

# DATA SCIENCE

## *Semester Project Report*

---

JUNE 5

---

**Abdullah Rauf (007)**

**Wajahat Masood (003)**

**Saif Ur Rehman (013)**



**To: Dr. Hikmat Ullah Khan**

**BSE 7A**

---

# Table of Contents

1. Introduction .....	3
2. Aim .....	3
3. Selected Algorithms .....	4
4. Data Set.....	5
5. EDA .....	7
6. Experimental Set up .....	9
7. Result .....	9
8. References and Acknowledgements.....	13
9. Appendix .....	13

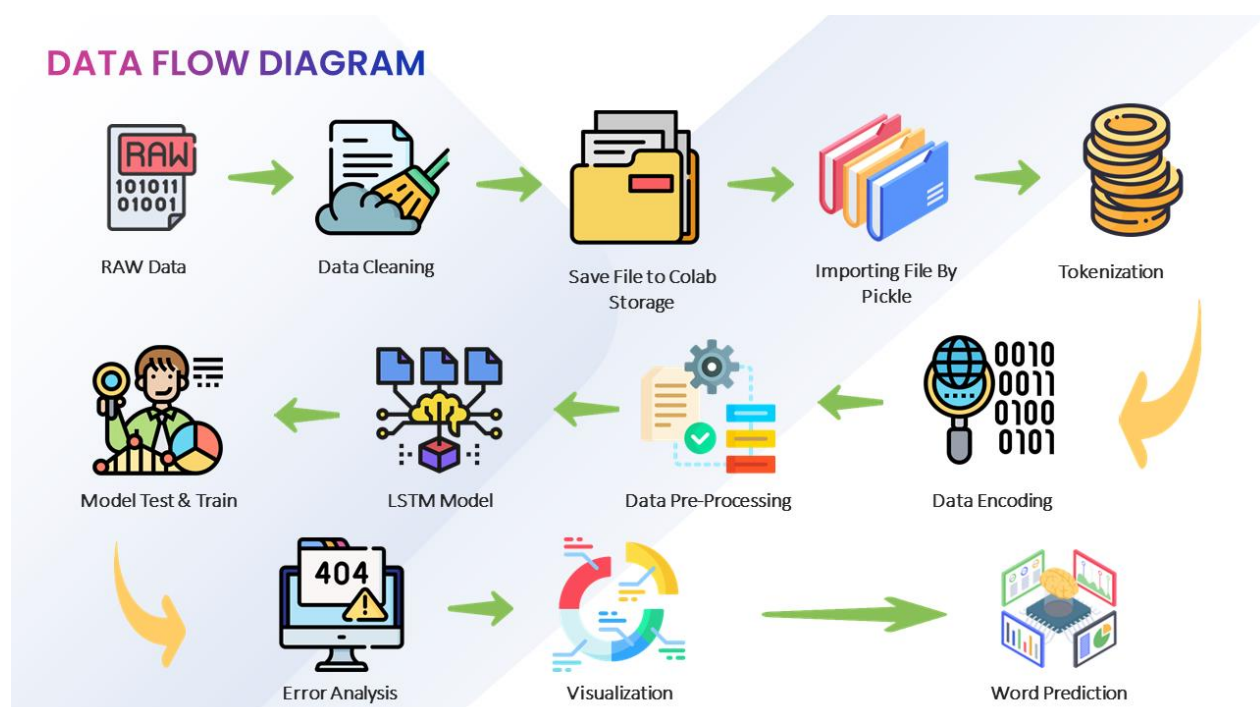
## INTRODUCTION

We often love texting each other and find that whenever we try to type a text a suggestion pops up trying to predict the next word we want to write. This process of prediction is one of the applications NLP deals with. We have made huge progress here and we can use Recurrent neural networks (RNN) & Long Short-Term Memory (LSTM) for such a process. This Project deals with how we can use a model better than a basic RNN and use it to predict the next word. We deal with a model called Long Short-term Memory (LSTM). We can use the TensorFlow library in python for building and training the model.

The next word prediction for a particular user's texting or typing can be awesome. It would save a lot of time by understanding the user's patterns of texting. This could be also used by our virtual assistant to complete certain sentences. Overall, the predictive search system and next word prediction is a very fun concept which we will be implemented.

## AIM

The Aim of the Project is to cover what the next word prediction model built will exactly perform. The model will consider the last word of a particular sentence and predict the next possible word. We will be using methods of natural language processing, language modeling, and deep learning. We will start by analyzing the data followed by the pre-processing of the data. We will then tokenize this data and finally build the model. The model will be built using LSTM's.



## SELECTED ALGORITHMS

### LSTM Model:

We will be building a sequential model. We will then create an embedding layer and specify the input dimensions and output dimensions. It is important to specify the input length as less than or equal to 4 since the prediction will be made on exactly four word and we will receive a response for those words. We will then add an LSTM layer to our architecture. We will give it 3 units and make sure we return the sequences as true. This is to ensure that we can pass it through another LSTM layer. For the next LSTM layer, we will also pass it through another 128 units, but we don't need to specify return sequence as it is false by default. We will pass this through a hidden layer with 16262 node units using the dense layer function with rely on set as the activation. Finally, we pass it through an output layer with the specified vocab size and a SoftMax activation. The SoftMax activation ensures that we receive a bunch of probabilities for the outputs equal to the vocab size.

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 3, 3)	48786
bidirectional (Bidirectional)	(None, 3, 128)	34816
attention (attention)	(None, 128)	131
dense (Dense)	(None, 16262)	2097798
Total params: 2,181,531		
Trainable params: 2,181,531		
Non-trainable params: 0		

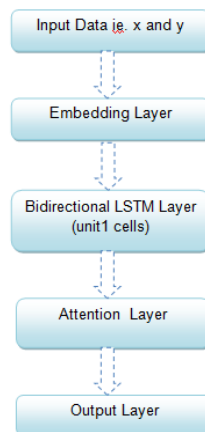
Figure 1: LSTM MODEL Summary

### Stacked LSTM Model:

Model developed by stacking a Bidirectional LSTM layer over a LSTM layer. Three such models corresponding to sequences of length 2, 4 and 7. In each case everything remains the same except the number of unit1 and unit2 cells. The idea is to retain the sequence information of the words in x to predict the next word class in y. The output layer has a SoftMax activation, so we got probability distribution of the output classes.

---

Below is the architectural flowchart of the Model:



**Figure 2:Model Architectural Diagram**

Predicting the next word for a sequence, all words of the sequence don't contribute equally to the prediction of that word.

For example: Consider the sentence "I went to see the doctor because I was quite sick". Suppose here, I must predict the word sick. For that, the words went, see, and doctor will carry more weightage as they contribute more to the probability for the next word to be sick.

Thus, to provide weights to each of the words in a sequence, we will need an attention layer. And then pass data to Output layer for display.

## Data Set:

The data set can be downloaded using this link,

<https://d396qusza40orc.cloudfront.net/dsscaphstone/dataset/Coursera-SwiftKey.zip>

This zip file contains four folders each containing data from four different languages namely English, Russian, Finnish and German. We Only Use US-English Data Set. The US file contain text file of all English language data from Twitter, Us Blogs, Websites, Channels & News of Last Year.

There are two text files document which we upload on Colab storage the text\_twt.txt file contain 2360147 Lines & text\_blogs.txt file contain 899289 Lines. These are 2 Raw files which we upload initially for further use. On these files following function performed,

- ✓ Removing Extra space
- ✓ Removing Special Characters
- ✓ Tokenizing text data
- ✓ Tokenizing Twitter data

After all Cleaning Process the files are pickled into “/content/sample\_data/Clean\_Data” folder using by Python Pickle Library.

Then File are named as,

Cleaned\_Text.txt & Cleaned\_Twitter.txt

Now these data sets are used further in LSTM Model.

```

with open("/content/sample_data/Clean_Data/cleaned_text.txt", "rb") as fp:
    cleaned_text = pickle.load(fp)
with open("/content/sample_data/Clean_Data/cleaned_twitter.txt", "rb") as fp:
    cleaned_twitter = pickle.load(fp)
cleaned_corpus=cleaned_text+cleaned_twitter
print(len(cleaned_corpus))

66904096

```

Figure 3: Cleaned Data Set

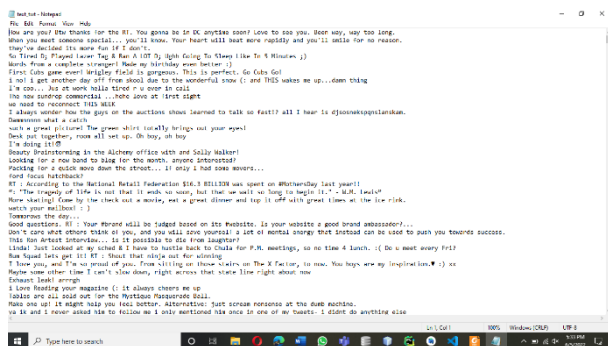


Figure 5: TEXT Data.txt

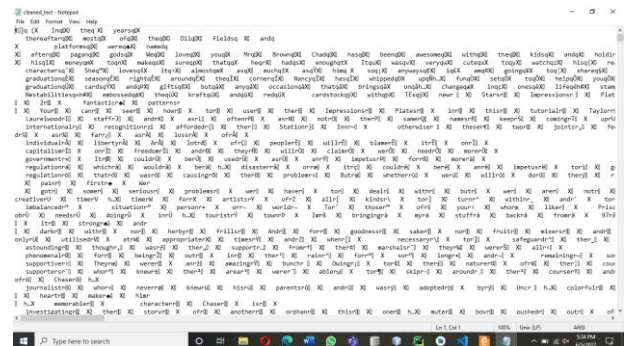


Figure 4: Cleaned Text.txt

The Size of Files are below in table,

Sr No.	File Name	Type	Size
1	Text_blogs	.txt	200 Mb
2	Text_twt	.txt	159 Mb
3	Cleaned_text	.txt	495 Mb
4	Cleaned_twt	.txt	390 Mb

Table 1: Data Set Size Table



## Exploratory Data Analysis (EDA):

### Error Analysis for Trigram:



Figure 6: Tri-Gram Code

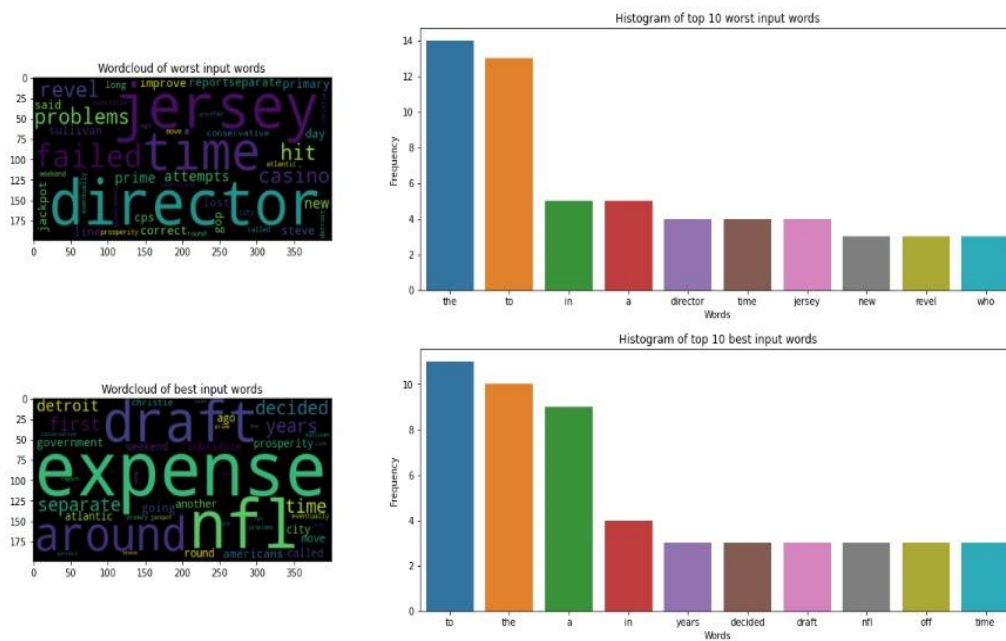
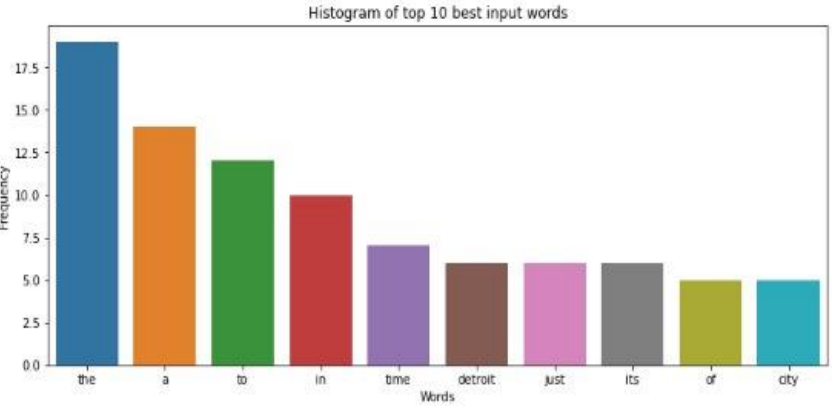
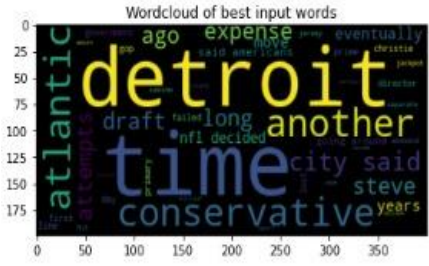
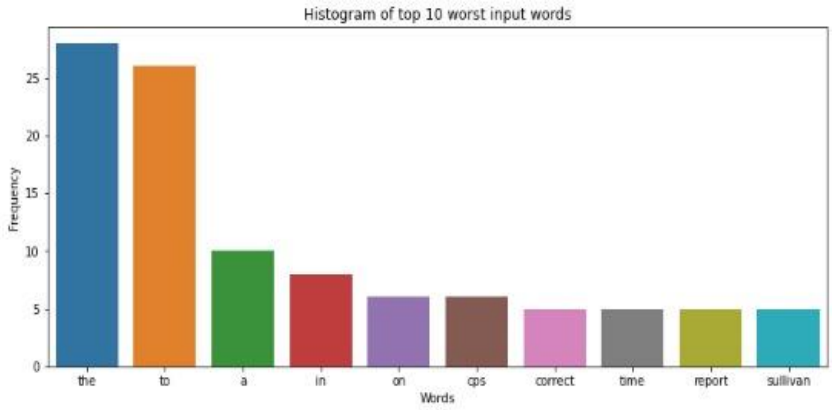
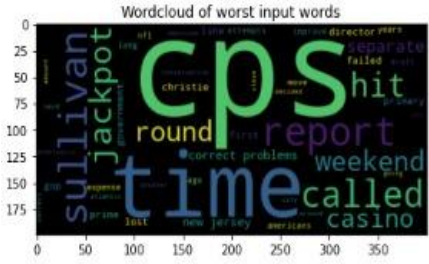


Figure 7: Tri-Gram Word Cloud & Best, Worst Word Histogram



**Figure 8: Code of Six Gran**



**Figure 9: Six-Gram Word Cloud & Best, Worst Word Histogram**



---

## Experimental Set Up:

The Libraries Used in Next Word Prediction are,

- + Pandas
- + NLTK
- + Word Tokenizer
- + Tweet Tokenizer
- + Pickle
- + Word Cloud
- + STOPWORDS
- + Image Color Generator
- + TensorFlow
- + NumPy
- + Keras
- + Matplotlib.pyplot

Tool, we have used for this problem is Google Colab, which is great for data science task.

Language used for the coding purpose is Python in Jupyter Notebook.

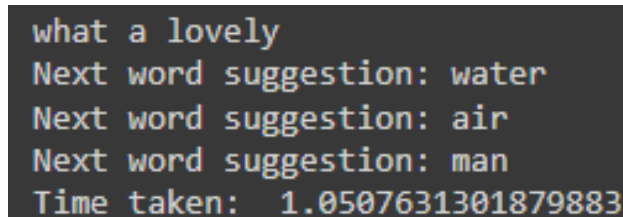
Ram Consumption is 11.56 Gb / 12.68 Gb.

Disk Space used is 56gb / 107gb.

LSTM Models are Used for Performance Evaluation Measures.

## Result:

For the prediction, we will load the tokenizer file which we have stored in the pickle format. We will then load our next word model which we have saved in our directory. We will use this same tokenizer to perform tokenization on each of the input sentences for which we should make the predictions on. After this step, we can proceed to make predictions on the input sentence by using the saved model.



```
what a lovely
Next word suggestion: water
Next word suggestion: air
Next word suggestion: man
Time taken: 1.0507631301879883
```

*Figure 10: Output*

The predictions model can predict optimally on most lines as we can see.

Now, we develop a high-quality next word prediction for the above dataset. We can reduce the loss significantly epochs. The next word prediction model which we have developed is fairly accurate on the provided dataset. The overall quality of the prediction is good. However, certain pre-processing steps and certain changes in the model can be made to improve the prediction of the model.

Model Training Chart is Provided below,

Sr No.	No of Word as Input	No of Tokens	Loss	Accuracy
1	Only One Word	16120	7.0989	0.0850
2	Only Three Words	16262	8.8459	0.0781
3	Only Six Words	16395	9.2400	0.07101

*Table 2: Error Analysis & Model Test Table*

Detail & Charts are,

```
Epoch 00001: loss improved from inf to 7.81460, saving model to lstmatt_len7.hdf5
Epoch 2/70
782/782 [=====] - 142s 181ms/step - loss: 7.3765 - accuracy: 0.0573 - loss: 7.3764

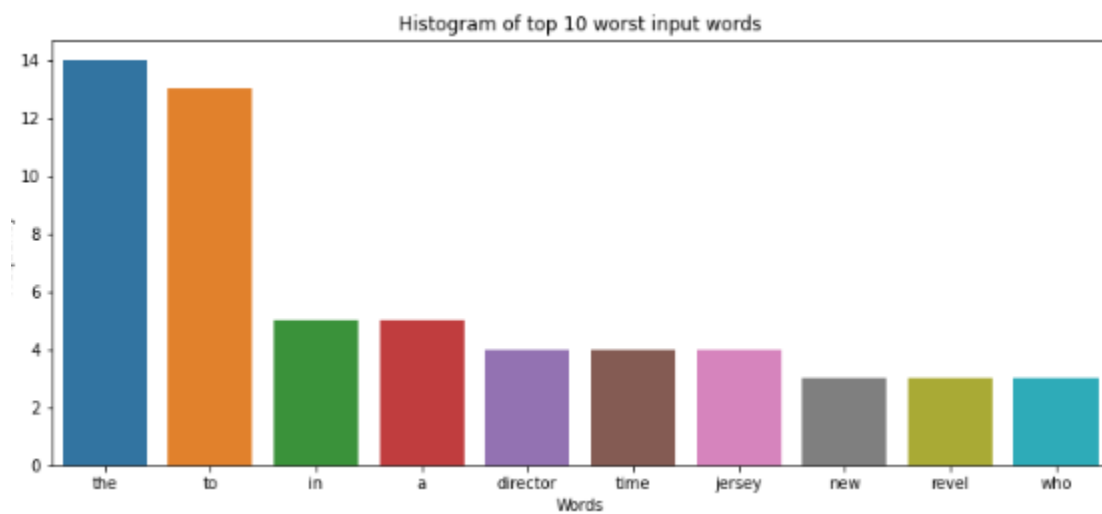
Epoch 00002: loss improved from 7.81460 to 7.39028, saving model to lstmatt_len7.hdf5
Epoch 3/70
782/782 [=====] - 139s 177ms/step - loss: 7.2547 - accuracy: 0.0577 - loss: 7.2545

Epoch 00003: loss improved from 7.39028 to 7.26817, saving model to lstmatt_len7.hdf5
Epoch 4/70
782/782 [=====] - 141s 181ms/step - loss: 7.1456 - accuracy: 0.0589

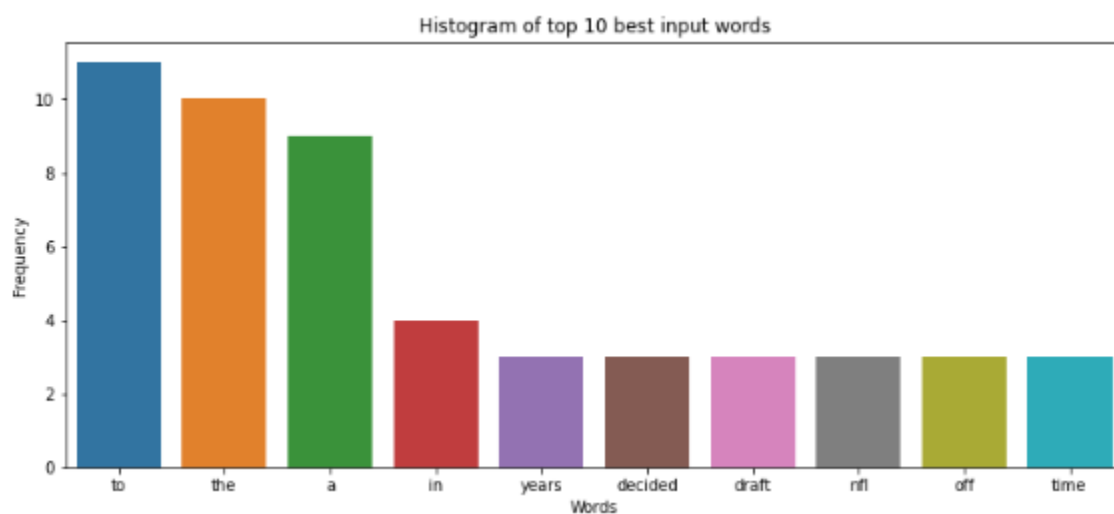
Epoch 00004: loss improved from 7.26817 to 7.16223, saving model to lstmatt_len7.hdf5
Epoch 5/70
782/782 [=====] - 147s 188ms/step - loss: 7.0674 - accuracy: 0.0616

Epoch 00005: loss improved from 7.16223 to 7.08898, saving model to lstmatt_len7.hdf5
```

*Figure 11: Model Testing & Saving Results in Files*



*Figure 12: TOP 10 Worst Words for Prediction*



*Figure 13: TOP 10 Best Words for Prediction*

---

Final Output Result for Some Predictions are,

```
[ ] what a lovely
Next word suggestion: water
Next word suggestion: air
Next word suggestion: man
Time taken: 1.0507631301879883
because there is no
Next word suggestion: heavy
Next word suggestion: towers
Next word suggestion: chance
Time taken: 1.575056791305542
maybe I should be
Next word suggestion: a
Next word suggestion: forced
Next word suggestion: charged
Time taken: 0.05898880958557129
very
Next word suggestion: be
Next word suggestion: than
Next word suggestion: have
Time taken: 1.1010732650756836
in
Next word suggestion: the
Next word suggestion: a
Next word suggestion: his
Time taken: 0.06171011924743652
There
Next word suggestion: and
Next word suggestion: was
Next word suggestion: said
Time taken: 0.08094501495361328
```

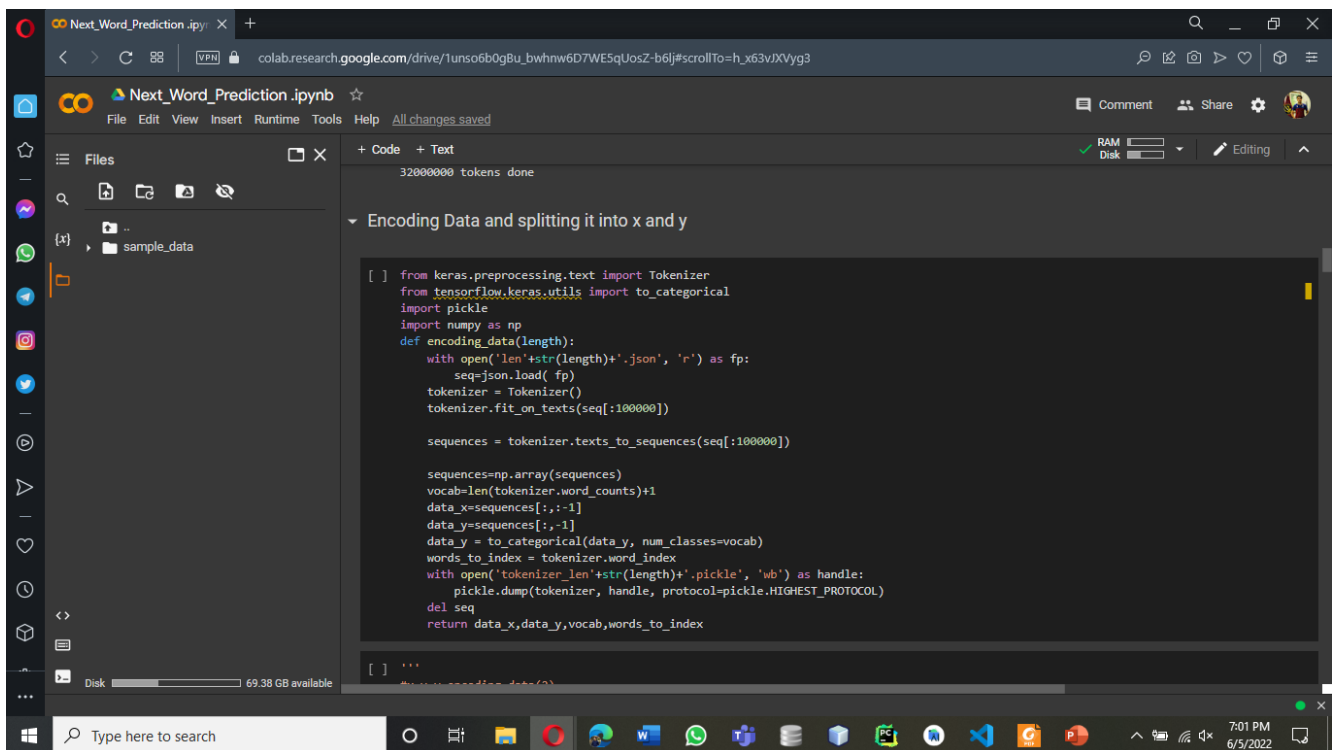
*Figure 14: Word Predictions & Suggestions*

## References and Acknowledgements:

- [https://www.itm-conferences.org/articles/itmconf/pdf/2021/05/itmconf\\_icacc2021\\_03034.pdf](https://www.itm-conferences.org/articles/itmconf/pdf/2021/05/itmconf_icacc2021_03034.pdf)
- [https://www.tutorialspoint.com/time\\_series/time\\_series\\_lstm\\_model.htm](https://www.tutorialspoint.com/time_series/time_series_lstm_model.htm)
- <https://www.youtube.com/watch?v=VB7bbFIEAhk>
- <https://insightimi.wordpress.com/2021/04/05/next-word-predictor-using-lstm/>

## Appendix:

Main Parts of the Code are,



```
[ ] from keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical
import pickle
import numpy as np
def encoding_data(length):
    with open('len'+str(length)+'.json', 'r') as fp:
        seq=json.load( fp)
        tokenizer = Tokenizer()
        tokenizer.fit_on_texts(seq[:100000])

        sequences = tokenizer.texts_to_sequences(seq[:100000])

        sequences=np.array(sequences)
        vocab=len(tokenizer.word_counts)+1
        data_x=sequences[:, :-1]
        data_y=sequences[:, -1]
        data_y = to_categorical(data_y, num_classes=vocab)
        words_to_index = tokenizer.word_index
        with open('tokenizer_len'+str(length)+'.pickle', 'wb') as handle:
            pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
        del seq
        return data_x,data_y,vocab,words_to_index

[ ] '''
#using encoding_data(5)
```

*Figure 15: Encoding Data into X & Y*

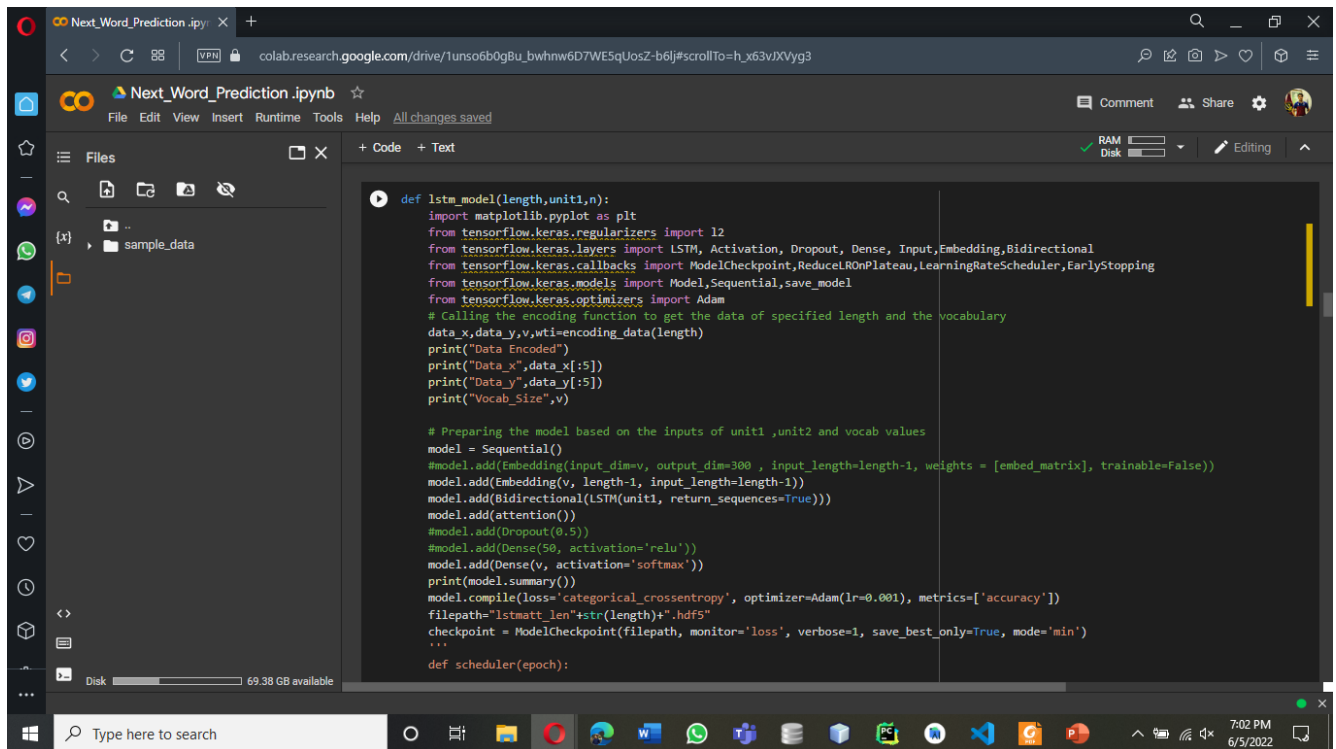


Figure 16: LSTM Model

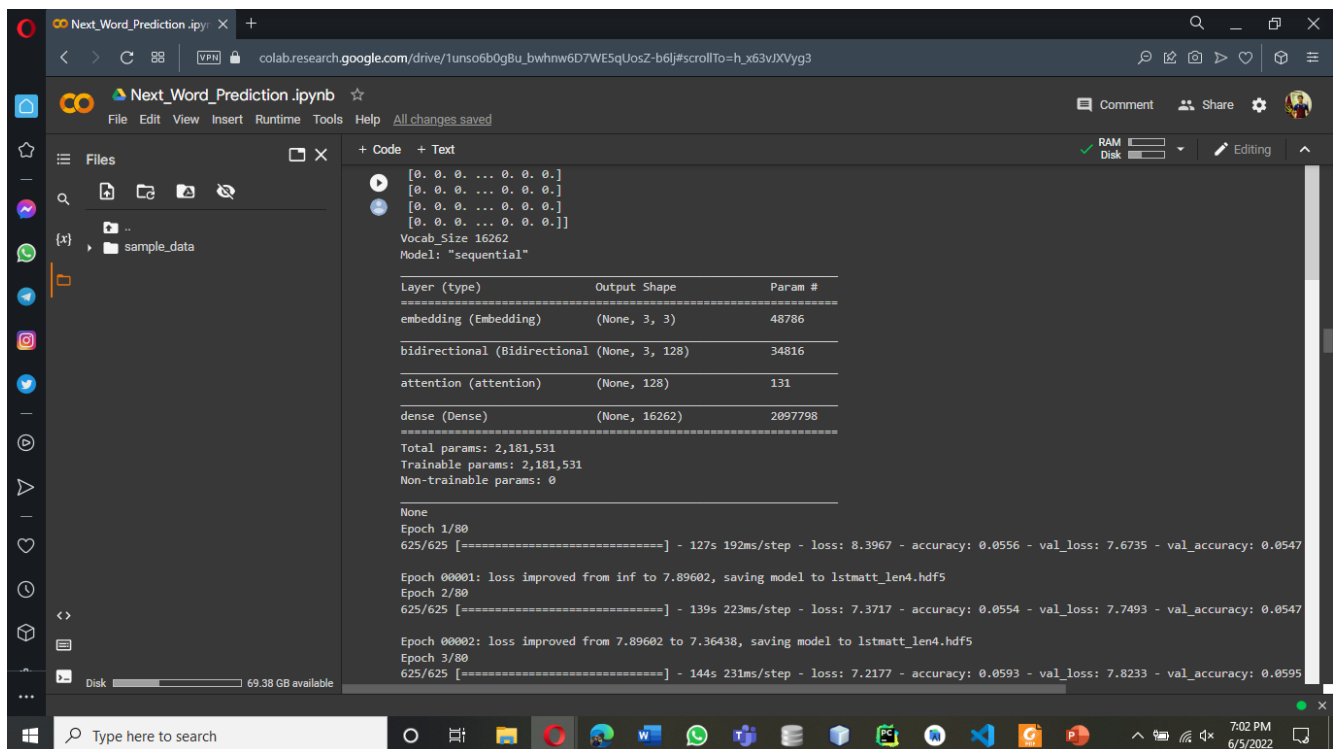


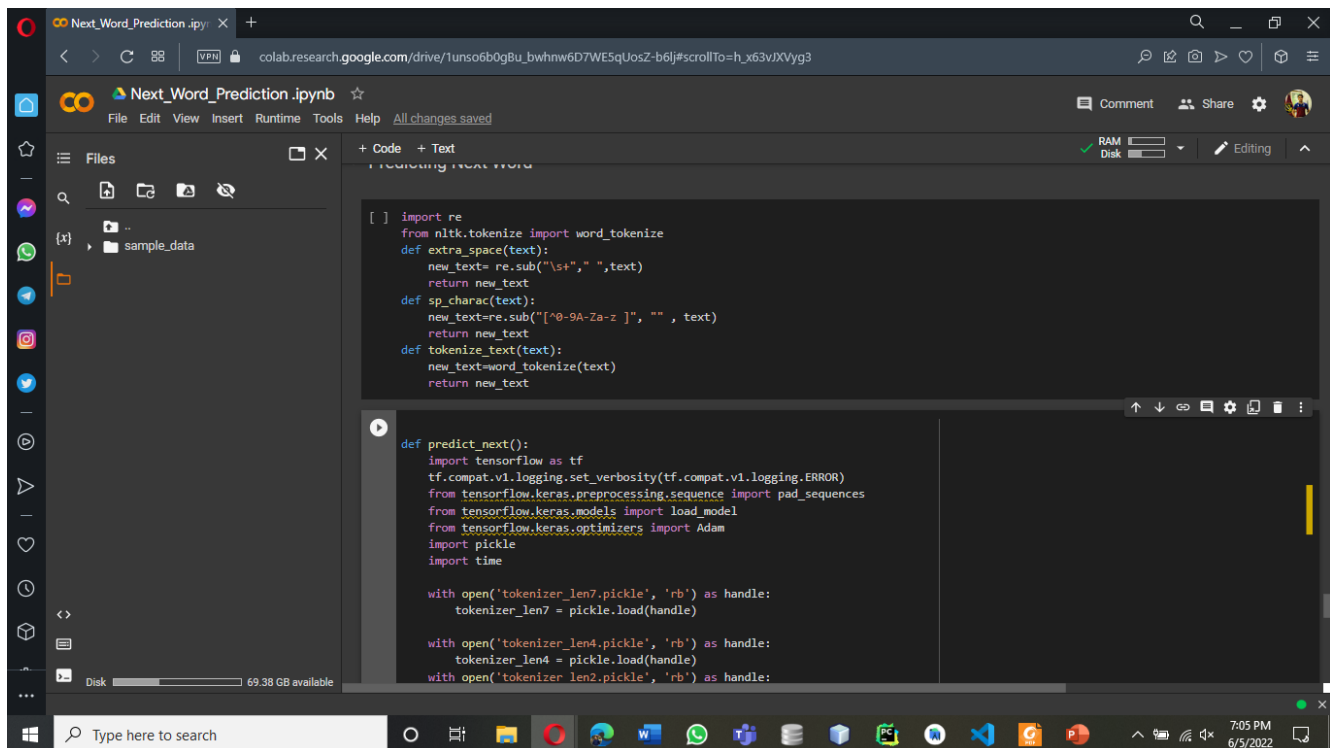
Figure 17: LSTM Model Output and Epoch



```
[ ] len2_x,len2_y=create_testdata2(start,temp_length,2)
model_len2.evaluate(x=len2_x,y=len2_y)
del len2_x,len2_y

10000 tokens done
20000 tokens done
30000 tokens done
40000 tokens done
16397
1563/1563 [=====] - 67s 37ms/step - loss: 7.0989 - accuracy: 0.0850
```

*Figure 18: Error Analysis for 1 Word*



```
[ ] import re
from nltk.tokenize import word_tokenize
def extra_space(text):
    new_text= re.sub("\s+"," ",text)
    return new_text
def sp_charac(text):
    new_text=re.sub("[^0-9A-Za-z ]", "", text)
    return new_text
def tokenize_text(text):
    new_text=word_tokenize(text)
    return new_text

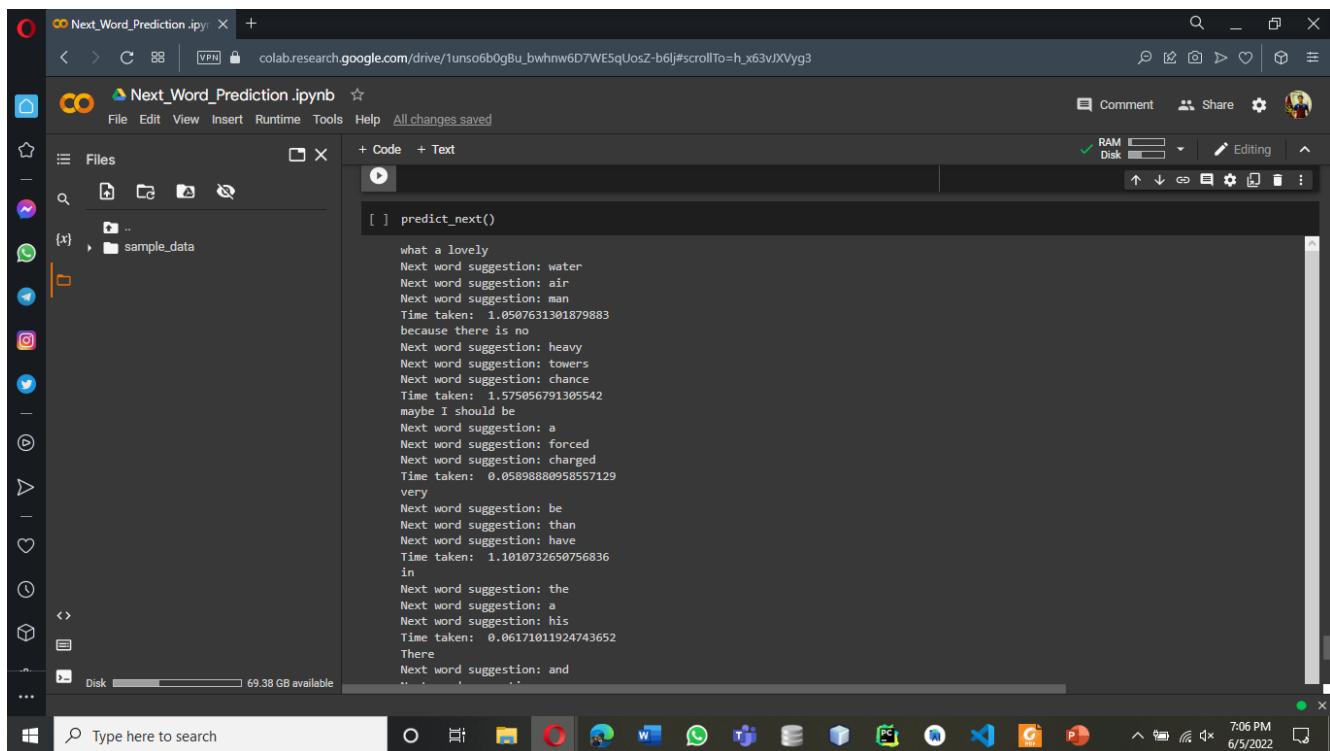
def predict_next():
    import tensorflow as tf
    tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    from tensorflow.keras.models import load_model
    from tensorflow.keras.optimizers import Adam
    import pickle
    import time

    with open('tokenizer_len7.pickle', 'rb') as handle:
        tokenizer_len7 = pickle.load(handle)

    with open('tokenizer_len4.pickle', 'rb') as handle:
        tokenizer_len4 = pickle.load(handle)

    with open('tokenizer_len2.pickle', 'rb') as handle:
```

*Figure 19: Prediction Code*



The screenshot shows a Google Colab notebook titled "Next\_Word\_Prediction.ipynb". The left sidebar displays a file explorer with a folder named "sample\_data". The main code editor area contains a single cell with the following output:

```
[ ] predict_next()

what a lovely
Next word suggestion: water
Next word suggestion: air
Next word suggestion: man
Time taken: 1.0507631301879883
because there is no
Next word suggestion: heavy
Next word suggestion: towers
Next word suggestion: chance
Time taken: 1.575056791305542
maybe I should be
Next word suggestion: a
Next word suggestion: forced
Next word suggestion: changed
Time taken: 0.0589880958557129
very
Next word suggestion: be
Next word suggestion: than
Next word suggestion: have
Time taken: 1.1010732650756836
in
Next word suggestion: the
Next word suggestion: a
Next word suggestion: his
Time taken: 0.06171011924743652
There
Next word suggestion: and
```

The bottom of the image shows the Windows taskbar with various application icons and the system clock indicating 7:06 PM on 6/5/2022.

*Figure 20: Result*

**THE END**