

# Text Summarization using Tensorflow

Vivsvaan Sharma

Department of Computer Science and Engineering  
Chandigarh College of Engineering and Technology  
Sector 26, Chandigarh

Abhinav Khetarpal

Department of Computer Science and Engineering  
Chandigarh College of Engineering and Technology  
Sector 26, Chandigarh

Charanjit Singh

Department of Computer Science and Engineering  
Chandigarh College of Engineering and Technology  
Sector 26, Chandigarh

**Abstract—** In this new era, where tremendous information is available on the Internet, it is most important to provide the improved mechanism to extract the information quickly and most efficiently and present it. It is very difficult for human beings to manually extract the summary of a large documents of text. There are plenty of text material available on the Internet. So, there is a problem of searching for relevant documents from the number of documents available, and absorbing relevant information from it. In order to solve the above two problems, the automatic text summarization is very much necessary. Text summarization is the process of identifying the most important meaningful information in a document or set of related documents and compressing them into a shorter version preserving its overall meanings. Search Engines like Google, Bing etc. are increasingly being used by people to search and get results of their interests. These search engines are employing very sophisticated tools to gather and make sense of the large amounts of data and summarize it and display it on these platforms. Also is giant tech companies or business corporations, they need to fetch the relevant information from databases and combine them in Datawarehouse to do the further processing and analysis so as to gain knowledge from that data which will be helpful in market analysis and decision making. Summarized data obtained from search engines or from such models has proven to be an effective way of capturing the relevant information and predict trends, thereby improving the decision making the process.

We propose a tensorflow model to summarize the articles and predict their corresponding headlines or titles. Collected articles or Mined data was cleaned and articles were fed into our model for training along with their correct titles. Then we tested our model with some new articles and summarized them and saved their titles.

**Keywords—** *summarizer; tensorflow; text-summarization;*

## I. INTRODUCTION

### 1.1 Motivation

In this new era, where tremendous information is available on the Internet, it is most important to provide the improved mechanism to extract the information quickly and most efficiently. It is very difficult for human beings to manually extract the summary of a large documents of text. There are

plenty of text material available on the Internet. So, there is a problem of searching for relevant documents from the number of documents available, and absorbing relevant information from it. In order to solve the above two problems, the automatic text summarization is very much necessary. Text summarization is the process of identifying the most important meaningful information in a document or set of related documents and compressing them into a shorter version preserving its overall meanings.

Before going to the Text summarization, first we, have to know that what a summary is. A summary is a text that is produced from one or more texts, that conveys important information in the original text, and it is of a shorter form. The goal of automatic text summarization is presenting the source text into a shorter version with semantics. The most important advantage of using a summary is, it reduces the reading time. Text Summarization methods can be classified into extractive and abstractive summarization. An extractive summarization method consists of selecting important sentences, paragraphs etc. from the original document and concatenating them into shorter form. An Abstractive summarization is an understanding of the main concepts in a document and then express those concepts in clear natural language. There are two different groups of text summarization: indicative and informative. Inductive summarization only represent the main idea of the text to the user. The typical length of this type of summarization is 5 to 10 percent of the main text. On the other hand, the informative summarization systems give concise information of the main text. The length of informative summary is 20 to 30 percent of the main text. Machine Learning and Deep Learning tools are perfect for summarization of such a process.

Search Engines like Google, Bing etc. are increasingly being used by people to search and get results of their interests. These search engines are employing very sophisticated tools to gather and make sense of the large amounts of data and summarize it and display it on these platforms. Also is giant tech companies or business corporations, they need to fetch the relevant information from databases and combine them in

Datawarehouse to do the further processing and analysis so as to gain knowledge from that data which will be helpful in market analysis and decision making. Summarized data obtained from search engines or from such models has proven to be an effective way of capturing the relevant information and predict trends, thereby improving the decision making the process.

The ability to extract insights from data is a practice that is widely gaining momentum throughout the world and forms a fascinating area of study with the ability to provide a wider view on how data is shaped.

### 1.2 Contribution

This work provides an alternative approach to tackle the task of summarizing and fetching relevant information from data. Neural Networks has been used to train the data, store associations of input parameters and output parameters.

The remainder of this paper has been organized into 3 sections. ‘Material and Methods’ section describes the approach used and provides a step by step account of the process followed as well as details about the neural network methods used. In the ‘Results and Discussions’ section analyzes the performance of the model. ‘Conclusion’ section summarizes the work done.

## II. MATERIALS AND METHODS

### A. Data Collection

This task included a series of decisions that needed to be made. First, we downloaded data set of articles and their corresponding headlines. Some articles we collected from news websites by using crawlers. The dataset contains a total of 3800000+ articles.

TABLE I

APPROXIMATE DATA FOR ARTICLES

Articles and Titles	Total Number
Articles	3993610
Titles	3993610

### B. Data Preprocessing

Real-world data is often incomplete, inconsistent and/or lacking in certain behavior/trends and is likely to contain many errors. The dataset or crawlers returns a massive volume of data.

The collected raw data were transformed into an understandable format, and stored in Text files. It included the following steps:

**Data Cleaning** - This process included filling in missing values, smoothing the noisy data (all the data were stripped off special characters like ‘@’ and URLs to overcome noise), resolving inconsistencies in data.

**Data Integration** - Data with different representation are put together and conflicts were resolved. We stored each article in a single line, then in next line there was next article. Similarly, titles were stored in each line. The line number for article and its corresponding title was same.

**Data Transformation and reduction** - Data is normalized, aggregated and generalized. Then it is represented in a reduced form and stored in Dataset. Separate files were created for training purpose and for testing purposes.

### C. Creating a Training and Testing Set

The Dataset for training was manually created by dividing the total articles file into two parts. For training purpose, we created 2 files. First is train.article.txt file which contains 38 lakh articles and second is train.title.txt file which contains their corresponding titles, i.e. 38 lakh titles. For testing purposes, we created 2 files, first is valid.article.filter.txt which contains 1.9 lakh of articles and second is valid.title.filter.txt which contains their corresponding titles, i.e. 1.9 lakh of titles

TABLE III

TRAINING AND TESTING DATASET

	Number of Articles	Number of Titles
Training Set	3803958	3803958
Testing Set	189652	189652

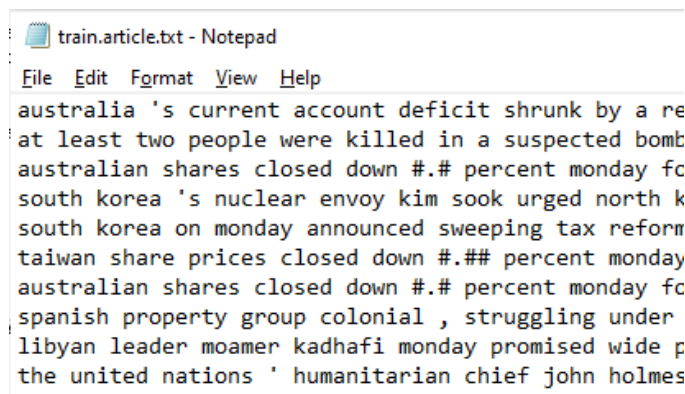


Fig 1 Articles data for training

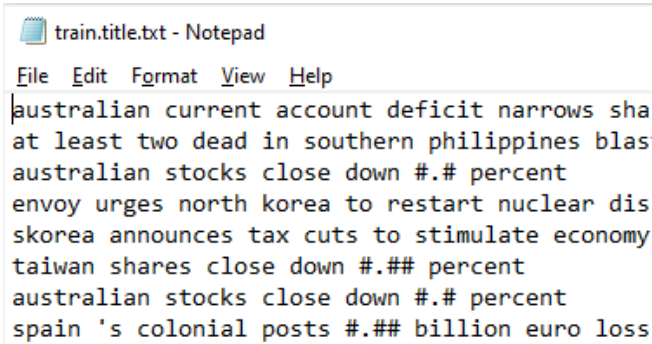


Fig 2 Titles data for training

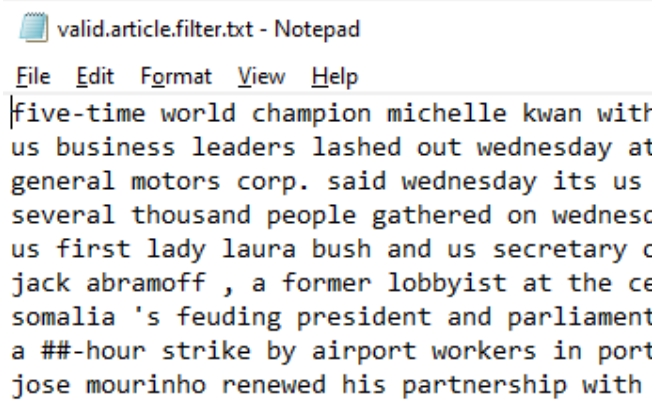


Fig 3 Articles data for testing

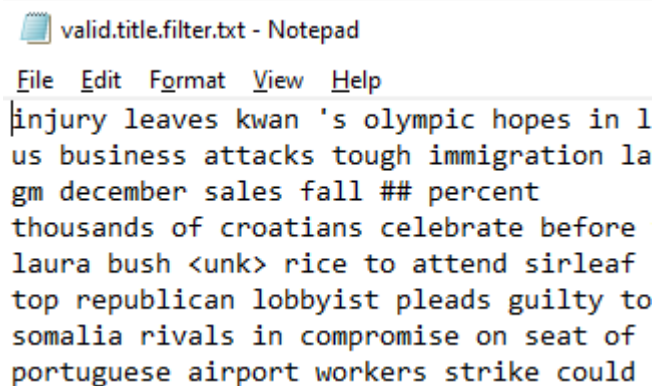


Fig 4 Titles data for testing

#### D. Design

The proposed model was simple tensorflow model using seq2seq library. Seq2Seq is a method of encoder-decoder based machine translation that maps an input of sequence to an output of sequence with a tag and attention value. The idea is to use 2 RNN that will work together with a special token and trying to predict the next state sequence from the previous sequence.

#### Tokenization

The first step of processing we make the texts go through is to split the raw sentences into words, or more exactly tokens. The easiest way to do this would be to split the string on spaces, but we can be smarter

- we need to take care of punctuation
- some words are contractions of two different words, like isn't or don't
- we may need to clean some parts of our texts, if there's HTML code for instance

The texts are truncated at 100 tokens for more readability. We can see that it did more than just split on space and punctuation symbols:

the "'s" are grouped together in one token

the contractions are separated like this: "did", "n't"

content has been cleaned for any HTML symbol and lower cased

there are several special tokens (all those that begin by xx), to replace unknown tokens (see below) or to introduce different text fields (here we only have one)

Our model is Encoder-Decoder model with attention mechanism.

Word Embedding – Used Glove pre-trained vectors to initialize word embedding.

Encoder – Used LSTM cell with stack\_bidirectional\_dynamic\_rnn.

Decoder – Used LSTM BasicDecoder for training, and BeamSearchDecoder for inference.

Attention Mechanism – Used BahdanauAttention with weight normalization.

#### E. Implementation

The Language Models can use a lot of GPU. A solution to that is to run this model on Google Colab. Google Colab is a free cloud service and now it supports free GPU! You can: improve your Python programming language coding skills. develop deep learning applications using popular libraries such as Keras, TensorFlow, PyTorch, and OpenCV.

But since our model had modules like tokenizations and it had to deal with parsers, google colab does not do good with parsers. It creates errors and we could't find help even on internet. Only the documentation of colab is provided which was not of much help. So what we did was we implemented this model on our local machines only and we had to reduce the batch size or the training and testing data size. Instead of taking 30 lakh articles for training and 1.9 lakh articles for testing, we took 10000 articles for training and tested it on 150+ articles.

This model's accuracy is estimated to be around 75%-82%. If we had more computing power or if colab had more

support of parsers, we could have increased the size of training and testing set and we could have achieved more accuracy. But due to lack of colab support and less computational power on our local machines, the overall accuracy over 10 epochs is estimated to be 60%-70%.

**Note – In text classification or summarization, our focus is more towards loss over the steps or epochs than the accuracy, due to the fact that calculating and working with loss in text classification is more reliable than the accuracy. So we will be talking in terms of loss only from now on.**

Below are some of the screenshots which shows model being trained.

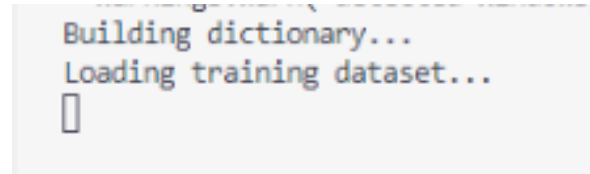


Fig 5 Building dictionary and loading training set

We were encountered with some warnings too.

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
	step 8000: loss = 32.86821746826172		
	step 8100: loss = 30.635047912597656		
	step 8200: loss = 35.5181884765625		
	step 8300: loss = 30.919998168945312		
	step 8400: loss = 28.195972442626953		
	step 8500: loss = 28.1325740814209		
	step 8600: loss = 23.47557258605957		
	step 8700: loss = 30.6511287689209		
	step 8800: loss = 26.22564697265625		
	step 8900: loss = 30.350683212280273		
	step 9000: loss = 30.52069854736328		
	step 9100: loss = 31.8193359375		
	step 9200: loss = 20.254005432128906		
	step 9300: loss = 28.99323272705078		
	step 9400: loss = 31.771085739135742		

Fig 5 Model being trained on 10 lakh articles

```
Iteration starts.
Number of batches per epoch : 79
OMP: Info #212: KMP_AFFINITY: decoding x2APIC ids.
OMP: Info #210: KMP_AFFINITY: Affinity capable, using global cpuid leaf 11
OMP: Info #154: KMP_AFFINITY: Initial OS proc set respected: 0-3
OMP: Info #156: KMP_AFFINITY: 4 available OS procs
OMP: Info #157: KMP_AFFINITY: Uniform topology
OMP: Info #179: KMP_AFFINITY: 1 packages x 2 cores/pkg x 2 threads/core (2
OMP: Info #214: KMP_AFFINITY: OS proc to physical thread map:
OMP: Info #171: KMP_AFFINITY: OS proc 0 maps to package 0 core 0 thread 0
OMP: Info #171: KMP_AFFINITY: OS proc 1 maps to package 0 core 0 thread 1
OMP: Info #171: KMP_AFFINITY: OS proc 2 maps to package 0 core 1 thread 0
OMP: Info #171: KMP_AFFINITY: OS proc 3 maps to package 0 core 1 thread 1
```

Fig 8 Iterations start

Number of Batches per epoch were 79. Number of epochs were taken to be 10.

```
step 9500: loss = 19.447298049926758
step 9600: loss = 23.290605545043945
step 9700: loss = 32.87354278564453
step 9800: loss = 30.574546813964844
step 9900: loss = 21.69025421142578
step 10000: loss = 30.162410736083984
step 10100: loss = 30.537364959716797
step 10200: loss = 30.906835556030273
step 10300: loss = 26.045326232910156
step 10400: loss = 36.70331573486328
step 10500: loss = 32.73112487792969
step 10600: loss = 26.743621826171875
step 10700: loss = 36.329193115234375
```

Fig 6 Model being trained on 10 lakh articles

```
Epoch 1: Model is saved. Elapsed: 00:04:10.41
step 100: loss = 52.89484405517578
Epoch 2: Model is saved. Elapsed: 00:05:53.30
step 200: loss = 47.321754455566406
Epoch 3: Model is saved. Elapsed: 00:07:29.73
step 300: loss = 40.98884582519531
Epoch 4: Model is saved. Elapsed: 00:09:06.15
Epoch 5: Model is saved. Elapsed: 00:10:42.15
step 400: loss = 36.060211181640625
```

Fig 9 Epochs 1 to 5

Five epochs are completed and total steps until now are 400. Average loss is 35%

However, this was taking many days to train, so we stopped it after 3 days. We trained our model on 5000 articles, due to lack of computational power and resources as discussed above.



```
Epoch 6: Model is saved. Elapsed: 00:12:17.54
step 500: loss = 29.413022994995117
Epoch 7: Model is saved. Elapsed: 00:13:52.56
step 600: loss = 29.978010177612305
Epoch 8: Model is saved. Elapsed: 00:15:28.33
step 700: loss = 24.95439910888672
Epoch 9: Model is saved. Elapsed: 00:17:04.20
Epoch 10: Model is saved. Elapsed: 00:18:40.65
```

Fig 10 Epochs 6 to 10

After all epochs are completed, average loss over the steps is estimated to be 16%-25%.

### III. RESULTS AND DISCUSSIONS

After training the model, we tested it on the articles which were in the valid.article.filter.txt file. We took 100 articles for testing. It predicted or summarized the articles and stored their titles in the result.txt file.

First, the dictionary data is loaded from the saved model, and then validation dataset or testing dataset is loaded.

```
Loading dictionary...
Loading validation dataset...
WARNING:tensorflow:From test.py
```

Fig 11 Loading dictionary and test set

After the loading of dictionary and validation set is complete, model starts summarizing the articles and it writes their corresponding titles in result.txt file.

```
Writing summaries to 'result.txt'...
OMP: Info #212: KMP_AFFINITY: decoding x2APIC ids.
OMP: Info #210: KMP_AFFINITY: Affinity capable, using glob
OMP: Info #154: KMP_AFFINITY: Initial OS proc set respecte
OMP: Info #156: KMP_AFFINITY: 4 available OS procs
OMP: Info #157: KMP_AFFINITY: Uniform topology
OMP: Info #179: KMP_AFFINITY: 1 packages x 2 cores/pkg x 2
OMP: Info #214: KMP_AFFINITY: OS proc to physical thread m
OMP: Info #171: KMP_AFFINITY: OS proc 0 maps to package 0
OMP: Info #171: KMP_AFFINITY: OS proc 1 maps to package 0
```

Fig 12 Writing summaries

```
OMP: Info #171: KMP_AFFINITY: OS proc 3 maps to package 0
OMP: Info #250: KMP_AFFINITY: pid 5784 tid 4600 thread 0
OMP: Info #250: KMP_AFFINITY: pid 5784 tid 4600 thread 1
OMP: Info #250: KMP_AFFINITY: pid 5784 tid 9976 thread 2
OMP: Info #250: KMP_AFFINITY: pid 5784 tid 960 thread 3 b
OMP: Info #250: KMP_AFFINITY: pid 5784 tid 4892 thread 4
OMP: Info #250: KMP_AFFINITY: pid 5784 tid 9884 thread 6
OMP: Info #250: KMP_AFFINITY: pid 5784 tid 9968 thread 5
OMP: Info #250: KMP_AFFINITY: pid 5784 tid 8608 thread 7
Summaries are saved to "result.txt"...
```

Fig 13 result.txt file is saved

The final summarized articles are now written in results.txt file and file is saved in the same directory path as that of train and test files.

Some of the summary outputs are shown below in the screenshots.

```
"general motors corp. said wednesday its us sales fell ##.## percent
> Model output: gm us sales down # percent in december
> Actual title: gm december sales fall # percent

"japanese share prices rose ##.## percent thursday to <unk> highest
> Model output: tokyo shares close # percent higher
> Actual title: tokyo shares close up # percent

"hong kong share prices opened ##.## percent higher thursday on foll
> Model output: hong kong shares open higher
> Actual title: hong kong shares open higher as rate worries ease
```

Fig 14 Sample Summary Outputs (1)

```
"the dollar regained some lost ground in asian trade thursday in what was
> Model output: dollar stable in asian trade
> Actual title: dollar regains ground in asian trade

"the final results of iraq 's december general elections are due within ti
> Model output: iraqi election results due in next four days
> Actual title: iraqi election final results out within four days

"microsoft chairman bill gates late wednesday unveiled his vision of the
> Model output: bill gates unveils new technology vision
> Actual title: gates unveils microsoft 's vision of digital lifestyle
```

Fig 15 Sample Summary Outputs (2)

### IV. CONCLUSION

In this paper we have shown the use of Text summarization for the process of identifying the most important meaningful information in a document or set of related documents and compressing them into a shorter version preserving its overall meanings. Search engines are employing very sophisticated tools to gather and make sense of the large amounts of data and summarize it and display it on these platforms. Also is giant tech companies or business corporations, they need to fetch the relevant information from databases and combine them in Datawarehouse to do the further processing and analysis so as to gain knowledge from that data which will be helpful in market analysis and decision making. Summarized data obtained from search engines or from such models has proven

to be an effective way of capturing the relevant information and predict trends, thereby improving the decision making the process.

In this paper we proposed a tensorflow model to summarize the articles and predict their corresponding headlines or titles. Collected articles or Mined data was cleaned and articles were fed into our model for training along with their correct titles. Then we tested our model with some new articles and summarized them and saved their titles. The use of text summarization for summarizing huge amounts of data poses challenges at different stages. In this paper, the scarcity of training data is tackled for text classification/summarization. Finally, a model is proposed for summarizing articles which uses the articles data. While this model alone may not be sufficient to efficiently summarize results, however, it becomes a crucial component when combined with other statistical models and offline techniques.

We implemented the proposed model on a very small dataset of 5000 articles and tested it on 100 articles due to lack of computation. However, this model can be extended in the future to work or train on 38 lakh or more articles to give the best and efficient results. Features should be extracted from newly mined data and compared with an existing set of features. Some similarity metric can be used to compare the new and old features.

## ***References***

- [1]. K. Mao, J. Niu, X. Wang, L. Wang and M. Qiu, "Text Classification", 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, 2015.
- [2]. Text Summarization by Jyoti Ramteke, Darshan Godhia, Samarth Shah and Aadil Shaikh
- [3]. D. Das and S. Bandyopadhyay, "Labeling Text at Sentence Level" Proceedings of the 8th Workshop on Asian Language Resources, pp. 47–55, Aug. 2010.
- [4]. A. Bakliwal, P. Arora, and V. Varma, "Hindi subjective lexicon: A lexical resource for Hindi polarity classification," Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC), pp. 1189–1196, May 2012.
- [5]. S. Mukherjee and P. Bhattacharyya, "Sentiment analysis on text with lightweight discourse analysis," Proceedings of the 24th International Conference on Computational Linguistics (COLING), pp. 1847–1864, Dec. 2012.
- [6]. M. Taboada, J. Brooke, M. Tofiloski, K. Voll, and M. Stede, "Lexicon based methods for sentiment analysis," Comput. Linguist., vol. 37, pp. 267–307, 2011.
- [7]. Neethu, M. S., and R. Rajasree. "Text Classification and Sentiment analysis in using machine learning techniques." Computing, Communications, and Networking Technologies (ICCCNT), 2013 Fourth International Conference on. IEEE, 2013