un-normalized vector, and normalizes it into a probability distribution. K^T represents the tranpose of K. d_k represents the dimension of the InceptionNet outputs which is same as the size of RNN (i.e., 400). $Attention(Q,K,V)$ represents the self-attended outputs of the InceptionNet layer. Figure 3.5[14] shows the process of the Self-Attention layer.

3.5.6 Gated Layer

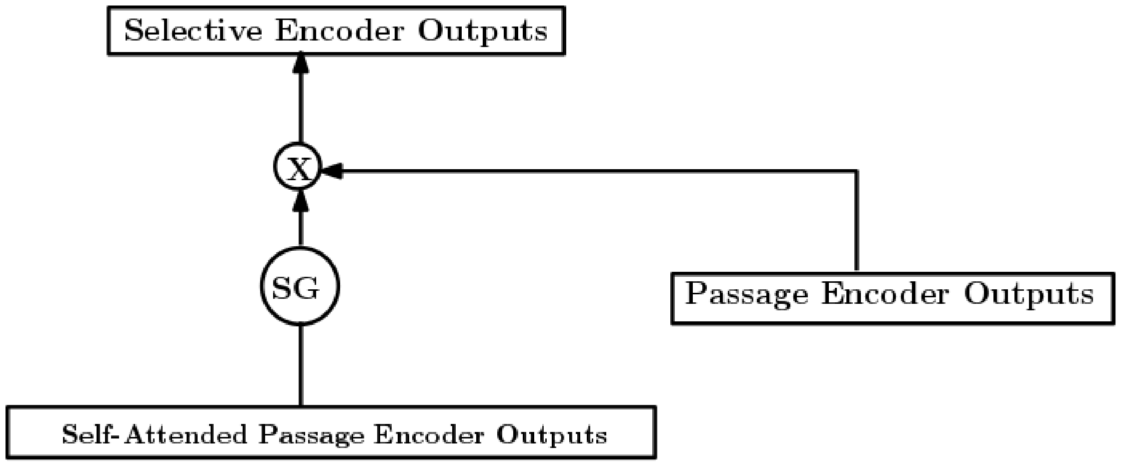
Selective Encoder Outputs
X
SG
Passage Encoder Outputs
Self-Attended Passage Encoder Outputs

Figure 3.6: Gated Layer For Self-Attended InceptionNet Outputs

The gated layer is the sixth layer of our proposed model and the third and final layer of our selective mechanism. This layer is applied only to the passage of the document as the length of the passage text is quite long. This layer is also new to the traditional EDA framework as most of the EDA frameworks do not use a selective mechanism approach. This Gated layer is responsible for filtering the outputs of the passage encoder in order to remove unnecessary and unimportant information and only select the information relevant to the local and global features of the document and the corpus. A gated mechanism is very popular in LSTMs and GRUs as well. This can be seen in Figure 3.6. The Gated layer for

[14]Source: https://medium.com/syncedreview/memory-attention-sequences-8522f531dd43

the passage of the document can be formulated as below:

$$\mathbf{SG = Sigmoid(SAINO)}$$

$$\mathbf{GEO} = \mathbf{SG} \odot \mathbf{PEO}$$

Here, *SAINO* represents Self-Attended InceptionNet Outputs which are the outputs from the previous self-attention layer. The outputs of previous self-attention layer are fed into this layer as input. *Sigmoid* represents the sigmoid activation function whose outputs ranges from [0, 1]. *SG* represents the selection gate just like the forget gate and input gate of an LSTM. $\odot$ is element-wise product of matrices also known as the Hadamard product. This is represented by X in Figure 3.6. *PEO* represents the outputs of the passage encoder. *GEO* represents the final passage encoder outputs after the gating mechanism removes unnecessary and unimportant information and only selects the information relevant to the local and global features. The gated passage encoder outputs are then fed into the attention layer as an input to generate the passage context.

3.5.7 Attention Layer

The attention layer is the seventh layer of our proposed model. This layer is responsible for aligning the input sequence for the purpose of decoding the output sequence. Our attention layer consists of two sub-layers: Query Attention and Passage Attention. The query attention aligns the query representations to the decoder whereas the passage attention aligns the passage representations to the decoder keeping in mind about the query representations. In a traditional EDA framework, the attention layer sits in between the encoder and the decoder. The attention mechanism takes the hidden encoder outputs and the hidden decoder state as input and gives out the context vectors to the decoder as output; whereas, our attention layer sits in between the gated layer and the decoding layer.

Proposed by Bahdanau et al. (2014), the attention mechanism is a recent trend for model improvement in the deep learning community. This process is loosely based on the human visual attention mechanism. Human eyes have the ability to focus on a certain region of an image with high intent while taking the surrounding image with low intent, and then adjusting the focal point over time. The idea of human visual attention applies to the summary generation as well. For example, when a reader is summarizing a document, the reader focuses on the words that are deemed important as a probable summary word and ignores the words that are deemed unimportant. This focus keeps on changing as the reader writes the complete summary of the document. Hence, the reader gives attention to different parts of the document at each step of the summary generation. As a result, the attention mechanism is suitable for our summarization task.

Without the attention mechanism, the encoder has the full burden to encode the whole document into a fixed vector. This fixed vector is then used at every time step to decode the summary. At every decoding step, the decoder focuses more on the end information as an RNN encoder has a problem with learning long range dependencies. Also, the focus is constant over all time steps because same vector is fed into the decoder at every time step. However, with the attention mechanism, the encoder does not have the full burden to encode the whole document into a fixed vector as this mechanism allows the decoder to attend to different parts of the source document (or query in case of query attention) at each step of the summary generation. The focus on different parts of the document at each decoder time step is learned with the help of this attention mechanism. The model learns what to attend to based on the input sequence and what it has produced so far. There are many variations to attention mechanism. In our work, we are using a scaled multiplicative attention mechanism. The concept of scaling an attention came from Vaswani et al. (2017), whereas the concept of multiplicative attention came from Luong et al. (2015).

Figure 3.7[15] shows the attention mechanism. Here, the y's are our summary words produced by the decoder, and the x's are our input words. Each decoder output word depends on a weighted combination of all the encoder outputs, not just the previous decoder state.

[15]Source: https://medium.com/syncedreview/memory-attention-sequences-8522f531dd43

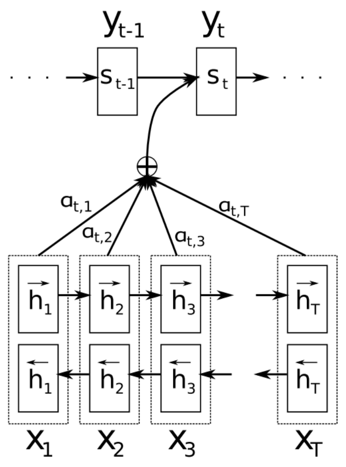

Figure 3.7: Example of Attention Mechanism

The a's are weights that define how much each encoder output should be considered for each summary output. For example, $a_{1,2}$ indicates what is the attention of decoder at time step 1 for input word x_2.

Query Attention

At each time step, the decoder produces an output word by focusing on different portions of the query. The query attention produces the query context vectors which are used for the passage attention and for the output layer. Passage context vectors calculated from the passage attention are used as an input to the decoding layer. As a result, query attention is used in the decoder layer in an indirect way. The query attention mechanism can be formulated as follows:

$$\mathbf{e}_{t,i}^{Q} = \mathbf{Score}(\mathbf{h}_t^{D}, \mathbf{h}_i^{QE})$$

$$\alpha_{ti}^{Q} = \mathbf{Scale}(\mathbf{Softmax}(\mathbf{e}_{t,i}^{Q}))$$

$$\mathbf{cv}_t^{Q} = \sum_{i=1}^{N} \alpha_{ti}^{Q} \mathbf{h}_i^{QE}$$

Here, h_i^{QE} represents the query encoder output at word position i. h_t^{D} represents the hidden decoder output at time step t. *Score* represents Luong's multipicative score function to calculate the energy of the attention. $e^{Q}_{t,i}$ represents the energy score at time step t for query sequence at position i. *Scale* represents the scaling operation. $\alpha^{Q}_{t,i}$ represents the scaled attention weights of query sequence at position i for the decoding time step t. cv^{Q}_{t} represents the query context vector at the decoding time step t. This context vector is then fed to the passage attention and to the output layer.

Passage Attention

At each time step, the decoder also produces an output word by focusing on different portions of the passage. The passage attention takes the query context vectors as input (along with hidden decoder state and hidden gated passage encoder outputs) and produces the passage context vectors and these vectors are used to decode the output word. The passage attention mechanism can be formulated as follows:

$$\mathbf{e}^{P}_{t,i} = \mathbf{Score}(\mathbf{h}^{D}_{t}, \mathbf{h}^{PE}_{i}, \mathbf{cv}^{Q}_{t})$$

$$\alpha^{P}_{ti} = \mathbf{Scale}(\mathbf{Softmax}(\mathbf{e}^{P}_{t,i}))$$

$$\mathbf{cv}^{P}_{t} = \sum_{i=1}^{N} \alpha^{P}_{ti}\mathbf{h}^{PE}_{i}$$

Here, the notations are almost identical to above query attention except for letter Q of query attention is replaced by letter P of passage attention. Also, the query context vector, cv^{Q}_{t} is added along with others to calculate the energy score for passage attention. The passage context vector is then fed to the decoding layer and to the output layer. For example, cv^{P}_{1} is fed into the decoding layer at time step 1.

3.5.8 Final State Concatenation Layer

The final state concatenation layer is the eighth layer of our proposed model. Just like input concatenation layer at Section 3.5.2, this layer is also a utility layer that combines the final hidden state of two encoders. This layer is responsible for combining the final hidden state of both the query and the passage encoder. The final hidden encoder state is the initial hidden decoder state for the traditional EDA framework. Since our task is query based summarization, we have two encoders in our proposed model. As a result, the final hidden state of both encoders need to be passed to the decoder as an initial hidden decoder state. Hence, the final hidden state of both encoders are combined in this final state concatenation layer followed by a linear layer. The combined final hidden state is the initial hidden decoder state for the decoding layer at time step 0. The final state concatenation layer can be formulated as below:

$$\mathbf{s}_0^D = \mathbf{Linear}(\mathbf{Concat}(\mathbf{s}_L^Q, \mathbf{s}_M^P))$$

Here, s_0^D represents the initial (at time step 0) hidden state of the decoder. s_L^Q represents the final hidden query encoder state assuming the query has L words. s_M^P represents the final hidden passage encoder state assuming the passage has M words.

3.5.9 Output Embedding Layer

The output embedding layer is the ninth layer of our proposed model. This layer is responsible for converting the summary words into vector space. Word level embedding layer is applied for the summary of the document using the same pre-trained word embedding matrix used for embedding the query and the passage. Character level and sentence level embeddings are not used in this layer. As a result, there is no concatenation layer for output embedding as there is only one word embedding for each word. The word embedding of each word in the summary is obtained via the lookup of that particular word on the word embedding matrix. The word level embedding layer for the summary can be formulated as follows:

WLSWE = Lookup (SWI, matrix)

Here, $WLSWE$ represents Word-Level Summary Word Embeddings. The SWI are the output sequence of summary words. During the training phase, these words are the target input words. During the inference phase, these are words predicted in the previous layer. $matrix$ represents the word embedding matrix mentioned above.

3.5.10 Decoding Layer

The decoding layer is the tenth layer of our proposed model. This layer is responsible for decoding the output sequence (i.e., the summary of the document). The decoder consists of a uni-directional LSTM with the same hidden-state size as that of the encoder LSTM. The decoding layer takes the concatenated final hidden state of both encoders and uses it as an initial state to its first recurrent layer. The inputs of the decoder are the previous hidden decoder state, the embedded target input word and the passage context vectors. The output of the decoding layer is the hidden decoder state. The hidden decoder output is passed to the output layer to predict the next output word. The decoding layer generates a hidden state one at a time until some maximum number of summary words are generated or a special end-of-sequence token is reached by the output layer. The decoding layer can be formulated as follows:

$$\mathbf{s}_t^D = \mathbf{LSTM}(\mathbf{s}_{t-1}^D, [\mathbf{e}(\mathbf{x}_t^D), \mathbf{cv}_t^P])$$

Here, cv_t^P represents the passage context vector at time step t. [ ] represents the concatenation of two vectors followed by a linear layer. The hidden output of this layer is fed to the output layer to predict the summary word. Along with the hidden decoder output, the query context and the passage context are also fed to the output layer as input. Figure 3.8[16] shows the process of decoding during the inference phase for a standard EDA framework. The blue line in the figure represents the passage context vector. During the decoding of the word "by", the figure shows that the passage vector is calculated by taking the weighted average of all hidden encoder outputs. In our model, the passage vector is calculated by

[16]Source: http://www.googblogs.com/author/open-source-programs-office/page/9/

taking the weighted average of gated encoder outputs.

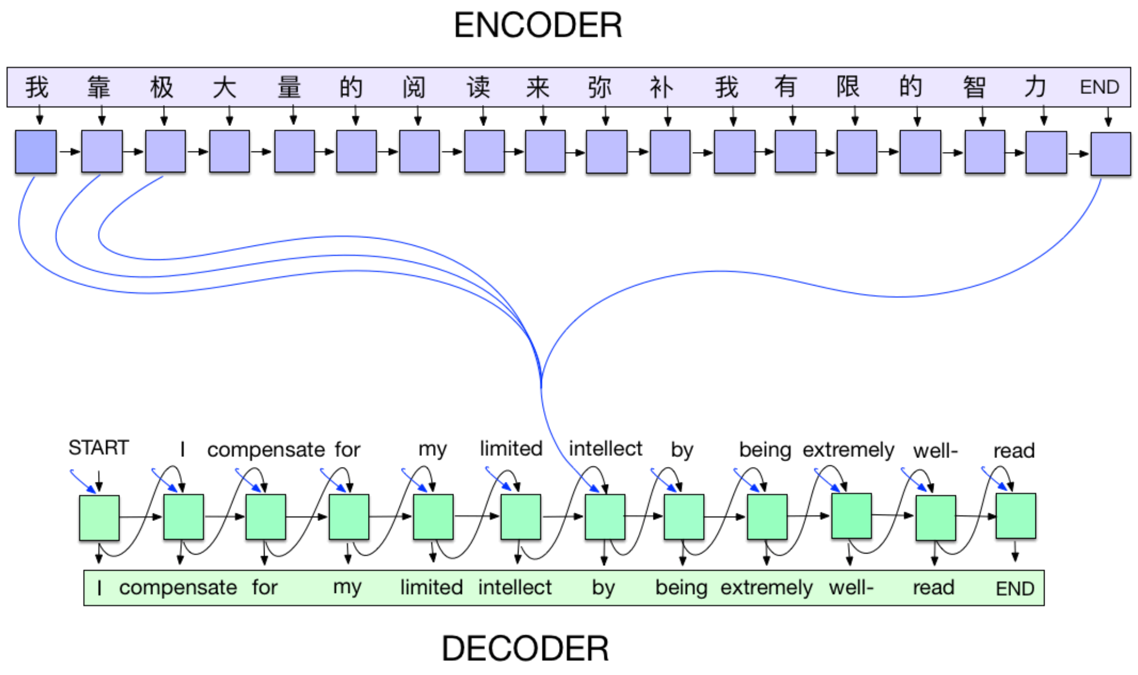
ENCODER
我 靠 极 大 量 的 阅 读 来 弥 补 我 有 限 的 智 力 END
START I compensate for my limited intellect by being extremely well- read
I compensate for my limited intellect by being extremely well- read END
DECODER

Figure 3.8: Decoder Source:

3.5.11 Output Layer

The output layer is the eleventh and the final layer of our proposed model. This layer is responsible for converting the hidden decoder output to an output word. This layer takes the hidden decoder output, the passage context and the query context as inputs and gives the predicted summary word as output. The output of the output layer is also the final output of the model. First, the output layer combines all the inputs into one vector and that vector is passed through two linear layers. Then, a softmax function is used to calculate the probability distribution of the word. Then an argmax function is used to determine the predicted word. This predicted word is compared with its corresponding target output as a part of a loss computation during training phase. The output layer can be formulated as below:

$$\mathbf{OV} = \mathbf{Projection}(\mathbf{Linear}(\mathbf{Concat}(\mathbf{h}_t^D, \mathbf{cv}_t^Q, \mathbf{cv}_t^P)))$$

$$\mathbf{OD} = \mathbf{softmax}(\mathbf{OV})$$

$$\mathbf{ID} = \mathbf{argmax}(\mathbf{OD})$$

$$\mathbf{y}_t = \mathbf{Id2Word}(\mathbf{ID})$$

Here, h_t^D, cv_t^Q, and cv_t^P represent the hidden decoder output, the query context, and the passage context, respectively at time step t. *OV* represents the output vector. $Projection$ represents converting the dense representation into the sparse representation matching the output size to the size of the vocabulary. *OD* represents the probability distribution of the output summary word. The *argmax* function takes the probability distribution and returns the index having highest probability. This index is the ID for the predicted word. This index is converted into word by using $Id2Word$ function which does a lookup on the model vocabulary. For the loss calculation at the training, the output distribution, *OD* is compared to the target output.

3.6 Training and Inference Details

Maximum log-likelihood estimation (MLE) is a common statistical technique to train a seq2seq model. Let us assume we have a parallel corpus D, where each sample in the corpus is a triplet (X^m, Q^o, Y^n) of passage, query, and summary of an individual document. Here m, n, o represent the length of passage, summary and query in a sample. Also, the passage and the query are the part of source sequence and the summary is the part of target sequence. Given any triplet from the corpus, the neural seq2seq model can compute the conditional log-probability of Y^n given X^m and Q^o: $\log P(Y^n|X^m, Q^o, \theta)$, where, θ is the training parameter. The training process can be can be formulated as below:

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

where

$$P(\mathbf{Y}|\mathbf{X};\mathbf{Q};\theta) = \prod_{t=1} \mathbf{P}(\mathbf{y}_t|\mathbf{y}_{1:t-1},\mathbf{X},\mathbf{Q})$$

Here, $L_t(\theta)$ is the loss function. The training process helps us find the global optimum of the loss function.

After setting up the neural seq2seq model, we need to train the model with enough training data. For this model to be trainable we need to decide what optimization algorithms to apply. We have used the Adam optimizer as our choice of optimizer. An optimizer is used to calculate the gradient descent of the loss. Since recurrent neural networks are known for vanishing/exploding gradients, a gradient clipping technique is used to address these issues. We have used sequence loss function to calculate the loss function. This loss function is just a weighted softmax cross entropy loss function. Cross entropy loss can be formulated as below:

$$\text{Cross Entropy Loss} = \sum_{i=1}^{n} \text{-log} p_i[y_i]$$

where, $p_i[y_i]$ represents the y_i-th entry of the probability vector p_i from the i-th decoding step and n represents the number of summary outputs.

As we used a 10 fold dataset to compare our result against the baseline model, we trained 10 models based on the respective 10 folds. The hyperparameters for all 10 models are kept same to test the robustness of model on different data combinations. We used a 300-dimensional FastText embeddings trained on the Debatepedia dataset as our embeddings initialization and the embeddings remained constant throughout the training and the inference phase. The same embedding matrix was used for the query, passage, and summary of the document. We used 512-dimensional USE embeddings for our sentence embedding. We limited the vocabulary size to 50k and we used the bucketing mechanism to speed up the computation. We fixed the model hidden size for both the encoders and the decoder at 4000. We used the batch size of 50 at the training time and did not shuffle the training data at every epoch. We used the Adam optimizer to train our model with an initial learning rate of 0.001. We employed gradient clipping, dropout, and batch normalization to regularize our model. Gradient clipping is adopted with the max global norm of 5. Dropout probabilities are separate for each encoder. For the inference phase, we used the beam search algorithm to predict the output sequence with a beam size of 12. We trained our models on a Nvidia TITAN X GPU card with 12G RAM and each model took on average about 24 hours to train. In total, the training took about 10 days. On average, each training phase consisted of about 700 epochs. Generating summaries at test time is reasonable (given our GPU type) with the throughout of about 2 summaries per second on a single GPU, using a batch size of 1. Our neural network based framework is implemented using Tensorflow (v 1.10) and Python (v 3.5).

3.7 Dataset

As there is no standard existing dataset for query-based abstractive summarization, we used the dataset created by Nema et al. (2017) from Debatepedia. Debatepedia is an encyclopedia of pro and con arguments and quotes on critical debate topics. There are 663 debates in the corpus. The creators of the dataset have considered only those debates which have at least one query with one document. These 663 debates belong to 53 overlapping categories such as politics, law, and crime. A given topic can belong to more than one category. The average number of queries per debate is 5 and the average number of documents per query is 4. The summary is an abstractive summary. The creators crawled 12695 such query, passage, summary triples from Debatepedia. They used 10 fold cross validation for all their experiments. Each fold uses 80% for training, 10% for validation, and 10% for testing. Each fold is used to train a separate model. We have used the exact 10 fold dataset so that we could evaluate our model against their model. Basically, we used the script provided[17] by the creators to download our required 10 fold data for the task. Table 3.1 reports the average length of the query, summary, and passage in this dataset. Before feeding

[17] https://github.com/PrekshaNema25/DiverstiyBasedAttentionMechanism

the model with data for the training, basic pre-processing was done to remove words and symbols like <eos>, <s>, #, [,], “, ″, and so on.

Table 3.1: Average length of passages, queries, and summaries in the DBPedia Dataset

Average number of words		
Passage	Query	Summary
66.4	9.97	11.75

3.8 Evaluation

We use different metrics to evaluate the performance of our proposed model. These metrics are ROUGE(1, 2, L), METEOR, EACS, GMS, and CR.

Metric for Evaluation of Translation with Explicit ORdering (METEOR) (Banerjee and Lavie, 2005) uses a combination of both precision and recall in the METEOR metric. Furthermore, the alignment is based on exact token matching, followed by WordNet synonyms, stemmed tokens and look-up table paraphrases.

Embedding Average Cosine Similarity (EACS) averages the cosine similarities of the embeddings. Higher EACS means improved abstractiveness in generated summaries.

Greedy Matching Score (GMS) indicates information coverage. Higher GMS means the generated summaries contain diversified contents.

We define Copy Rate (CR) as how many tokens are copied to the abstract sentence (or target sentence) from the source sentence without modifying the tokens. A lower Copy Rate score means more abstraction is involved in the generated abstract sentence. A Copy Rate of 100% means no abstraction is involved in the process.

3.8.1 Baseline Models

We compare our model with the current state-of-art model on query based abstractive document summarization model called “Diversity driven attention model for query-based abstractive summarization” (Nema et al., 2017). For simplicity, let us call it **Diversity** model. Besides the tradition EDA framework, their model augments two key additions (i) a query attention model which learns to focus on different portions of the query at different time instead of using a static representation for the query (ii) a new diversity based attention model which aims to alleviate the problem of repeating phrases in the summary. They have compared various diversity models and have shown that soft LSTM based diversity attention model gives the best performance. We will compare our model against their best performing Diversity model. Just like their model, we will also report average scores across the 10 folds.

Also, we also compare our results against the standard EDA framework to show the readers how far we have come for the task of query based abstractive summarization. This model does not contain the encoder and attention module for the query.

3.8.2 Results

Table 3.2: Performance of various model using ROUGE(1, 2, L), METEOR, EACS, GMS, and CR metrics.

Model	R-1	R-2	R-L	METEOR	EACS	GMS	CR
Standard EDA	13.73	2.06	12.84	-	-	-	-
Diversity	41.26	18.75	40.43	-	-	-	-
Our Model	**43.22**	**27.40**	**42.73**	**25.72**	**85.49**	**72.59**	**40.47**

Table 3.2 summarizes the result of our experiment. We have compared various model using full-length ROUGE metrics. F-measure scores are reported. All standard or default options are chosen to evaluate the model. We have reported the average scores of ROUGE-1, ROUGE-2, ROUGE-L across the 10 folds. This result clearly shows that our proposed model outperforms the current state-of-art (Diversity) model for the task of query based abstractive summarization.

Also, we have reported our model's scores using METEOR, EACS, GMS, and CR metrics. The script by Maluuba [18] was used to obtain the evaluation scores. These scores

[18]https://github.com/Maluuba/nlg-eval

are also reported as the part of results because these scores are more popular evaluation metrics than ROUGE scores. It will be easier in future for upcoming work to compare their results against ours.

3.9 Summary

In this chapter, we proposed our query-focused abstractive document summarization model using neural network. We described our model of novel selective mechanism for the task in full detail. Then, we presented our training and testing details and described the dataset that was used to train the model. Finally, we evaluated our proposed model against the current state-of-art model. In the next chapter, we will apply document level query-focused abstractive summarization to multi-document level query-focused abstractive summarization.

Chapter 4

Query Focused Abstractive Multi-Document Summarization Using Neural Network

4.1 Introduction

query-focused abstractive multi-document summarization (QFAMDS) is the process of summarizing the main points from multiple documents which are focused on the context of the query into a condensed summary. Similar to QFADS, sequence-to-sequence learning can be applied to query-focused neural abstractive multi-document summarization, whose model consists of two encoders (i.e., one each for the query and the documents), two attention models (query attention and documents attention), and a decoder.

Some of the current problems in this field are limited research and the shortcomings to the attention mechanism. These problems are mentioned in detail in Section 3.1.

To solve the above problems, we propose three models for QFAMDS. These models differ from our previous QFADS model. The input sequence of a QFAMDS model contains multiple documents and a query whereas the input sequence of QFADS model contains single document and a query. A simple way to solve QFAMDS task is to view it as a QFADS task, and treat all documents as the input to seq2seq models. Unfortunately, the documents are too long to be the input to this simple seq2seq model. We used a selective mechanism in our single document model because a single document was too long. Now, imagine how long the input sequence of combined multiple documents would be? As a result, using the previously implemented selective mechanism is not enough for this multi-document model. Also, a significantly long input sequence will make the model too hard to train. Training on multiple documents will mostly confuse the model and produce poor results. Inspired by Tan et al. (2017), we propose a coarse-to-fine framework to tackle the challenge of QFAMDS. Our approach first extracts ordered top-k important sentences from multiple documents. Then, these extracted sentences are passed as an input to our previously implemented QFADS model to further abstract these sentences to a multi-doc summary. The extraction methods for all proposed methods are different during the training phase and are same during the inference phase. During the training phase, the top-k sentences are extracted based on the similarity comparison of all sentences of the multiple documents to the query for the first QFAMDS model and to the summary for the second QFAMDS model and to the combined query and summary of the document for the third QFAMDS model. During the inference phase, the top-k sentences are extracted based on the similarity comparison of all sentences of the multiple documents to the query for all proposed models. We conducted experiments on the MS-MARCO (Nguyen et al., 2016) dataset, a recently developed dataset by Microsoft for reading comprehension, and have reported the various scores as our evaluation metrics.

4.2 Task Description

In this section, we formalize our QFAMDS task. Given a query q = q_1, q_2, …, q_k containing k words, a list of document d = d_1, d_2, …, d_o containing o documents, a document (passage) s = s_1, s_2, …, s_n containing n sentences, and a sentence w = w_1, w_2, …, w_l containing l words, the task of QFAMDS is to generate a context based summary y = y_1, y_2, …, y_m containing m words. This task can be achieved by finding the probability $y*$ such that it maximizes the probability $p(y|q,d)$. Using Bayes theorem, it can be further decomposed into following:

$$\mathbf{y}^* = \mathbf{argmax}_y \prod_{t=1}^{m} \mathbf{p}(\mathbf{y}_t|\mathbf{y}_1,\dots,\mathbf{y}_{t-1},\mathbf{q},\mathbf{d})$$

The above equation for $y*$ can be modeled using the neural network framework.

4.3 Proposed Models: A Complete General Diagram

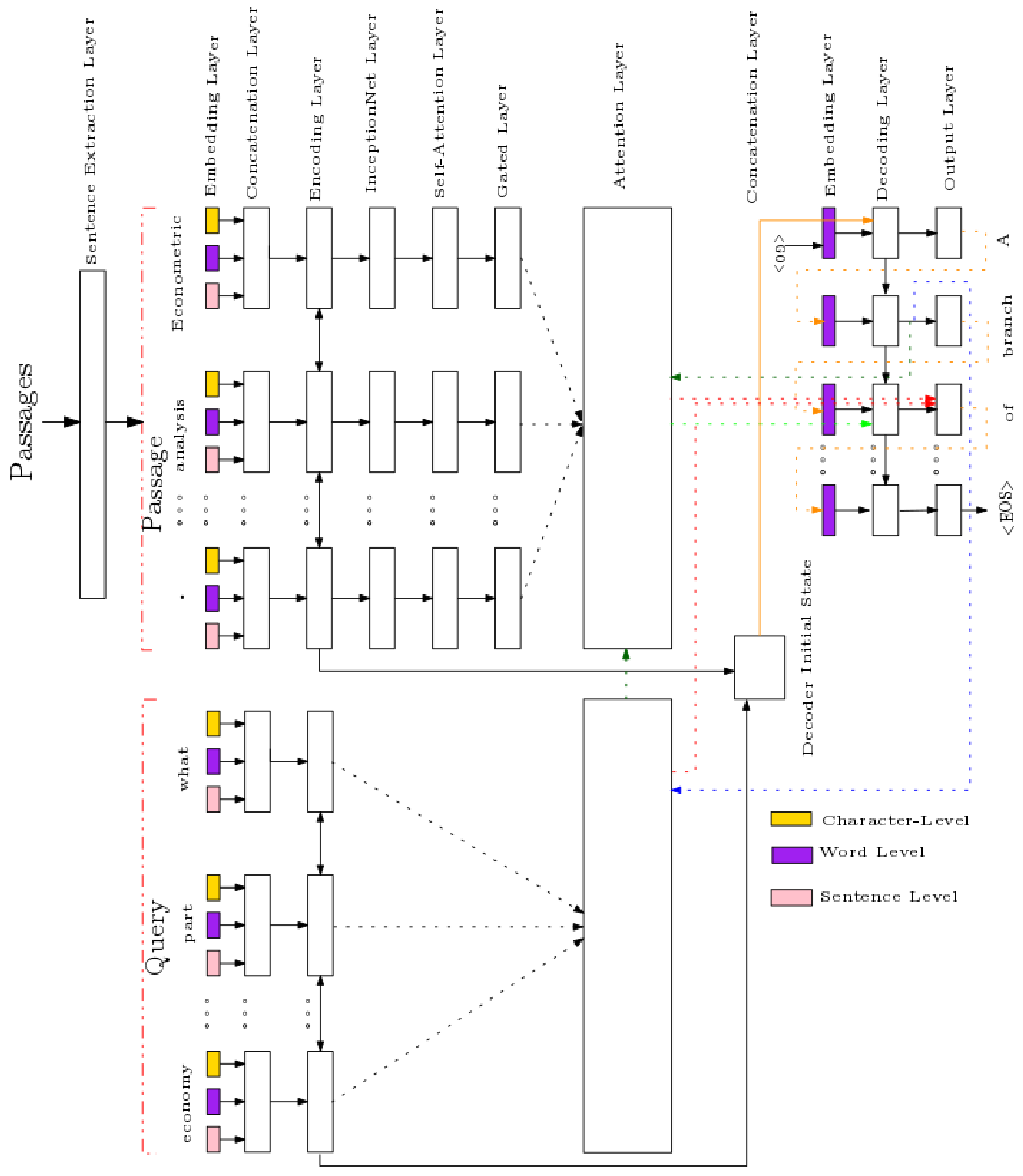

Figure 4.1: A general diagram to all our three proposed models for Query Focused Abstractive Multi-Document Summarization.

Figure 4.1 shows the general diagram for all three of our proposed models for the

QFAMDS task. Passages and the query are the input sequence to the QFAMDS models. The multi-document summary is the output sequence to the QFAMDS models. Here, passages are passed to these models along with the query. The reduction of size in passages is performed at the sentence extraction layer where each sentence from all documents are scored, ranked, ordered, and selected. The Sentence Extraction layer is different for all three proposed models. Details about each sentence extraction layer will be explained under their respective sections. The rest of the diagram is very similar to the QFADS model from the previous chapter. The description about the QFADS model diagram is mentioned in Section 3.3.

4.4 Proposed Models: An Overview

Similar to our previous QFADS model, our models use sequence to sequence network having an EDA architecture to solve the problem of QFAMDS. The differences between our proposed QFAMDS models and the traditional EDA framework are mentioned in Section 3.4. Also, these QFAMDS models differ from our previous QFADS model. The input sequence of QFAMDS models contain multiple documents whereas the input sequence of previously implemented QFADS model contains a single document. The rest of this section shows the flow of the data of our proposed models for QFAMDS task:

- First, 10 passages are combined together to form a big, single document. Then, the combined document is tokenized into several sentences. Then these tokenized sentences are compared to the query and/or summary (depending on the model and its phase) to extract important sentences. These extracted sentences are combined together in their original order and send to a seq2seq model as an input for training. During inference, the comparison is made to the query.

- Then, the rest of the data flow is similar to the data flow mentioned in Section 3.4.

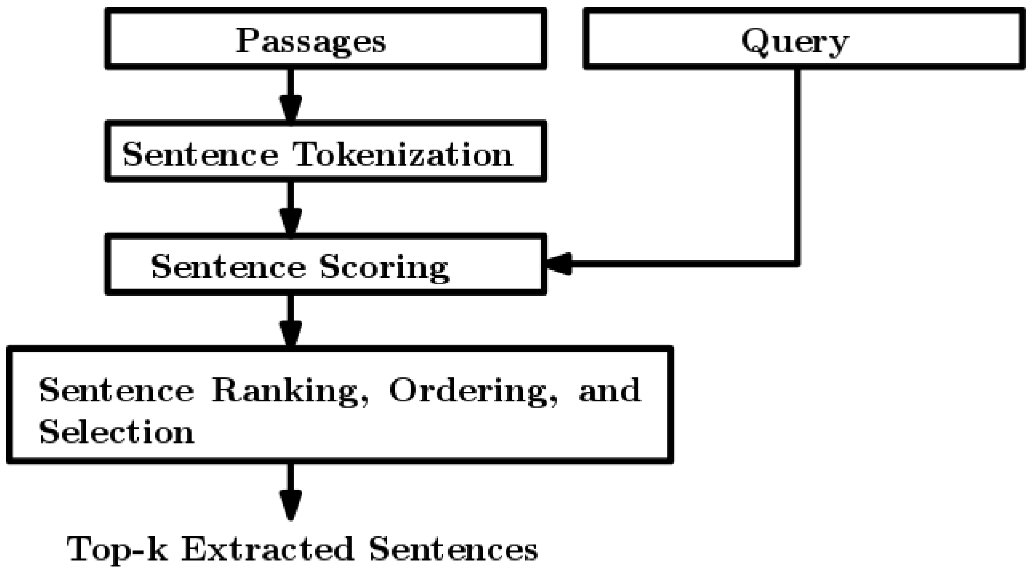
Passages
Query
Sentence Tokenization
Sentence Scoring
Sentence Ranking, Ordering, and Selection
Top-k Extracted Sentences

Figure 4.2: Sentence Extractor for Sent2Query.

4.5 First Proposed Model: Sent2Query

In this section, we propose our first model for the QFAMDS task. This model will be referred to as Sent2Query as the top-k sentences are extracted based on the similarity comparison of the sentences to the query. The key idea to all our proposed models is take coarse-to-fine approach and reduce the amount of text from multiple documents. But there exist multiple viable solutions for this coarse-to-fine approach. In the Sent2Query model, we filter out the unnecessary sentences by only selecting those sentences which are most similar to the query and treat those selected sentences as a part of input sequence to the seq2seq model instead of all documents related to the sample data.

4.5.1 Sentence Extractor

The sentence extractor of this model is responsible for extracting the most similar sentences to the query from the multiple documents while preserving the ordering of the sentences. These top-k extracted sentences are usually the most important sentences of all the documents. Shortening the input sequence by just keeping the most important sentences will make the model easier to train and give better results. The sentence extractor can be visualized in Figure 4.2. The sentence extractor contains the following procedures.

Sentence Tokenization

A sample data for this task is a triplet containing a query, ten passages associated with the query, and a summary summarizing 10 passages based on the query. Once 10 documents (passages) are loaded from the dataset for the query, they are combined to form a single document. Then, the combined document is broken down into sentence tokens using the sentence tokenization operation by the Natural Language Toolkit (NLTK). Sentence tokenization is the process of splitting a paragraph or document into a list of sentences. In general, tokenization is the process of splitting a stream of text into a list of pieces or tokens. In case of sentence tokenization, the piece or token is a sentence. Once tokenized, these tokenized list of sentences are fed to the sentence similarity model to calculate the sentence similarity score.

Sentence Scoring

The sentence similarity model is used to calculate the similarity of a given sentence with respective to its reference sentence. We are using Gensim's KeyedVectors module as our sentence similarity model. Introduced by Řehůřek and Sojka (2010), Gensim is a robust open-source vector space modeling and topic modeling toolkit implemented in Python. Before using this module, our choice of word2vec embedding needs to be loaded into this KeyedVectors model. We used pre-trained FastText word vectors. Once the embeddings are loaded, then KeyedVectors model can be used to perform similarity lookup on word, sentence, paragraph, and even whole document. The KeyedVectors model is used instead of full word2vec model for faster and easier training of our proposed model.

The list of tokenized sentences along with their corresponding reference sentence is passed to the loaded KeyedVectors model to calculate the sentence similarity score. In the case of the Sent2Query model, the query of the sample data is considered as the reference sentence. Therefore, every sentence is compared to its corresponding query of the sample data to find the sentence similarity between the sentences and the query. There are various ways to calculate the sentence similarity between two sentences. We are using the Word Mover's Distance (WMD) for this purpose. Introduced by Kusner et al. (2015), the WMD is a distance function that measures the distance between two texts as the cumulative sum of the minimum distance each word in one text must move in vector space to the closest word in the other text. Here, we are comparing the similarity of the sentences to the query with the assumption that the sentences which are the most similar to the query are the most important sentences from multiple documents, which will be fed into the seq2seq model. The sentence scoring function for Sent2Query model can be formulated as below:

LSDS = WMD (LTS, Query)

Here, *LTS* refers to the list of tokenized sentences. *Query* refers to the query corresponding to the list of tokenized sentences of the sample data. *WMD* refers to the WMD function mentioned above. *LSDS* represents the list of sentence distance scores for each sentence from the list of tokenized sentences. For example, *LSDS[1]* is the sentence distance score between the sentence *LTS[1]* and the *Query*. In this example, [1] refers to the index 1 of the list. A lower distance score means that the two sentences are considered more similar. Once the distance scores between each sentence and the query are calculated, these scores are forwarded for ranking based on their distance scores, for ordering based on their occurrences in their documents, and for selection based on one of the hyperparameters (the number of sentences to be selected).

Sentence Ranking, Ordering, and Selection

Once the distance scores are calculated, we need to find the most similar sentences to the query and use them as an input to the seq2seq model. However, there are few things to keep in mind while finding the most similar sentences. First, we need to know how many sentences are we selecting. This is one of the hyperparameters of our proposed model. This is important to the performance of the model as choosing too few sentences might lead to missing important information and choosing too many sentences might lead to a confused model. Then, we need to make sure that there is an order to the sentences in a logical manner. The original occurrence order to the sentences must be preserved. This is done to make sure that the flow of all the documents is not broken because having a different order to the sentences could mean something completely different or unreadable. This is also very important while generating a comparatively sensible summary. The ranking function gives a rank to each sentence based on their sentence distance scores. This function gives the top ranks to the least sentence distance scores meaning the sentences having the least distance score to the query are ranked among the most similar sentences to the query. An ordering function ensures that the original occurrence order of the sentences are preserved. Finally, a selection function selects the top ranked sentences with their original order based on the hyperparameter mentioned above. Once the top ranked and ordered sentences are selected, they are combined to form a single document. From the next section, we will call the combined sentences as document or passage. Along with the query, these shortened but important combined sentences (passage or document) are treated as an input sequence to the query based summarization model.

4.5.2 Encoder

The encoder of this model is responsible for encoding the input sequence (the query and the passage) using an LSTM into a fixed dense representation. Before passing the query and the combined sentences (let us call it a passage) to the encoder, they are converted into vector space using the character-level, word-level, and sentence-level embeddings mentioned in Section 3.5.1 and concatenated into a single word embedding as mentioned in Section 3.5.2. Then the concatenated query word embeddings and the concatenated passage word embeddings are fed to two separate encoders. Both encoders are deep bi-directional LSTMs. The query embedding fed to the query encoder produces hidden query encoder state. Similarly, the passage embedding fed to the passage encoder produces hidden passage encoder state. The exact formula governing the query encoder and the passage encoder are both mentioned in Section 3.5.3.

4.5.3 Feature Selector

The feature selector of this model is responsible for learning strong global and local features of the document as well as the corpus and selecting the most important information among the passage while considering the relevant features. Despite having the coarse-to-fine approach to reduce the number of inputs, the outputs to the passage encoder still contains significant noise as each sentence still contains unnecessary or redundant information. As a result, we add the previously implemented selective mechanism to help reduce more redundancy. The feature selector goes through the InceptionNet layer, Self-Attention Layer, and Gated Layer. These layers are mentioned in Section 3.5.4, Section 3.5.5, and Section 3.5.6, respectively. The addition of these layers results in a comparatively shorter but important passage text which the LSTM can handle comfortably. Once all the gated passage encoder outputs and the hidden query encoder outputs are calculated, they are passed to the attention layer in order to calculate the passage attention and the query attention.

4.5.4 Attender

The attender of this model is responsible for aligning the input sequence for the purpose of decoding the output sequence. Our attender consists of two parts: Query Attention and Passage Attention. The query attention aligns the query representations to the decoder whereas the passage attention aligns the passage representations to the decoder keeping in mind the query representations. The key difference between the query attention and the passage attention is how both calculate the energy score. For the query attention, the hidden query encoder outputs and the previous hidden decoder state are used to calculate the query attention energy score. Then, this query attention energy score is used to calculate the query context. For the passage attention, the hidden gated featured passage encoder outputs, the query context, and the previous hidden decoder state are used to calculate the passage attention energy score. Then this passage attention energy score is used to calculate the passage context. The formula governing the query attention and the passage attention are both mentioned in Section 3.5.7.

4.5.5 Decoder

The decoder of this model is responsible for decoding the output sequence (the multi-document summary). The decoder consists of a uni-directional LSTM with the same hidden-state size as that of the encoder LSTM. The decoder takes the concatenated final hidden state of both encoders and uses it as decoder's initial state to its first recurrent layer. For all remaining recurrent layers, the inputs of the decoder are the previous hidden decoder state, the embedded target input word (or previously predicted word) and the passage context vectors. At each decoding step, based on its input, the decoder LSTM generates the hidden state. The hidden output along with the query context and the passage context at that time step are passed through linear layer, projected to sparse representation of vocabulary size, normalized, and used to predict the probable summary word. The decoder can be formulated as below:

$$\mathbf{s}_t^D = \mathbf{LSTM}(\mathbf{h}_{t-1}^D, [\mathbf{e}(\mathbf{x}_t^D), \mathbf{cv}_t^P])$$

$$\mathbf{y}_t = \mathbf{Output}(\mathbf{h}_t^D, \mathbf{cv}_t^P, \mathbf{cv}_t^Q)$$

Here, s_{t-1}^D represents the previous hidden decoder state. It contains hidden output (h_{t-1}^D) and hidden cell memory (c_{t-1}^D). $e(x_t^D)$ represents the embedding of summary word at position t. cv_t^P represents the passage context vector at time step t. [ ] represents the concatenation of two followed by a linear layer. During the training period, the embedded target input will be the input to the decoder whereas during the inference period the embedding of the previously predicted word will be the input to the decoder. cv_t^Q represents the query context vector at time step t. *Output* represents the steps performed at the output layer in Section 3.5.11. Finally, y_t represents the predicted word at decoding step t. During training, the probability distribution of the predicted word is used to calculate the loss function and whereas during the inference, the predicted word becomes the input to the decoder at next decoding step and is also a part of the multi-document summary.

4.6 Second Proposed Model: Sent2Summary

Sent2Summary is our second proposed model for the QFAMDS task. In this model, important sentences are extracted based on the comparison of the sentences to its corresponding summary. Here, sentences having higher distance measures to the summary are discarded as these sentences are considered least similar to the summary. Once the unnecessary sentences are filtered out, the length of the passage is shortened. These extracted sentences having a reduced length are then fed into query-focused seq2seq model as a part of input sequence.

4.6.1 Sentence Extractor

This sentence extractor is responsible to extracting the most similar sentences to the summary from the multiple documents while preserving the ordering of the sentences. The visualization of the sentence extractor is very similar to Figure 4.2 except the query in the figure is replaced by the summary. The sentence extractor contains following procedures.

Sentence Tokenization

The sentence tokenization of the Sent2Summary model is the same to the sentence tokenization of Sent2Query model discussed in Section 4.5.1. Once tokenized, these tokenized list of sentences are fed to the sentence similarity model to calculate the sentence similarity score.

Sentence Scoring

A similarity comparison of all sentences is done to its corresponding summary of the sample data to find out the sentence similarity score. We use the same Word Mover's Distance (WMD) to calculate the sentence similarity between two sentences. The sentence scoring function for Sent2Summary model can be formulated as below:

LSDS = WMD (LTS, Summary)

Here, *Summary* refers to the summary corresponding to the list of tokenized sentences of the sample data. The representation of LST, WMD, and *LSDS* are already mentioned before.

Sentence Ranking, Ordering, and Selection

The sentence ranking, ordering and selection function of the Sent2Summary model is the same as the sentence ranking, ordering and selection function of Sent2Query discussed in Section 4.5.1. Once top ranked and ordered sentences are selected, they are combined to form a single document. Along with the query, these shortened but important combined sentences (passage or document) are treated as an input sequence to the query based summarization model.

4.6.2 Encoder, Feature Selector, Attender, Decoder

The encoders, feature selector, attenders, and decoder of the Sent2Summary model are the same to the encoders, feature selector, attenders, and decoder of the Sent2Query model discussed in Section 4.5.2, Section 4.5.3, Section 4.5.4, Section 4.5.5, respectively.

4.7 Third Proposed Model: Sent2CQS

Sent2CQS is our third proposed model for the QFAMDS task. This model is almost similar to other proposed models except for its sentence extractor component. The extraction of salient sentences is based on the similarity comparison of the sentences to its corresponding combined query and summary of the sample data.

4.7.1 Sentence Extractor

The sentence extractor is responsible for extracting sentences which are the most similar to the combined query and summary of the sample data from the multiple documents while preserving the ordering of the sentences. Very similar to Figure 4.2, this model's sentence extractor can be visualized by replacing the query with combined query and summary from the figure. The sentence extractor contains following procedures.

Sentence Tokenization

The sentence tokenization process is same as previous two models.

Sentence Scoring

Sentence scoring is performed by calculating the distance similarity score of all sentences to its corresponding combined query and summary. The same WMD function is used to calculate the distance similarity score. The sentence scoring function for the Sent2CQS model can be formulated as below:

$$\mathbf{LSDS = WMD\ (LTS,\ CQS)}$$

Here, *CQS* refers to the combined query and summary corresponding to the list of tokenized sentences of the sample data.

Sentence Ranking, Ordering, and Selection

In this model, the idea behind the sentence ranking, ordering, and selection functions is same and hence they are also same to previous two models.

4.7.2 Encoder, Feature Selector, Attender, Decoder

Once separate input are generated based on previous sentence extraction layer, the idea now is to pass them to a seq2seq model for training and then compare the results to other models. As a result, all three models have same seq2seq components like encoder, feature selector, attender, and decoder.

4.8 Training and Inference Details

Maximum log-likelihood estimation (MLE) is a common statistical technique to train a seq2seq model. More about the MLE technique is mentioned in Section 3.6.

We used the Adam optimizer as our choice of optimizer. We have used sequence loss function to calculate the loss function. The gradient clipping technique is used to alleviate the problem of vanishing/exploding gradient.

For each proposed model, the hyperparameters are the same. This was done to check the effectiveness of each model. The hyperparameters and choice of these models are mostly same as the previously implemented summarization model for the single document setting. We fixed the model hidden size for both the encoders and the decoder at 400. We used the batch size of 32 at the training time. We trained our model on a Nvidia TITAN X GPU card with 12G RAM. The Sent2Query model took about 5 days and 5 hours to train. Its training phase consisted of about 1900 epochs. The Sent2Summary model took about 3 days 13 hours to train. Its training phase consisted of about 1300 epochs. The Sent2CQS model took about 1 day 21 hours to train. Its training phase consisted of about 700 epochs. Generating summaries at test time is reasonable (given our GPU type) with the throughput of about 2 summaries per second on a single GPU, using a batch size of 1. Our neural network based framework is implemented using Tensorflow (v 1.10) and Python (v 3.5).

4.9 Dataset

Introduced by Nguyen et al. (2016), The Microsoft Machine Reading Comprehension (MS MARCO) is a large scale dataset focused on machine reading comprehension, question answering, and passage ranking. We are using this dataset for the task of QFAMDS. In the MS MARCO dataset, all questions (queries in our case) have been generated from real anonymized Bing user queries. The context passages, from which the answers in the dataset are derived, are extracted from real web documents using the most advanced version of the Bing search engine. The answers to the queries are human generated, and the dataset contains such question, passages, answer triples. For each query, the dataset also contains a query intent type across five different categories (a) description, (b) numeric, (c) entity, (d) person and (e) location. The dataset contains 1,010,916 real Bing user queries. The dataset also provides a well-formatted 182,669 natural language answers. About half of the dataset contains “No Answer” in their answer. There are 10 passages per query making it a dataset for multi-document summarization. All the 10,109,160 passages in the dataset are extracted from 3,213,835 full web documents. For our task, we used a subset of the MS MARCO dataset. This subset contains only “DESCRIPTION” query intent type samples whose length of answer was more than 20 words. Our selected subset was divided into 13400 training pairs, 1000 validation pairs, and 1000 testing pairs.

4.10 Evaluation

We use different metrics to evaluate the performance of our proposed models. These metrics are ROUGE-L, BLEU (1, 2, 3, 4), METEOR, EACS, GMS, and CR.

4.10.1 Baseline Models

As there is not much research conducted on the task of QFAMDS using the MS MARCO dataset, we do not have a good baseline to compare our proposed models against. As a result, we compare our results against the standard seq2seq model. Introduced by Sutskever et al. (2014), the Seq2Seq model is one of the most commonly used RNN models. The team for the MS MARCO dataset trained a standard Seq2Seq model similar to the one described in Sutskever’s paper with a query as a source sequence and the answer as a target sequence. The team trained this model on a subset of MS MARCO dataset but the MS MARCO paper does not explain the subset very well. What examples or samples does this subset contain remains unclear in the MS MARCO paper. As a result, our proposed models' comparison to the vanilla seq2seq is not straightforward since our model's training and testing data might be different than the vanilla seq2seq model's training and testing data.

4.10.2 Results

Table 4.1: Performance of various model using ROUGE-L, BLEU (1, 2, 3, 4), METEOR, EACS, GMS, and CR metrics.

Model	R-L	B-1	B-2	B-3	B-4	M	EACS	GMS	CR
Standard Seq2Seq	8.9	-	-	-	-	-	-	-	-
Sent2Query	17.72	18.35	5.65	1.57	0.46	6.95	85.17	61.47	**56.84**
Sent2Summary	18.05	19.94	6.49	2.16	0.88	**7.49**	**86.73**	62.53	59.88
Sent2CQS	**18.71**	**21.83**	**7.04**	**2.52**	**1.03**	6.25	84.22	**62.64**	64.88

Table 4.1 summarizes the result of our experiments. First, we have compared our various proposed models using the ROUGE and BLEU evaluation metrics. The official evaluation script[19] from the MS MARCO team was used to evaluate our proposed models. Both the ROUGE-L and BLEU scores are required for the evaluation of the system which used MS MARCO dataset for training and testing. All our proposed models outperform the standard Seq2Seq model in ROUGE-L score. Sent2CQS model has the best ROUGE-L score among all our proposed models. BLEU scores are only provided for our proposed models as the MS MARCO paper did not report BLEU score for Vanilla Seq2Seq model. The Sent2CQS model outperforms both the Sent2Query and Sent2Summary models wuth respect to all BLEU(1, 2, 3, 4) scores. In overall, Sent2CQS outperforms all other compared models in all evaluation metrics. Sent2CQS is the best performing system because during sentence extraction this model takes in account both the query and the summary rather than only the query or summary. Information of both the query and summary are represented in the extracted sentences. Also, our proposed models are significantly better than Standard Seq2Seq model because our proposed models encode the passage as well as the query

[19]https://github.com/dfcf93/MSMARCOV2/tree/master/Q+A/Evaluation

giving some extra useful information about the document to the decoder to process.

Secondly, we have compared our various proposed models using METEOR, EACS, GMS and CR evaluation metrics as well.

4.11 Analysis

Table 4.2: Performance of extraction methods using ROUGE-L, BLEU (1, 2, 3, 4), METEOR, EACS, GMS, and CR metrics on training data.

Model	R-L	B-1	B-2	B-3	B-4	M	EACS	GMS	CR
Sent2Query	20.22	20.42	16.17	7.38	6.57	9.11	71.38	71.33	**100**
Sent2Summary	**73.66**	**70.59**	**68.56**	**67.14**	**65.94**	**43.99**	**96.42**	**93.59**	**100**
Sent2CQS	71.67	68.05	65.79	64.28	63.06	42.86	96.01	92.83	**100**

If we compare our single document model against our the multiple document models, we can see that the single document model outperforms the multiple document models. This section contains our analysis of the poor performance of multi document models.

The first reason could be the encoding of long text on the multi document models. As compared to the single document model, the multiple document models contain longer passages of text which need to be encoded using an LSTM encoder and we know that encoding long text leads to noisy and confused encoder outputs. Using a selective mechanism might help in reducing the length of passage text; but, in the case of the multiple document model where the length of the passage was too long, the selective mechanism might not be a sufficient approach. As a result, the encoder outputs could still be noisy despite being passed through the selective mechanism.

The second reason could be the selection of bad sentences by the sentence extraction layer which were eventually forwarded to the seq2seq model for training and inference. To check the quality of the selected sentences, we extracted the sentences using the same WMD distance function and evaluated those extract sentences against the reference summaries. Table 4.2 shows the performance of different models' extraction methods using different evaluation metrics on the training data from the MS MARCO data set. This shows that except for the query based selection, the quality of sentence selection for extraction layer are generally good. One negative effect of selecting sentences from multiple source documents despite capturing the original sentence occurrence order is that the structure of the passage is destroyed. Reading the combined ordered selected sentences might not make sense as much as the complete passage would. This effect has negatively affected the generated outputs as well. The outputs generated from the multi document models having sentence extraction layer are less coherent than the outputs generated from the multi document model having no sentence extraction layer.

The third reason could be the use of query based sentence selection layer during inference for all models. During the training, sentence selection is done based on the query for the Sent2Query model, and based on the summary for the Sent2Summary model, and based on the combined query and summary for the Sent2CQS model. However, during inference all models' sentence selection layer is based on the query as the summary is not available during prediction. As a result, there is a discrepancy between the selected sentences of the training phase and the inference phase. For example, for the Sent2Summary model, during training, the selected sentences based on the summary are passed to the seq2seq model and during inference, the selected sentences based on the query are passed to the seq2seq model for prediction. But in reality, the selected sentences based on the summary are trained for the seq2seq model. The same is the case for the Sent2CQS model. Also, we know from Table 4.2 that the performance of the Sent2Query model's extraction method is poor as compared to the performance of other models extraction method. As a result, good quality sentences are not selected during the inference which are passed to the seq2seq model for prediction. This could also lead to degradation of the performance of the multi document models.

4.12 Summary

In this chapter, we proposed three different models for the task of query-focused abstractive multi-document summarization model using neural networks. Then, we presented our training and testing details and described the dataset that was used to train the model. Then, we evaluated our proposed models. Finally, we analyzed the poor performance of our proposed models. In the next chapter, we will conclude this thesis with conclusion and future work.

Chapter 5

Conclusion & Future Work

5.1 Conclusion

We proposed four models for the task of query-focused abstractive document summarization and query-focused abstractive multi-document summarization. Sequence to Sequence mapping using neural networks were applied for all these models. Firstly, we implemented a model for query based abstractive document summarization. The unique feature of this model is its novel selection mechanism based on an Inception Network. Convolutional gated units are used for this selection mechanism. The concept of the novel selective mechanism was introduced to reduce the noise present in long text of the input (passage) sequence. Advanced techniques like residual connection, highway connection, dropout, batch normalization, bucketing and beam search were used to improve the performance of the model. Experiments were conducted on the Debatepedia dataset which showed that our model outperforms the state-of-the art model in all ROUGE scores. Secondly, we proposed three models for query-focused abstractive multi-document summarization. The unique feature of these models is their coarse-to-fine approaches. Multiple passage inputs are refined by extracting the top-k sentences and those extracted sentences were passed as a passage to the single document level summarization model. In addition to previously mentioned advanced techniques, the Word Mover's Distance (WMD), a new promising distance function, was used to improve the models' performance. We conducted experiments on the MS-MARCO dataset and have reported our scores using various evaluation metrics. As there is not any good baseline model to compare our results we hope that our proposed model will be a good baseline model for upcoming works on the query-focused abstractive multi-document summarization task.

5.2 Future Work

Although the results we obtained have shown the effectiveness of the proposed query based abstractive summarization model for both single and multiple document settings, it could be further improved in a number of ways:

- Data augmentation is a regularization technique which helps improve a model's performance and helps reduce the overfitting problem. We plan to have this step before the input sequence are passed to the embedding layer.
- Due to the lack of time, we did not get enough time to experiment with all the hyperparameters of the model. We plan to tune the hyperparameters which could lead to improvement of the model's performance.
- Currently our top-k sentence extraction is based on Gensim's Word Mover's Distance algorithm. We plan to train the sentences to a neural network to extract top-k sentences.
- Currently, we replace our <UNK> words at the inference phase with word corresponding to having the maximum probability of attention distribution. We plan to implement pointer-generator model proposed by See et al. (2017).
- Currently our model avoids the document structure hierarchy except for the embedding layer. We plan to implement a model which captures the hierarchical document structure with hierarchical attention.
- Even though our model is good at capturing local and global features of the document and the corpus, we think our model is still not good enough to capture keywords. We plan to implement feature rich encoder.

- The training of our model took longer than expected. We plan to use large vocabulary trick to reduce the training time and to improve model's performance.
- Currently our model implements single headed attention. We plan to use multi-headed attention to improve model's performance.
- Inspired by Paulus et al. (2018), we plan to implement a hybrid model which combines standard supervised word prediction and reinforcement learning.
- Currently, our embedding layer consists of character level, word level, and sentence level embedding. We plan to extend our embedding layer by adding paragraph level and document level embedding to it.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. USENIX Association, Berkeley, CA, USA, OSDI'16, pages 265–283. http://dl.acm.org/citation.cfm?id=3026877.3026899.

Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. 2014. Provable bounds for learning some deep representations. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*. PMLR, Bejing, China, volume 32 of *Proceedings of Machine Learning Research*, pages 584–592. http://proceedings.mlr.press/v32/arora14.html.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR* abs/1409.0473.

Satanjeev Banerjee and Alon Lavie. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. pages 65–72.

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.

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.

Ziqiang Cao, Wenjie Li, Sujian Li, Furu Wei, and Yanran Li. 2016. Attsum: Joint learning of focusing and summarization with neural attention. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. The COLING 2016 Organizing Committee, pages 547–556. http://aclweb.org/anthology/C16-1053.

Jaime Carbonell and Jade Goldstein. 1998. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, USA, SIGIR '98, pages 335–336. https://doi.org/10.1145/290941.291025.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Universal Sentence Encoder. *arXiv e-prints* page arXiv:1803.11175.

Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, and Hui Jiang. 2016. Distraction-based neural networks for modeling documents. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. AAAI Press, IJCAI'16, pages 2754–2760. http://dl.acm.org/citation.cfm?id=3060832.3061006.

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, pages 484–494. https://doi.org/10.18653/v1/P16-1046.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. 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.

H. P. Edmundson. 1969. New methods in automatic extracting. *J. ACM* 16(2):264–285. https://doi.org/10.1145/321510.321519.

Karl Pearson F.R.S. 1901. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2(11):559–572. https://doi.org/10.1080/14786440109462720.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1631–1640. https://doi.org/10.18653/v1/P16-1154.

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

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. JMLR.org, ICML'15, pages 448–456. http://dl.acm.org/citation.cfm?id=3045118.3045167.

Mikael Kågebäck, Olof Mogren, Nina Tahmasebi, and Devdatt Dubhashi. 2014. Extractive summarization using continuous vector space models. In *Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)*. Association for Computational Linguistics, pages 31–39. https://doi.org/10.3115/v1/W14-1504.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, pages 1746–1751. https://doi.org/10.3115/v1/D14-1181.

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.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. 1999. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*. page 319. https://doi.org/10.1007/3-540-46805-6_19.

Piji Li, Wai Lam, Lidong Bing, and Zihao Wang. 2017. 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, pages 2091–2100. https://doi.org/10.18653/v1/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.

Junyang Lin, Xu SUN, Shuming Ma, and Qi Su. 2018. Global encoding for abstractive summarization. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, pages 163–169. http://aclweb.org/anthology/P18-2027.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1412–1421. https://doi.org/10.18653/v1/D15-1166.

Shulei Ma, Zhi-Hong Deng, and Yunlun Yang. 2016. An unsupervised multi-document summarization framework based on neural document model. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. The COLING 2016 Organizing Committee, pages 1514–1523. http://aclweb.org/anthology/C16-1143.

Warren S. McCulloch and Walter Pitts. 1988. Neurocomputing: Foundations of research. MIT Press, Cambridge, MA, USA, chapter A Logical Calculus of the Ideas Immanent in Nervous Activity, pages 15–27. http://dl.acm.org/citation.cfm?id=65669.104377.

Ryan McDonald. 2007. A study of global inference algorithms in multi-document summarization. In *Proceedings of the 29th European Conference on IR Research*. Springer-Verlag, Berlin, Heidelberg, ECIR'07, pages 557–564. http://dl.acm.org/citation.cfm?id=1763653.1763720.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Neural and Information Processing System (NIPS)*. https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2017. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14636/14080.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*. Association for Computational Linguistics, pages 280–290. https://doi.org/10.18653/v1/K16-1028.

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 M. Khapra, Anirban Laha, and Balaraman Ravindran. 2017. Diversity driven attention model for query-based abstractive summarization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1063–1072. https://doi.org/10.18653/v1/P17-1098.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. In *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016.*. http://ceur-ws.org/Vol-1773/CoCoNIPS_2016_paper9.pdf.

J. Niu, H. Chen, Q. Zhao, L. Su, and M. Atiquzzaman. 2017. Multi-document abstractive summarization using chunk-graph and recurrent neural network. In *2017 IEEE International Conference on Communications (ICC)*. pages 1–6. https://doi.org/10.1109/ICC.2017.7996331.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. http://aclweb.org/anthology/P02-1040.

Romain Paulus, Caiming Xiong, and Richard Socher. 2018. A deep reinforced model for abstractive summarization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HkAClQgA-.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, Valletta, Malta, pages 45–50. `http://is.muni.cz/publication/884893/en`.

Pengjie Ren, Zhumin Chen, Zhaochun Ren, Furu Wei, Liqiang Nie, Jun Ma, and Maarten de Rijke. 2018. Sentence relations for extractive summarization with deep neural networks. *ACM Trans. Inf. Syst.* 36(4):39:1–39:32. https://doi.org/10.1145/3200864.

Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. 2000. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision* 40(2):99–121. https://doi.org/10.1023/A:1026543900054.

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, pages 379–389. https://doi.org/10.18653/v1/D15-1044.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1073–1083. https://doi.org/10.18653/v1/P17-1099.

G. L. Shaw. 1986. Donald hebb: The organization of behavior. In Günther Palm and Ad Aertsen, editors, *Brain Theory*. Springer Berlin Heidelberg, Berlin, Heidelberg, pages 231–233.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15:1929–1958. http://jmlr.org/papers/v15/srivastava14a.html.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. MIT Press, Cambridge, MA, USA, NIPS'14, pages 3104–3112. http://dl.acm.org/citation.cfm?id=2969033.2969173.

C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. 2015. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. pages 1–9. https://doi.org/10.1109/CVPR.2015.7298594.

Jiwei Tan, Xiaojun Wan, and Jianguo Xiao. 2017. From neural sentence summarization to headline generation: A coarse-to-fine approach. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. pages 4109–4115. https://doi.org/10.24963/ijcai.2017/574.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Y. Zhang, M. J. Er, and M. Pratama. 2016. Extractive document summarization based on convolutional neural networks. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*. pages 918–922. https://doi.org/10.1109/IECON.2016.7793761.

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, pages 1095–1104. https://doi.org/10.18653/v1/P17-1101.

Appendix A

Abbreviations

Table A.1: Abbreviations used in this thesis

Abbreviations	**Full Forms**
ANN	Artificial Neural Network
Bi-RNN	Bidirectional Recurrent Neural Network
BLEU	Bi-lingual Evaluation Understudy
CBOW	Continuous Bag of Words
CNN	Convolutional Neural Network
CR	Copy Rate
DAN	Deep Averaging Network
D-Bi-RNN	Deep Bidirectional Neural Network
DP	Deep Learning
EACS	Embedding Average Cosine Similarity
EDA	Encoder-Decoder Framework with Attention Mechanism
EMD	Earth Mover's Distance
FCFFNN	Fully Connected Feed Forward Neural Network
FFNN	Feed Forward Neural Network
GMS	Greedy Matching Score
GRU	Gated Recurrent Unit
LCS	Longest Common Sub-Sequence
LSTM	Long Short Term Memory
METEOR	Metric for Evaluation of Translation with Explicit ORdering
ML	Machine Learning
MLE	Maximum Likelihood Estimation
MLP	Multiple Layered Perceptron
MS MARCO	Microsoft Machine Reading Comprehension
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
nBOW	Normalized Bag of Words
NMT	Neural Machine Translation
QFADS	Query Focused Abstractive Document Summarization
QFAMDS	Query Focused Abstractive Multi-Document Summarization
RNN	Recurrent Neural Network
ReLU	Rectified Linear Unit
ResNet	Residual Network
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
S2S or Seq2Seq	Sequence to Sequence
USE	Universal Sentence Encoder
WMD	Word Mover's Distance

Appendix B

Sample System Generated Query Focused Abstractive Summaries

Here, we show some examples of our system-generated summary using our query focused abstractive document summarization model described in Chapter 3 along with their query, document, and original summary from the Debatepedia dataset. Some of them are as follows:

Sample 1:
QUERY: proliferation/npt : does the us-india nuclear deal undermine non-proliferation ?
DOCUMENT: jayshree bajoria . the u.s.-india nuclear deal . council on foreign relations . july 21 2008 - what effect will the u.s.-india deal have on the npt ? it could gut the agreement experts say . article 1 of the treaty says nations that possess nuclear weapons agree not to help states that do not possess weapons to acquire them . albright says that without additional measures to ensure a real barrier exists between india 's military and civilian nuclear programs the agreement could pose serious risks to the security of the united states by potentially allowing indian companies to proliferate banned nuclear technology around the world . in addition it could lead other suppliers including russia and china to bend the international rules so they can sell
ORIGINAL SUMMARY: a us-india nuclear deal will gut the npt .
GENERATED SUMMARY: a us-india nuclear deal will gut the npt .

Sample 2:
QUERY: vs. harm reduction : is a war a better idea than harm reduction ?
DOCUMENT: many of the health dangers associated with recreational drugs exist or are made worse precisely because they are illegal . the government can not enforce quality control on products sold and manufactured illegally . examples include : the easier to make derivative mda being sold as mdma heroin users unintentionally injecting brick dust quinine or fentanyl with which their heroin had been cut ; and heroin/cocaine overdoses occurring as a result of users not knowing exactly how much they are taking .
ORIGINAL SUMMARY: legalizing drugs would improve quality control
GENERATED SUMMARY: legalizing drugs would improve quality control

Sample 3:
QUERY: history : does history prove democracy to be the best form of government ?

DOCUMENT: the existence of an overweening state machine that meddles in everything can tempt leaders to use it against their political foes . total control of the economy also sucks the air away from what istvan bibo a hungarian political thinker called the little circles of freedom the free associations and independent power centres that a free economy allows . the economist crying for freedom january 16th 2010
ORIGINAL SUMMARY: democracy has never endured in countries with mainly non-market economies .
GENERATED SUMMARY: democracy has never endured in countries with mainly non-market economies

Sample 4:
QUERY: victims : is allowing victims to see trials important ?
DOCUMENT: do n't 'trust ' court to try terrorists . lancaster online . nov 20 2009 : instead of a straightforward finding of guilt and execution in a military court americans who suffered the loss of loved ones on 9/11 will have to suffer through a lengthy trial designed to show the world that americans are fair to terrorists . ... and they will have to suffer that trial within blocks of where two planes commandeered by fellow terrorists flew into and destroyed the twin trade towers .
ORIGINAL SUMMARY: nyc civilian trial re-opens wounds of 9/11 victims
GENERATED SUMMARY: york civilian trial re-opens wounds of / victims

Sample 5:
QUERY: markets : are market forces insufficient in demanding sound public discourse ?
DOCUMENT: michael gerson . where the mines are . washington post . 14 nov. 2008 - during the campaign obama signaled that he did not support the reimposition of the fairness doctrine . but house speaker nancy pelosi and other democratic leaders are big fans of this regulation . and talk radio is already preparing for a showdown . if obama were to endorse this doctrine even reluctantly the resulting fireworks would obscure every other topic .
ORIGINAL SUMMARY: fairness doctrine would open costly political battle
GENERATED SUMMARY: fairness doctrine would open costly political battle

Appendix C

Sample System Generated Query Focused Abstractive Multi-Document Summaries

Here, we show some examples of our system-generated summary using our query focused abstractive multi-document summarization model (Sent2CQS) described in Chapter 4 along with their query, documents, and original summary from the MS MARCO dataset. Some of them are as follows:

Sample 1:
QUERY: why is coconut oil good for the skin
DOCS: using coconut oil on your skin can be the difference between flaky , dry skin , and soft , glowing skin , as this oil is known for its moisturizing benefits.widely used in skin creams , soaps and lotions for its effectiveness in fighting dry skin , coconut oil also helps to sooth various other types of skin conditions.kin benefits of coconut oil . using coconut oil on your skin can be the difference between flaky , dry skin , and soft , glowing skin , as this oil is known for its moisturizing benefits . coconut oil 's origins : in parts of the world , using coconut oil on your hair and skin has been an ancient practice . the amount of coconut oil you 'll need depends on the length and thickness of your hair . you can also put the coconut oil on a piece of cotton and rub the cotton onto your skin.oconut oil is a great skin softener and helps you do away with dry and hard skin conditions . rub the coconut oil into your hair . use your fingers to massage it in and smooth it down your hair shaft all the way to the tips.easure 3 - 5 tablespoons of coconut oil into a bowl . coconut oil 's origins : in parts of the world , using coconut oil on your hair and skin has been an ancient practice.he sees the effects by way of shiny hair and breakout-free skin . when we apply chemical gels on the lips , it is possible to consume those gels accidentally , even though they are somewhat toxic.oconut oil is a great skin softener and helps you do away with dry and hard skin conditions . if you have long , thick hair , use 5 ; if your hair is shorter and thinner , you only need 3 or 4.easure 3 - 5 tablespoons of coconut oil into a bowl .
ORIGINAL SUM: Because a great skin softener and helps you do away with dry and hard skin conditions. Simply take some coconut oil on your palms, rub your palms against each other once or twice and then on your face, hands, or wherever you want to see the moisturizing effect.
GENERATED SUM: massaging is good to help your skin and it helps your skin and it is

good to smooth your skin .

Sample 2:
QUERY: what is the purpose of a condenser on a car
DOCS: best answer : the condensor is a coil of wire inside a metal canister that has the function of capturing voltage spikes or surges that occur when the breaker points ... the refrigerant enters the condenser as a high-pressure vapor , but as it flows through the condenser and cools , it turns back into a cooler high-pressure liquid . the condenser can be compared to a radiator in an engine cooling system : the radiator releases heat from the hot engine coolant passing through it , to the atmosphere . things that can go wrong with condensers the air temp outside is cooler than the air inside the car , so moisture forms on the inside of the glass surface . your breath contains a lot of moisture and it lands on the inside of the car windows . purpose of a condensor the purpose of a condenser in the cycle of compression refrigeration is to change the hot gas being discharged from the compressor to a liquid prepared for use in the evaporator . the condenser accomplishes this action by the removal of sufficient heat from the hot gas , to ensure its condensation at the pressure available in the condenser . the condensor , or capicator , stores the emf discharged by the coil when the points open , and the collapsing force field generates a current in the secondary winding of the coil . without the condenser , there is no spark when the points open . the condenser acts like a capacitor and stores and boosts the spark until it is sent to the spark plug .
ORIGINAL SUM: In cars A/C system, the condenser is the other heat exchanger in a mobile A/C system; it condenses the refrigerant while releasing heat that was in the car.
GENERATED SUM: the purpose of a gas vehicle is to treat the air and the system is to make the seed .

Sample 3:
QUERY: what is the weather in muskogee?
DOCS: average weather muskogee , ok - 74401 - 1981-2010 normals . mostly cloudy , with a high near 75 . new rainfall amounts between a tenth and quarter of an inch , except higher amounts possible in thunderstorms . damaging winds , hail , perhaps a tornado or two will be possible from the lower great lakes into the ohio valley . elsewhere , heavy snow will impact the upper elevations of the northern and central rockies . these one-page documents provide a quick glimpse at the historical climate of the county . summarized data for weather observing stations in the county . average weather in muskogee oklahoma , united states . muskogee has a warm humid temperate climate with hot summers and no dry season . the hot season lasts for 103 days , from june 6 to september 17 , with an average daily high temperature above 84f .
ORIGINAL SUM: A warm humid temperate climate with hot summers and no dry season. The temperature typically varies from 29F to 93F over the course of the year and is rarely below 15F or above 101F. The hot season lasts for 103 days from June 6 to September 17 with an average daily high temperature above 84F.
GENERATED SUM: the weather in march , poland has a warm humid climate . there is a hot with air in 9 , and a year on clear from 0 . during summer , with a close or more up . from its park in summer , with a little on july 9 .