Abstractive Text Summarization with Neural Networks

Shubham Mukherjee
University of Massachusetts Amherst
shubhammukhe@cs.umass.edu

Abhyudai Nouni
University of Massachusetts Amherst
anouni@cs.umass.edu

Kartik Chhapia
University of Massachusetts Amherst
kchhapia@cs.umass.edu

Abstract

Abstractive Sentence Summarization generates a shorter version of a given sentence while attempting to preserve its meaning. In this research project, we aim to generate a news headline for various news articles. We do this by casting abstractive text summarization as a sequence-to-sequence problem and employing the framework of Attention Encoder-Decoder Recurrent Neural Networks (RNNs) to this problem. We also compare our model's performance with select extractive models and analyze and contrast the performance, deriving interesting insights into the efficiency of neural network based summarization.

1. Introduction

Every day, people rely on a wide variety of sources to stay informed – from news stories to social media posts to search results. Being able to develop Machine Learning models that can automatically deliver accurate summaries of longer text can be useful for digesting such large amounts of information in a compressed form can be a very useful.

Text summarization can also serve as an interesting reading comprehension test for machines. To summarize text well, machine learning models need to be able to comprehend documents and distill the important information, tasks which are highly challenging for computers, especially as the length of a document increases.

Text Summarization is the task of taking an input text and creating a supplementary text which contains the same meaning as the original text, but in lesser words. This is essentially the task of being able to present the most important pieces of information from the text, in other words, the gist or the crux of the whole text.

In Natural Language Processing, there are three primary approaches to automatic text summarization. The first is reducing the length of the input text by simply deleting words in the original text, but preserving word order. This is known as deletion or compressive summarization. The second type of summarization is called extractive, and refers to generating a summary using words in the input text only, but without regard to word order. Finally, the third and most general type of summarization is abstractive, which produces a summary using any words. The output in abstractive text summarization is not at all constrained by the input text. We personally prefer the abstractive approach because it produces a condensed version of an input text that uses words which are not just present in the original text, capturing the core meaning of the input.

Automatic text summarization is inherently hard because it traditionally requires semantic understanding and grouping of content using world knowledge. In the past, researchers have had some success with heuristic and semantic approaches that don't use machine learning. But, given enough supervised data, machine learning methods, especially neural networks show significant performance gains.

In this project, we aim to perform abstractive text summarization on news articles

2. Formalization

We can formally define the task as follows: The input is a string $x \in X$ of length M, where $X \subset (\{0,1\}^V, \{0,1\}^V, \cdots \{0,1\}^V)$, represents the set of all strings of length M and V is the size of our vocabulary. Each word x_i is represented as an indicator vector.

Now, an abstractive system finds a second-string $y \in Y$, where Y is defined similarly as X , such that

$$y = \operatorname*{argmax}_{y \in X} s(x, y)$$

under some scoring function s such that $s(x, y) \in \mathbb{R}$.

Now, an extractive or compressive system finds a summary using only words that are extracted from the original input.

An extractive summarizer is more formally expressed as:

$$y = \operatorname*{argmax}_{m \in \{1,\ldots,M\}^N} s(x, x_{[m_1,\ldots,m_N]})$$

Where, $x_{[i,j,k]}$ represents a string consisting of the i^{th}, j^{th} and k^{th} word of the input string x.

As mentioned above, we will be be performing an abstractive text summarization to generate news headlines using neural network approaches. As well as, implementing more traditional approaches of extractive summarization before the advent of deep neural networks and do a comparative study.

We will be using a corpus consisting of 300 thousand news articles, as collected by [Hermann et al, 2015]. The choice of the dataset and it's specifications are detailed in Section 4.

3. Related Work

Past work in summarization has mainly been extractive. Key sentences or passages in the source document were identified and reproduced as a summary. Humans on the other hand, tend to paraphrase the original story in their own words. As such, human summaries are abstractive in nature and seldom consist of reproduction of original sentences from the document.

The task of abstractive summarization has been standardized using the DUC-2003 and DUC-2004 competitions[[Over et al.2007]]. The data for these tasks consists of news stories from various topics with multiple reference summaries per story generated by humans. The best performing system on the DUC-2004 task, called TOPIARY, developed by [Bonnie et al.2004], used a combination of linguistically motivated compression techniques, and an unsupervised topic detection algorithm that appends keywords extracted from the article onto the compressed output. Some of the other notable work in the task of abstractive summarization includes using traditional phrase-table based machine translation approaches, compression using weighted tree-transformation rules and quasi-synchronous grammar approaches. In [Banko et al.2000], the authors approximated the problem of text summarization as a machine translation problem and come up with a statistical model that learns summaries from a training corpus using a count based noisy channel approach. They demonstrated the main drawback of extractive summarization : being unable to generate a summary that is shorter than a single sentence. This motivated us to concentrate on abstractive summatization with current research.

With the emergence of deep learning as a viable alternative for many NLP tasks, researchers have started considering this framework as an attractive, fully data-driven alternative to abstractive summarization. In a paper by [Rush et al.2015], the authors use convolutional models to encode the source, and a context-sensitive attentional feed-forward neural network to generate the summary, producing state-of-the-art results on Gigaword and DUC datasets. Later, in [Chopra et al.2016], the same authors turn to encoder-decoder Recurrent Neural Networks(RNN) and build upon their previous work with feed-forward networks. In this paper, they discuss a method of generating abstractive summaries using a conditional RNN approach. The conditioning is provided by a novel convolutional attention based encoder which ensures that the decoder focuses on the appropriate input words at each step of the generation. This model is similar to our model although we do not use the convolutional conditioning.

This architecture was first introduced by [Cho et. al. 2014] for the purpose of machine translation. This novel RNN model formed the basis if recent state-of-the-art neural methods for summarization including the paper by [Chopra et al.2016], which we described above. A limitation of this architecture is that it limits the encoding of the input sequence to a fixed length vector. To overcome this [Bahdanau et al.2014] proposed a attention mechanism to focus on certain sub-sequences while decoding the output sequence. All the neural methods for abstractive summarization employ this as a crucial part of the algorithm.

Our work concentrated on this attention based encoder-decoder RNN architecture, and we rely on a very similar research out of Google, also based on [Cho et. al. 2014] for implementation details:

[Sutskever et. al. 2014] proposed a sequence to sequence encoder-decoder RNN model, parallely from [Chopra et al.2016]. Even though, this paper didn't directly focus on text summarization, it led to the open source TensorFlow Model code that is provided by Google for generating news headlines. We employ this for our implementation and experimental design.

A important piece of research extending the neural netowrk approaches we mentioned was done by [Nallapati et al.2016] In this work, the authors take the same attention encoder-decoder RNN to this problem and and add to it performance enhancing features. It is from this paper, that we get the idea of using the TF-IDF frequency, POS tags, and named entity recognition of words to boosting the performance of the system.

In almost all the preceding neural approaches we mentioned, the Gigaword corpus is used

[[Graff et al. 2003]]. For licensing reasons we used a newswire text corpus composed of articles from CNN and DailyMail, from [Hermann et al, 2015]. The details of the architecture and the results of our experiments are described in the detail in the next sections.

4. Data

As mentioned earlier, all the related work conducted, especially those focusing on deep neural networks used the Gigaword corpus [[Graff et al. 2003]] which consists of about 5 million newswire articles. However, due to licensing issues we were unable too gain access the Gigaword corpus. Now, all past neural network models have only been able to generate one sentence length summaries and chose to do so on news articles only. Hence, we made use of the largest public domain news article dataset available.

We decided to utilize another corpus consisting of about 320 thousand articles from CNN/ DailyMail, initially collected for question answering by [Hermann et al, 2015]. The dataset consists of more than 320kURLs to online news articles from CNN and DailyMail and a script provided in their Github repository github.com/deepmind/rc-data to download the `html` articles and generate question answer pairs from them. We stop short of extracting the question-answer data.

The raw data had to processed considerably to allow us to use it as training data. These steps are explained in the following section.

4.1. Pre-processing of the Data

During the download process a portion of the URLs were not downloaded and missing. This might have been due to obsolete links in the metadata. We proceeded with about 290 thousand downloaded files. Now, our experimental setup involved using a modified version of a open source tensorflow code, which required the data to be in a tensorflow struct binary with the article and the body separated.

However, the articles were mostly composed of irrelevant text, from which two values needed to be extracted (i) the Headline, labeled ’abstract’ and (ii) the content of the news labeled ’article’. Due to the inconsistent format of the `html` files we used the python library Goose: github.com/grangier/python-goose to extract the relevant content. This ended up taking up significant time and effort. The extraction was done in parts and aggregated together later.

Analysis of the binary files showed that possibly due to the inconsistent nature of the raw data, a significant portion of extracted string were empty. The resulting tensorflow object had little more than 200 thousand processed articles, which we used to train our model.

5. Background

Our work fundamentally is about generating natural language,which is inherently sequential, from another sequential natural language words. The difference being the input sequence is considerably larger than the output sequence. The human brain also processes information sequentially while reading text where input at one time step is essential for understanding the next time step. Recurrent Neural Networks (RNNs) have been highly successful in recent past in modeling sequential tasks. So, RNNs were the natural choice for the model for both our tasks (i) capturing (and encoding) words from the input sentence and (ii) generating output summarized sentences.

Almost, all of the state-of-the art algorithms which employ RNNs, use Long Term Short Networks (LSTMs) as a RNN unit instead of a vanilla unit. We employ the LSTM-RNN network but in a specialized architecture know was the sequence-to-sequence RNN.

We provide a brief overview of this terms/models in the following sections and the detailed application details are given in Section 6.

5.1. Recurrent Neural Networks : An Overview

RNNs are essentially neural networks with loops where the previous hidden state is used to generate the next hidden state. Recurrent Neural Networks (RNNs) can be thought of as an extension of the traditional feed forward neural network. Once viewed as being 'unfolded' over time, it can be understood as multiple copies of the same network, passing messages from time step t to $t+1$.

Given a input x_t at a given time step and the corresponding hidden state h_t is implemented as follows in equation:

$$h_t = g(Wx_t + Uh_{t-1})$$

where $g(.)$ is a non linear differential function such as the sigmoid or the hyperbolic tangent function.

A generative RNN outputs the probability distribution over the next element of the sequence, given it's current state h_t. Our model, which we explain in Section 6, uses this type of generative RNN to generate the final summarized sentence. A generative RNN can theoretically capture a distribution over sequences of variable length by using a special output symbol to represent the end of the sequence: <EOS>.

However [Bengio et al, 1994] showed that RNNs suffer from the vanishing gradient problem, which makes gradient based learning algorithms intractable, and RNNs are unable to capture "long term dependencies". A solution is to use a more sophisticated activation unit, such as a Long Short Term Memory (LSTM) unit introduced by [Hochreiter and Schmidhuber, 1997], which has gained popularity.

5.2. Long Short Term Memory Unit

RNNs employing either LSTMs have been shown to perform well in tasks that require capturing long-term dependencies. These tasks include machine translation, which is closely tied to our task of summarization.

Unlike traditional RNN units, Long Term Short Network (LSTM) units are designed specifically for capturing long range dependencies . We will follow the analysis as done by [Chung et. al. 2014]

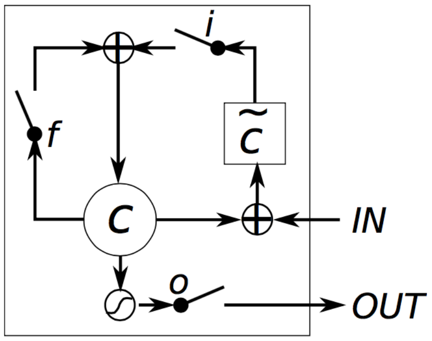

Figure 1: LSTM Internal Architecture

LSTMs carry a memory element c_t . This memory element is updated at each time step, where the unit makes a decision as to what new information to add and what to forget. These decision are made on the basis of three 'gates' i_t the input gate , f_t the forget gate and and o_t the output gate . The model diagram is explained in Figure 1 The gates i_t and f_t are used to calculate the new current memory from the previous:

$$c_t = c_{t-1} \odot f_t + \bar{c} \odot i_t$$

where $\bar{c}$ is the new information, which in turn is calculated from h_{t-1} and x_t

The final output of each LSTM unit is given by:

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

Where o_t is the *output gate* . As we can see the gates are binary variable vectors which itself are derived from the following 3 similar equations:

$$i_t = \sigma(x_t W_i + h_{t-1} U_i)$$

$$f_t = \sigma(x_t W_f + h_{t-1} U_f)$$

$$o_t = \sigma(x_t W_o + h_{t-1} U_o)$$

Unlike traditional recurrent unit which overwrites its state at each time-step, an LSTM unit decides which part of the current state to retain and which parts to forget via the introduced gates. An LSTM unit can potentially detect a important input and retain it over longer time steps to encode information in a better way.

5.3. RNN Sequence to Sequence: Encoder Decoder Models

Here, we describe briefly the underlying framework, called Sequence to Sequence RNN, which essentially acts as a Encoder followed by a Decoder. Sequence to Sequence models were first introduced for the purpose of machine translation by [Cho et. al. 2014], from which parallels can be drawn to text summarization. In the Encoder - Decoder framework, an encoder reads the input document, a sequence of vectors $x = (x_1, \cdots x_N)$, into a fixed length vector c. More generally, the x_i need not be a word depending on the application, for example, video captioning has also been achieved using the same architecture.

The most common approach is to use an RNN with LSTM units, which we explained in the previous sections, and $c = h_t$, where h_t is the hidden state at the final time-step t in the encoding stage. The decoder is often trained to predict the next word y_t given the encoded vector c and all the previously predicted words $(y_1, \cdots y_{t-1})$.

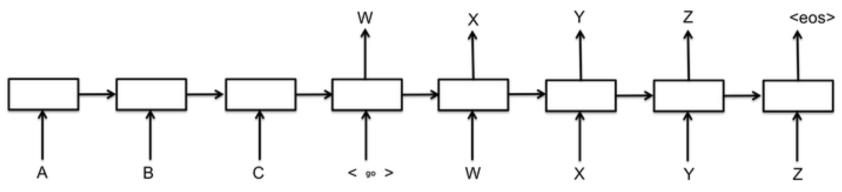

Figure 2: Sequence to Sequence Encoder Decoder Network

The core idea behind the model is to use recurrent units to encode all the information into a fixed length representation, and then use this 'encoded state representation' to decode into words sequentially. In other words, an encoder processes the input and a decoder generates the output.

Mathematically, in the encoder, the input sequence $(x_1, x_2 \cdots x_n)$ is fed into the encoder RNN sequentially and produces a series of hidden states $(h_1, h_2 \cdots h_n)$, where x_i, h_i represent the input and the hidden state at the i^{th} time step. Note that we only have access to h_n after the encoding stage is complete.

Next. the decoder produces a series of outputs $(y_1, y_2 \cdots y_m)$ of length say m. The conditional probability distribution of the output sequence can be given by

$$P(Y|X) = P((y_1, y_2 \cdots y_m)|(x_1, x_2 \cdots x_n)) = \prod_{t=1}^{m} p(y_t|h_{n+t-1}, y_{t-1})$$

Note that there exists other models which access the encoded vector c is combined along with intermediate hidden steps. Those are explained more in Section 6.3.

6. Method

With our aim of generating abstractive summaries we focus on using the concepts explained above in Section 5 and apply it to summarization, while also exploring the efficacy of non neural network approaches. The non neural network methods are known to perform well only in the extractive summarization problem.

6.1. Non-Neural Network Approaches: Extractive Summarization

We explored two different non-neural network approaches to summarization. Although a lot of work has been done in this area, the domain of extractive summarization is considered entirely separate since the approach at the task is entirely different.

The first method that is considered is a variation of the SumBasic algorithm as described by [Vanderwende et al.2007]. The original SumBasic method proposed by same authors in 2005 was a system that produced multi-document summaries and was motivated by the simple observation that words which are mentioned often in the article occur more frequently in human-generated summaries. Their algorithm first computes the probability distribution of all the words in the input based on the entire article word count. Then within every sentence, weights are given to each word based on the average probability of that word in the sentence. Finally they pick the best scoring sentence which contains the word which has the highest probability, wherein the sentence score is a collection of the probabilities of its words. They keep repeating this process to generate summaries of variable length. However, we only need to generate one sentence for our case. So, we will always find the best sentence according to SumBasic. In their modified approach, they perform sentence simplification using a previously learned thesaurus, which is a simplified version of a word embedding. It is worth mentioning that they do not use stopwords.

The second approach is called TextRank and was proposed by [Mihalcea et al.2004]. Their model is a graph-based approach for processing text which can be applied for text summarization as well. They have implemented their model with unsupervised learning wherein they are basically assigning importance to a vertex in a graph based on the information drawn from the entire graph. Here every vertex of the graph is a word in the article. They also follow a ranking model which assigns votes to every vertex based on how many edges it has. This is also in essence, finding the important keywords in the article.

6.2. Neural Network Model

As mentioned earlier we will be using the encoder decoder RNN. The motivation being that these models are desgined to map a variable length input sequence $(x_1, x_2 \cdots x_n)$ is mapped to an output sequence $(y_1, y_2 \cdots y_m)$. This was demonstrated by [Cho et. al. 2014] in machine translation, which has the same input - output paradigm. The sequence to sequence architecture has gained a lot of attention in recent years, because of it's effectiveness in deriving one sequential output from another. Our implementation is closely associated with [Sutskever et. al. 2014] and it's implementation for summarization.

Specifically, we feed 3 - Layered LSTM units the text of the news article one word at a time. Consistent, with the open-source implementation we are following. We restrict ourselves to the first 2 lines of the article. As preprocessing each word is passed through an embedding layer to get a embedded representation. The hidden state h_i constantly keeps getting updated at each time step.

h_t after all the input words have been fed, is treated as the 'encoded state representation'. The subsequent decoding stage is responsible for mapping this vector to a sequence of output words.

To prompt the decoding a start-of-sentence symbol is fed in as input. Then, the decoder generates, using attention mechanism, which we explain in the next section, the next section, each of the words of the headline, ending with an end-of-sentence symbol. After generating each word that same word is fed in as input when generating the next word like a standard generative RNN.

Our model, shown in 3 depicts our model. Note that this figure assumes a lack of attention mechanism, so the intermediate hidden states are ignored.

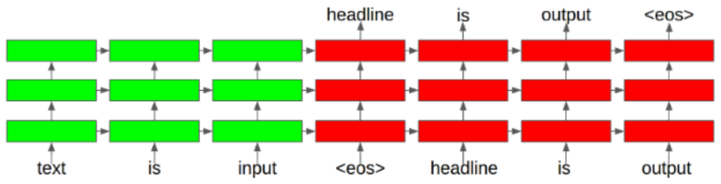

Figure 3: Sequence to Sequence RNN-LSTM for Abstractive Summarization

It's important to note that, we are one set of recurrent units stacks for both the encoding and the decoding stage. Figure 3 shows that set of LSTM stacks unrolled over time.

From a probabilistic perspective, summarization is equivalent to finding a target summary y that maximizes the conditional probability of y given document x. In neural summarization, we fit a parameterized model to maximize the conditional probability of summary using a training corpus. Once the conditional distribution is learned by a model, given a source document a corresponding summary can be generated by searching for the summary that maximizes the conditional probability.

This is exactly what the decoder tries to achieve. During decoding, the model maximizes the log-likelihood of the output word sequence given the encoded hidden state the frame sequence, and the previous words it has seen. A stochastic gradient descent algorithm is used to maximize the log-likelihood over the entire training data. The outputs y_t is used to calculate the series of most probable words. The generated words are compared against the gold standard summaries and loss is back propagated in time, and eventually the model learns.

6.3. Adding Attention Mechanism

As noted in 5 the most common approach is to use an RNN such that the encoded vector $c = h_t$, where h_t is a hidden state after final time-step of encoding stage. However, a novel feature was proposed by [Bahdanau et al.2014] for machine translation. The new architecture consists of a decoder that emulates searching through a source sentence during decoding a summary.

In a new model architecture, we define each conditional probability for an output y_t, $P(y_i|y_1 \cdots y_{i-1}, x)$ is also dependent on a weighted sum of the intermediate hidden units.

It should be noted that unlike the existing encoder-decoder approach (see section 5.3), here the probability is conditioned on a distinct vector for each target word y_i . The encoded vector c_i depends on a sequence of hidden states $(h_1, \cdots, h_t)$ to which an encoder maps the input document. Each annotation h_i contains information about the whole input sequence with a strong focus on the parts surrounding the i-th word of the input document.

The encoded vector c_i is, then, computed as a weighted sum of these annotations h_i :

$$c_i = \sum_{i}^{t} \alpha_i h_i$$

The weights α_i are in turn calculated from the both the hidden states in the encoder and the current state in the decoder. For more details the reader is encouraged to refer to [Bahdanau et al.2014].

We can understand the approach of taking a weighted sum of all the hidden states as computing an expected state, where the expectation is over possible alignments. Let α_i be interpreted as a probability that the target word y_i is aligned to, or translated from, a source word x_j. Then, the i-th encoding vector is the expected state over all the state with probabilities α_i. The probability reflects the importance of the state h_i in generating y_i.

Intuitively, this implements a mechanism of attention in the decoder. The decoder decides parts of the source document to pay attention to. By letting the decoder have an attention mechanism, we don't force the encoder to encode all the information in a fixed length vector. With this new approach the information can be spread throughout the sequence of states, which can be selectively retrieved by the decoder accordingly.

7. Experimental Setup

We use tensorflow to set up our model. As we have explained in the previous section, we use a slighlty modified version of the open-source encoder-decoder implementation by Google research and pre-process the CNN/ DailyMail corpus to extract a tensorflow object.

It is essential to compare the trained neural network from non neural approaches, which were more suited to extractive summarization. We provide the details below.

7.1. Baseline

Out of our dataset, we randomly sampled some articles to run a standard simple extractive algorithm to serve as our baseline. The naive summarizer first calculates the word frequencies for each article. Then, it breaks down the article into a list of sentences. To create a summary naively, we will find the sentences that contain the most frequent words in the article. We used a NLTK Python library based implementation from github.com/thavelick/summarize.

7.2. Extractive methods

For consistency, we took the same articles as the ones we ran the baseline on to evaluate summaries generated by the extractive methods. These articles and their original published headlines were compared with the summaries generated by extractive methods. The evaluation in this case was done manually as the size of the sample was manageable.

The SumBasic approach and the TextRank method generated unique summaries for most cases and the results are mentioned in the later sections. The methods were implemented in python, and while they are not an exact representation of the original methods, but they serve as the closest alternatives to the original methods.

7.3. Neural Net Training

The RNN Network training was conducted on the entire extracted portion of the data from Section 4. Training per article proceeds in two phases - the encoding phase and the decoding phase.

- Encoding: The first 2 sentences of each article is fed into the inputs of the RNN and the RNN generates an encoding of text. After the sequence has been completely fed, we feed a <Start> tag to denote the start of the decoding phase.
- Decoding: The RNN generates a set of words and using the gold standard headlines in the corpus of the articles the error gradient is back-propagated using the Adagrad Gradient Descent method.

The RNN layers were each trained using a set of hyperparameters. While training LSTM unit based RNN the choice of hyper parameters is critical. The behavior of the average loss was tracked after we began experimented with hyper parameters. The estimated optimum parameters are listed in Section 8.1

We also extracted the vocabulary of 200k words after parsing the entire training data. The decoding phase outputs will only be mapped to this vocabulary.

7.4. Neural Net Testing

The dataset is split into 90:5:5 - 90% of it is used as training data , 5% for validation and the remaining 5% is used as testing data. The relatively low amount validation data was due to our efforts to maximise the amount of training data. Considering the fact that the extracted number of articles = 200 thousand is way less than the Gigaword corpus used to train similar models.

Training a deep neural net requires large of amounts of computational resources and time, as noted by the recent research work in this area. After initial runs to estimate the probable optimum hyper parameters, we ran the network for a period of 36 hours at a time on an AWS Elastic Cloud CPU Instance.

For comparative qualitative study we used the same articles as the non neural network methods. The results are listed in the next section.

8. Results

In this section, we present our results for our abstractive summarization model and analyze the performance of our model. We will also look at evaluation metrics we chose to evaluate our model. We will then present a subjective comparison of our results against the extractive models that we implemented to run on the same dataset.

8.1. Abstractive summarization model - RNN

For our RNN based text summarization model, we used up $102.58 worth of Amazon Web services (AWS) credits for running a remote instance and training our model on that instance. We could train our model 5 times with different hyperparameters and we are going to analyze the results of our experiments in the first and summarize our results in the end.

Like most machine learning models, the RNN model needs to be trained with different settings, or what is formally called as hyperparameters. We tried experiments with different hyperparameters. First we changed only the number of hidden units in the RNN. The number of hidden units in the model is kept as a perfect multiple of the size of the word embedding vector. This was observed to yield the best results. First we kept the word embedding as a vector of size 128 and varied the number of hidden units between 128,256,512 and 1024. The complexity of the model increases with every step and it convergence time is increased with each increment. We did not observe significant gains of using 512 over 256 hidden units and hence did not go ahead with our experiment with 1024 hidden units.

The optimum hyperparameters are summarized as :

- Word embedding dimensions = 128
- Hidden layer dimensions of the RNN units = 256
- Summary Length Limit = 30
- Article length: First 2 sentences
- Batch size = 64
- Vocabulary word: Most Frequent 200k words.

Using our RNN model, we found that for generating output summary as a headline, we need only 1 sentence, and it works best if input length is comparable to output length. This practice was adopted by many other works on abstractive text summarization, where people have restricted the input to 1-3 sentences of the article. For our model, we found that the best results were obtained when we passed two input sentences as input. The evaluation of best results is subjective here since we manually checked the quality of the summaries generated by our model. We will talk about different evaluation metrics shortly.

We get the following deprecating loss function graph 4 after 15 hours of training our model with the hyper-parameters mentioned.

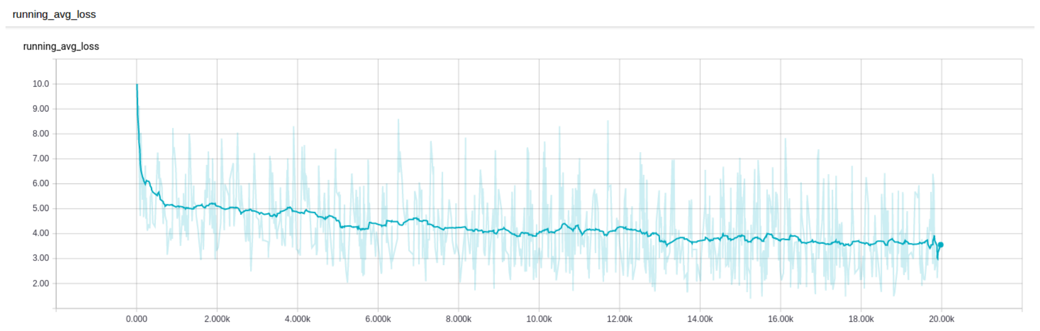

Figure 4: Running average loss vs training cycles

Running average loss is an important metric to look at to understand the progress of learning of a deep neural network. It can also be viewed as an equivalent of perplexity or negative log likelihood. There is a good amount of disturbance in the loss function. This can be attributed to the batch size. When the batch size is 1, the disturbances are relatively high. When the batch size is the full dataset, disturbances are minimal because every gradient update improves the loss function monotonically, if the learning rate is not too high. The hockey stick nature of this function is as per expectations and we saw convergence at a value close to 3.4.

While the running average loss converges to 3.4, it is expected to go even lower with more training time. With more training, the model seemed to perform better, however we had to stop our training midway to test different approaches.

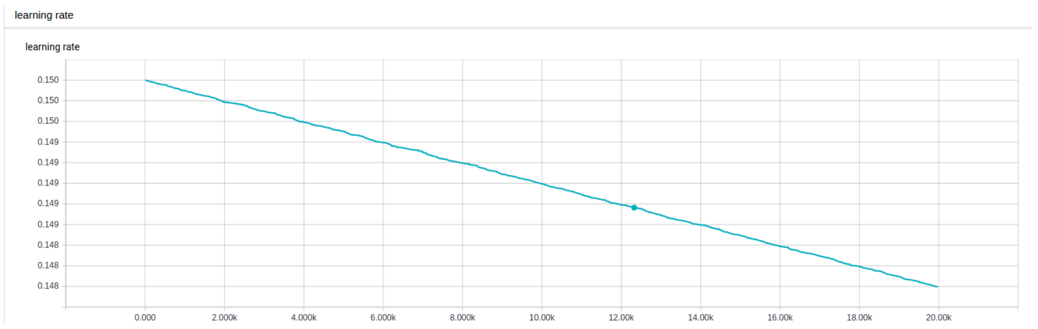

Figure 5: Learning rate

The average learning rate of our model was 0.149. It was set to decrease very slowly, so that the model can converge in an optimum manner. Generally, with low learning rates the improvements are linear, and with high learning rates they will start to look more exponential. But, higher learning rates perform worse at worse values of loss because there is too much chaos in the optimization and the parameters are bouncing around too much.

8.2. Evaluation Metric

There are multiple ways of evaluating the generated text descriptions:

(i) BLEU (bilingual evaluation understudy): evaluating the quality of text which has been machine-translated from one natural language to another. It is know to have a high correlation to human judgments of quality.

(ii) ROUGE (Recall-Oriented Understudy for Gisting Evaluation): the most widely used metric for evaluating summarization quality. It gives to a higher score when there are more overlapping N-grams. Hence, it inherently gives priority to extractive summarization.

Although, the ROUGE evaluation method is known to prefer extractive summaries. Hence, to compare our summaries across extractive and abstractive methods, we opted for a manual judging approach. Basically, this is based on the assumption that humans are the best evaluators of a summary.

Table 1 compares the summaries generated by our model against the baseline and other extractive approaches. It also compares all summaries against the gold standard which is the actual headline of the corresponding news article when it was published.

Actual headline	Baseline	TextRank	Our model
Man was already culturally diverse before he left Africa: Differences in stone tools hint at a variety of traditions	The tools, discovered in the region between sub-Saharan Africa and Eurasia, were made in different ways, reflecting a diversity of cultural traditions.	These stone tools reveal how early populations of modern humans dispersed across the Sahara just before they left North Africa.	man tools <UNK> how they <UNK>.
Police hunt murderer who has gone on the run after absconding while on day release	She has done much to me help and realise my full potential as a human being,' he said.	First jailed for the killing of Mr Smales alongside his brother in 1998, Maxwell made headlines in 2011 when he unexpectedly pleaded guilty at a retrial after his conviction was overturned on appeal.	<UNK> Man <UNK> guilty of prison <UNK> <UNK>.
Floyd Mayweather vs Manny Pacquiao will be shown on Sky after broadcaster beats BoxNation for right to show	The current Sky record stands at 1.2million buys for Ricky HattonâĂŹs Vegas loss to Mayweather in 2007.	The richest fight of all time will not come cheap either for Sky Sports or their subscribers even though Sky are keeping faith with their core following by keeping the base price below	battle with <UNK> Why <UNK> next to all.

Table 1: A comparison of summaries with the original headline

The summaries generated by the extractive models (including baseline), look better than the summaries generated by our model, but they are also lacking in conveying the gist of the articles. This affirms our initial belief that automatic text summarization is inherently a very hard task.

The summaries generated by our model contain a token <UNK>, which refers to unknown or out of vocabulary word. This limits the evaluation of our model-generated summaries, since it does not refer to a word in the dictionary. There are ways to overcome this problem which we will discuss in Section 9.

8.3. Analysis of Results

As can be seen from our results, the summaries generated match the original headlines of the article poorly. Though the model did generate state-of-the-art results, we believe that the reason that our model trained poorly is that Recurrent Neural Networks are notoriously hard to train.

There is noticeable matching in the semantics of most of the predicted words shows that the model is essentially able to capture the semantic embeddings of the key words in the document, but it's struggling to generate a cohesive sentence.

A crucial and most likely the central reason of the sub-satisfactory performance was less amount of data. The converge and stagnation of the averge loss shows that the neural network is able to learn but the learning seems incomplete with less amounts of data. We are fairly confident that the performance will improve considerably given a larger dataset and more time to train.

9. Conclusion and Future Work

We implemented extractive summarization techniques using different methods, which were mostly a variation of TextRank and Classifier. The generated summaries are grammatically correct but do not always generate a meaningful gist of the article. This led us to experiment with the deep-learning based approaches.

We experimented initially with abstractive text summarization with RNN model which takes word embeddings of article text with first two sentences passed as input. Our results with the RNN model are not satisfactory, but they are encouraging. Given enough training time and data, we are confident that our model can yield much better results. However, for the scope of this project, we were limited to implementing a basic version of the model.

The summaries generated by the RNN are lacking full grammatical sense. It is intuitive to think about a hybrid approach. Future work can involve adding sematic knowledge to our model by adding part-of-speech tags, TF-IDF word frequencies, and NER tags to the input vector. This model is expected to perform better than our model which had only word embeddings as input.

To give the abstractive model more expressive power, we can add multiple layers of RNNs to process the data. The output of the first layer will become the input of the second and so on. This could result in interesting results and similar work has been carried out with promising results in similar domains.

Another extension to this work could be to train the RNN model on a different, preferably larger dataset and see the output. The model if trained on larger dataset like Gigaword corpus is bound to learn well and generate much more meaningful summaries. We saw good improvements by adding semantics to the plain word-embeddings, however there were a lot of unknown tokens. The unknown token problem can be handled via a switcher-generator model as described by [Nallapati et al.2016].

References

[Hermann et al, 2015] K M Hermann, T Kočiský, E Grefenstette, L Espeholt , W Kay, M Suleyman and P Blunsom (2015). Teaching Machines to Read and Comprehend *Advances in Neural Information Processing Systems (NIPS)*

[Bengio et al, 1994] Y. Bengio, P. Simard, and P. Frasconi. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157-166.

[Hochreiter and Schmidhuber, 1997] S. Hochreiter and J. Schmidhuber. (1997). Long short-term memory. *Neural Computation*, 9(8):1735-1780.

[Cho et. al. 2014] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation *Proceedings of the Empiricial Methods in Natural Language Processing* 1724-1734

[Chung et. al. 2014] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling *arXiv preprint* arXiv:1412.3555

[Sutskever et. al. 2014] I. Sutskever, O. Vinyals and Q. V. Le (2014). Sequence to Sequence Learning with Neural Networks *Advances in Neural Information Processing Systems (NIPS)*

[Chopra et al.2016] Sumit Chopra, Michael Auli, and Alexander M. Rush. (2016). Abstractive sentence summarization with attentive recurrent neural networks. *arXiv preprint* In HLT-NAACL.

[Nallapati et al.2016] Ramesh Nallapti, Bowen Zhou, Cicero Nogueira dos santos, Caglar Gulcehre, and Bing Xiang. (2016). Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond. *arXiv preprint*:1602.06023

[Rush et al.2015] Alexander M. Rush, Sumit Chopra,and Jason Weston. (2015). neural attention model for abstractive sentence summarization. CoRR,abs/1505.00487.

[Bahdanau et al.2014] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. (2014). Neural machine translation by jointly learning to align and translate. CoRR,abs/1409.0473

[Banko et al.2000] M. Banko, V.O. Mittal,and M.J. Witbrock. (2000). Headline generation based on statistical translation. In Proceedings of ACL, pages 318-325.

[Over et al.2007] P. Over, H. Dang, and D. Harman. (2007). DUC in Context. Information Processing and Management *arXiv preprint* 43(6):1506-1520

[Bonnie et al.2004] D. Zajic, B. Darr, and R. Schwartz (2004). BBN/UMD at DUC-2004: Topiary (2004). *In Proceedings of the 2004 Document Understanding Conference (DUC 2004) at NLT/NAACL 2004*

[Graff et al. 2003] D. Graff, J. Kong, and K. Chen (2003). English Gigaword.. *Linguistic Data Consortium, Philadelphia*

[Mihalcea et al.2004] R Mihalcea, P Tarau. (2004). Textrank: Bringing Order into Texts *digital.library.unt.edu*

[Vanderwende et al.2007] Beyond SumBasic: Task-focused summarization with sentence simplification and lexical expansion. (2007). Beyond SumBasic: Task-focused summarization with sentence simplification and lexical expansion *Elsevier*