ABSTRACTIVE TEXT SUMMARIZATION ON WIKIHOW DATASET USING SENTENCE EMBEDDINGS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BAHATTIN TOZYILMAZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2019

Approval of the thesis:

ABSTRACTIVE TEXT SUMMARIZATION ON WIKIHOW DATASET USING SENTENCE EMBEDDINGS

submitted by **BAHATTIN TOZYILMAZ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Assist. Prof. Dr. Hande Alemdar
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Assist. Prof. Dr. Elif Sürer
Graduate School of Informatics, METU _____

Assist. Prof. Dr. Hande Alemdar
Computer Engineering, METU _____

Assist. Prof. Dr. Bahri Atay Özgövde
Computer Engineering, Galatasaray University _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Bahattin Tozyılmaz

Signature :

ABSTRACT

ABSTRACTIVE TEXT SUMMARIZATION ON WIKIHOW DATASET USING SENTENCE EMBEDDINGS

Tozyılmaz, Bahattin
M.S., Department of Computer Engineering
Supervisor: Assist. Prof. Dr. Hande Alemdar

September 2019, 45 pages

Summarization is a well known natural language processing task that is used in our day-to-day lives. The field saw recent research using neural networks and word embeddings. We use WikiHow dataset and show that we can match performance of a similar model using sentence embeddings, and using abstractive summarization. We show that we can use sentence embeddings and lower input data size without impacting performance too much.

Keywords: automatic summarization, artificial neural networks, sentence embedding, task embedding

ÖZ

WİKİHOW VERİ SETİ ÜZERİNDE CÜMLE GÖMMELERİ İLE SOYUTLAMALI ÖZETLEME

Tozyılmaz, Bahattin
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Dr. Öğr. Üyesi. Hande Alemdar

Eylül 2019 , 45 sayfa

Özetleme günlük hayatta karşımıza çıkan, iyi bilinen bir doğal dil işleme görevidir. Yakın zamanda bu alanda yapay sinir ağları ve kelime gömmeleri kullanan çalışmalar yapılmıştır. Bu çalışmamızda, cümle gömmeleri ve soyutlamalı özetleme yaparak, WikiHow veri seti üzerinde benzer diğer çalışmaların performansını yakalayan bir özetleme sistemi geliştirdik. Bu çalışmamız ile, performansı aşırı düşürmeden cümle gömmeleri kullanıp girdi veri boyutunu düşürebileceğimizi gösterdik.

Anahtar Kelimeler: otomatik özetleme, yapay sinir ağları, cümle gömme, görev gömme

To my family

ACKNOWLEDGMENTS

I would like to express my sincerest gratitudes to my advisor Assoc. Prof. Dr. Hande Alemdar. She shaped this thesis from the initial ideas we had and supported me from thousands of kilometers away. I always felt her support, encouragement and motivation besides me.

I like to thank Prof. Dr. Fatoş Yarman-Vural for her invaluable mentorship and help in shaping up this thesis.

I would like to show my gratitude to defense jury members, Assoc. Prof. Dr. Elif Sürer and Assist. Prof. Dr. Bahri Atay Özgövde for evaluating my thesis and their valuable feedbacks.

I would like to thank to all my friends and my colleagues for their support throughout my thesis.

I specially thank my dear Işıl for filling my life with joy and being with me all through this process.

Finally, I must heathily thank my parents, Zeynep and Necdet, and my sister, Özlem for their endless care, support and love.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

BLEU	Bilingual evaluation understudy
CSV	Comma seperated values
GRU	Gated recurrent unit
GUID	Globally unique identifier
LSTM	Long short-term memory
NLL	Negative log likelihood
NLP	Natural language processing
NLTK	Natural language processing toolkit
ReLU	Rectified linear unit
RNN	Recurrent neural network
ROUGE	Recall-oriented understudy for gisting evaluation
Seq2Seq	Sequence to sequence

CHAPTER 1

INTRODUCTION

1.1 Motivation

Text summarization is an established field in natural language processing. Text summarization can produce summaries for both sentences and longer documents. Summarization has practical uses in our everyday life. The title for the news article you read may have been produced by a summarization system. The twitter post you saw for a news article was probably written by a summarizer and edited by a human. We see text written by machines all the time. Automatic summarization helps us glance at some text and get the gist of it.

There are two main types of summarization methods. First type is extractive summarization. In this type of summarization, summary words or sentences are selected from source text. The other type is abstractive summarization. This type of summarization constructs its own sentences by using information from input sentences. For most time, hand tuned algorithms and feature engineered solutions domineered text summarization field. Up until recently, extracrtive summarization was getting state-of-art results in this field. Recently, there has been research using neural networks and these research started giving fruits in both extractive and abstractive summarization tasks.

In this thesis, we focus on document summarization using abstractive sentence generation. Unlike previous research (detailed in chapter two), we use sentence embeddings for our inputs. We apply our solution to a comparatively new dataset and compare our results with previous solutions.

1.2 Contributions

In this thesis, we use sentence embeddings to create summaries. Sentence embeddings are not used widely in abstractive summarization systems. Also, there is no other previous research done using sentence embeddings with WikiHow dataset. Our system greatly reduces input sequence length, thus lowers time and other resources needed to train a summarization system. Use of pretrained word embeddings boosted many NLP task results compared to embeddings learned during training in dataset. We hope that by using external sentence embeddings, we can also use embeddings trained on billions of sentences and leverage other research. Also, we try to show that by getting similar performance using sentence embeddings, sentence embeddings are a viable option to word embeddings.

We also propose a new task for wikihow dataset. We dub this new task "*task to title*". Task to title tries to regenerate titles from task description. It produces a very high level summary of task at hand. By doing this, we are trying to generate embeddings that can encode the whole task description.

1.3 Outline

This thesis is divided into seven chapters. In introduction chapter, we list our motivations for selecting this subject and contributions to the research field. Related work chapter follow introduction with some information on sentence embeddings, text summarization and some previous work related with our research. After this, in artificial neural networks chapter, we give a primer on neural networks and embeddings. We follow that with a chapter to explore our dataset and give examples and metrocs about it. In sentence vector approach chapter, we explain our approach, our network architecture and inputs-outputs. After this, we present our results and example predictions in experiments and results chapter. We follow this with conclusion and future work chapter, which we provide a conclusion and potential future work.

CHAPTER 2

RELATED WORK

We build our solution on already existing ones. We use encoder-decoder, networks, pretrained sentence embeddings and a readymade dataset. We apply our solution to the well known summarization problem and we explore a new viewpoint.

Our code is built using PyTorch [5]. We use PyTorch's builtin neural network constructs with minimal additions.

2.1 Sentence Embeddings

Words and sentences are basic building blocks of human languages. In natural language processing tasks, we usually need to process words and sentences. Words are usually converted to **token**s for NLP tasks, where a word or a symbol is converted to an integer. Using these tokens as they are introduces complexity to network, so they are usually embedded, i.e. converted to a vector of numbers. These embedding are learned along with task at hand. There are also word embeddings that are unsupervised and generic. Examples being word2vec, GLoVe and fastText.

Sentences are the main building blocks of meaning and sentiment. Most NLP tasks require researchers to work on sentences. Creating a token for each and every possible sentence would be impractical. Thus, sentences are either treated as a sequence of word or converted to an embedding in NLP tasks. Sentence embeddings build on word embeddings. There are several methods of building sentence embeddings. The simplest baseline can be defined as the average of word embeddings for the words constituting the sentence. There are other methods of using the words in the sen-

tence in a sequence to sequence network or treating the sentence as the context for its constituting words. There has been an interest in building sentence (and document) embeddings lately, so there are recent research in the area. Examples for sentence embeddings would be skip-thought vectors, sent2vec, inferSent, universal-sentence-encoder.

One of the well known and earlier unsupervised sentence embedding solutions is **skip-thought vectors** [6]. Given a document collection, skip-thought vectors train a neural network to generate previous and next sentences using current sentence. System is modelled as three recurrent neural networks. An encoder that reads current sentence one by one using word embeddings and two decoders each generating previous and next sentences in the document. Since sentence order is kept, the system also learns temporal relations. One major drawback of this system is training time. Since the system is actually a recurrent neural network, it is computationally expensive to train. **InferSent** [7] is another sentence embedding solution based on neural networks. This solution builds a sentence embedding using a natural language interference dataset, and then uses these embeddings for other NLP tasks. This solution uses a supervised learning algorithm and performs better than skip-thoughts. Due to the way this network is trained, it is suitable for transfer learning. **Universal-sentence-encoder** [8] is another sentence embedding solution that is also suitable for transfer learning between different tasks. It is also trained on a natural language interference task. It uses a deep averaging network in encoder stage. After then, learned embeddings can be used in different NLP tasks.

2.1.1 sent2vec

Sent2vec [9] was developed by EPFML and is defined as an extension of fastText [10] and continuous bag of words (CBOW) [11] model for sentences. This is an unsupervised method that tries to minimize recreation error for a word in a given sentence. This model also works on character level, so it can also generate embeddings for sentences that include words it hasn't seen before. This model combines speed with accuracy, so we have selected this model in our research.

2.2 Sequence to Sequence Systems

Sequence to sequence [12], also known as encoder-decoder, is a neural network architecture that models both inputs and outputs as sequences. This architecture is developed first to be used in a translation task. While it did not beat the state-of-art system for the task it was developed, it beat any other pure neural network approaches. This architecture is robust enough that it has been used from 3D pose estimation [13] to stock prediction tasks successfully.

2.3 Summarization Systems

Text summarization is an NLP task that has a long history. These systems strive to keep key aspects of some text while limiting length. Some systems also strive to keep content interesting. Summarization is an area that has every day usage. Nowadays, most news sites show a small summary when linking to actual article page. When using social media, news outlets use a summary to catch attention and link to actual article. These summaries are already or can be written using a summarization system. Dataset in this area also follow this trend and are based on news articles or some meeting notes.

Summarization systems usually take one of two approaches. First approach is extractive. Extractive approach uses the exact sentence in source text and tries to select the subset of sentences that best cover the information in source text. Second approach is the abstractive one. That approach tries to generate sentences that cover the information in source text. With the rise of deep learning, summarization tasks are also seeing more research using neural networks. There are research in both extractive and abstractive neural network solutions to summarization task.

2.4 Text Similarity Metrics

There are a few established metrics for text similarity. Summarization tasks also use text similarity. In such tasks, there are usually several human written summaries

for each sample. Human written summaries are called **references** while summary produced by the system is called **hypothesis** or **candidate**.

2.4.1 BLEU

Bilingual evaluation understudy (BLEU) [14] is a widely used metric for text similarity. Despite being originally developed for machine translation tasks, BLEU is an established metric for text summarization. BLEU score as shown in Equation 21 is calculated using a modified n-gram precision and a brevity penalty, which is in turn calculated using candidate and reference lengths.

$$BLEU(x) = BP \cdot exp(\sum_{n=1}^{N}(w_n \cdot log(p_n))) \tag{21}$$

where

$$BP = \begin{cases} 1 & r < c \\ e^(1 - r/c) & r \geq c \end{cases} \tag{22}$$

Take "deal with people talking about you behind your back" as reference and "deal with people who always complain" as candidate for $N = 1$ case. The n-gram precision, shown as p_n in Equation 21, works out to be $3/6$ since only "deal", "with", "people" from candidate occur in reference. Following the calculation from Equation 22, brevity penalty is $e^(1 - 9/6)$, which is 0.6065. BLEU score for these two sentences which only the verb part is related, is 0.3033.

In the example where reference is "plant an herb pot" and the candidate is "grow herbs in containers", we see a different case. N-gram precision is 0, since there are no common words in reference and candidate. Thus, for these two sentences which contain very similar meanings, BLEU score is 0.

2.4.2 ROGUE

Recall-oriented understudy for gisting evaluation (ROUGE) [15] is another text similarity metric widely used for summarization. ROGUE was specially developed with text summarization in mind. It has a few versions. ROGUE-n versions are based on n-gram cooccurence. ROGUE-L is based in longest common subsequence. ROUGE-N is defined as:

$$\text{ROUGE-N} = \frac{\sum_{S \in ReferenceSummaries} \sum_{gram_n \in S} (count_{match}(gram_n))}{\sum_{S \in ReferenceSummaries} \sum_{gram_n \in S} (count(gram_n))} \tag{23}$$

Take "deal with people talking about you behind your back" as reference and "deal with people who always complain" as candidate for $N = 1$ case. ROUGE-1 is then $1/3$, when we follow Equation 23, since only "deal", "with", "people" from reference occur in candidate. In the example where reference is "plant an herb pot" and the candidate is "grow herbs in containers", we see a different case. ROGUE-1 is 0, since there are no common words in reference and candidate. Thus, for these two sentences which contain very similar meanings, ROGUE-1 score is 0.

2.5 Related Works

Since we are implementing a neural network based solution, we will be looking into such other solutions here. There has been several solutions using neural networks in summarization.

In 2004, Kaikhah [16] proposed a system to extractively summarize text. This system generates seven features for each sentence in the document. First four features represent placement of the sentence in the document. These feature encode if the sentence or paragraph is the first one, and its relative position. Rest of the features express sentence length and its content with respect to most used words in the document and the title. After these features are generated, a neural network is trained to classify the sentence that should appear in the summary. Then the system is modified to generalize and filter the sentences that belong to the summary. This approach mixes feature engineering with machine learning.

Rush et al [17] in 2015 proposed a neural network based system for abstractive sentence summarization with attention. Paraphrasing their abstract, they built a simple system that can scale with data and train end to end that showed significant performance gains. This system has a neural network to score next word given previous words as the generation step and has another encoder for information encoding. These two models are trained jointly and then beam searching is used to generate the final summary.

In 2016, Building on research from Rush et al, Chopra et al [18] built a system for abstractive sentence summarization with attention using *recurrent* neural networks. This system improves on Rush [2015] research. Main contribution of this model on top of Rush et al research is its use of a convolutional attention based recurrent encoder.

Also in 2016, Nallapati et al [19] proposed a system for document level abstractive summarization using a seq2seq network. This system outperformed previous state-of-art systems in datasets it was applied to.

Later on in 2017, Nallapati et al [20] released another system that improved on 2016 research. This newer system could be trained both extractively and abstractively. While abstractive system outperformed current state-of-art, extractive systems performance was better.

We use the research done by Koupaee et al [21] as our dataset and baselines. This research collected the wikihow dataset and applied several document summarization solutions to it.

CHAPTER 3

ARTIFICIAL NEURAL NETWORKS

Neural networks are mathematical constructs that solves many problems by approximations. The concept was coined by McCulloch and Pitts in 1943 [22]. The idea stems from emulating biological brain, thus named artificial neural networks. Since their inception, they were used for multiple classification and regression tasks.

A neural network poses a supervised or unsupervised learning problem as an optimization problem.This way, numerical optimization techniques can also be used in neural networks, greatly accelerating training speed.

3.1 Artificial Neuron

Neural constructs, such as a brain, a spinal cord or a ganglion consist of neuron connected to each other. Artificial neural networks also consist of artificial neurons, shown in Figure 3.2, connected to each other in some configurations. Some network types constrict how these connections are made and some network types have special connections.

These artificial neurons act on a set of inputs and produce an output. Generally, these inputs and outputs are real valued scalars, but there has been work on binary or complex values too. Artificial neurons take a weighted sum of their inputs and generate a single scalar value. As such, a neuron is modelled as

$$output = f_{activation}(\hat{input} \cdot \hat{w} + bias) \tag{31}$$

where $input$ is a vector of inputs, $output$ is the output value, w is the weighting vector parameter, $bias$ is the biasing parameter and $f_{activation}$ is the activation function

that defines how our neuron behaves. There are plenty of activation functions, each working better on some specific problems. Parameters w and $bias$ need to be adjusted if we want our network to perform well. Otherwise, the neuron would spit out random values. We call this adjusting of parameter **training**. There are numerous ways to train a neural network.

Artificial neurons and artificial neural networks are generally divided into layers. These layers are made of similar neurons that act the same way, but on different inputs and weights. Using this knowledge, we can merge multiple neurons calculations defined in (31) into a single calculation. Thus, a layer of artificial neurons are modelled as

$$\hat{output} = f_{activation}(\hat{input} * W + \hat{bias}) \tag{32}$$

where $input$ is a vector of inputs, $output$ is the output vector, W is the weighting matrix parameter, $bias$ is the biasing vector parameter and $f_{activation}$ is the activation function that defines how our layer behaves. This way, we substitute multiple vector calculations with a single matrix calculation, which speeds up calculations on physical devices.

All artificial neurons follow this pattern, but they perform differently depending on their activation function. Most common activation functions being $linear$, $sigmoid$ and $tanh$; there are numerous functions. We graph some of those activation functions in Figure 3.1.

This gives a simple summary of how artificial neurons work. We use these basic neurons and layers as building blocks and construct network that can accomplish complex tasks. We call these artificial neural network. A simple neural network consists of a layer of neurons connected to a result layer. This is a directed connection and data flows from input to result layer, which is also called a feed-forward neural network.

3.2 Artificial Neural Network

Artificial neural networks build on artificial neurons. In an artificial neural network, artificial neurons become inputs of other artificial neurons. As such, neurons are

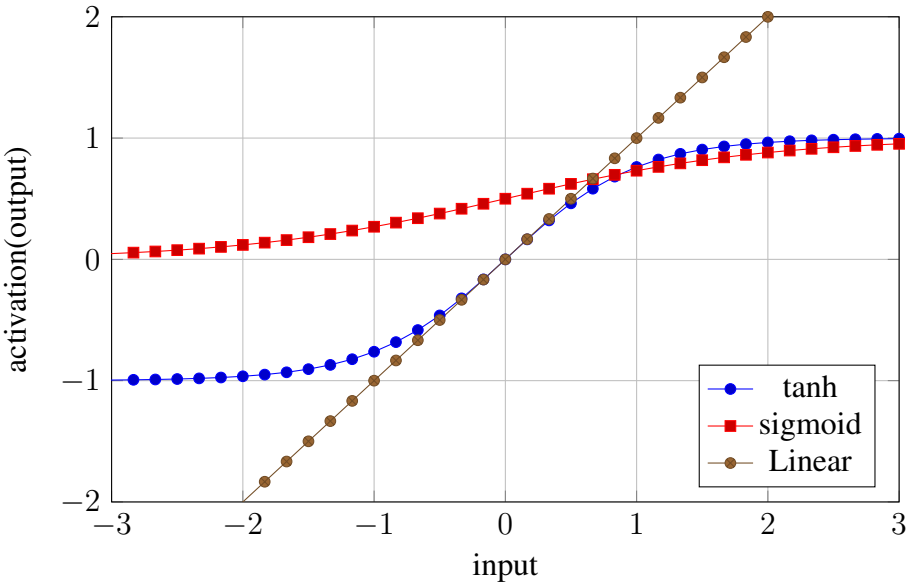


Figure 3.1: Elementary activation functions

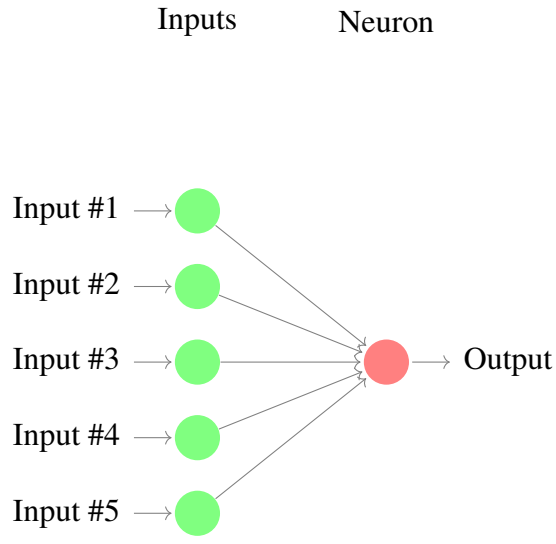


Figure 3.2: An artificial neuron

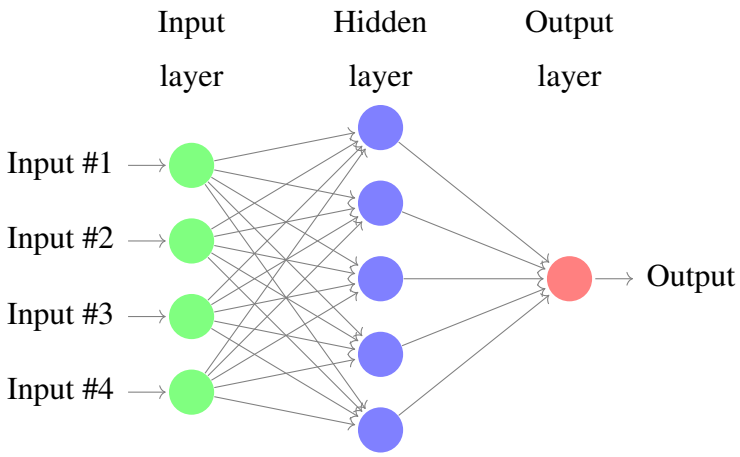


Figure 3.3: Simple neural network

grouped in layers in the network. So, first layer gets inputs from actual inputs and creates its output. This first layers output then becomes second layers input and so forth. This continues until the final output is calculated. The layers apart from the actual output layer are called **hidden layer**s. When we connect inputs, hidden layers and outputs as seen in Figure 3.3, we get an artificial neural network.

When used with a linear activation function, any number of layers can be reduced into a two-layer network. To overcome this limitation, we use non-linear activation functions. When used with non linear activation functions, multilayer neural networks with arbitrary number of layers are universal function estimators. In other words, there is a neural network for each and every function imaginable.

We can't use simple linear regression to train an artificial neural network. It becomes problematic to transfer errors on the current layer to previous layers. We need a way to propagate errors back to earlier layers. We do this with back propagation and gradient descent.

3.3 Gradient Descent and Back Propagation

To train neural networks more complex than simple perceptrons, we need a more powerful algorithm. The perceptron algorithm we used does not work with activation functions other than step functions. If we used the $-\mathcal{H}$ function as our activation

function, then perceptron algorithm would give totally wrong results. Furthermore, it does not work with more than two layers, there is no way to transfer the error on the result layer to previous layers. We need an algorithm to handle all kinds of activation functions and a way to propagate errors back to previous layers and use that errors on current layer to update weights. Lets first start with the optimization algorithm we are going to use.

The algorithm we are going to use is the gradient descent algorithm. Gradient descent is an algorithm to find local minima of a function. It uses gradient of the function in question to move closer to minima. Since gradient of a function gives us how much it changes in a given dimension, if we move our point by a negative amount of that gradient like Figure 3.4, we would arrive at a point of local minima.

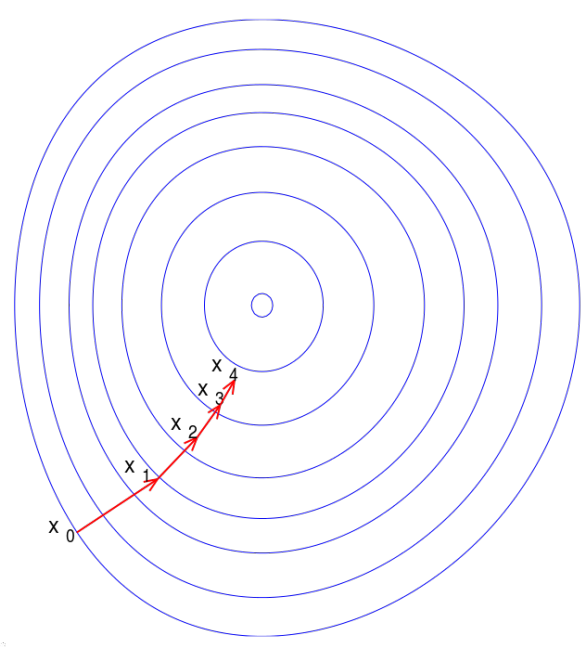


Figure 3.4: An example of gradient descent steps

By taking steps on the gradient, we do arrive at a local minima.

This version, Algorithm 1, being the simplest one, there are more advanced versions like momentum usage or adaptive versions. However, gist of the algorithm, usage of gradients to reduce error stays the same. One catch of using gradient descent is, you are guaranteed to reach "a" minima, but it is not guaranteed to be the global one.

Algorithm 1 Gradient descent algorithm

Require: F a function with a gradient

$X \leftarrow 0$

repeat

$X \leftarrow X - \alpha \cdot \nabla F(X)$ {move X towards minima by a small amount α}

until satisfied

Gradient descent itself works well for finding error and then updating final layer of a neural network. To train deeper networks, we need a method to propagate errors in the final layer back to previous layers. This is where back propagation comes into play.

In neural network design, we model the **loss function** between our predictions and ground truth. This functions result is called **error**. Using this error, we convert our learning task to an optimization task that targets to lower the error. However, error is only defined at output layer. We then take partial derivatives for each layer and parameter of our network with respect to this error. Using gradient descent, then we update weights of our network to minimize error.

Applying gradient descent and backpropagation to output layer is fairly easy. To backpropagate the error to earlier layer, we use *chain rule*. We apply chain rule to our error so we find out the partial derivative of the error with respect to the parameter we want to update. Then we update that parameter just like in regular gradient descent.

3.4 Deep Neural Networks

We call any neural network with more than two layers a deep neural network. Deep neural networks can learn very complex non linear functions. As of this writing, most of the state of the art solutions in image processing, image classification and natural language processing use deep neural networks.

Deep neural networks are powerful tools that can learn very complex and layered relationships. However, they are not without drawbacks. One drawback is, they need more data and time to train.

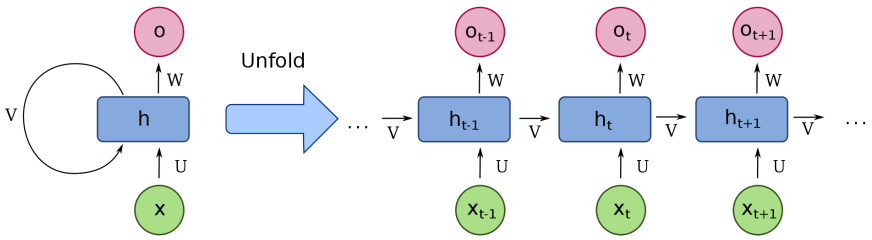


Figure 3.5: A fully recurrenct network unfolded in time [1]

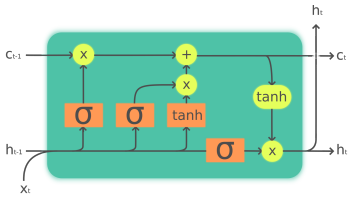


Figure 3.6: An LSTM network [2]

3.5 Recurrent Neural Networks

Recurrent neural networks (RNN) are a type of deep neural networks. They are used for modeling temporal sequences. In a simple recurrent neural network, the same layer is unfolded in time and applied to the input sequence. Figure 3.5 shows an unfolded RNN applied to an input. These are also called fully recurrent networks. Fully recurrent networks do not fare well in propagating errors that happen latter steps to earlier steps. As a result, the network shows a bias towards learning using its later inputs and makes it harder to learn long term dependencies. This is caused by something known as vanishing or exploding gradient problem.

Hochreiter et al [23] introduced Long Short Term Memory (LSTM) to solve vanishing gradient problem. LSTM solves this problem by allowing the error gradient to be propagated back to earlier steps without changing. An LSTM unit consists of a **cell** and three gates named **input**, **output** and **forget**. Cell is used for keeping an internal state and gates control when and how information moves from or into cell. Figure 3.6 shows LSTM cell and its workings.

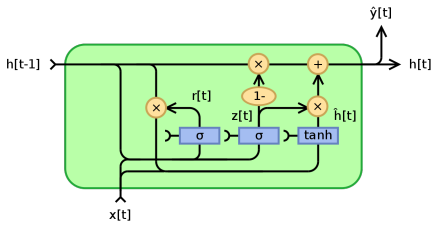


Figure 3.7: A GRU network [3]

Cho et al [24] proposed another unit with gates that is named Gated Recurrent Unit (GRU). GRU's are similar to LSTM but has fewer parameters to train. A GRU does not have a cell and only has **update** and **reset** gates. Performance can be compared to LSTM in many tasks. GRU's offer a good trade-off between ease and speed of training and performance when compared to LSTM's. Inner workings of a GRU cell is shown in Figure 3.7.

3.6 Embeddings

Embedding is a mathematical method of mapping an object in one domain to another object in another domain. Embedding is widely used as a tool in NLP tasks. There are supervised and unsupervised ways to generate embeddings. In this thesis, we are more interested in unsupervised embeddings. Unsupervised embeddings tend to be more generic and have a wider range of uses.

3.6.1 Word Embeddings

Word embeddings map a word to a multidimensional vector. Most of these mapping methods build upon vector space model, saying an object can be described by its relations to other objects. Word embeddings have been in use since 60s. However, they became popular with word2vec paper by Mikolov et al [25]. Word2vec allowed researchers to use huge datasets to train word embeddings in a short time.

The driving idea behind word embeddings is, words can be encoded using other words they are used together. Word2vec uses two methods for such use: first is CBOW [11], and the second is skip grams [25]. In CBOW method, current word is predicted

using neighboring words. In skipgram method, neighboring words are predicted using current word.

In 2017, Bojanowski et al released a word embedding solution named fasttext [10]. Fasttext introduced sub word embeddings that enabled it to generate an embedding even when it did not see that word before.

3.6.2 Sentence Embeddings

Sentence embeddings turn a sentence into a vector. They are useful in comparing sentences to each other, classifying sentiment or answering questions. These embeddings can be built upon words or characters[26]. Sentence embeddings can be built upon word embeddings, or they can be generated using source document and words in the sentence. Sentence embeddings provide a middle ground between abstraction of the sentence and the performance.

There are some advantages of using sentence embeddings over word embeddings. First, sentence embeddings can deal with polysemy better than word embeddings. A word embedding solution does not have access to information needed to distinguish between different meanings of a word, but sentence embeddings can use the whole sentence to figure out which meaning or use was intended. Second advantage is, when using word embeddings, word order has to be processed by our solution. When using sentence embeddings, word order is processed by sentence embedding solution. Most word embedding solutions use CBOW or skip grams and these methods do not capture word order at all. Furthermore, sentence embeddings can be trained on very large corpora. This way, sentence embeddings can learn relationships that are not easily seen in our documents. These relationships can help in our final task.

One can use a variety of ways to generate sentence embeddings. The most basic way of creating sentence embeddings involves using a vector space model. We can model each sentence using the words it has. Then, we can assign each word an index and set those indices to one when that particular word is used in the sentence. Number of different words possible poses a challenge, but we can just limit the number of words we include in our dictionary. For example, if we choose our dictionary as <*study,*

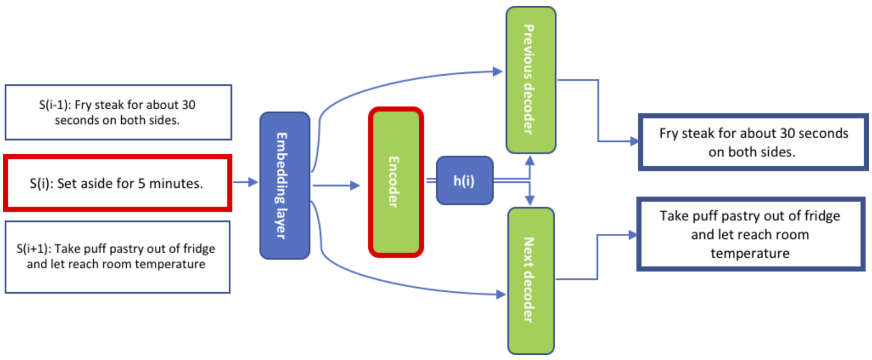


Figure 3.8: Skip-thought vectors training [4]

medicine, engineering>, *"He choose to study engineering"* becomes <*1, 0, 1*>, and, *"She feared if she didn't score well on tests, she would have to study medicine"* becomes <*1, 1, 0*>. This simple method does not always catch sentence meaning, but it is adequate for simple use cases.

Another way of obtaining sentence embeddings is combining word embeddings linearly. Most common is to use average pooling, i.e. taking the centroid of word embeddings. This actually generates a good baseline for several NLP tasks, however does not fare well in others, like sentiment analysis. There are several tricks to employ when doing this. One can remove stop words from corpora, combine word embeddings using weights, or use the sentences parse tree order to combine word embeddings [27].

The methods we discussed so far only use the sentence itself to generate sentence embeddings. There is another way to generate sentence embeddings that involve adjacent sentences. Skip-thought vectors [6] use this approach to generate sentence embeddings. This system takes words in current sentence and feeds them to an encoding RNN. Then, two other RNNs are run to recreate the previous and the next sentences using information from encoding RNN. This forces the network to learn ordering information between sentences and creates an information bottleneck that produces sentence embeddings. Sent2vec [9] also uses this approach. This system uses the vector space model and tries to recreate the *context* of the surrounding sentences. This makes the system more performant since there are no RNNs to run.

CHAPTER 4

DATASET

We use human description of tasks and summarize these. Using human descriptions poses a problem: it is very hard to parse descriptions into a standard format. Moreover, different people have different writing styles and use different words. To circumvent these, we use WikiHow that is community governed, and has a certain format every article follows.

4.1 WikiHow

WikiHow is a community governed wiki that has human readable instructions for tasks. Tasks range from very basic ones like "How to make peanut butter and jelly sandwich" to ambiguous, open ended ones like "how to trust yourself" to technical ones like "How to Connect Excel to an Oracle Database". WikiHow is governed by a community, so every instruction is refined by users multiple times. WikiHow also has guidelines that instructions follow. With these, WikiHow has a fairly consistent writing style with a defined instruction structure.

A WikiHow article consists of three parts:

- **Title:** A short description of task at hand.
- **Steps:** A list of text that, if followed in given order, should accomplish the task at hand.
- **Bolds:** These are a part of **steps**. Each bold summarizes a single step.

Unfortunately, WikiHow does not provide exports of their content. There exists some

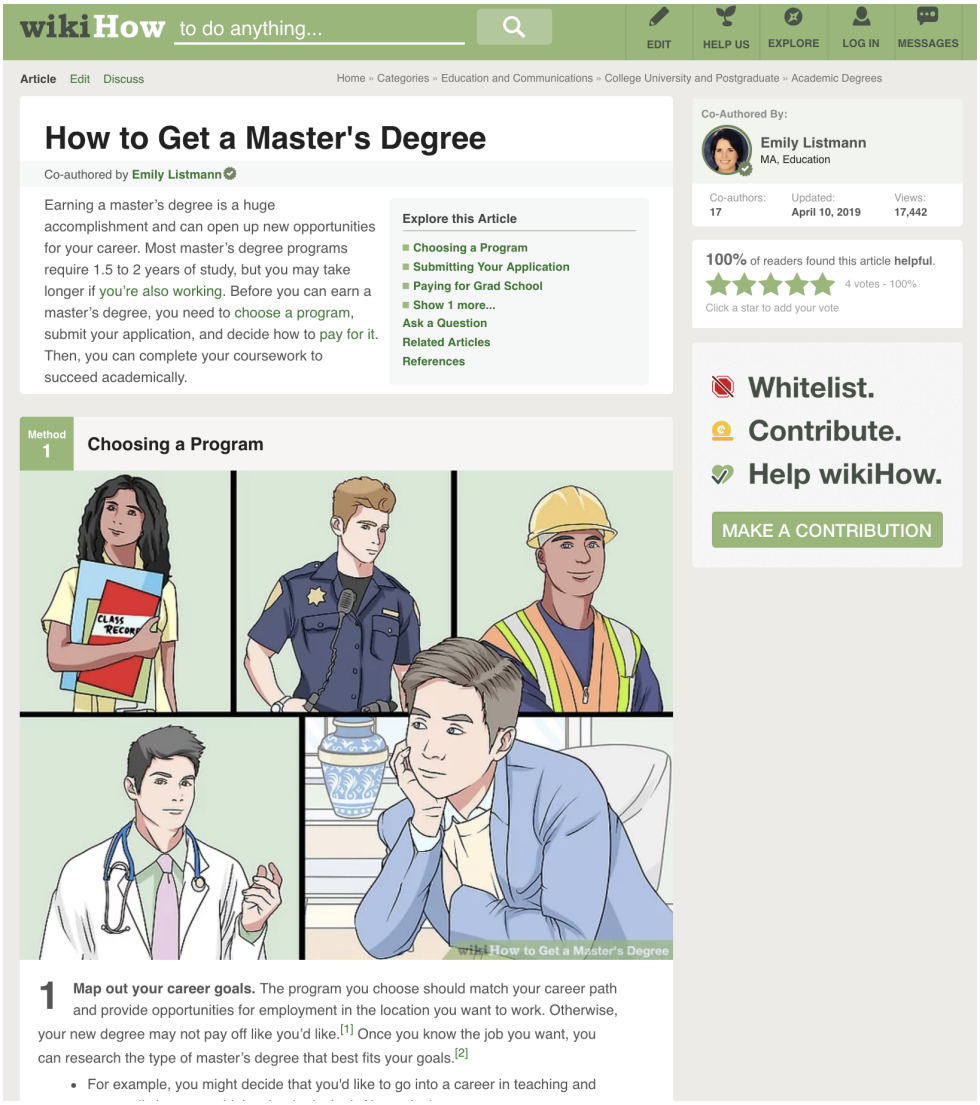


Figure 4.1: Screenshot of a WikiHow article. "How to get a Master's Degree" is the title. "Map out your career goals" is the first element in bolds and rest of that paragraph is the first element of steps.

dumps of WikiHow content in different formats. But when we started this research, most of them were old content since WikiHow is always in constant change. Thus, we started collecting instructions from WikiHow ourselves. While doing our research, we found another team [21] created a new dataset and produced some baselines, so we used that in our final solution.

Before we started using this dataset, we generated our own dataset. That was because when we started this research, there were no recent datasets on WikiHow. We used Python to crawl through top 100000 articles on WikiHow. Some articles actually contained more than one way to achieve the goal, so we futher divided these articles into seperate samples. Then we assigned Globally Unique Identifiers (GUIDs) to each sample and run our algorithms on that.

4.2 Dataset Details and Examples

Our dataset comes from research done by Koupaee et al [21]. This dataset comes in the comma seperated values (CSV) format. This file contains titles, instructions and summaries of instructions. We present some statistics about dataset in Table 4.1

Title is a short description of the task at hand. It is seen as the link text that points to actual article page and the header at article page. Title can also be seen as a high level summary since it encapsulates all task description. Bolds, or summary, is the next largest data. They provide a summary while not being detailed as instructions, also includes more information than title. Bolds can be found at the beginning of every step. Instructions make the bulk of article text. They provide detailed instructions to complete the task at hand. They are pulled from each step of the task description. We list some samples from WikiHow dataset in Table 4.2

Table 4.1: Dataset metrics

Part	Num sentences	Average words per sentence	Num unique words
Title	214293	8.09	41000
Summary	1552195	9.58	148430
Instructions	5142664	20.97	625181

Table 4.2: WikiHow sample examples

Title	Instructions	Summary (Ground truth)
How to Order Decimals from Least to Greatest1	The quickest way to determine the least or greatest number is to compare their whole numbers. If one number has a larger whole number than the others, it is automatically the greatest number. If one number has a smaller whole number than the others, it is automatically the least number. For example, ...	Convert both percentages to decimal., Multiply the decimals., Multiply the product by 100 to get a percent., Add a percent (%) sign so others know that the number is being compared to 100. , Rewrite both percentages as fractions with a denominator of 100., Multiply the fractions., ...
How to Administer a Flu Shot	The term "pre-filled vaccine syringes", in this case, is not referring to influenza vaccine syringes specifically manufactured as individual doses by the vaccine manufacturer, and, instead, refers to multiple, individual dose syringes filled from either single-dose or multi-dose ...	Check your medicine., Prepare the powder medicine (skip this step if you have liquid medicine)., Get the vial ready., Get the syringe ready., Insert the syringe into the vial., Take the medicine from the vial., Remove air bubbles., Remove the syringe from the vial., Locate an area for injection., ...

CHAPTER 5

SENTENCE VECTOR APPROACH

In this thesis, we summarize instructions for tasks. Our inputs are instructions content and our output is its summary. Since both are a sequence of tokens (i.e. words), we are converting a sequence into another sequence. This type of problems include machine translation, speech to text and text generation [28]. These are well researched areas and there exist several great solutions to the problem. One of these solutions is Seq2Seq networks.

5.1 Seq2Seq Networks

We convert instructions content to summaries with a Seq2Seq [12] network. Seq2Seq metworks are also called encoder-decoder networks. A Seq2seq network consists of two networks working together to turn a sequence into another sequence. Encoder network works on input and turns it into a lower dimensional data that we will call **state**. Then, decoder part of the system uses that **state** to construct another sequence, sometimes with some additional inputs.

An encoder-decoder network creates an information bottleneck by limiting the number of parameters that can pass between two networks, i.e. the size of state variable. Seq2Seq architecture aim to encode input data to encoder network with as little dimensions as possible such that decoder network can still perform well with given **state**.

While we create an information bottleneck, this may cause problems in some cases. For example, in machine translation, transferred data alone may not be enough to

actually select the correct word in a given sentence. In that case, we insert another information channel named **attention**. Attention lets some internal state from encoder to be passed to decoder stage. This improves decoder performance by a large margin.

5.2 Data Preparation

We do not do much preprocessing to our dataset. The few preprocessing steps we apply are:

- removing unnecessary whitespace,
- standardizing quotes and symbols (numbers are left as is),
- turning everything lowercase.

We use NLTK [29][1]'s word tokenizer to do most of these. Then we turn these words into tokens and use them.

We can get by doing very small preprocessing because of our selection of input transformation! In a typical implementation of Seq2Seq network, the input for our network would be tokenized words or characters. While we use tokenized words for decoder input, we use a sentence embeddings for encoder stage input. This lets us gain some advantages.

5.3 Sentence Embeddings as Input

A regular application of seq2seq architecture would use word tokens for both encoder and decoder stage. Then these tokens would run through an embedding layer to become a vector and get processed.

Instead of doing this, we decided to leverage one of the general purpose sentence embedding solutions. These solutions are generated using much more computation and data we can hope to use, they are multi-purpose and generally fare well in other

[1] https://nltk.org/

benchmarks. If we used word tokens (and word embeddings like word2vec), our system would have to learn to distinguish between different meanings of the same word.

By using a sentence embedding, we are also showing that it is not necessary to know the exact words used to create a summary of a text. A sentence embedding that captures the generic context of a text is also enough to create an adequate summary.

5.4 Our Architecture

Our architecture is a standard seq2seq architecture. We run our encoder network as a GRU and then use its outputs and final hidden state as inputs to decoder stage. Decoder stage outputs tokens which is then converted to words. We concatenate these word and use that as our final results.

5.4.1 Inputs and Outputs

By definition of the problem, our inputs are task definition and our outputs are summary text. These correspond with **steps** as input and **bolds** as output. Our system parses inputs as sentences and outputs as words, then feeds these to our neural network. We show inputs, outputs and how they are connected in Figure 5.1, with some examples.

With sentence vector approach, our inputs differ from a standard seq2seq network. A vanilla seq2seq network would take words as input. Then, using an embedding layer, these words would be converted to a multidimensional vector. However, some implementations move this embedding layer out of training phase and use a standard word embedding system like word2vec or GloVe. This gives implementations a boost since embeddings start with meaningful initials instead of random values.

We take this approach one step further and feed sentence vectors directly. Our inputs are vectors of sentence embeddings and our outputs are one hot encoded word tokens. For decoder stage embedding, we also get our weights from our sent2vec solution. This way, we completely externalize input transformations. Our encoder solely works

on sentence embeddings. Decoder part utilizes word embeddings that are derivated from sentence embeddings.

5.4.2 Data Preparation

We first apply sentence tokenization on input and word tokenization on output. Input sentences are then converted to sentence vectors using EPFMLs sent2vec solution. Outputs, i.e. words are then counted to find most common ones. Then we select most common 25000 words and assign then ids. We have three special tokens/ids; a start of sentence token, an end of sentence token and a placeholder "unknown" token. The unknown token replaces all other words that did not make it into the top 10000 words.

We convert output sequence to numbers using ids assigned to words. The words that are not assigned an id are converted to "unknown" token. Output sequence is then prepended with a start of sentence and appended an end of sentence token. We also take advantage of our embeddings and set decoder input embeddings to sent2vec results of those single words.

5.4.3 Network Architecture

In this research, we used a simple architecture. This architecture is a modified seq2seq network. While it is a simple architecture, it is fairly powerful. The architecture seen in Figure 5.2 is the actual architecture used in our experiments.

Our network is composed of two sub networks: an encoder and a decoder. Decoder is a two layer RNN network with ReLU nonlinearity after each layer. Note that we do not need an embedding layer since our encoder inputs are already sentence vectors of . This network has two outputs. One is last state of RNN after all inputs. We call this a task vector. The other one is the results produced by encoder while processing the input sequence. While this is not nearly as important, using the output history as attention input increases performance. We pass both of these to our decoder network.

The decoder network consists of an attention layer, an embedding layer and an RNN, followed by a linear layer and a ReLU unit. We use attention model proposed by

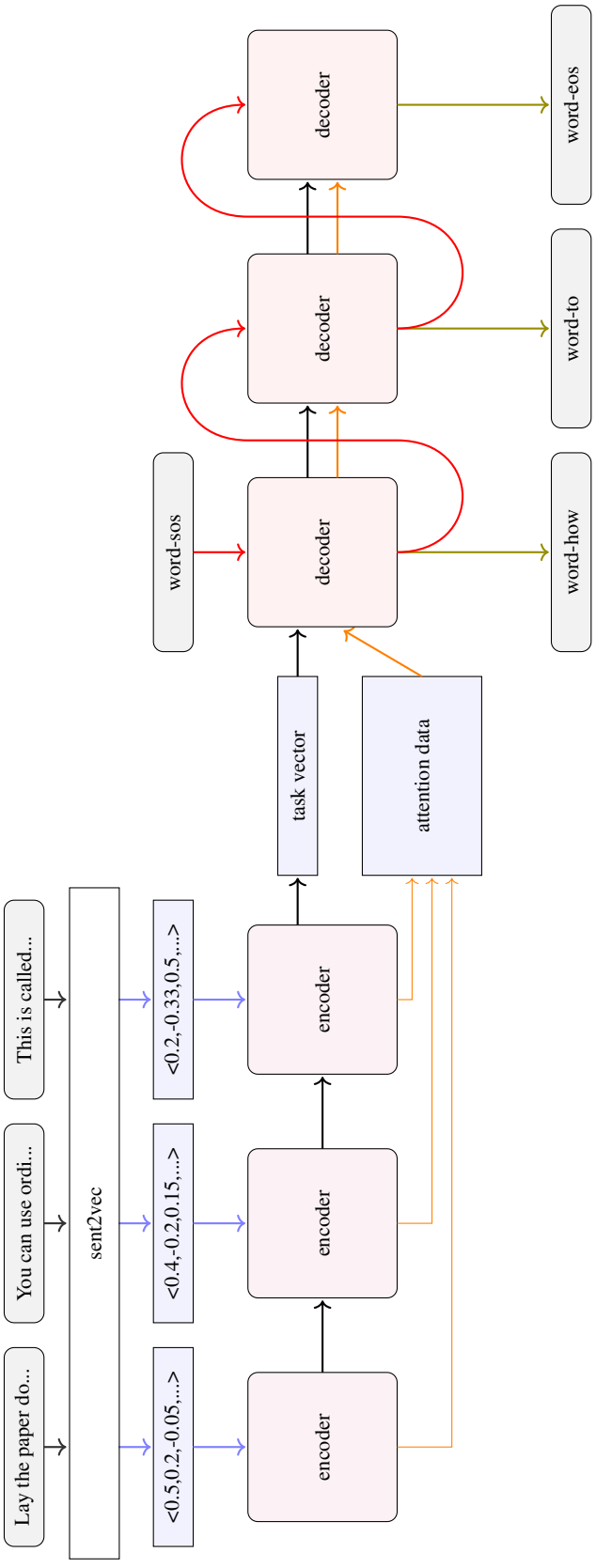


Figure 5.1: Seq2seq network and its inputs outputs

Luong et al[30] in our decoder. This network starts with final output and output history from encoder network. Final output is set at starting hidden state of RNN and output history is fed to attention network. This networks input is a series of word tokens and its out is a new word token. We start by feeding start of sentence token and get first word of the sentence. Then, keeping internal state of RNN, we feed the first result and generate second word of our output and so forth. This way, we generate new words until we see an end of sentence token of we exhaust a set number of words.

5.4.4 Training Tricks

We employ a few training tricks to keep our network from overfitting. We apply gradient clipping and batching to keep our network from overfitting.

During our experiments, we found out that gradient clipping had an important effect on final error rates. Gradient clipping is achieved by limiting the final gradient of the network to a set value. If magnitude of the output layer exceeds this set value, then the gradient is scaled so its magnitude is equal to that set value. We show the effects of gradient clipping in Figure 5.3. When applied, gradient clipping keeps network from performing sudden big jumps. This in turn keeps hystherisis to a minimum.

Batching is also applied when possible. Batching in our case does not have a very significant effect like gradient clipping. However, it increases performance. PyTorch supports batching in its core, so the workload needed on our side to support batching is pretty low. We only handle batching in training sample creation and masked negative log likelihood (NLL) loss calculation.

5.5 Task to Title Network

Before we found our current dataset, we also tried to summarize the steps down to article title. This is a very high level summary and decoder part has less things to learn.

This version of the network is virtually identical to version for long summaries (**bolds**).

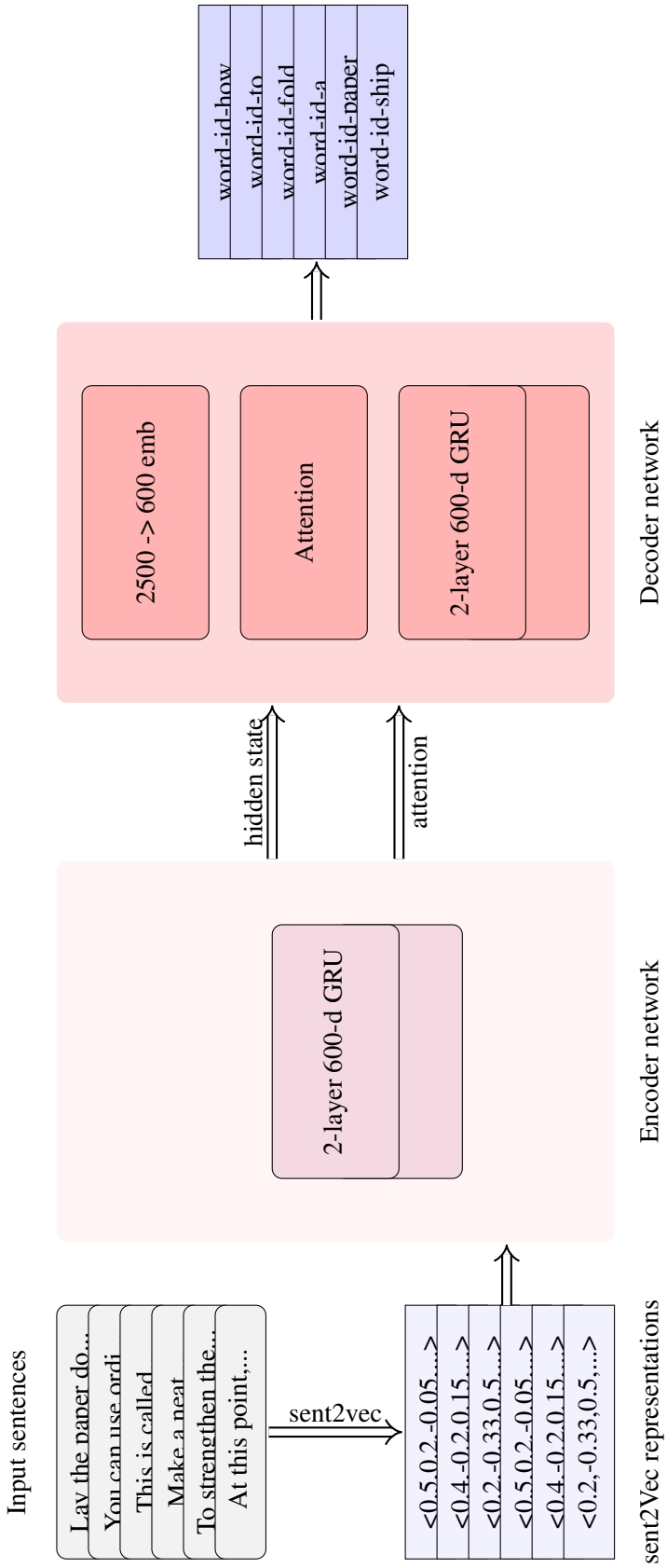
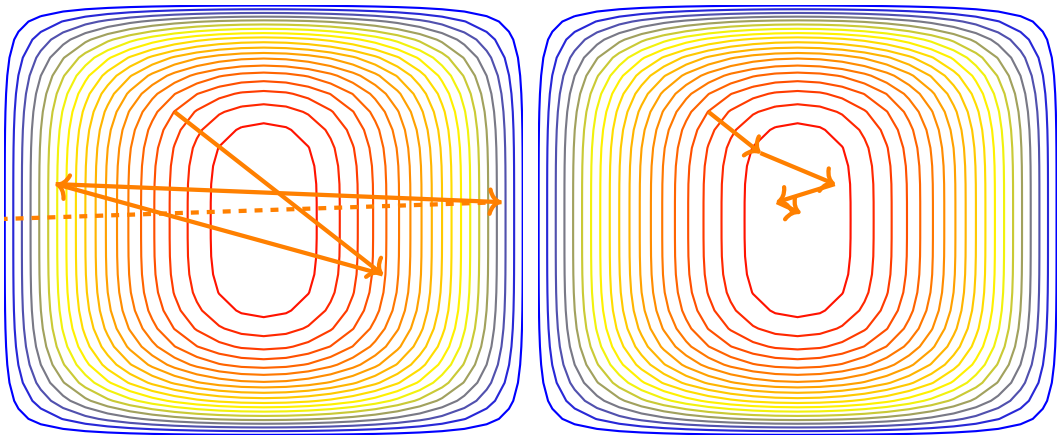


Figure 5.2: Network architecture



(a) Without gradient clipping (b) With gradient clipping

Figure 5.3: Effect of gradient clipping

Only hyperparameters are changed. Number of output words are changed to acommodate lower number of total different words and learning rate parameters are adjusted. In fact, all network code is reused without any changes.

Output processing was changed to acommodate multiple sentences being generated. We remove any word received after an end of sentence token was generated. Title network is run without that, since we only create a single sentence.

CHAPTER 6

EXPERIMENTS AND RESULTS

In our research, we aimed to use sentence embeddings instead of word vectors. There are other research that successfully uses word embedding word summarization. By using sentence vectors, we both hide actual word information and lessen the RNN sequential data.

We did two experiments, one being a standard summarization task, which tries to summarize the instruction down to bolds and the other one trying to summarize instructions dows to title. The last task does not fare well in many standard metrics used in text summarization since these metrics are based on exact or near exact matches while our system sometimes produces summaries with.

6.1 Summarization Task

Text summarization is a popular task in natural language processing. This is a well researched task with several great solutions over the years. With developments in deep learning, this field also saw a renewed interest, especially with usage of RNN's and embeddings. Summarization is fairly standard and well defined task. The task is generally performed on a dataset consisting of news articles. We report our results in BLEU [14] and ROGUE [15] metrics when available.

Our task uses wikihow dataset [21] and trains a summarizer to produce **bolds** from **steps**. Since this is not a popular dataset yet, there is not much other research we can directly compare to. We use baselines from Koupaee et al. [21]. We train our network to summarize the dataset.

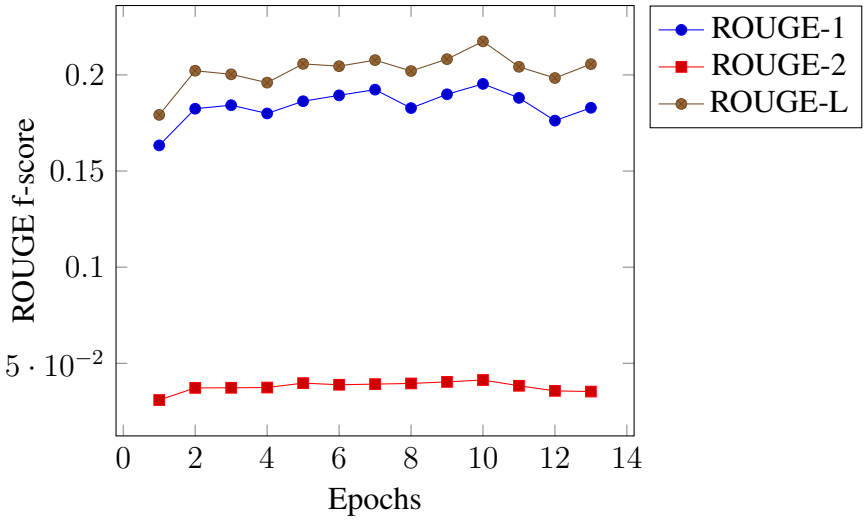


Figure 6.1: Summarization task ROUGE scores

Table 6.1: Comparison with other solutions(scores are f-scores)

Solution	ROUGE-1	ROUGE-2	ROUGE-L
Our solution (Seq2Seq with sent. emb.)	19.53	4.13	21.74
Seq2Seq with attention	22.04	6.27	20.87
TextRank	27.53	7.4	20.00
Pointer generator+coverage	**28.53**	**9.23**	**26.54**
Lead-3 baseline	26.00	7.24	24.25

We train our network with EPFMLs sent2vec 600 dimensional sentence vectors. Both encoder and decoder uses two layer GRU's with 0.4 dropout. We use PyTorch's Adam optimizer with a learning rate of 0.001 and betas of 0.9 and 0.999. We clip gradients to a length of 10.

Network is trained for 10 epochs and then results are reported. Figure 6.1 has a graph of several training and evaluation metrics by training batch.

We demonstrate our results in Table 6.1. Our solution shows a similar result with regular seq2seq solution using word embeddings. We are able to produce similar results using less information and using an abstractive neural network.

Some examples of ground truth and our summaries are presented in Table 6.2.

Table 6.2: Example results

1	list the author 's name . state the title of the book . add publication details . cite an e-book from a library database . cite a book you found on the web .	start with the author 's name . add the title of the book . add the edition of the book . add the edition of the book . add the edition . add the edition .
2	market yourself and your talent . be persistent in your marketing . be highly organized . connect with people . negotiate contracts that benefit all parties .	find a talent . sell your work . sell your work . sell your work . sell your work . sell your work .
3	learn a few at a time . practice switching between these chords . get a teacher . print out chord diagrams . learn simple songs .	learn the basics of the song . practice the piano . practice the chord . try to find out what you are going to do .
4	remove debris from the drain weekly . use a <unk> drain cleaner monthly . clean monthly with household products .	clean your shower curtain . use a plunger to remove buildup . use a plunger to remove buildup .

From these results, it can be seen that we indeed can learn to summarize an article without actually knowing which words were used. Some examples also show that our system uses synonyms and rewords some summaries while keeping the meaning same.

6.2 Title Summarization Task

This task is also a summarization task but there is no other directly comparable research done on the same dataset. Instead of trying to summarize to **bolds**, we try to summarize **steps** to **title**s in this task. This is a very high level summary and does not capture the details in **steps**. In fact, since there is more than one way to reach the same result, and for some titles there is more than one way in WikiHow, this task poses a different challenge.

We train our network with EPFML's 600 dimensional sentence vectors. Our network uses two-layer GRU's for both encoder and decoder with 0.4 dropout. We limit our vocabulary to 8000 words and max title word length to 15 words. We again use PyTorch's Adam optimizer with a learning rate of 0.001 and betas of 0.9 and 0.999. We clip gradients to a length of 10. Unlike our normal summarization task, we use our own dataset we scraped from wikihow.

When preparing test and training batches for this task, we used two different approaches. One is, we randomly divide test and training batches. The second approach is, we make sure every title that appears in test set also appears in training set, albeit with a different **steps** list. We show results from first approach in Table 6.3 and results from second approach in Table 6.4.

Network is trained for 40 epochs and results are reported. A plot of evaluation metrics can be seen in Figure 6.2 for first approach and Figure 6.3 for second approach. From these grahps, we see that both systems manage to achieve similar performance for BLEU scores. However, when checking their output, the system with modified splits produces better results albeit marginally.

Table 6.3: Example results with regular strategy

#	Ground truth	Our result
1	soothe your legs after stress or long walk <eos>	get rid of leg pain <eos>
2	skip a class in middle school <eos>	skip a class in high school <eos>

Table 6.3: Example results with regular strategy (continued)

3	stay organized <eos>	organize your day <eos>
4	start a pet shop <eos>	choose a country <unk> <eos>
5	study for a science exam <eos>	pass the bar exam <eos>
6	tell if your guinea pig is pregnant <eos>	tell if your guinea pig is pregnant <eos>
7	teach your child to read <eos>	teach reading <eos>
8	talk your boyfriend out of breaking up with you <eos>	break up with your boyfriend <eos>
9	tie a headscarf <eos>	tie a cute girl <eos>
10	tell if a boy loves you <eos>	determine if a person is interested in you
11	tell the difference between planets and stars <eos>	find planets in the night sky <eos>
12	treat tick bites <eos>	remove a tick <eos>
13	treat a caterpillar sting <eos>	remove a tick <eos>
14	use lemon juice to lessen acne and heal acne scars <eos>	use a lemon to lighten acne <eos>
15	use the internet to promote your business <eos>	make money online <eos>
16	write your first program in java <eos>	open a php database <eos>
17	write a comedy sketch <eos>	make a joke <eos>
18	be healthy <eos>	lose belly fat <eos>
19	change your start page on mozilla firefox <eos>	change your home page on safari <eos>
20	get california unemployment benefits <eos>	apply for unemployment compensation benefits <eos>
21	get pen stains out of clothing <eos>	remove pen ink from clothes <eos>
22	install a bathtub <eos>	replace a bathtub in dishwasher

Table 6.3: Example results with regular strategy (continued)

23	prevent emotional abuse <eos>	recognize an abusive relationship <eos>
24	start a conversation with a stranger <eos>	make friends at a new school <eos>

Table 6.4: Example results with duplicates strategy

#	Ground truth	Our result
1	make a personal minecraft server <eos>	make a personal minecraft server <eos>
2	increase estrogen <eos>	increase testosterone levels <eos>
3	break into a car <eos>	open a locked door <eos>
4	deal with people talking about you behind your back <eos>	deal with people who always complain <eos>
5	copy favorites <eos>	download bookmarks <eos>
6	come up with a nickname <eos>	name a main character <eos>
7	deal with religious people if you are an atheist <eos>	deal with someone who is always right <eos>
8	apply for food stamps in the us <eos>	apply for welfare <eos>
9	plant an herb pot <eos>	grow herbs in containers <eos>
10	write a graduation thank you speech <eos>	write a graduation speech <eos>
11	use linkedin <eos>	create an online linkedin
12	tile a bathroom floor <eos>	install a brick floor <eos>
13	convert inches to feet <eos>	calculate your feet by hand <eos>
14	get permanent marker stain out of hardwood flooring <eos>	remove floor stains <eos>

Table 6.4: Example results with duplicates strategy (continued)

15	make a storm <unk> helmet from a milk jug <eos>	make a storm 's <unk> costume <eos>
16	get rid of rats without harming the environment <eos>	get rid of rats <eos>
17	calculate the circumference of a circle <eos>	calculate the volume of a <unk> prism <eos>
18	sharpen a knife <eos>	sharpen a kitchen knife <eos>
19	use a coffee maker <eos>	make coffee with a coffee maker
20	block and unblock internet sites (on a mac) <eos>	block unwanted site from your computer <eos>
21	get dogs to gain a healthy weight <eos>	increase appetite in dogs <eos>
22	make your hamster trust you <eos>	hold a hamster <eos>
23	replace a ceiling fan pull chain switch <eos>	replace a tail light in a kitchen sink <eos>
25	get a newborn to sleep through the night <eos>	stop breastfeeding at night <eos>
26	relieve itchy hands and feet at night <eos>	treat an eczema flare up <eos>

From these results, it can be seen that we indeed can learn to summarize an article without actually knowing which words were used. Some examples also show that our system uses synonyms and rewords some summaries while keeping the meaning same.

6.3 Discussion

We present many examples in each task. There are many interesting and uninteresting examples, but we can classify them in four categories. Namely, good summaries with

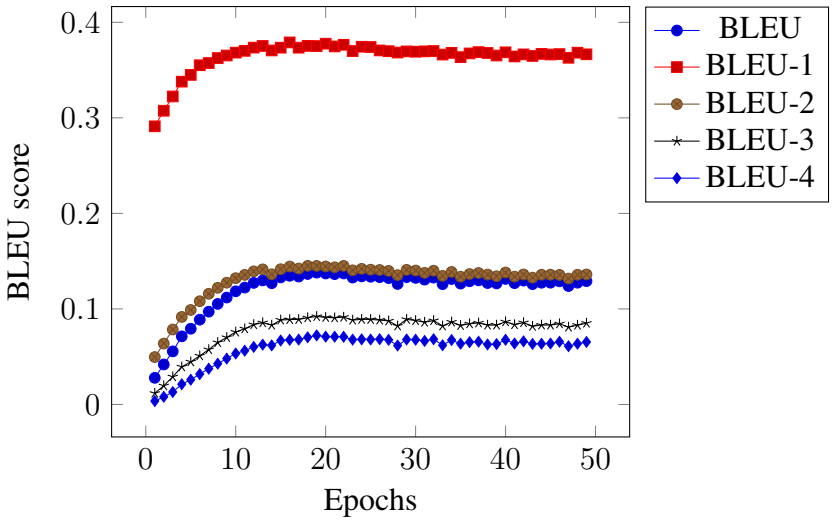


Figure 6.2: BLEU scores for normal split

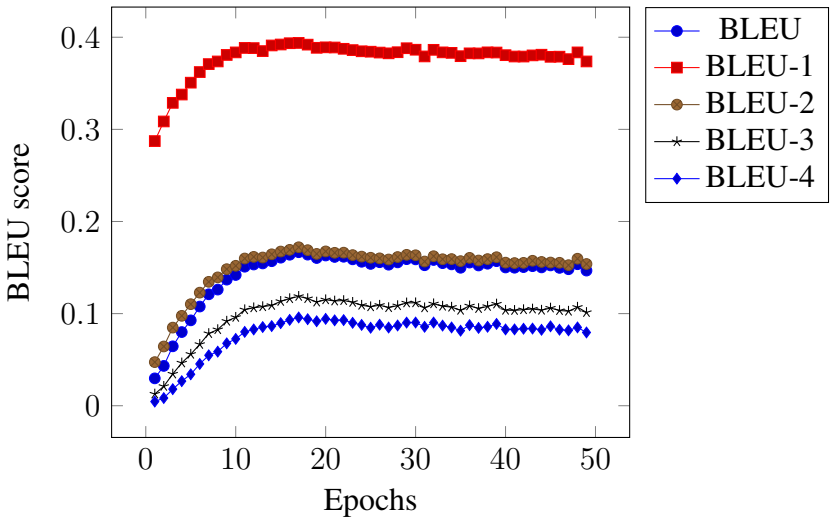


Figure 6.3: BLEU scores for modified split

good scores, bad summaries with bad scores, good summaries with bad scores and bad summaries with good scores.

Good summaries with good scores and bad summaries with bad scores are easiest examples to explain. Take *"make a personal minecraft server"* as an example, where we manage to completely recreate the ground truth. Another example would be *"write a graduation thank you speech"* vs. *"write a graduation speech"*. These are good summaries with good BLEU and ROUGE scores. They capture the original meaning well and also match the words used. On the other hand, take *"start a pet shop"* and *"choose a country <unk>"*, or *"copy favorites"* vs. *"download bookmarks"*. These have bad summaries with low BLEU and ROUGE scores. They do not capture the original meaning and do not reuse the same words with ground truth. In these examples, we can say that our model could or could not learn the underlying information.

Then there are bad summaries with good scores. Examples would be *"tie a headscarf"* vs. *"tie a cute girl"*, or, *"increase estrogen"* vs. *"increase testosterone levels"*. While these match many of the words used, the meaning is irrelevant or even completely opposite as can be seen in the latter example. These examples show that our model was not able to extract the underlying information.

The most interesting case is good summaries with bad scores. In these cases, our system generates summaries with a very similar meaning to ground truth. An example would be *"plant an herb pot"* vs. *"grow herbs in containers"*. These two sentences have very similar meanings while managing to get 0 scores from both BLEU and ROUGE. Another example is *"stay organized"* vs. *"organize your day"*. This example also manages to get 0 scores, while correctly summarizing the source document.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this thesis, we focused on generating task summaries from WikiHow dataset. We opted to perform abstractive summarization using sentence embeddings. While there are several previous research done using word embeddings to perform abstractive summarization using neural network, there is none that used sentence embeddings.

In our standard summarization task, we show that sentence embeddings are viable alternatives to word embeddings for abstractive summarization. Our ROGUE scores are lower than corresponding seq2seq model with attention using word embeddings, however, using sentence embeddings instead of word embeddings, we are able to reduce input size by near an order of magnitude. This order of magnitude processings gain can instead be used to process a bigger dataset. It is also worthwhile to note that this system uses beam search which has better performance than the straightforward greedy search our work uses. We believe with a beam searching decoder, our systems performance would rise.

Our task to title task results show that while we have low BLEU scores, we still are able to generate summaries that has the same or a similar meaning. We have two possible explanations for this behaviour. First one is the case that our decoder memorizes some common phrases that are likely to occur, or even complete titles, and acts as a classifier that outputs the memorized sequence based on input from encoder. Seeing that network performance does not differ greatly between our two different splitting strategies, we believe this explanation is not the case. The other explanation is that our system is actually capable of encoding task information a vector and then recreate the titles from that vector.

For future work, one of the low hanging fruits is changing our sentence embeddings. The embeddings we use are trained on wikipedia unigrams. Pagliardini et al shows in their paper [9] that there are other models that perform better. One alternative is using the same embedding solution with another model, for example bigrams trained on twitter data. Another alternative would be completely changing the embedding solution we use.

As the next step, we think jointly training regular summarization task with task to title task might yield interesting results. For this network, we could share the encoder and have seperate decoder stages for title and regular summarization tasks. WikiHow somehow has multiple articles for the same title. They explain different ways to reach the same results. A joint trained network would learn to generalize the task at hand while also encoding nuances of different ways of doing the same thing.

Another future work would be improving underlying architecture. For starters, we can implement a beam searching decoder. It is important to state that all other models that we compare our research to use a beam searching decoder. As a bigger change, we can start using some newer architectures like self attentive networks. If we can use a better architecture and sentence embeddings, we can lower the amount of recurrent cycles we need and use these cycle gains to process a bigger dataset.

REFERENCES

[1] "Recurrent neural network - Wikipedia." https://en.wikipedia.org/wiki/Recurrent_neural_network. Accessed: 2019-03-22.

[2] "Long short-term memory - Wikipedia." https://en.wikipedia.org/wiki/Long_short-term_memory. Accessed: 2019-03-22.

[3] "Gated recurrent unit - Wikipedia." https://en.wikipedia.org/wiki/Gated_recurrent_unit. Accessed: 2019-03-22.

[4] "Building recipe skill representations using skip-thought vectors." https://sourcediving.com/building-recipe-skill-representations-8a6e4c38ae6c. Accessed: 2019-09-01.

[5] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[6] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler, "Skip-thought vectors," *arXiv preprint arXiv:1506.06726*, 2015.

[7] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes, "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data," *arXiv e-prints*, p. arXiv:1705.02364, May 2017.

[8] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. St. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil, "Universal Sentence Encoder," *arXiv e-prints*, p. arXiv:1803.11175, Mar 2018.

[9] M. Pagliardini, P. Gupta, and M. Jaggi, "Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features," in *NAACL 2018 - Conference of the North American Chapter of the Association for Computational Linguistics*, 2018.

[10] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv e-prints*, p. arXiv:1301.3781, Jan 2013.

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.

[13] M. Rayat Imtiaz Hossain and J. J. Little, "Exploiting temporal information for 3D pose estimation," *arXiv e-prints*, p. arXiv:1711.08585, Nov 2017.

[14] K. Papineni, S. Roukos, T. Ward, and W. jing Zhu, "Bleu: a method for automatic evaluation of machine translation," pp. 311–318, 2002.

[15] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, (Barcelona, Spain), pp. 74–81, Association for Computational Linguistics, July 2004.

[16] K. Kaikhah, "Automatic text summarization with neural networks," pp. 40 – 44 Vol.1, 07 2004.

[17] A. M. Rush, S. Chopra, and J. Weston, "A Neural Attention Model for Abstractive Sentence Summarization," *arXiv e-prints*, p. arXiv:1509.00685, Sep 2015.

[18] S. Chopra, M. Auli, and A. M. Rush, "Abstractive sentence summarization with attentive recurrent neural networks," pp. 93–98, 01 2016.

[19] R. Nallapati, B. Zhou, C. Nogueira dos santos, C. Gulcehre, and B. Xiang, "Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond," *arXiv e-prints*, p. arXiv:1602.06023, Feb 2016.

[20] R. Nallapati, F. Zhai, and B. Zhou, "Summarunner: A recurrent neural network based sequence model for extractive summarization of documents," 2017.

[21] M. Koupaee and W. Y. Wang, "WikiHow: A Large Scale Text Summarization Dataset," *arXiv e-prints*, p. arXiv:1810.09305, Oct 2018.

[22] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, Dec 1943.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.

[24] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *arXiv e-prints*, p. arXiv:1406.1078, Jun 2014.

[25] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," *arXiv e-prints*, p. arXiv:1310.4546, Oct 2013.

[26] A. Radford, R. Jozefowicz, and I. Sutskever, "Learning to Generate Reviews and Discovering Sentiment," *arXiv e-prints*, p. arXiv:1704.01444, Apr 2017.

[27] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," *EMNLP*, vol. 1631, pp. 1631–1642, 01 2013.

[28] I. Sutskever, J. Martens, and G. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, (USA), pp. 1017–1024, Omnipress, 2011.

[29] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, (Stroudsburg, PA, USA), pp. 63–70, Association for Computational Linguistics, 2002.

[30] M.-T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," *arXiv e-prints*, p. arXiv:1508.04025, Aug 2015.