ABSTRACT
In this paper, we present an implementation of search engine integrated with a text summarizer, which is built utilizing of deep learning techniques. The text summarization problem is a sequence to sequence problem and follow the typical encoder-decoder architecture. We use word embedding to give each word semantic representation and use recurrent neuron network (RNN) for both encoding and decoding. Also, we adopted an attention mechanism to make the model learn how much attention each word should be paid for a specific corresponding generated word. Finally, we integrated everything to our search engine to enhance its user experience.

Keywords
search engine, text summarization, recurrent neuron network, attention mechanism

1. INTRODUCTION
Auto text summarization has been a topic discussed and studied by more and more people nowadays due to the rapid development of deep learning techniques. In the past days, text summarization systems were mostly focused on text deletion. Systems tend to use some hard-coded rules to decide which word to keep and which word to discard. Consequently, each of the generated word will be guaranteed to be found in the original text, which means the generated words were actually a subset of the original text.

However, the deep learning approach tries to encode the original text into a semantic representation via an encoder and then tries to decoder it by another decoder with fewer words, pretty much like what a normal human being will handle this task, thus lead to much more acceptable results.

A sad fact about this is that at current stage, we are still quite far from a genuine artificial intelligence, which can make a summarization that is so good that it’s even hard to tell whether it’s from a real human being or not.

Despite the fact, it’s still of great significance to try this topic, especially integrate it with a search engine. If you look at the resulting page of Google, or DuckDuckGo, you will find that most of the searching entities consist of a title, an url and a snippet. The problem is that in some certain case, a title maybe too short or un-relevant (and some page even don’t have a title) and a snippet may be just an un-relevant chunk of information which contains the keyword. Giving all this information, a user may still be confused about what’s the gist of the web page. If we can build a search engine with an additional piece of information, which tells the user what’s the web page is about in a very short sentence, it will surely improve the user experience.

Recurrent Neuron Network (RNN) has gain quite a lot of popularity in the past year as a matter of the fact that Google adopted this method for its own translation system and achieved state-to-art performance. RNN is perfect for sequence processing problem, as well as sequence generation. There are lots of similarities between a text summarization system and a translation sequence. Both deal with the sequence to sequence problem and share the encoder-decoder architecture, except for the fact that after understanding the content and encoding it into some semantic representation, a summarization system tries to rephrase it using fewer words while a translation system tries to express the meaning using another language.

Attention Mechanism has also been applied as suggested in this paper [1]. The advantage of using this technique is that we are able to get an inside view of how the final data is being generated. For each of the generated data we are able to know a weight distribution among all the input words, thus make result more sensible.

In this paper, we will first introduce how we preprocess the data, how we trained our own word embedding matrix given the dataset. Then we will give a basic introduction of the idea of RNN and attention mechanism. Finally, we will present how we integrated everything to make it works and things succeed and fails for our project.

2. DATA PROCESSING AND WORD2VEC
2.1 Data source
We used the ECIR 2016 dataset, which you can it here: http://research.signalmedia.co/newsir16/signal-dataset.html. The dataset includes mainly two kinds of articles, namely news and blogs. And most of those articles are written in English, while some portion of the dataset is written in other languages. In our project, we mainly use the content of the article as our input data (X) and the headlines of the data as our labels (Y).

2.2 Preprocessing
First of all, to make things simple, we convert all the letters to lowercase, and then split the content by various delimiters. After this step, we’ve got a vector of words. Then we give each word a unique integer index. Thus, to avoid string manipulation which is highly computing power consuming.

Then, we set a maximum length for both the headlines and the content. The reason for this is also due to a limited memory space and computing power. If the length of the headlines or the content
exceeds the maximum length set, then we simply cut it, and discard
the rest.

2.3 Word Embedding
2.3.1 Background
Understanding the meaning of a word in specific context is
of central importance in NLP. While it is impossible to
construct a human like understanding of word in context,
approaches have been developed to discover word similarity.
As opposed to traditional “count-based” methods (Pantel et
al., 2010) like PPMI and SVD, representative neural network
models (Bengio et al., 2003) including SGNS and GloVe
embedded words into a low dimensional space. Each word
is represented as a vector. Words with semantic similarity
have closer vectors.

2.3.2 Implementation and performance of SGNS
2.3.2.1 Dataset
word2vec models* were trained based on skip-gram with
negative-sampling training method (SGNS) based on
Wikipedia dump xml data file and Signal Media One-
million news articles dataset (SMO dataset). Signal Media
One-million news articles dataset contains 265,512 Blog
articles and 734,488 News articles, while Wikipedia dump
xml file contains 918 texts and titles. 10% of the JSON
objects in Signal Media One-million news articles dataset
were randomly chosen as training dataset and the other 10%
was chosen as test dataset.

2.3.2.2 Result
Two models are trained. One is based on Wikipedia dump
xml file. The other one is based on training dataset from
SMO dataset. SMO dataset is thirty times larger than
Wikipedia dump file, and manual tests of the model shows a
little advantage comparing to Wikipedia dump file based
model. Accuracy tests according to questions-words.txt (test
file) reveals that SMO dataset based training bears 4.93%
correctness, while wiki dump based dataset has 2.09%
correctness.

<table>
<thead>
<tr>
<th>Test code</th>
<th>Wikipedia dump dataset</th>
<th>SMO dataset</th>
</tr>
</thead>
</table>
| model.most_similar(positive=['android',
  'apple'], negative=['google'])          | (‘ios’, 0.774983763694
  7632)                  | (‘4s’, 0.847942590713
  501)                   |
| model.wv.most_similar(positive=['woman',
  'king'], negative=['man'])             | (‘louis’, 0.730951607277
  3254)                   | (‘godly’, 0.817936539649
  9634)                   |
| model.wv.doesnt_match("break
  fast cereal dinner lunch").split())   | ‘dinner’               | ‘cereal’          |
| model.wv.similarity("woman", ‘man’)      | 0.629240477699 62521   | 0.962907828097 92563 |

Table 1. Results of different models

2.3.2.3 Conclusion
While the test accuracy results are highly correlated to local
corpus, both manual and machine testing show that a larger
corpus does bring higher accuracy rate to word2vec model.

3. ARCHITECTURE OF THE
SUMMARIZER
3.1 The Big Picture
Before we dive into any details, it’s beneficial to understand the big
picture first. It is illustrated in the picture below which is a typical
encoder-decoder architecture [2]:

![Encoder-Decoder Architecture](image)

This is the basic of our fundamental, on which we will add more
stuff to make it close to the architecture what we’ve been using in
our real implementation.

3.2 Recurrent Neuron Network(RNN)
The idea of RNN [3] can be akin to human being’s behavior of
reading a paper. Think about the scenario when you are reading,
you read the first paragraph of the paper. Then when you switch to
the second paragraph, you are not reading without any pre-requisite
knowledge of the paper, you are reading with your understanding
of the first paragraph instead.

However, for the structure of the classic Neuron Network, each
input data is treated identically without consequence and this may
lead to some problems. The classic data flow look like this:
What RNN did is that it adds a shortcut from the output of the neuron back to its input:

So, what happens right now is that when you get your data input, you also get the data from the previous step.

Here is an unrolled version of RNN [4], which pretty much do the same thing:

From this structure, we can see we have got a basic sequence to sequence structure, with $X_i (0 \leq i \leq n)$ being the input and $h_i (0 \leq i \leq n)$ being the output. And we can also get the sense that while $h_0$ is generated only from $X_0$, $h_1$ is generated from both $X_1$ and the output of the first neuron. Finally, the last output $h_n$ represents all the semantic encoding of everything from $X_0$ to $X_n$, which makes $h_n$ a powerful generalization of the original text.

We can also stack the unrolled version of RNN vertically to make it more powerful:

In this example, we have a 3 layers RNN and $h_n$ should have all the information encoder into one output.

### 3.3 Training Architecture

A little wired thing we have done here is that we actually have a different paradigm for training and testing, where usually they are the same.

In the training process, when we are trying to generate a word, we actually feed into the correct previous word from the labels [5], while in testing, we don’t do that and let the model generate everything by itself.

That being said, we need to feed our labels into our RNN models as well. Meanwhile the decoder part is also a RNN and we need to chain the encoder and decoder together. After made those revision to our model, we got a model looks like this:
Therefore, before our training process, we concatenate our content and our headline and put them together as our input data. We feed them into the RNN as a whole.

3.4 Attention Mechanism

Attention mechanism in deep learning have been widely used and studied. The good part of attention mechanism is that we can have a direct impression that for each of the generated word, how much attention should each of the original text be paid. As illustrated by the following picture, which is how it might work for generating the first word “this”:

![Attention Mechanism Example](image)

In previous work, the output of the last RNN layer compress all the information into one value $h_n$. What we have added to our model is an attention vector to produce the weighted average of the last RNN layer and feed into decoder to produce the output sequence. After we add this, the model looks like this:

![Model With Attention Mechanism](image)

4. EVALUATION

4.1 Hardware

We use a quite powerful machine which are from our friends, which have 4 GTX Titan GPU.

4.2 Training Loss

Have been said in the previous section, when we are training we actually feed into the correct previous word from the label set and try to predict what the next word will be. Therefore, we can also use this metric to measure how good our model actually is.

We trained our models for hours and we get a loss like this:

![Training Loss](image)

From which we can find the loss goes rapidly down in the very first epochs and then it saturates after 1500-2000 epochs.

4.3 Summarization Performance

We can provide a few examples to see how our models works.

<table>
<thead>
<tr>
<th>Original Text</th>
<th>Generated Text</th>
<th>Actual Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>VETERANS saluted Worcester's first ever breakfast club for ex-soldiers which won over hearts, minds and bellies.</td>
<td>veterans names jose mcclain as next ceo</td>
<td>Jumpshot Gives Marketers Renewed Visibility Into Paid and Organic Keywords With Launch of Jumpshot Elite</td>
</tr>
<tr>
<td>Apple has announced a series of hardware upgrades for the iPhone, iPad and Apple TV. iPhone's price sees steep increase</td>
<td>apple tv what does it are the best</td>
<td>Harwood Feffer LLP Announces Investigation of VASCO Data Security International, Inc</td>
</tr>
</tbody>
</table>
The Chinese online retailer plans to buy back up to $1 billion of its own American depositary shares over the next 24 months.

Yen drops as European and Japanese stocks start week higher. * U.S. data fails to provide clarity on Fed rate-hike timing. * Speculators cut U.S. dollar longs to lowest in over a year - CFTC

Meredith and Walt Disney are big market movers

FOREX-Dollar rises vs yen and Swiss franc as stocks stabilise 2013

4.4 The reason why it succeeds and why it fails

First of all, the generated headlines do feel like have the same style with the actual headlines due to our usage of “teacher forcing”. We can see it do capture some semantic meaning of the original text and still adhere to the English Grammar.

But what it fails is that it still not focused on the original text enough and it seems throw out some random words in the training dataset.

Here are some main reasons why it fails:

4.4.1 Limitation of resource

We actually do have a powerful machine with the most advanced GPUs on market. But it turns out that it’s still not powerful enough. We have a dataset of about 6 Million records. But training on the entire dataset may takes a month and as a matter of fact we can only train on 60 k records, which is only 1% of our entire dataset.

As a matter of fact, it do help improving the result a lot when you have more data. In the beginning of training period the model outputs some non-sense like “the the the the”. Then after we trained it on more data, it throws some random words. And right now the good thing is that it can produce some sentence which feel like an English sentence.

4.4.2 The nature of the dataset

The articles in our dataset mainly come from the real-world blogs and articles without any selection. As a consequence, there are a lot of special words and appears only once. We have a huge amount of unique words with most of it being only appears for one time. This caused a lot of preprocessing problem to our model and may cause big issues.

We believe it’s better to start from a smaller dataset with only the most commonly used words.

5. ACKNOWLEDGES

We would like to thank everyone who has helped us in this project, especially my friend to let us use their expensive hardware to do our experiments.

6. FUTURE WORKS

We will try to extend our project in the following aspects:

The first one is try to improve the architecture. The attention mechanism is too expensive for us to train, thus we need to find a better structure which allow us to train on more data.

The next is that we should try another dataset with the reasons mentioned in Section 4.

7. REFERENCES


[3] Ilya Sutskever, Oriol Vinyals, Quoc V. Le, Sequence to Sequence Learning with Neural Networks.


[6] Minh-Thang Luong, Hieu Pham, Christopher D. Manning, Effective Approaches to Attention-based Neural Machine Translation