QUERY-FOCUSED ABSTRACTIVE SUMMARIZATION USING SEQUENCE-TO-SEQUENCE AND TRANSFORMER MODELS

MD MAINUL HASAN POLASH
Bachelor of Science, Military Institute of Science and Technology, 2017

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

MASTER OF SCIENCE

in

COMPUTER SCIENCE

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

QUERY-FOCUSED ABSTRACTIVE SUMMARIZATION USING SEQUENCE-TO-SEQUENCE AND TRANSFORMER MODELS

MD MAINUL HASAN POLASH

Date of Defence: December 19, 2019

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

Dedication

I dedicate this thesis to the Almighty ALLAH who has made every step of my life smoother than expected.

Abstract

Query Focused Summarization (QFS) summarizes a long document with respect to a given input query. Creating a query-focused abstractive summary by using a neural network model is a difficult task which is yet to be fully solved. In our thesis, we propose two neural network models for the query-focused abstractive summarization task. We propose a model based on the sequence-to-sequence architecture with a pointer-generator mechanism. Furthermore, we also use the transformer architecture to design a model for the abstractive summarization. Afterward, we train both our models with the Debatepedia dataset so that the model can learn to summarize a long document with respect to a query. We evaluate the output of our models against the human-created reference summary. Our transformer model outperforms our sequence-to-sequence model in all ROUGE scores.

Acknowledgments

"Surely, Allah is the best of planners." - [Quran, 3:54]. If I recall my undergraduate days, I cannot remember from when I started to dream of coming to a foreign country for higher studies. Thank a lot to Allah for giving me the opportunity to finish my M.Sc. thesis.

I would like to thank Professor **Dr. Yllias Chali** for giving me the opportunity to work under his supervision at the University of Lethbridge. During the last two years, his guidance, advice, motivation, and encouragement helped me a lot to complete this thesis. Whenever I face any problem regarding my research and personal life, his door was always open for me to discuss my problems.

I would like to thank my thesis committee members, **Dr. John Anvik** and **Dr. John Zhang** for their valuable time on my thesis. Their support and inspiration during course works were also helped me a lot to develop my skills.

I also would like to thank University of Lethbridge and GSA for their support to make this journey smoother for me.

I would like to thank my parents without whom I can not be the human being I am today. Their continuous supports always inspire me to work harder. I hope I could fulfill all their dreams one day.

Last but not the least, I would like to express my heartiest gratitude to all the vaia-apu for their continuous support throughout this journey, no matter whether the problem was regarding kitchen or study.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Summarization means creating short and concise sentences which contain the most relevant information from a larger body of text. These sentences can be categorized into one of the two ways based on the process - extractive or abstractive. An extractive model extracts the most important sentences from the original document to create a response. For example, we highlight the most important information of a document with a highlighter and use it to create an extractive summary of the document. Conversely, an abstractive model creates summary using words which may not be present in the original document. For example, we highlight the most important information of a document, paraphrase the original highlighted information, and use it to create an abstractive summary of the document. Query Focused Summarization (QFS) summarizes a long document with respect to a given input query where the summary can be extractive or abstractive. For example, the query "how was the cake?" would only create a summary based on the cake and not cover all the food items that were served on the table.

With increase of data being stored worldwide, it becomes next to impossible to obtain important information from this data manually. Moreover, this is tedious and economically expensive for humans to generate summary from large text document. To overcome these difficulties, researchers are trying to develop automatic summarization tools to summarize large text documents with the help of a computer system to extract the required information. In recent years, researchers refer summarization equivalent to machine translation because

summarization task has input document and output summary which are equivalent to source and destination language respectively in a machine translation system. There are some deep learning based popular translation models such as sequence-to-sequence (Sutskever et al., 2014) and Transformer (Vaswani et al., 2017). These models have gained attention from the researchers to use for the summarization task because of their ability of text generation. However, researchers are improving constantly but yet to produce a model which is able to summarize a document as accurately as human.

To become a part of this progress, we decide to develop a query focused abstractive summarization model which can save man-hours, provide impartial summaries, and make the search for information easier.

1.2 Goals

The goal of this thesis work is to design a neural network model for query-focused abstractive summarization task which can make proper use of the query. We make the following contributions:

- We design a neural sequence-to-sequence model for the query focused abstractive summarization task. Our model uses a sequence-to-sequence architecture, which has the usual encoder-attention-decoder architecture. In addition, we use a pointer mechanism to copy words directly from the input sequence.

- We design another model which uses tensor2tensor for the abstraction of the documents. To the best of our knowledge, we are the first group to use the transformer architecture to build a model for query focused abstractive summarization. This model is computationally very fast and it can capture the context for longer sequences.

1.3 Thesis Overview

The rest of the thesis is organized as follows: Chapter 2 reports the current progress of the summarization task and gives a brief description of various terms and concepts that are

used throughout this thesis. Chapter 3 presents our two proposed models and Chapter 4 describes the experiments and evaluation. Chapter 5 concludes this thesis work, along with some future directions.

Chapter 2

Related Works and Background Studies

In this chapter, we give an overview of the related works and brief description of various terms and concepts that are used throughout this thesis.

2.1 Automatic Summarization: An Overview of Previous Works

With data collection quickly becoming one of the most important industries, the ability to automate the summarization of large text documents is a huge asset. In the following section, some significant works on different summarization models have been stated which we have studied for the purpose of achieving the goal of this thesis.

2.1.1 Extractive Summarization

Text preprocessing is a mandatory step for the summarization task, which changes the text into a form that is easier to analyze. Luhn (1958) proposed an unsupervised approach which introduces some new ideas, such as stemming and stop word filtering, which are now understood as universal preprocessing steps for text analysis. Previously, researchers emphasized on the frequency of a word in a document, as the number of occurrences of a word in a document is a useful feature to measure its importance within the document. Instead of working with words in a single document, Blei et al. (2003) proposed to work with collections of discrete data such as corpora and multiple documents. Mihalcea and Tarau (2004) introduced TextRank, which is a graph-based ranking model for text processing, where every sentence is represented by a vertex and the similarity between two sentences is represented by an edge. The vertex which is connected with more vertices is recognized as

the most important vertex. Mihalcea and Tarau (2004) implemented the TextRank mechanism in extractive summarization by extracting the key phrases, calculate the score of each sentence by using Jaccard similarity between the sentence and key phrases, and the most scored sentences were selected to create the extractive summary of the document.

In the recent years, Nallapati et al. (2016a) introduced Recurrent Neural Network (RNN) based model for extractive summarization which can capture both salient and nonredundant characteristics of a sentence. In this work, the task of extractive summarization is treated as sequential classification task, where sentences are attended in the same sequence as they are in the main document. Authors used the re-purposed version of the CNN/Daily Mail corpus (Cheng and Lapata, 2016) for extractive summarization that was originally developed by Hermann et al. (2015). This model achieved the state-of-the-art results for the extractive summarization task at that time. Previously, most of the models were relying on human-engineered features such as surface features, content features, event features, etc. To reduce the dependency on the human-engineered features, Cheng and Lapata (2016) introduced a data driven approach for the extractive summarization task, which had a hierarchical encoder and an attention based extractor. Authors also proposed a large scale dataset by modifying the CNN/Daily Mail dataset which eliminated the lack of training data for extractive summarization. In order to solve the redundancy problem, Tarnpradab et al. (2017) introduced hierarchical attention networks which were able to represent the document using neural attention mechanism, along with some recurrent and convolution neural networks. However, Vaswani et al. (2017) focused only on attention mechanism that connects the encoder and decoder, where the recurrent and convolution neural networks were ignored. This attention based architecture is known as Transformer which outperforms the state-of-the-art results for majority of the NLP tasks such as questions answering, summarization, and machine translation (Liu et al. (2018), Lan et al. (2019)). Liu (2019) fine tuned Bidirectional Encoder Representations from Transformers (BERT) for extractive summarization and obtained better outcomes.

2.1.2 Abstractive Summarization

Abstractive summarization is more challenging compared to extractive summarization as we need to generate a sentence by paraphrasing, sentence compression, or creating new words which might be absent in the source document. Rush et al. (2015) described a sequence-to-sequence neural attention model which has an encoder-decoder architecture with some recurrent networks. This model is able to take only one sentence and generate abstraction of the sentence where the length of the output sentence is very short. Considering these problems, Nallapati et al. (2016b) proposed a system which could summarize two sentences with a maximum of 120 words. They used encoder-decoder based architecture with bidirectional neural net to reduce the perplexity by modifying Large Vocabulary Trick (LVT), which was originally proposed by Jean et al. (2015). The most interesting part of this encoder-decoder based architecture is to add a pointer layer in the decoder which can decide which words need to be copied from the original document and which words need to be generated based on the context. This pointer also keeps track of the words which need to be copied. Repetition of same word being generated is one of the main challenge of RNN based encoder-decoder architecture which was solved by Suzuki and Nagata (2017), where an upper bound was set for the occurrence of each target vocabulary in the encoder and control the output words in the decoder. However, the researchers have not looked after the saliency, nonredundancy, information correctness, and fluency while developing neural network based summarization models. Tan et al. (2017) addressed these problems and proposed a novel sequence-to-sequence architecture for the abstractive summarization task. They proposed a new graph-based hierarchical decoding algorithm which was able to accept and generate longer sequences than the previous sequence-to-sequences models and solve all the problems which are mentioned above.

2.1.3 Query-Focused Summarization

Query-focused summarization refers to create a short summary of a document, where a query is given to specify which information is needed to be focused in the summary. Goldstein et al. (1999) presented some early methods which used selection method for query-focused summarization. One of the main flaws of the selection mechanism is that it considers only short queries. There are few methods for retrieving a portion of a passage according to the context of a question, which are classified as the query-focused extractive summarization (Otterbacher et al., 2009; Wang et al., 2013).

However, question answering can be categorized as query-focused summarization, where the question is referred to as the query and the passage carrying the answer of the question is termed as the source document which needs to be summarized in the context of the query. This field of research has not been explored properly by the researchers previously due to the difficulty of sentence generation mechanism. Hermann et al. (2015) presented a neural network based model for question answering where they were able to generate full sentences and authors modified the CNN/Daily mail dataset into a question answering dataset. Later, Hermann et al. (2015) adopted their question answering model to the query-focused summarization task. One of the main drawbacks of these models are they generate repeated phrases during summarization which was reported by Chen et al. (2016). Nema et al. (2017) solved this repeating phrases problem by using a diversity based attention mechanism. In addition to a document attention model, Nema et al. (2017) also used a query attention model. Besides, Nema et al. (2017) also presented a new dataset named Debatepedia for their training purpose. In our thesis, we also use the Debatepedia dataset to train our models. We also present a query-focused abstractive model which uses the Transformer (Vaswani et al., 2017) architecture.

2.2 Artificial Neural Networks

Artificial neural networks are used to solve computational problems in the way how the brain operates. The basic block of a neural network is called neuron. In a neural network, there are input and output layers, as well as hidden layers in most of the cases. The hidden layers take the inputs, transform the inputs, and pass them to the output layer.

2.2.1 Build a Neuron

Figure 2.1 shows a 2-input neuron, where each input has a corresponding weight denoted by w_1 and w_2. Then each input is multiplied by its corresponding weight, summed up the multiplication results, and add a bias to calculate the output value. The output can be anything ranging from -inf to +inf. Hence, it is not possible to determine whether the neuron should be activated or not based on the output value.

Finally, the output is passed through an activation function (Chen and Chen, 1995) to decide whether the neuron should be activated or not. We apply a sigmoid function (Ito, 1991) over the output. Sigmoid function is a commonly used activation function in neural networks, which converts the output into the range [0, 1], where 0.5 and above is considered as 1 while below 0.5 as 0. If we do not use the activation function, the neural networks cannot map the complicated data such as image, video, speech, etc. The value of a sigmoid function can be defined as follows:

$$Sigmoid(x) = \frac{1}{1+e^{-x}}$$

For example,

$$In_1 = 1, W_1 = 0$$

$$In_2 = 2, W_2 = 1$$

$$bias = 4$$

$$x = (1 \times 0) + (2 \times 1) + 4 = 0 + 2 + 4 = 6$$

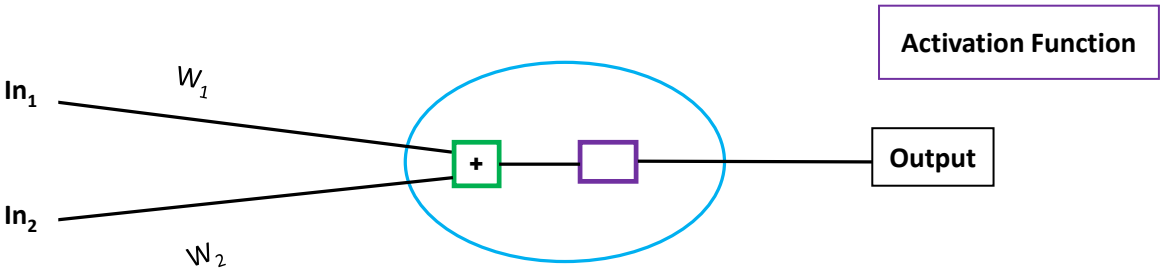

Figure 2.1: A simple 2-input neuron.

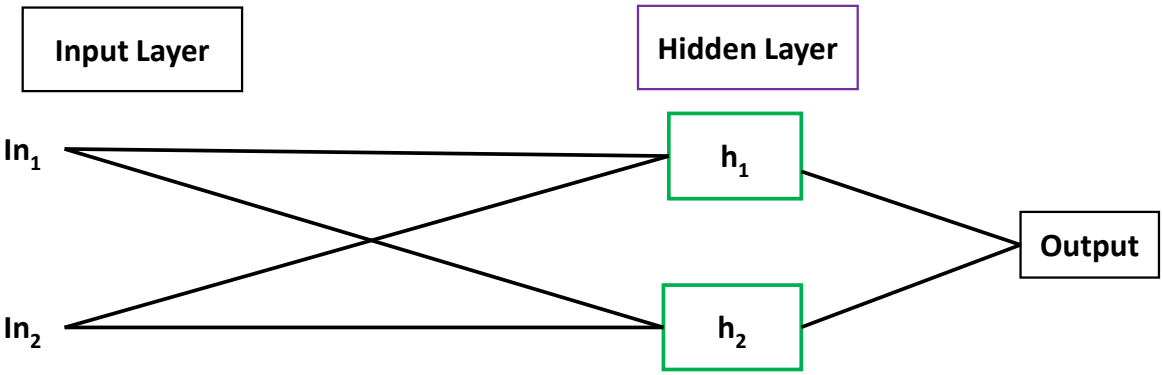

Figure 2.2: A simple Neural Network.

$$Sigmoid(6) = 0.9975$$

As the output of the sigmoid function is greater than 0.5, the neuron should be activated.

2.2.2 Build a Neural Network

Once a neuron is formed, a set of neurons are then connected to form a neural network. The number of neurons to construct a neural network can be hundreds or even millions which are arranged in a series of layers. The hidden layers stay in between the input and output layers. The number of hidden layer can be zero or more. Figure 2.2 demonstrates a simple neural network.

2.3 Deep Learning

Deep learning has become one of the popular research topics in computer science in recent years. Training a neural network architecture with millions of examples is termed as deep learning. Hence, neural networks are sometimes referred to deep neural networks as

well. Usually there are 3-4 hidden layers in a simple neural network while a deep neural network usually can have tens or hundreds of hidden layers. Deep learning models with neural network architectures are trained, so that the models can learn features automatically by analyzing the labeled examples. Hence, deep learning requires a very large amount of data in order to perform better.

Deep learning has a variety of techniques and models which are designed based on the type of the problem, size and structure of the data. As deep learning requires training the model with millions of data, the processing time is very important. Graphics Processing Unit (GPU) can process a large amount of data at a rapid speed. A deep learning system can be used in the following ways:

- **Training from Scratch:** We need to have a large amount of labeled data and a neural network architecture to train a deep network. Training a deep network can take extensive amount of time to train.

- **Transfer Learning:** We can fine-tune pretrained deep networks by feeding new data into the existing model. We can use the network to solve new tasks by making some adjustments to the existing architecture.

The performance of a deep learning system increases with the increase of the amount of data. In traditional machine learning approaches, the performance becomes constant after certain steps, while the performance of the deep learning systems increases with the increase of the amount of data.

2.4 Backpropagation Algorithm

Backpropagation is a supervised learning algorithm which is used during the training process of an artificial neural network to minimize the cost for achieving the desired output (Yu et al., 2002). In a neural network, we choose a random weight of an input initially to start our training process. As a human, it is impossible for us to predict the accurate weight.

We have to derive a function which can map the input to the output through trial and update. To determine the error in each training step we use a backpropagation algorithm. We now describe the a backpropagation algorithm with respect to the Figure 2.3. First, we start the propagation forward. The steps of the forward propagation are described below:

- Choose some random weights for W_1,W_2,...,W_8.
- Calculate the $Output_1$ and $Output_2$.
- Find the difference between the desired outputs and the calculated outputs. This difference is termed as error.

Now we start backward propagation by changing the values of the weights in order to reduce the error. The steps of the backward propagation are described below:

- Calculate the change of error with respect to the $Output_1$ and $Output_2$.
- Calculate the change of $Output_1$ and $Output_2$ with respect to its total input.
- Calculate the change of total input of $Output_1$ with respect to W_5.
- Calculate the change of total input of $Output_2$ with respect to W_8.
- Calculate the updated value of W_5 and W_8.
- Similarly, calculate the updated value of other weights.

After updating all the weights, we will again continue the forward propagation with the updated values. We will continue this process until the error falls below the threshold. This threshold for error has been determined at the beginning of the iteration.

2.5 Different Neural Network Architectures

In the following section, we describe some of the popular neural network architectures.

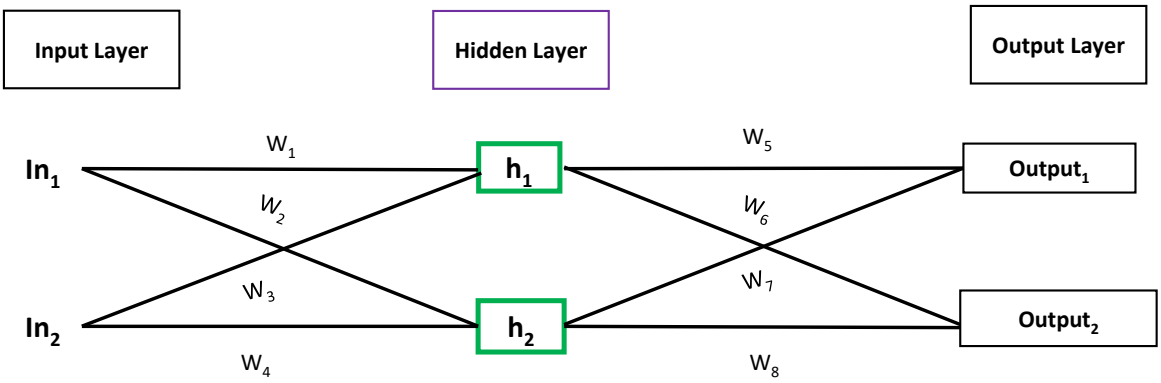

Figure 2.3: Multi-layer Neural Network.

2.5.1 Convolutional Neural Network

Convolutional Neural Network (CNN) (Sermanet et al., 2012) is a type of neural network which uses back-propagation algorithm during the learning phase. It was initially proposed for hand-written and machine-printed character recognition tasks (Lawrence et al., 1997). Furthermore, CNN has achieved massive success in image classification task such as automatic photo tagging feature of Facebook. Recently, researchers have got some interesting facts which encourage them to use CNN in natural language processing problems (Dos Santos and Gatti (2014), Santos et al. (2015)). The following are the basic layers of a CNN:

Convolutional Layer

CNN is a multi-layer neural network, where the hidden layers are called convolutional layers. These layers take the input, transform the input, and then feed the transformed input into the next layer. The layers can detect patterns by using filters. Each layer has a large number of filters. Filters are represented by a matrix, which are moved over the original matrix and multiplied with the corresponding index of the original matrix, and sums them up to convolve the required features. In case of image processing, original matrix represents the whole image. In figure 2.4, the left matrix represents the full image, the yellow portion (3×3 matrix) inside the original matrix is the filter, and the right matrix represents the convolved features after sliding the filter through the whole matrix.

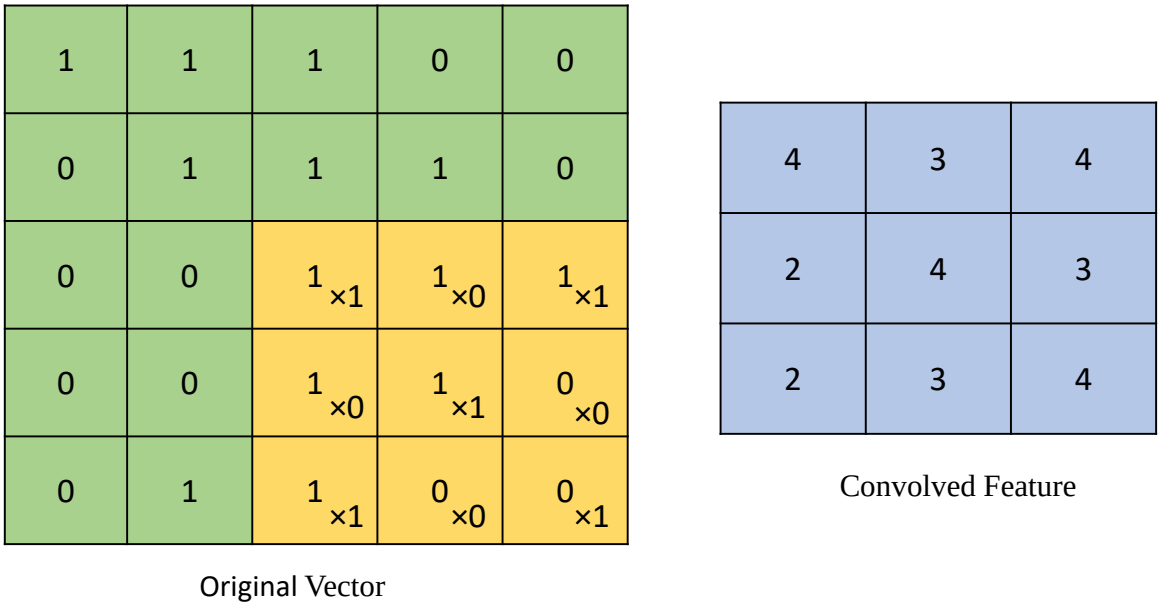

Figure 2.4: Example of a Convolution with a 3×3 Filter (Britz, 2015).

Pooling Layer

Pooling layers reduce the size of the spatial dimension by sub-sampling their inputs, which is applied after the convolutional layers. They are normally inserted periodically between two successive convolutional layers. This layer reduces the spatial size of the input which leads to reducing the number of parameters in the network, in order to reduce the amount of computation. Pooling is very important to provide a fixed size output matrix and reduce the complexity of the output dimensions.

2.5.2 Using CNN in NLP

In most of the NLP tasks, the inputs are sentences or documents. The sentences are usually represented as a matrix where each row corresponds to a word, where in image classification task, the image is represented by a matrix where each entry corresponds to a pixel. CNN performs better in classification, clustering, and regression tasks. For summarization task, it is very important to get the position of a word in a sentence to understand the context properly. Given this circumstance, CNN may not be a good fit for summarization task. Figure 2.5 shows a CNN model for sentence classification, where the sentence

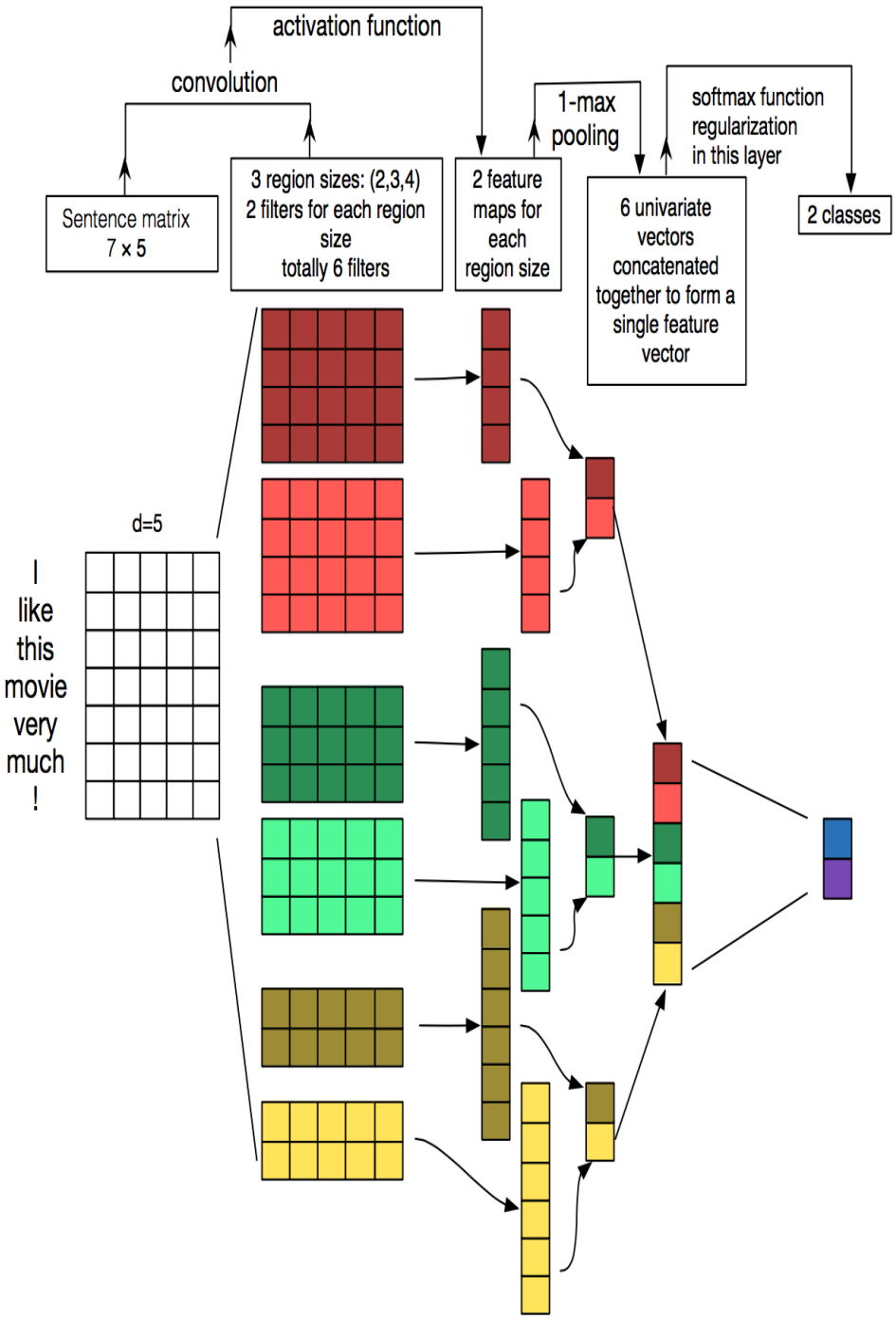

Figure 2.5: Illustration of a CNN architecture for sentence classification proposed by Zhang and Wallace (2017).

is represented by a 7×5 matrix. There are three different regions with filter sizes: 2, 3, and 4, where each region has 2 filters; 6 univariate feature maps are created by applying convolution over the sentence matrix; 1-max pooling is applied over the maps and the maps are concatenated together to create one feature vector; finally a softmax layer is performed over the feature vector which maps the values into the range [0, 1]. The output of the softmax layer is used to classify the sentence; we use two binary output states to represent the output (Zhang and Wallace, 2017).

2.5.3 Recurrent Neural Network

Recurrent Neural Network (RNN) (Yao et al., 2013) is more important in NLP, since RNN takes care of sequence rather than the individual items. RNN differs from other neural networks by refactoring some layers into a cycle. In the normal neural network, all inputs and outputs are independent to each other, where in RNN the current element is dependent on the previous elements. For example, we take a sequence of two words. We are told to predict the third word of the sequence. It would be a lot easier for the model to predict the third word if it knew the first two words. RNN serves this purpose by keeping track of the previous words in form of a memory which differs RNN from the traditional neural networks. We will design a neural network with two hidden layers to solve the problem. The first hidden layer takes the vector representation of the first word and produces an output. The output is then passed to the second layer and this layer also takes the vector representation of the second word as input. Both the inputs are added or concatenated. The output is then passed through an activation function and predicts the next word. This is how a node is computed in the next layer in RNN. This advantage is absent in traditional fully connected neural networks.

If we are given the third word and told to predict the fourth word we need to use one more hidden layer. The third hidden layer would take the output of the second layer and add it with the vector representation of the third word. The operations in the second and

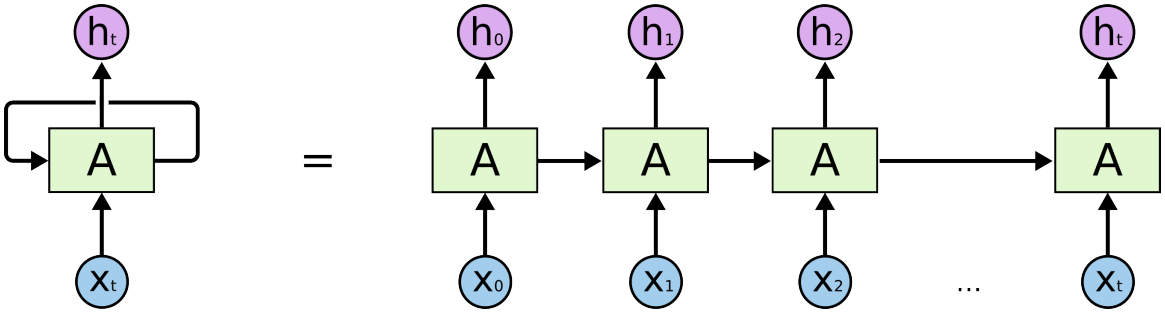

Figure 2.6: A Loop in a Simple RNN (Olah, 2015).

third layer are quite similar. We can replace both the layers with another layer which could be iterated twice to serve the purpose. In each iteration, the vector representation of the next word and the output of the last layer is concatenated as the input of the iterated layer. We can use this network for predicting next word, given a sequence of words with arbitrary length. Figure 2.6 represents the loop of the network described above.

For learning purpose, we need to train this model through trial and error. We can store the predicted word after each iteration and use them as new information in the next iteration which retain the context of the words properly, and improve the prediction capability of the network.

Suppose we have a song sequence of words and the words contain a variety of context. It is easy to capture and store the context of a three word sequence after each training step. But it is not possible if the sequence has hundreds of words in a sentence. In the broader context, at some point, RNN cannot keep track of all the past information. This problem of a standard RNN is termed as the vanishing gradient problem. This exception of RNN is briefly described by Bengio et al. (1994).

2.5.4 Gated Recurrent Unit

Gated Recurrent Unit (GRU) was first introduced by (Cho et al., 2014), which adds to a RNN and aims to solve the vanishing gradient problem of the RNN. GRU is a small neural network which connects to a RNN after the output of each iteration. A GRU consists of an update gate and a reset gate. These two gates can decide which information to carry

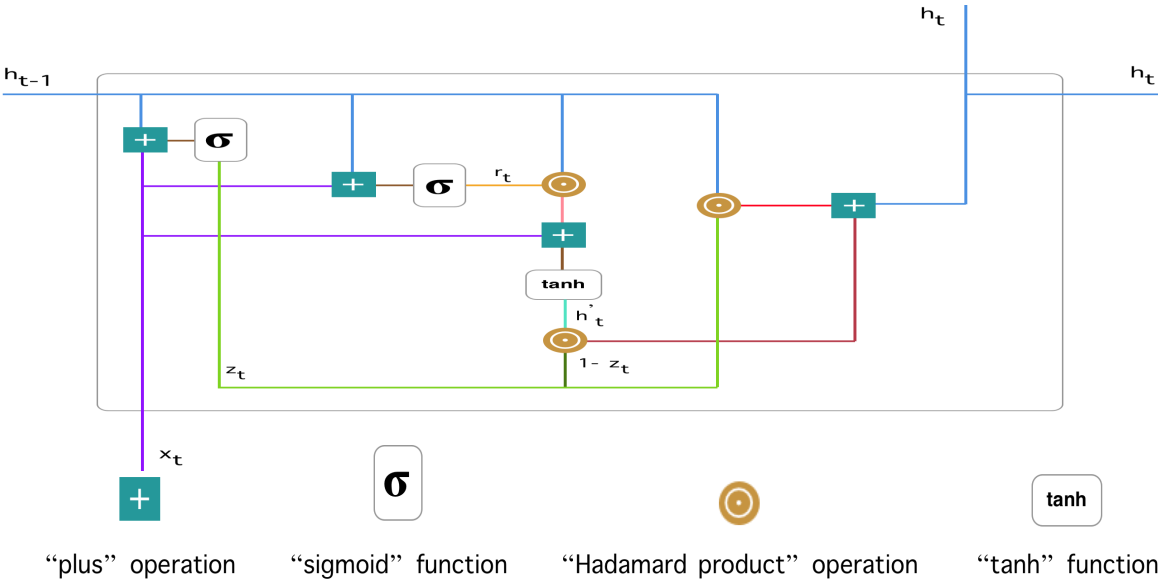

Figure 2.7: A Gated Recurrent Unit (Kostadinov, 2017)

forward for passing on to the next iteration. The main advantage of using GRU is that they can be used to keep track of long sequences without vanishing it, even if it is not relevant to the prediction. Figure2.7 shows a basic structure of single GRU unit.

In the following section, we describe a GRU with an update and a reset gate by considering a simple GRU unit [1].

- **Update Gate:** We update the gate z_t for each time step by the following formula:

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

where x_t is the input in time step t, and h_{t-1} is the activation output of the previous time step. Both of them are multiplied by their corresponding weight ($W^{(z)}$, and $U^{(z)}$) and added together. Then they are passed through a sigmoid activation function (Sibi et al., 2013). The update gate decides which previous information should be passed for the future. It also eliminates the risk of vanishing gradient by making a copy of all the previous information.

[1] The equations are collected from https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

- **Reset Gate:** The information which can be removed from the past information is determined by this gate. We calculate this by the following formula:

$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

where x_t is the input in time step t, and h_{t-1} is the activation output of the previous time step. Both of them are multiplied by their corresponding weight ($W^{(r)}$ and $U^{(r)}$), and added together. Then they are passed through a sigmoid activation function.

- **Current Memory Content:** A memory content is required in the reset gate to store the relevant information from the past which we need to use in the future. This is calculated by the following equation:

$$h_t^{'} = tanh\left(Wx_t + r_t \odot Uh_{t-1}\right)$$

where x_t is the input in time step t, and h_{t-1} is the activation output of the previous hidden layer, and r_t represents the reset gate. Both of them are multiplied by their corresponding weight (W and U). As discussed earlier, each input representing a word or character is represented as a vector in neural network. We then perform the Hadamard product (Horn, 1990) between the reset gate and the weighted hidden layer from the previous time step to determine what to forget from the past information. Finally, we use a non-linear activation function, *tanh* (Karlik and Olgac, 2011), to determine what to store from the previous information. For example, we are told to design a model to rate a video game by analyzing the user's review. Let us assume the game is Fifa20. One of the reviews is as follows:

> Fifa20 brings so many changes in the game....I feel like playing game in real life.....As they have reduced the game speed, I don't like the game much.

We do not need the whole comment to rate the game, we just need to read the last

sentence to predict the ratings from user's perspective. The reset gate will remove everything from the past and pass the last sentence to the next layer for the prediction.

- **Final Memory:** At the end, we need to calculate the final memory at the current time step which is passed to the next layer. This is calculated by the following equation:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t^{'}$$

Here, we do a Hadamard product between the update gate output (z_t) and the output from the previous time step (h_{t-1}). Again, we do a Hadamard product between (1-z_t) and $h_t^{'}$. Finally, sum both the results.

It clearly shows that for every time step, the network calculates the relevant information, passes the information to the next layer for further computation, and copy all the previous information. This is how GRU eliminates the gradient vanishing problem of an RNN.

2.5.5 Long Short Term Memory Network

Long Short Term Memory (LSTM) network which was invented by (Hochreiter and Schmidhuber, 1997), has become one of the most popular networks used instead of RNN to solve the vanishing gradient problem. The LSTM unit contains three gates to let information through after several calculations. Figure 2.8[4] represents a single LSTM unit.

The workflow of an LSTM network is described in the following section[2]:

- **Forget Gate Layer:** First, we have to determine which information is irrelevant to our prediction and leave the irrelevant information out. The information we need to forget is determined by the following equation:

$$f_t = \sigma\left(W^{(f)} x_t + U^{(f)} h_{t-1}\right)$$

[2]The equations are collected from https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57

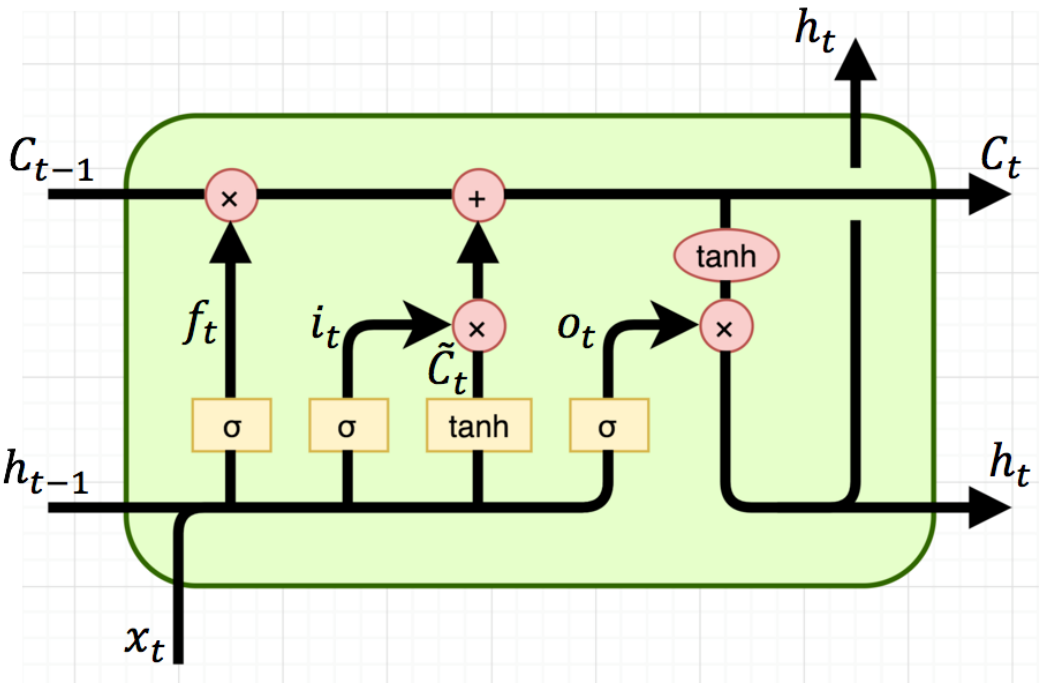

Figure 2.8: A Long Short Term Memory (Rathor, 2018).

where x_t is the input in time step t, and h_{t-1} is the activation output of the previous time step. Both of them are multiplied by their corresponding weight ($W^{(f)}$ and $U^{(f)}$) and added together. Then they are passed through a sigmoid activation function.

- **Update Gate Layer:** In this layer, we need to determine which new information we need to store in the current time step. First, we determine which value we need to update using the following equation:

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$

The new candidate values, $C_{t_{temp}}$ is calculated using the following:

$$C_{t_{temp}} = tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$

where *tanh* is an activation function for this layer which represents the output within -1 and 1. Finally, the memory content in the current time step is determined by the

following equation:

$$C_t = f_t \odot C_{t-1} + i_t \odot C_{t_{temp}}$$

where C_{t-1} is the memory content from the previous time step, and $\odot$ represents the Hadamard product operation.

- **Output Gate Layer:** Once we get the final memory content of the current step, we need to decide what information we are going to pass through output. We determine this using the following two equations:

$$o_t = \sigma\left(W^{(o)} x_t + U^{(o)} h_{t-1}\right)$$

$$h_t = o_t \odot tanh\left(C_t\right)$$

2.6 Word Representation

It is an important task to convert the words into a form which we can use as the inputs of a neural network so that the computer can understand them. In the current NLP research, people use word embedding which was first proposed by Bengio et al. (2003). Word embedding refers to a learned representation of words where the similar words have similar representation. In word embedding, the words are represented as vectors in predefined vector space. In the test and validation phase, deep learning models are expecting the same word representation that they have seen during the training phase. So it is better to use the same embedding system throughout the model. In the following section we present some of the popular word embedding techniques.

2.6.1 One-hot vector

One-hot vector is a method to represent words numerically which is also known as count vectorizing. Here, we need to create a vector whose dimension is equal to the number of unique words of a corpora. Each word is represented by a unique dimension has a 1 on that

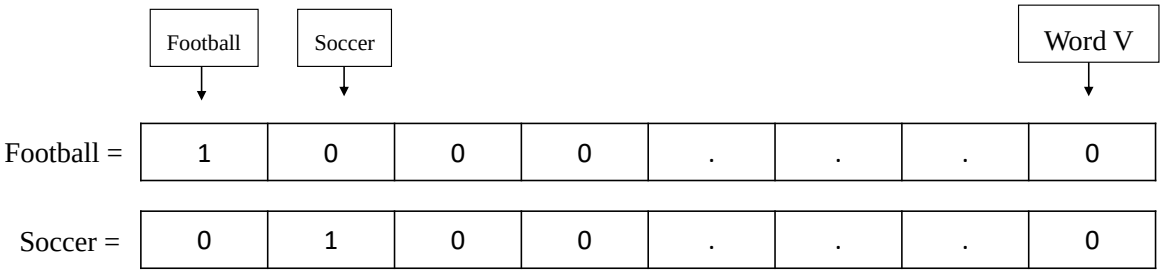

Figure 2.9: Vectors representation of word 'Football' and 'Soccer'.

dimension and 0s in every other dimensions.

For example, we have two words – 'Football' and 'Soccer'. They are represented using the following two vectors (Figure2.9) where the dimension of each vectors is 'V'. There are two problems in this system. At first, football and soccer have the same meaning but seeing their representation it is hard to determine whether they have similarity between them. And then, we need to increase the size of the memory with the increase of the number of words in the vocabulary as well.

2.6.2 Co-occurrence Matrix

This technique represents as its name. A giant $(V \times V)$ matrix is required if the size of the vocabulary is V. If two words occur together in a corpora then that position is marked with a positive entry and if there is no occurrence, then the position is marked with a 0. The positive entry represents how many times the two words occur together. Again, the dimension puts a big question mark over this technique as the dimension is squared, compared to the dimension of one-hot vector.

2.6.3 Word2Vec

The purpose of a neural network is to predict some targeted outputs given some contexts. For example, if we consider finding the similarity between words or sentences to solve a problem, then the above embedding methods will not help much because the one-hot vectors do not carry any special characteristics which can determine the semantic similarity between two words. The Word2Vec (Mikolov et al., 2013) is such a kind of embedding

system which solves the problem of one-hot vector by predicting a word's meaning based on past appearances. They proposed two methods:

- **Common Bag of Words (CBOW):** The input of this model is the context of a word and the model predicts the targeted word corresponding to the context. For example:

  I eat rice.

  We need to get the one-hot vector representation of the input words. Here, 'eat' is the context. We need to pass the one-hot vector representation of the word 'eat' to a neural network which has three layers - an input layer, a hidden layer, and an output layer. The task of the neural network is to predict the word 'rice' based on the context 'eat'. There are two weight matrices - one from the input layer to the hidden layer with a dimension of $[V \times N]$, and another from the hidden layer to the output layer with a dimension of $[N \times V]$. Here, V=3 and N is the number of dimensions in which we want to represent our word. Then the forward propagation works by multiplying the inputs of each layer with their corresponding weights and calculating the hidden layer inputs. There is no activation function in the hidden layer. One softmax layer is used at the output layer to sum the probabilities achieved in the output layer. After calculating the error, we need to re-adjust the weights to reduce the error. Figure 2.10 represents such an architecture. Here, $x_1, x_2, ..., x_V$ is the vector representation of a single input word (context) and $y_1, y_2, ..., y_V$ is the vector representation of a single target word.

  Instead of using a single context, the network can operate with multiple context. Then the one-hot representation of all the words are passed to input layer, and the rest of the networks remains the same. One of the main problems of this approach is, if a word has two different contexts they make an average of the contexts and represent the target word with the averaged vector. For example, we have a word 'drink' and the word 'drink' has two contexts - 'tea' and 'coffee'. If the model averages the two

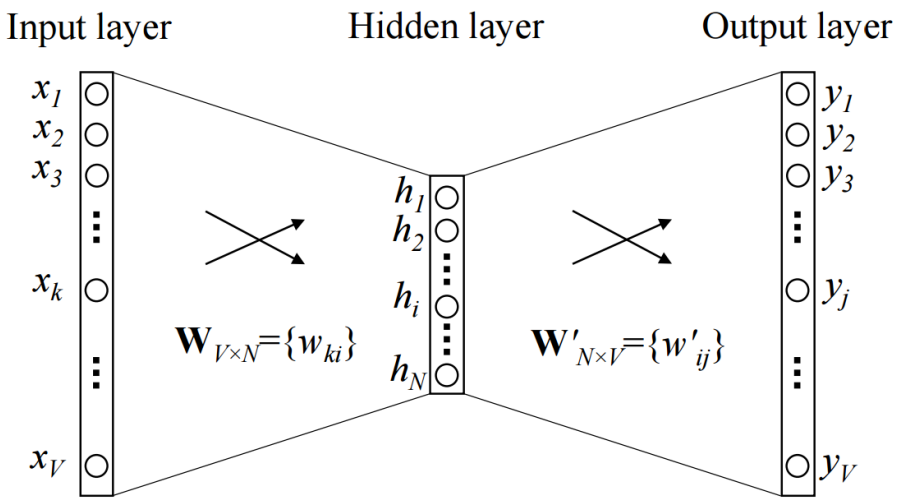

Figure 2.10: A Simple CBOW model with only one word in the context (Karani, 2018).

contexts, we will not find the exact context 'tea' or 'coffee' when we are finding them, we will get something in between which cannot predict the word 'drink' accurately.

- **Skip-Gram Model:** If we just flip the architecture of a CBOW model we will get the skip-gram model. The aim of this model is to predict the context when a word is given. The input layer with 1-context CBOW and calculations up to hidden layer are the same. The target variable will change depending on the window size of the context. As there will be more than one target vector, we will calculate the error separately for each of the target vectors. Then they are added element-wise to determine the final error. Figure 2.11 demonstrates the skip-gram model. Here, $x_1, x_2, ..., x_V$ is the vector representation of a single input word and $y_1, y_2, ..., y_C$ represent C target words (contexts), which are related to the input word.

  This model can produce two vector representation of a single word which eliminates the problem of CBOW model.

2.6.4 FastText Embedding

FastText embedding is a popular way of representing words, which is proposed by (Bojanowski et al., 2017). This is an extension of word2vec embedding system, where the

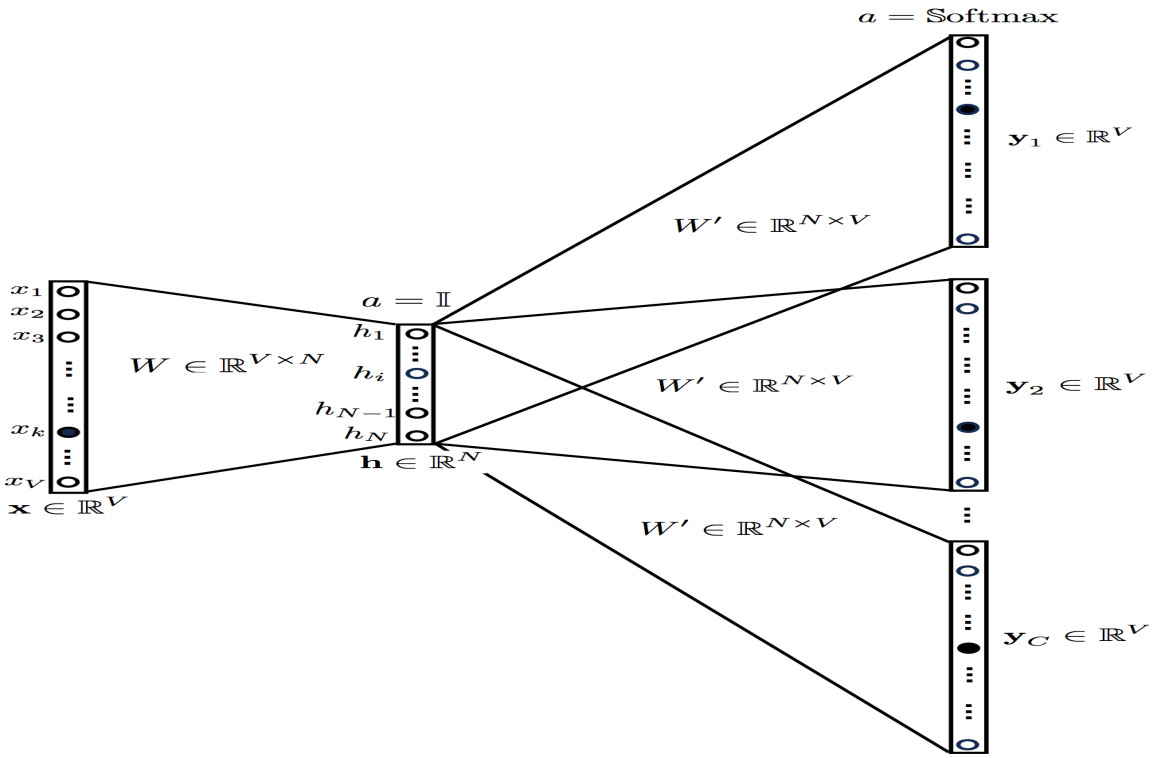

Figure 2.11: A Simple Skip-Gram model (Karani, 2018).

words are considered as several n-grams (sub-words) and fed to the model in order to obtain the vector representation of the n-grams. Finally, the vectors representing the n-grams are summed up to determine the vector representation of the word. For example, let us consider the word "thesis", which can be represented as a bag of character of 3-grams by the following: $< the, hes, esi, sis >$. Here, '<' and '>' are boundary symbols, which are used to distinguish the 3-grams from the word itself. If sub-word "the" is present in the vocabulary, then it is represented as $< the >$, which helps to preserve the meaning of the sub-words. By doing this, the rare words can be represented because their n-grams may exist in any other words.

2.7 Sentence Representation

As we know how the words are represented to feed into the neural network, we need to know how to represent a sentence. As a sentence is constructed by words, we can sum up the

word embedding of the words which construct the sentence. But in this case, the sequence of words and sentence semantics are completely ignored, which leads to this technique as a wrong approach to represent a sentence. In the following, we discuss a couple of widely used sentence embedding techniques:

2.7.1 Universal Sentence Encoder

Universal Sentence Encoder (USE) (Cer et al., 2018) maps texts into high dimensional vectors for feeding the texts to a neural network. This model is trained on many data sources and tasks, and allows it to dynamically accommodate many different natural language understanding tasks. Iyyer et al. (2015) trained the model with a deep averaging network (DAN) encoder. In the DAN encoder, the input embedding of the words and bi-grams are averaged together in order to send them through a feed-forward deep neural network. Thus, the sentence embedding of a sentence is produced. Researchers use this network as the computation time is linear to the input sequence. USE represents a sentence with a 512-dimensional vector.

2.7.2 Word Mover's Distance

Our initial plan was to use USE to measure the semantic similarity between the query and the sentences of a document. Unfortunately, USE did not perform well. In that case, Word Mover's Distance (WMD) (Kusner et al., 2015) came into rescue to help us to measure the semantic similarity between two sentences. WMD is able to determine both the semantic and syntactic similarity between two sentences. WMD considers text documents as a weighted point cloud of embedded words. The similarity between two sentences is measured by the minimum cumulative distance, where the distance is calculated by the space traveled by the words from one sentence to match the point cloud of other sentence. We discuss the details of WMD in section 4.3.1.

2.8 Evaluation of Automatic Summarization System

Once we generate the summaries with the neural network models, our main task is to evaluate the system generated summaries to measure the performance of the system in terms of generating summaries. We do this by comparing the system generated summaries with the reference summaries (mostly they are human-created). Recall-Oriented Understudy for Gisting Evaluation (ROUGE) (Lin, 2004) and Bilingual Evaluation Understudy (BLEU) (Papineni et al., 2002) are the two widely used evaluation matrices for the summarization task. We evaluate our models using ROUGE. There are four types of ROUGE matrices. They are:

- **ROUGE-N** measures the n-gram overlap between the system generated summary and the reference summary. Researchers mostly measure the unigram (ROUGE-1) and bigram (ROUGE-2) overlap. ROUGE-N can be calculated using the following:

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

  here, S_g is the generated summary sentence, s_r is the reference summary sentence, g_n is the n-grams, *Count(g_n)* is the maximum number of n-grams in the generated summary, and $Count_{match}(g_n)$ is the maximum number of n-grams in the generated summary that matches with the reference summary (Lin, 2004).

- **ROUGE-L** measures the longest sequence of words that matches between the system generated summary and the reference summary. LCS (Longest Common Subsequence) is used to find the longest sequence, which allows in-sequence matches rather than consecutive matches.

- **ROUGE-W** measures the consecutive in-sequence matches between the system generated summary and reference summary by assigning weights to the in-sequence matches in LCS.

- **ROUGE-S** measures the overlapping skip-bigram if any pair of words is in order and maintain any arbitrary distance between the pair of words. For example, the skip-bigram of the phrase "I love football" would be "I love, I football, love football".

In our thesis, we use ROUGE-N and ROUGE-L for our evaluation. Let us see the example for better understanding:

System Summary#1: I live in Lethbridge, Alberta.

Reference Summary#1: I live in Lethbridge.

First, we calculate the recall. Recall is used to measure how much of the reference summary is captured by the system generated summary. We consider only the 1-gram overlapping in this example. The recall for 1-gram is calculated by the following equation:

$$R-1_{recall} = \frac{no\_of\_overlapping\_words}{total\_words\_in\_reference\_summary} = \frac{4}{4} = 1.0$$

Now, consider the scenario, where the length of the system summary is way more than the reference summary and the system summary has all the words of the reference summary. In this case, the recall would be 100% but the summary is not concise and the system summary contains lots of irrelevant information. Let us consider a different system summary:

System Summary#2: I live in Lethbridge, which is in Canada.

So, the recall would be 1.0 but this time the system summary also contains some words which are not required. In this case, we calculate the precision which captures how much the system summary is relevant to the reference summary.

$$R-1_{Precision} = \frac{no\_of\_overlapping\_words}{total\_words\_in\_system\_summary} = \frac{4}{8} = 0.5$$

Precision and recall can be represented together by the following equation:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

We use only recall-oriented ROUGE-N for our evaluation as our models are able to produce concise summary of a given document.

2.9 Summary

In this chapter, we try to give a brief description about some of the significant works on automatic text summarization. We have introduced the basic concepts of a neural network and their functionality, and some word and sentence representation methods which convert the texts into high-dimensional vector spaces before passing them as the input of a neural network. We conclude the chapter by introducing the evaluation matrices to evaluate the output of our proposed model which is presented in the next chapter. We also discuss the model architectures, in addition to solving the query-focused summarization task using the theories that we have discussed in this chapter.

Chapter 3

Proposed Models

We propose two models for query-focused abstractive summarization. The first one is based on the sequence-to-sequence architecture and the second one is based on the transformer architecture. Our models take a document and a query as input and produce a summary of the document with respect to the query.

3.1 Problem Definition

Suppose, we have a document $d = w_1, w_2, ..., w_{\mathrm{n}}$ containing n words, a query $q = q_1, q_2, ..., q_{\mathrm{p}}$ containing p words. Our task is to find a summary $y = y_1, y_2, ..., y_{\mathrm{m}}$ containing m words where m is always less than n. We can formulate the problem by using the Bayes formula:

$$y \; = \; argmax_t \prod_{t=1}^{m} \; p \; (y_t | y_1, ..., y_{t-1}, \; d, \; q)$$

3.2 Sequence-to-sequence Model

We have designed a model based on the sequence-to-sequence architecture with attention mechanism. The input of the model is the word embedding of the document and the query, which are then passed as a sequence to the encoder and the encoder converts the embedding vectors into corresponding hidden states. The hidden states are then passed to the decoder. The decoder then generates the output summary as a sequence of words after applying a soft attention layer over the hidden states. Different components and variables which are used to design the model are discussed briefly throughout this section.

3.2.1 Document Encoder

The input of the document encoder is a sequence of words and the encoder generates a hidden state for each words. In our model, we use a bi-directional LSTM in the encoder. We use a bi-directional LSTM because it is proved that the bi-directional LSTM can capture a better context of a word rather than a unidirectional LSTM and achieved a better results for some of the text generation tasks (Bahdanau et al., 2015). The LSTM take a sequence of tokens as input and transform them into a sequence of hidden states. We need to calculate the hidden state for both the readers, and combine the result using the following equations:

$$\vec{h_t} = LSTM(\vec{h_{t-1}}, E(\vec{w_t}))$$

$$\overleftarrow{h_t} = LSTM(\overleftarrow{h_{t-1}}, E(\overleftarrow{w_t}))$$

$$h_t = [\vec{h_t}, \overleftarrow{h_t}]$$

Here, $E(w_t)$ denotes the embedding vector of the input word w at the time step t; $\overleftarrow{w_t}$ is word w in the reversed sequence. The dimension of the final hidden state is two times of the embedding dimension of the input word in both unidirectional LSTMs.

We update the gates of the LSTM in every time step according to the following equations:

$$i_t = \sigma\left(W_{(i)}E_{x_{t-1}} + b_i + W_{hi}h_{t-1}\right) + b_{hi}$$

$$f_t = \sigma\left(W_{(f)}E_{x_{t-1}} + b_f + W_{hf}h_{t-1}\right) + b_{hf}$$

$$o_t = \sigma\left(W_{(o)}E_{x_{t-1}} + b_o + W_{ho}h_{t-1}\right) + b_{ho}$$

$$c_{t_{temp}} = tanh\left(W_{(c)}E_{x_{t-1}} + b_c + W^{(hc)}h_{t-1}\right) + b_{hc}$$

$$c_t = f_t \odot C_{t-1} + i_t \odot c_{t_{temp}}$$

$$h_t = o_t \odot tanh\left(c_t\right)$$

Here, W represents the weight matrices and b represents bias vectors respectively which are learnable parameters, t represents the current time step, x represents the input tokens to the LSTM, E represents the word embedding of the input token, and c represents the memory content of the LSTM. Both h_t and c_t are initialized as 0.

3.2.2 Query Encoder

As discussed earlier, we have a query$q = q_1, q_2, ..., q_\mathrm{p}$ containing p words. The query encoder encodes the query into a hidden query state. As the length of the query is small compared to the document, we choose to use uni-directional LSTM for the query encoder. The hidden query state (h_t^q) at the time step t is updated according to the following equations:

$$h_t^q = LSTM(h_{t-1}^q, E(w_t^q))$$

$$q = h_p^q$$

Here, $E(w_t^q)$ represents the embedding vector of the input query word at the time step t, and p is the length of the query. The dimension of the hidden query encoder is equal to the dimension of the query embedding.

3.2.3 Decoder

The inputs of the decoder are the final states of the encoder and the query. We use the same embedding matrix for the encoder and the decoder. The decoder generates an output summary with respect to the query by making proper use of soft attention, and a pointer mechanism. At first, we need to feed the query encoder to the decoder input at time step t. This is similar to the question answer generation model described by Kumar et al. (2016). Instead of GRU, we use LSTM in the decoder of our model as LSTM cells outperform the GRU cells in terms of longer sequence (Britz et al., 2017). The hidden state of the decoder

is updated by using the equation:

$$h_{d_t} = LSTM(h_{d_{t-1}}, [c_t, q, E(y_{t-1})])$$

here $h_{d_o} = h_{n_d}$, the final encoder state, n is the number of input word, y_0 indicates the initial prediction which is represented as the token $< SOS >$, c_t is the current memory content, q is the query, $E(y_{t-1})$ represents the embedding of the predicted output at the previous time step. The predicted output from the previous time step comes from the pointer or generator. The main purpose of using the query in the hidden state of the decoder, is to predict a word related to context of the query. By doing so, the generated output must predict a word which is related to the context of the query.

The generator then generates an output word which belongs to the vocabulary, V. The vocabulary distribution is done using the following equation:

$$P_{vocab,t} = softmax(W_{d2v}h_{d_t} + b_{d2v})$$

where the dimension of the vector $P_{vocab,t}$ is equal to the size of the vocabulary V, W and b are learnable model parameters, and $softmax(v_t)$ is calculated as follows:

$$softmax(v_t) = \frac{exp(v_t)}{\sum_r exp(v_r)}$$

for each element v_t for vector v, v_t can be found when we apply a linear transformation on the decoder hidden state and the context vector, which is calculated by the following equation:

$$v_t = W^2_{gen}(W^1_{gen}[h_{d_t}, c_t] + b^1_{gen}) + b^2_{gen}$$

Then we calculate the probability of the generated word w, being the target word as follows:

$$y^t_{gen} = P_{vocab,t}(w)$$

3.2.4 Attention Mechanism

The main problem of an encoder-decoder based model is that the encoder does not train well, as the path between the encoder and the output is too far away to predict the target word accurately. To solve this problem, we use an attention mechanism which is designed based on the mechanism proposed by Bahdanau et al. (2015) for machine translation. In the attention mechanism, the decoder not only takes the final encoding state as input, but also focuses on the specific portions of the input document to find out the target word. The inputs of the attention mechanism are all the encoder hidden states and the current decoder hidden state. The attention layer then generates the context vector containing the current memory content which is computed using the following equations (Hasselqvist and Helmertz, 2015):

$$c_t = \sum_i \alpha_{ti} h_i$$

$$\alpha_{ti} = \frac{exp(e_{ti})}{\sum_r (e_{tr})}$$

$$e_{ti} = score(h_i, h_{d_{t-1}}, E(y_{t-1}), q)$$

$$score(h, s, x, q) = V_{att} tanh(W_{att}[h, s, x, q] + b_{att})$$

where h_i denotes the encoder hidden state at index i, $h_{d_{t-1}}$ denotes the previous decoder hidden state, E denotes the embedding vector of a word, $V_{att} \in \mathbb{R}^{d_{att}}$, W_{att} denotes a weight matrix where $W_{att} \in \mathbb{R}^{d_{att} \times (d_{enc} + d_{dec} + d_{emb} + d_q)}$, b_{att} is a bias vector. We concatenate the query (q) while computing the score so that the attention layer can focus on the query words.

3.2.5 Pointer-Generator Mechanism

One of the main drawbacks of the generator is that it may not generate some of the query words. In this case, some target words need to be copied directly from the input sequence based on their weights in the attention layer. Also, it is difficult to train a model

to learn to output names as discussed earlier in this thesis. Another reason behind using the pointer mechanism is to deal with unknown ($< unk >$) words during training (See et al., 2017; Gu et al., 2016; Nallapati et al., 2016b). To solve these issues, the generator mechanism is combined with a pointer mechanism. Nallapati et al. (2016b) introduced a model named pointer-generator, which can help to solve the issues mentioned above. Later, Merity et al. (2017) proposed the same model to solve the similar problem. We follow the implementation provided by Hasselqvist et al. (2017).

The pointer mechanism has a switch which can determine whether to copy a word from input sequence or generate a word from the vocabulary. The pointer switch is denoted by p_{swt} where $p_{swt} \in (0,1)$. The switching network calculates the probability of generating a token from the vocabulary based on the context vector (c_t) and current decoder hidden state (h_{d_t}). They are passed through a sigmoid activation function. This can be represented using the following equation:

$$p_{swt_t} = \sigma(v_{swt}[h_{d_t}, E(y_{t-1}), c_t] + b_{swt})$$

where, $\mathrm{v_{swt}} \in \mathbb{R}^{(d_{dec} + d_{emb} + d_{doc})}$ and b_{swt} are learnable parameters. If this probability is greater than 0.5, it means the switch is activated and ready to copy the word from the input sequence, otherwise it uses the generator to generate the output.

We have to determine what is copied from the input sequence by using the attention distribution. We select the word at the index (i'_t) at the time step t,

$$i'_t = \underset{i}{\mathrm{argmax}}\ \alpha_{ti}$$

Here, the index i'_t denotes the position of the word having the highest attention in the main document. Then the embedding of the word is passed to the next decoding step. Finally,

we retrieve the word from its embedding, which can be defined by:

$$y_{swt_t} = w(i_t^{'})$$

The whole scenario can be described as follows:

$y_t = y_{swt_t}$ if $p_{swt_t} > 0.5$

$y_t = y_{gen_t}$ if $p_{swt_t} \leqslant 0.5$

3.2.6 Training Loss

Let us assume that we have a corpus and each sample of the corpus has a triplet (D^n, Q^p, Y^m) where the length of the document is n, the length of the query is p, and the length of the summary is m. The document (D) and the query (Q) is passed to the model as input and the target of the model is to find the summary (Y). Our sequence-to-sequence model can compute the conditional log probability when any triplet from the input is given. The log-probability of Y^{m} is: $logP(Y^m|X^m, D^n, Q^p, \theta)$ where the document (D^{n}) and the query (Q^{p}) are given. Here, θ is training parameter.

As we follow the pointer-generator mechanism of Hasselqvist et al. (2017), we also follow the technique of finding the global minimum using their loss function. We can define the total loss function during the training phase of our model by the following:

$$L = \frac{1}{N_s}(L_{swt} + L_{attn} + L_{gen})$$

where N_s is the length of the target summary.

Let us calculate the total loss function of our training process. In the pointer-generator mechanism, we need to train the model to decide when the switch of the pointer needs to be activated in a supervised manner. To do so, we introduce an additional input x_{swt} at the time step t during training, which is set to 1 if we copy the t th word to the summary, otherwise the output of the generator is used. The loss function for this phase can be calculated using

the following equation:

$$L_{swt} = \sum_{t=1}^{N_s} (x_{swt_t}(-logp_{swt_t}) + (1-x_{swt_t})(-log(1-p_{swt_t})))$$

We use this to train the pointer of the switching mechanism (p_{swt_t}) to predict x_{swt_t} at the time step t.

We define a loss over the softmax layer of the decoder to determine the loss during the generation phase of the decoder.

$$L_{gen} = \sum_{t=1}^{N_s} (1-x_{swt_t})(-logP_{gen_t}(w^*))$$

here, w^* is the t th word in the target summary where $w^* \in V_{gen}$. We multiply by $(1-x_{swt_t})$, which helps to avoid any addition to the loss when we choose to go on with the pointer mechanism.

When the model decides to choose output by using the pointer mechanism, we use a supervised attention. The loss function at this phase is defined using the following equation:

$$L_{attn} = \sum_{t=1}^{N_s} x_{swt_t}(-log\alpha_{t_i})$$

where, i is the index of the word α_t in the input document. Finally, we sum up all the three loss functions to get the final loss, L.

3.2.7 Training Details

At first, we initialize the weight matrices and the bias vectors, which we use throughout our model. We use the uniform initialization proposed by Glorot and Bengio (2010) to initialize the weight matrices, and initialize the bias vectors as zero vectors.

We use a pre-trained FastText embedding to initialize the embedding matrices. The same embedding is used for both the input and the output words throughout the model. The dimension of the embedding vector is 512. The size of the vocabulary is 50k which is

denoted by V in the model description. A subset of V, named V_{gen}, contains three special tokens - $< UNK >, < SOS >, < EOS >$. $< UNK >$ is used to represent a word which is absent in the vocabulary. $< SOS >$ indicates the word which is fed at first in the decoder steps, which actually means the start of the summary generation process. $< EOS >$ indicates the end of the input sequence.

In order to increase the training speed, we feed the last output word to the next decoder step. In this case, we consider the last output from the target summary assuming that our predicted output from the last time step is accurate. We follow the technique described by Bengio et al. (2015).

We train our model on a machine using a TITAN GPU card along with 64GB of RAM. We use a 10 fold dataset (Bengio and Grandvalet, 2004) to train this model by training each fold, where each training phase contains 700 epochs on average. The whole training process took 7 days to complete the training process. We use Tensorflow and Python to implement our work.

As the document length of our dataset is very small, reducing document length will not save significant amount of processing time.

3.3 Transformer Model

3.3.1 Extractive Phase

We use an unsupervised method to manipulate the Debatepedia dataset which was created by Nema et al. (2017). We use Word Mover's Distance (WMD) which finds the similarity between the query and the sentences of a document. We choose the best ranked sentence according to their score and create a new version of the document. We use this new version of the document as the input of our transformer model. The workflow of this approach is described below:

Word Mover's Distance proposed by Kusner et al. (2015) is a widely used tool in machine learning which can take an input query and find out the documents related to the

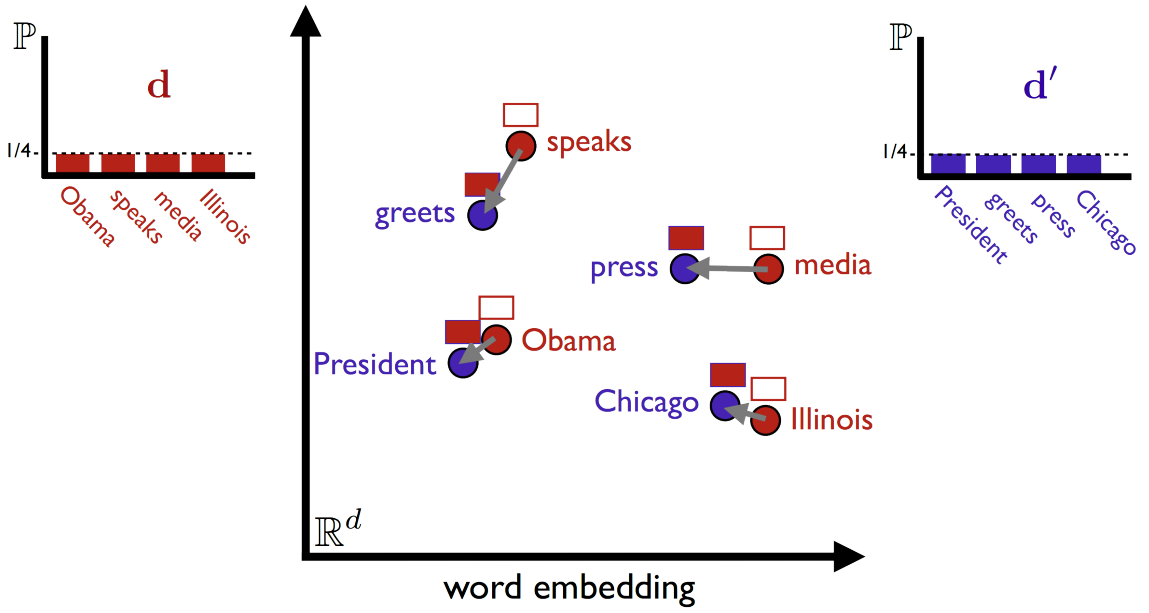

Figure 3.1: Finding the similarity between two sentences using WMD (Kusner et al., 2015).

query. WMD generally calculates the semantic similarity between two documents or two sentences even if there are no common words between them. WMD can be illustrated below for two semantically similar sentences [3].

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

There is no common word between these two sentences. Therefore, if we calculate the distance between them using common word-based metrices, we will get the maximum distance between them. But, WMD can be very useful in this context by measuring the similarity between two sentences even if there is no common word. However, they use CBOW representation to find the word frequencies in the sentences, which is denoted as d in the figure 3.1. WMD calculates the minimum "travelling distance", where "distance" represents the path to move the distribution of sentence 1 to the distribution of sentence 2.

We use the gensim's[4] WMD functionality to compute the similarity between the query

[3] http://vene.ro/blog/word-movers-distance-in-python.html
[4] https://markroxor.github.io/gensim/static/notebooks/WMD_tutorial.html

and the sentences of the document.

3.3.2 Abstractive Phase

In the sequence-to-sequence model we use a sequence of words as input and transform the input into an output sequence. But problem arises when we try to process the input in parallel manner for GPU processing using recurrent and convolution neural networks. These traditional neural networks cannot handle the long term dependencies either when the input/output sequences are too long (Kolen and Kremer, 2001). When the input and output are represented in a sequence of words and the calculations in the hidden layers are computed in parallel, the number of calculation grows with the positional distance between the input and output while trying to map the input sequence into the output sequence. Transformer (Vaswani et al., 2017) successfully reduces the number of calculations by using the multi-head attention mechanism. Therefore, we use the transformer for the abstraction phase of our model as the aim of both abstraction and machine translation is to generate words. We now describe some of the attention mechanisms that we use throughout the model for better understanding.

3.3.3 Positional Encoding

The main purpose of using transformer is to represent a model without using RNN or CNN. In the sequence-to-sequence model, we encode each input word and decode that encoders' output at the same time step in the decoder. In the transformer, the authors represent the time in the form of a sine wave which is fed as an extra input to represent the time step. As the model is designed for translation task the position of the input word is very important. Positional encoding helps the model to understand which portion of the input the model is currently processing. Positional encoding can be fixed or learnt during training. The authors prefer to use the fixed positional encoding as choosing any of them does not affect the result. We use a pre-trained positional encoding which is trained based on the following two equations proposed by the authors to let the model learn about the

position of the currently processing word in the input sequence:

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

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

We add this positional encoding with the word embedding of the words to represent the embedding vector of the words.

3.3.4 Self-Attention

The size of the embedding vector of the input words is 512, which are the inputs of the first encoder. The inputs of the remaining encoders are the outputs generated from the previous encoders. These vectors are passed through the two layers of the encoder (self-attention and feed-forward). These vectors are passed through their own path and these paths have dependencies in the self-attention layer. Therefore, various paths can be processed in parallel which is the main advantage of the transformer architecture over the sequence-to-sequence architecture.

We start from creating three vectors from each input embedding vector, which are - a query vector (Q), a key vector (K), and a value vector (V). These vectors are three different projections of the input embedding vector and their size is smaller than the input embedding vector. Here, the size of the word embedding is 512 and the size of W^Q, W^K, *and* W^V is 64. The aim of the self attention mechanism is to calculate the importance of each word with respect to other words. This is done using the following steps:

- Multiply the query vector and the key vector of the word.
- Divide the multiplication result by $\sqrt{d_k}$, where d_k represents the dimension of the key vector.
- Apply softmax over the division results.

- Multiply the softmax output with the value vector of the currently processing word.

These steps can be formulated using the following equation:

$$Self-Attention\,(Q,\,K,\,V)\;=\;softmax\,(\frac{QK^T}{\sqrt{d_k}})V$$

We have to continue this calculation for all the words of the input sequence. For example, we have two words in the sequence - 'play' and 'cricket'. In the first step, we calculate the importance of the words 'play' and 'cricket' with respect to the word 'play'. At first, we calculate the the score of the word 'play' with respect to the word 'play' following the steps described previously. When we calculate the score of the word 'cricket' with respect to the word 'play', we repeat the same steps, except we use the query vector of the word 'play' and key and value vector of the word 'cricket'. As we are considering the query vector of the first word, we get the importance of the second word of the sequence with respect to the first word.

3.3.5 Multihead Attention

The transformer does not only look at each other at each position only but also uses multihead attention mechanism. We have 8 different sets of query, key, and value vectors, which contains all the information of the input embedding. There might be some information missing when we calculate the self-attention with one set of projection vectors. These sets are called head. The mechanism of multihead attention is described below:

- There are 8 attention layers which are also called heads. They represent the linear project of key (K) and query (Q) into d_k dimension and value V into d_v dimension in order to reduce the dimension.

$$head_i\;=\;Self-Attention(QW_i^Q,\,KW_i^K,\,VW_i^V)V$$

Where, $W_i^Q \in \mathbb{R}^{d_{model}\times d_k}$, $W_i^k \in \mathbb{R}^{d_{model}\times d_k}$, $W_i^V \in \mathbb{R}^{d_{model}\times d_v}$.

- Then we calculate the self-attention for all the heads using the following equation:

$$Self-Attention\,(Q,\,K,\,V)\;=\;softmax\,(\frac{QK^T}{\sqrt{d_k}})V$$

  We choose this dot-product attention because this takes less time and space compared to the additive attention which is proposed by Kumar et al. (2016).

- We packed together all the queries, keys, and values into three matrices and compute the attention by concatenating all the layers output.

$$MultiHeadAttention(Q,K,V)\;=\;Concat(head_1,...,head_h)W^O$$

  where $W^O \in \mathbb{R}^{hd^v \times d_{model}}$, *Concat* function concatenates 8 heads into one head which is multiplied by the weight matrix W^O.

Figure 3.2 illustrates the transformer architecture that is built out of N=6 identical layers.

3.3.6 Encoder

The architecture of a transformer is composed of six encoders and six decoders. The encoders are almost similar to each other. Each encoder has two sub-layers. They are:

- **Multi-head self-attention mechanism:** The encoders' input is the embedding of the input words which pass through a self-attention layer. The purpose is to pass them through a self-attention layer so that the encoder can look at the other words of the input.

- **Position-wise fully connected feed forward network:** The self-attended layers are passed through a fully connected forward neural network. This network attends at each position. This network consists of two linear transformation, and a Rectified Linear Unit (ReLU) (Li and Yuan, 2017) activation function. This network can be

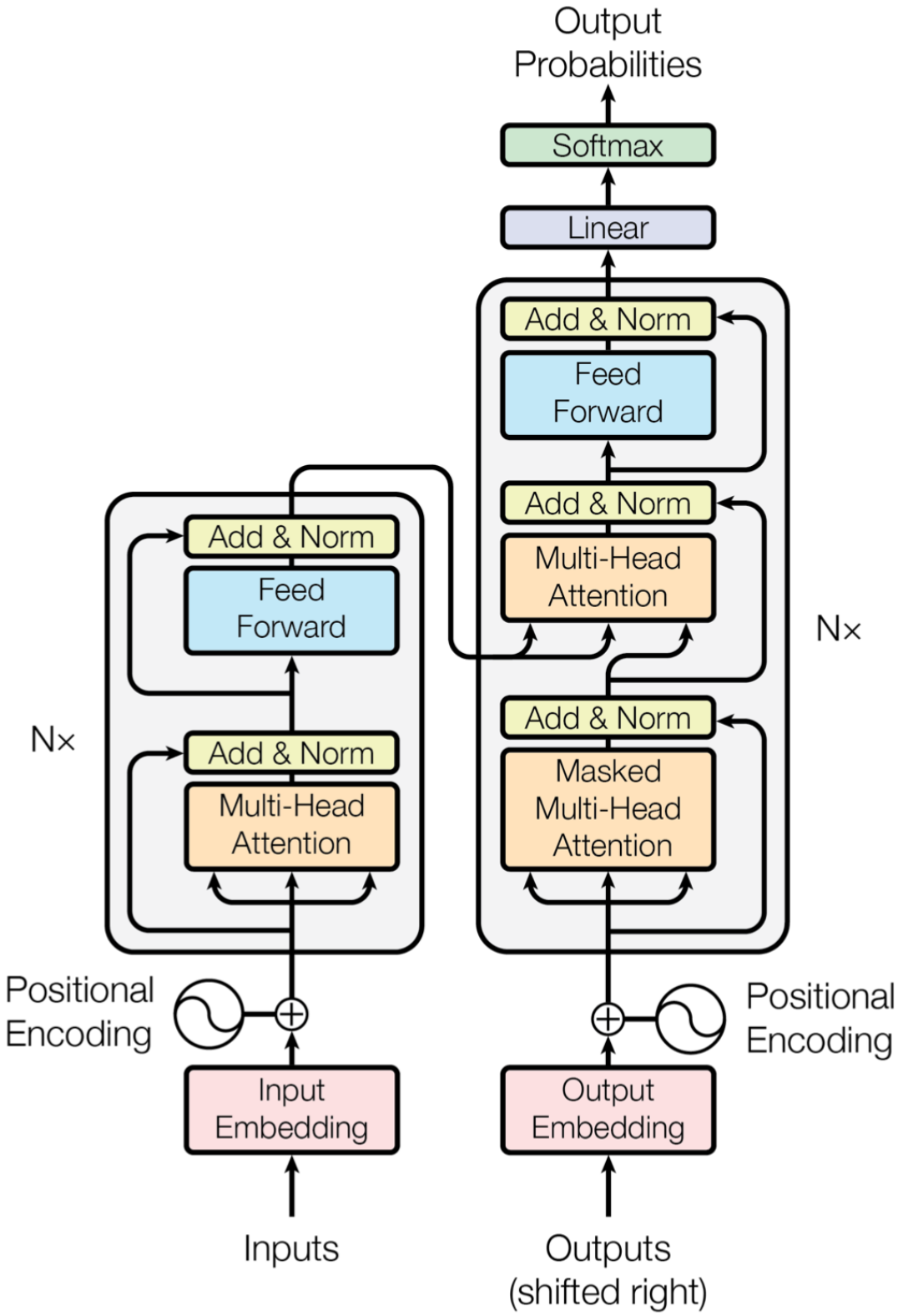

Figure 3.2: Single layer of Encoder (left) and Decoder (right) (Vaswani et al., 2017).

defined using the following equation:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

The workflow of the encoder can be divided into three stages which are described as follows:

- The input of this step is the embedding of the input words. We add the word embedding and the positional encoding in this stage. The size of both the embedding vectors is 512.

- In this step we take the output of the previous step and pass the output through multihead attention layer to calculate the attention. Then we normalize the result using the following function:

  $$LayerNorm(x + Sublayer(x))$$

  Where, Sublayer(x) refers to the function implemented by each sub-layer itself. This residual connection is applied over all the sub-layers in the model and the embedding layers so that the dimension of the vectors remains the same throughout the model. In our experiment we use, $d_{model} = 512$

- We pass the output of the previous step through the feed forward network and apply the normalization function in this stage also.

3.3.7 Decoder

The decoder of a transformer mechanism is also composed of six individual decoders. Each of the decoders has three sub-layers:

- **Multi-head self-attention mechanism:** This layer has the same mechanism as the encoders' self-attention layer.

- **Position-wise fully connected feed forward network:** This layer has the same mechanism as the encoders' feed forward network.

- **Encoder-Decoder attention:** Decoder has this extra attention layer which focuses on the specific part of the input sentence. This attention mechanism uses the query from the previous decoder layer, keys and values from the output of the encoder so that the decoder can attend all the words of the input sequences.

The workflow of the decoder is identical to the encoder.

- In the first stage, we take the output embedding as the input.

- In the next stage, we use masked multihead attention so that the future words cannot be the part of the attention.

- At the end the decoder predicts one word at a single time step. The decoder attends all the previously generated words by the decoder which is similar to our sequence-to-sequence model but computationally this model is much faster than the sequence-to-sequence model.

3.3.8 Training Details:

We train our model using a TITAN GPU card along with 64GB of RAM. We use a 10 fold dataset to train this model by training each fold, making slight change to the base model where each training phase contains 700 epochs on average. The whole training process took 6 days to complete.

We use the Adam optimizer(Bengio and LeCun, 2015) and use the equation described by Vaswani et al. (2017) to determine the learning rate.

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

where, warmup_steps = 4000 and initial learning rate is 0.001. Here, learning rate is in-

creasing linearly until the warmup_steps and then decreasing the learning rate, which is proportional to the inverse square root of the step numbers.

3.4 Summary

In this chapter, we propose two model architectures for the query-focused abstractive summarization task. We also describe the training details of the models. In the next chapter, we discuss the dataset which is used to train our models and the experimental results.

Chapter 4

Experiments and Evaluations

In this chapter, we describe the dataset that we use for training our models, evaluate our models, and comparison of our results with some baseline models.

4.1 Dataset

There are no standard datasets which are developed for the query-focused summarization task. We use the dataset created by Nema et al. (2017), which is known as Debatepedia. This dataset contain a large number of debate topics consisting of pros and cons and quotes on the topics. Nema et al. (2017) considered the debates which have at-least one query per document. After filtering, there are 663 debates left in the dataset. These 663 debates belong to 53 overlapping categories such as Politics, Religion, Environment, Law, Health, Crime, Morality, etc. Here one debate topic can belong to more than one category[5]. Table 4.1 summarizes the dataset, where the information is taken from the analysis of the authors.

Table 4.1: Average length of documents/ queries/ summaries in the dataset (Nema et al., 2017).

Average number of words per		
Document	Query	Summary
66.4	9.97	11.16

A debate has 5 queries on average, while a query has 4 related documents on average. There are 12695 $(Document, Query, Summary)$ triplets in the Debatepedia, which are crawled by the authors to create this dataset. Table 4.2 shows some of the examples.

[5]For details: `https://github.com/PrekshaNema25/DiverstiyBasedAttentionMechanism`

Table 4.2: Some examples of (*Document*, *Query*, *Summary*) triplet from the Debatepedia dataset.

Document Snippet: The "natural death" alternative to euthanasia is not keeping someone alive via life support until they die on life support. That would, indeed, be unnatural. The natural alternative is, instead, to allow them to die off of life support. **Query:** Is euthanasia better than withdrawing life support (non-treatment)? **Ground Truth Summary:** The alternative to euthanasia is a natural death without life support.
Document Snippet: Legalizing same-sex marriage would also be a recognition of basic American principles, and would represent the culmination of our nation's commitment to equal rights. It is, some have said, the last major civil-rights milestone yet to be surpassed in our two-century struggle to attain the goals we set for this nation at its formation. **Query:** Is gay marriage a civil right? **Ground Truth Summary:** Gay marriage is a fundamental equal right.

Nema et al. (2017) used 10 fold cross validation for all their experiments where the training set has 80% of the data, test set has 10% of the data, and validation set has 10% of the data. Before using the data we pre-process the data by removing different tags, stop-words, and special characters. We use 10 fold cross validation for all our experiments.

4.2 Evaluation

ROUGE produces a value between 0 and 1. When the predicted summary is same as the reference summary then it is represented by 1 and 0 represents the prediction is totally wrong. For better understanding, we represent the value in %. If there are multiple reference summaries, the system calculates the ROUGE score separately for all the reference summaries, and takes the highest score to represent the ROUGUE score for the evaluation. In our evaluation, we use ROUGE-1, ROUGE-2, and ROUGE-L. We present some samples of our system generated outputs in Appendix A.

4.2.1 Baseline Models

We compare the performance of our models with several baseline models. We implement the pointer mechanism proposed by Hasselqvist et al. (2017). One of the main

Table 4.3: ROUGE(1, 2, L) scores of the different models on the test set.

Model	ROUGE-1	ROUGE-2	ROUGE-L
vanilla e-d-a	13.73	2.06	12.84
seq-to-seq-pointer	18.25	5.04	16.17
distraction	17.80	3.45	16.38
diversity	**41.26**	**18.75**	**40.43**
seq-to-seq (our)	30.03	12.20	20.12

drawbacks of the neural sequence-to-sequence models is that the generator cannot produce some of the query words. We use the pointer mechanism to point to the words which needs to be copied from the input sequence. Another reason behind using the pointer mechanism is to deal with unknown ($< unk >$) words during training (See et al., 2017; Gu et al., 2016; Nallapati et al., 2016b). To handle these issues, the generator mechanism is combined with the pointer mechanism. We follow the implementation of the pointer mechanism proposed by Hasselqvist et al. (2017). We refer to their model as **seq-to-seq-pointer** in the rest of this section. We also compare our models with the current state-of-the-art model which is proposed by Nema et al. (2017), which is referred to as the **diversity** model during the comparison. Furthermore, we will compare our result with the result of the model proposed by Chen et al. (2016) on the DUC-2007 dataset. We refer to this model as the **distraction** model. Finally, we will compare our model with the vanilla encode-decode-attention model, which is referred as the **vanilla e-d-a**. Our two models are identified as: seq-to-seq model and transformer model. We take the average of the 10 folds to calculate the overall score of the model.

4.2.2 Results

We calculate the ROUGE scores on the test set using previously mentioned models. Table 4.3 reports the ROUGE scores of all the models.

The results show that we have improved the ROUGE scores a lot compared to the vanilla e-d-a model. In fact, all our results are better except for the diversity model. As our main goal is to design a model based on the transformer, our seq-to-seq model is a basic seq-

Table 4.4: ROUGE(1, 2, L) scores of the different abstractive models on the test set.

Model	ROUGE-1	ROUGE-2	ROUGE-L
Nallapati et al. (2016b)	28.97	9.46	25.24
Chopra et al. (2016)	28.97	8.26	**24.06**
transformer model (our)	**33.18**	**19.75**	20.87

to-seq model. The reason behind the failure of beating the score of the diversity model is discussed later.

As we have done our abstraction using the transformer, we compare our model with some abstractive models (Nallapati et al., 2016b; Chopra et al., 2016). Table 4.4 reports the ROUGE scores. We can see our transformer model performs better than the other baseline models.

4.3 Discussion

We observe that our sequence-to-sequence model does not achieve good ROUGE score. One of the main reasons behind this problem is that the attention mechanism attends only a few words at each time step. Mostly, they focus on the beginning of the document if they find the query words at the beginning of the document. As a result, the summary may capture the topic of the document properly, but they cannot generate the truthful summary by paraphrasing the main document. This problem is addressed by Hasselqvist et al. (2017). This problem can be solved using the transformer. In self-attention mechanism of the transformer, we calculate the score of each word in the input sequence with the current processing word in order to determine how much focus needs to be given on the other parts of the sentence. By implementing this mechanism, we increase the ROUGE score of the transformer model.

A common problem of a sequence-to-sequence model is that the decoder can get the same context vector as the input at several time steps, which forces the decoder to generate the same word repeatedly. Nema et al. (2017) described a mechanism to solve this issue. They proposed to introduce a checker before feeding the context vector to the decoder so

that the successive context vectors are orthogonal to each other. They do this by removing the contents from the current context if the contents are present in the previous context vector in the same direction. This mechanism is known as the diversity mechanism.

Another problem with the Debatepedia dataset is that the average words in a query is 10. In some cases, the queries contain 3/4 words which do not even form a proper sentence. The length of the documents are also very small compared to the CNN/Daily mail dataset. Unfortunately, these two datsets are the only existing datasets in this field. Moreover, both of them are not originally created for the query-focused abstractive summarization task.

4.4 Summary

In this chapter, we have discussed about the Debatepedia datset which we have used for training our models. We also have show the results of our proposed models. Finally, we have discussed the poor performance of our models and proposed some potential improvement techniques. In the next chapter, we conclude our thesis with some future directions.

Chapter 5

Conclusion & Future Work

5.1 Conclusion

We have designed two query-focused abstractive summarization models based on the neural network in this thesis work. We have evaluated our models on the test set of the Debatepedia datset and compare our results with several baseline models. Our aim of this thesis work is to use the query effectively during summary generation of a long text document. Our transformer based model achieves better results than the baseline models. Our sequence-to-sequence model could not outperform the result of the diversity model. We have noticed some issues in the output of our model, all of them are already discussed in different research works. The issues are: (1) the summaries sometimes are not relevant and truthful to the query, (2) sometimes the decoder produces same word repeatedly, and (3) the length of the document and the query sometimes prevents to produce meaningful summary.

5.2 Future Work

Our proposed models obtain better performance than all the baselines models except the diversity model. We have discussed the potential improvement areas of our current models in Section 4.3. The improvements can be done by the following:

- The problem of generating summaries which are not related to the query can be solved by using the co-reference resolution. This mechanism not only considers the identical occurrences, but also takes similar occurrences of the same entity and focuses on all the similar occurrences during attention.

- The repetition of words problem can be solved by using the diversity cell in the decoder at each time step.
- In our transformer model, we can try to feed the query to the transformer architecture along with the document which can improve the result of our transformer based model.
- We can create a dataset for the query-focused abstractive summarization task considering the lack of availability of dataset for this particular work.

Bibliography

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. http://arxiv.org/abs/1409.0473.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. MIT Press, Cambridge, MA, USA, NIPS'15, pages 1171–1179. http://dl.acm.org/citation.cfm?id=2969239.2969370.

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.

Yoshua Bengio and Yves Grandvalet. 2004. No unbiased estimator of the variance of k-fold cross-validation. *Journal of machine learning research* 5(Sep):1089–1105.

Yoshua Bengio and Yann LeCun, editors. 2015. *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. https://iclr.cc/archive/www/doku.php%3Fid=iclr2015:accepted-main.html.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3(Jan):993–1022.

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.

Denny Britz. 2015. Understanding convolutional neural networks for nlp. http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/.

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Copenhagen, Denmark, pages 1442–1451. https://doi.org/10.18653/v1/D17-1151.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Brian Strope, and Ray Kurzweil. 2018. Universal sentence encoder for English. In *Proceedings of*

the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. Association for Computational Linguistics, Brussels, Belgium, pages 169–174. https://doi.org/10.18653/v1/D18-2029.

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.

Tianping Chen and Hong Chen. 1995. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks* 6(4):911–917.

Jianpeng Cheng and Mirella Lapata. 2016. Neural summarization by extracting sentences and words. *Association for Computational Linguistics* pages 484–494. https://doi.org/10.18653/v1/P16-1046.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, pages 1724–1734. https://doi.org/10.3115/v1/D14-1179.

Sumit Chopra, Michael Auli, and Alexander M. Rush. 2016. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, San Diego, California, pages 93–98. https://doi.org/10.18653/v1/N16-1012.

Cicero Dos Santos and Maira Gatti. 2014. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. pages 69–78.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*.

Jade Goldstein, Mark Kantrowitz, Vibhu Mittal, and Jaime Carbonell. 1999. Summarizing text documents: Sentence selection and evaluation metrics. In *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, USA, SIGIR '99, pages 121–128. https://doi.org/10.1145/312624.312665.

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, Berlin, Germany, pages 1631–1640. https://doi.org/10.18653/v1/P16-1154.

Johan Hasselqvist and Niklas Helmertz. 2015. A neural attention model for abstractive sentence summarization.

Johan Hasselqvist, Niklas Helmertz, and Mikael Kågebäck. 2017. Query-based abstractive summarization using neural networks. *arXiv preprint arXiv:1712.06100* .

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, Curran Associates, Inc., pages 1693–1701. http://papers.nips.cc/paper/5945-teaching-machines-to-read-and-comprehend.pdf.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

Roger A Horn. 1990. The hadamard product. In *Proc. Symp. Appl. Math*. volume 40, pages 87–169.

Yoshifusa Ito. 1991. Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory. *Neural Networks* 4(3):385–394.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. volume 1, pages 1681–1691.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On using very large target vocabulary for neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Beijing, China, pages 1–10. https://doi.org/10.3115/v1/P15-1001.

Dhruvil Karani. 2018. Introduction to word embedding and word2vec. https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa.

Bekir Karlik and A Vehbi Olgac. 2011. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems* 1(4):111–122.

J. F. Kolen and S. C. Kremer. 2001. *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*, IEEE. https://doi.org/10.1109/9780470544037.ch14.

Simeon Kostadinov. 2017. Understanding gru networks. https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be.

Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask me anything: Dynamic memory networks for natural language processing. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*. PMLR, New York, New York, USA, volume 48 of *Proceedings of Machine Learning Research*, pages 1378–1387. http://proceedings.mlr.press/v48/kumar16.html.

Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. 2015. From word embeddings to document distances. In *International conference on machine learning*. pages 957–966.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *arXiv e-prints* page arXiv:1909.11942.

Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. 1997. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks* 8(1):98–113.

Yuanzhi Li and Yang Yuan. 2017. Convergence analysis of two-layer neural networks with relu activation. In *Advances in Neural Information Processing Systems*. pages 597–607.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, pages 74–81. https://www.aclweb.org/anthology/W04-1013.

Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198* .

Yang Liu. 2019. Fine-tune bert for extractive summarization. *arXiv preprint arXiv:1903.10318* .

H. P. Luhn. 1958. The automatic creation of literature abstracts. *IBM J. Res. Dev.* 2(2):159–165. https://doi.org/10.1147/rd.22.0159.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. https://openreview.net/forum?id=Byj72udxe.

Rada Mihalcea and Paul Tarau. 2004. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*. pages 404–411.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. Curran Associates Inc., USA, NIPS'13, pages 3111–3119. http://dl.acm.org/citation.cfm?id=2999792.2999959.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2016a. In *SummaRuNNer: A Recurrent Neural Network based Sequence Model for Extractive Summarization of Documents*. volume abs/1611.04230. http://arxiv.org/abs/1611.04230.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar GuÌ‡lçehre, and Bing Xiang. 2016b. 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, Berlin, Germany, pages 280–290. https://doi.org/10.18653/v1/K16-1028.

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, Vancouver, Canada, pages 1063–1072. https://doi.org/10.18653/v1/P17-1098.

Christopher Olah. 2015. Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Jahna Otterbacher, Gunes Erkan, and Dragomir Radev. 2009. In *Biased LexRank: Passage retrieval using random walks with question-based priors*. volume 45, pages 42–54. https://doi.org/10.1016/j.ipm.2008.06.004.

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*. Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, pages 311–318. https://doi.org/10.3115/1073083.1073135.

Saurabh Rathor. 2018. Simple rnn vs gru vs lstm :- difference lies in more flexible control. https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 379–389. https://doi.org/10.18653/v1/D15-1044.

Cicero Nogueira dos Santos, Bing Xiang, and Bowen Zhou. 2015. Classifying relations by ranking with convolutional neural networks. *arXiv preprint arXiv:1504.06580* .

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, Vancouver, Canada, pages 1073–1083. https://doi.org/10.18653/v1/P17-1099.

Pierre Sermanet, Soumith Chintala, and Yann LeCun. 2012. Convolutional neural networks applied to house numbers digit classification. *arXiv preprint arXiv:1204.3968* .

P Sibi, S Allwyn Jones, and P Siddarth. 2013. Analysis of different activation functions using back propagation neural networks. *Journal of Theoretical and Applied Information Technology* 47(3):1264–1268.

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

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

Jiwei Tan, Xiaojun Wan, and Jianguo Xiao. 2017. Abstractive document summarization with a graph-based attentional neural model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, pages 1171–1181. https://doi.org/10.18653/v1/P17-1108.

Sansiri Tarnpradab, Fei Liu, and Kien A Hua. 2017. Toward extractive summarization of online forum discussions via hierarchical attention networks. In *The Thirtieth International Flairs Conference*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. pages 5998–6008.

Lu Wang, Hema Raghavan, Vittorio Castelli, Radu Florian, and Claire Cardie. 2013. A sentence compression based framework to query-focused multi-document summarization. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Sofia, Bulgaria, pages 1384–1394. https://www.aclweb.org/anthology/P13-1136.

Kaisheng Yao, Geoffrey Zweig, Mei-Yuh Hwang, Yangyang Shi, and Dong Yu. 2013. Recurrent neural networks for language understanding. In *Interspeech*. pages 2524–2528.

Xinghuo Yu, M Onder Efe, and Okyay Kaynak. 2002. A general backpropagation algorithm for feedforward neural networks learning. *IEEE transactions on neural networks* 13(1):251–254.

Ye Zhang and Byron Wallace. 2017. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Asian Federation of Natural Language Processing, Taipei, Taiwan, pages 253–263.

Appendix A

Sample System Generated Summaries

Here, we show some system generated summary of our sequence-to-sequence and transformer models.

Sequence-to-Sequence Model:
Sample1:
Document Snippet: israel and the flotilla . chicago tribune editorial . june : video shows the israeli commandos were surrounded and attacked as they reached the ship 's deck . the israelis tried to avoid a lethal confrontation . israeli authorities reportedly offered the vessels the same deal that was accepted by at least one previous flotillas divert to the israeli port of ashdod and unload the cargo for inspection . as long as the cargo does n't contain weaponry it will be shipped into the gaza strip by land .
Query: self-defense : were israeli soldiers protect themselves ?
Ground Truth Summary: israel commando acted in self-defense once on gaza flotilla.
System Generated Summary: israeli commandos acting in self-defense once on gaza.

Sample2:
Document Snippet: a boys school will usually have a largely male staff where women may feel uncomfortable or denied opportunity and vice versa .
Query: teachers : do teachers work well in single-sex schools ?
Ground Truth Summary: teachers are often discriminated against in single-sex schools .
System Generated Summary: teachers are often discriminate against in co-ed schools .

Sample3:
Document Snippet: while sanctions may be having unfortunate effects on the mexican people the impact of sanctions on the morale of the mexican people and their respect for the united states can be mollified through various media sources broadcasting from the united states to cuba . the intention of the sanctions to punish the mexican regime and to helping protect the freedoms of cubans can be better explains to the mexican people through these messages .
Query: are sanctions a faulty policy in the effort to democratize cuba ?
Ground Truth Summary: media helps soften mexican most impression of us sanctions.
System Generated Summary: media help softens cuban popular impressions of us sanction.

Sample4:
Document Snippet: most farming fish are omnivorous so wild fish are caught to feeds them . greenpeace estimates that for every pounds of farming salmon produced six pounds of wild fish are caught to feeds the salmon .
Query: effect on wild fish : does fish farming harms wild fish ?
Ground Truth Summary: salmon farming does not stop the catch of wild fish.
System Generated Summary: salmon agriculture are often to stop .

Transformer Model:
Sample1:
Document Snippet: even if hiroshima was necessary the u.s. should have waited for word on the devastation of hiroshima to filter out to the people and leadership of japan . if they had waited and playing more diplomatic cards in the interim it would have been possible to persuade japan to surrender.
Query: was it necessary to drop the third bomb on nagasaki ?
Ground Truth Summary: the us should have waited long before bombing nagasaki .
System Generated Summary: us should have waiting long before bombing nagasaki.

Sample2:
Document Snippet: irrespective of what kind of work they are doing this deprives them of something so important that we make it compulsory for all children . although the maximum law requirements can often be provided by tutors on the set or sports academies it can be hard to keeping performance and education in proper balance when one appears to bring so many immediate rewards both in terms of notoriety and either .
Query: learning : does performing depriving child of valuable teaching experiences such as time-spent on education ?
Ground Truth Summary: if children are work or perform they are not spending their time in informal education .
System Generated Summary: if children are simply to know they not spending their time in formal education .

Sample3:
Document Snippet: miscarriage can only occur after the implantation of the fertilised egg into the uterus - and once this has happened emergency contraception does not work anyway . this interpretation was upheld in followed a case came by the society for the protection of unborn children and opposed by the departments of health . the possibility of make the argument at all however does expose the inadequacy of the abortion act which does not enshrine a woman s right to abortion but creates certain exceptions to the state s basic continued right to controls her fertility .
Query: united kingdom : should emergencies contraception be banned throughout the rubric of uk law ?
Ground Truth Summary: uk-specific argument : this arguments is irrelevant because emergencies contraception does not cause miscarriage.
System Generated Summary: uk-specific argument : this is irrelevant because emergency contraception does not causes miscarriage.

.

Sample4:
Document Snippet: regional party organizations do not assign delegates on the basis of population but through a confusing mix of political considerations . this means that two states with the same size populations may be given very different numbers of delegates . this means that the votes of citizens in different states count correctly and that states typically are value unequally .
Query: state interests : are state interests upholding in the primary election process ?
Ground Truth Summary: secondary elections delegates are not distributed fairly between state.
System Generated Summary: primary election delegates is not distributed fairly between all.

Sample5:
Document Snippet: if us troops leave there will no longer be a foreign target for terrorists in iraq and from the region . certainly they may try to attack the united state and other western countries through various means but his would be more difficult than fighting us forces in iraq . withdrawing from iraq would deprive terrorists of as many opportunities to strike as they presently enjoy .
Query: war on terror : is it wrong to think the war in iraq protect the homeland ?
Ground Truth Summary: iraq is only a jihadist-terrorist threat because we soldiers are there to be targeted .
System Generated Summary: iraq is only a many threats because us troops are there to be targeted .