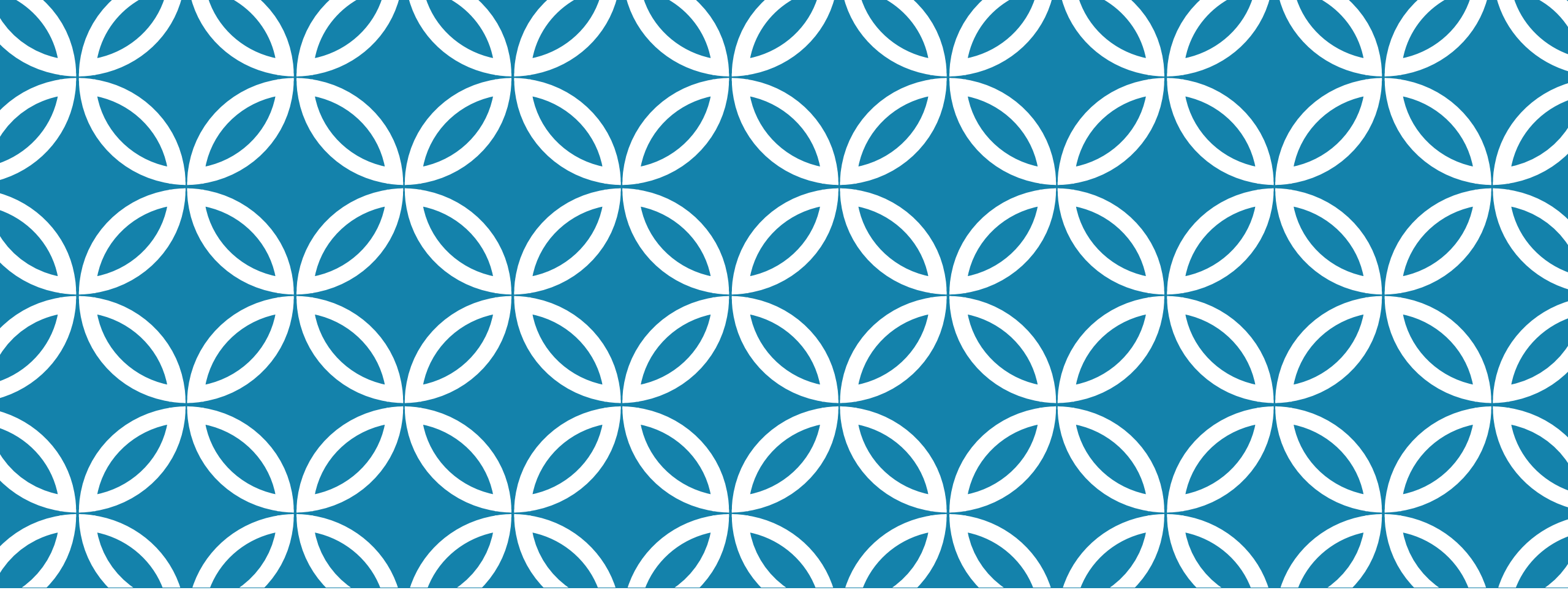


PREDICTIVE TEXT FOR CONTEXT ANALYZATION

Gadicherla Sameer
01FB15ECS101
G Manasa
01FB15ECS099
UE15CS333 project submission

SUBMISSION GUIDELINES

1. Since ESA time table in preponed (starting on 9th May), last date of submission is 6th May 2018 (Sunday) EOD. No extension will be granted.
2. Dataset, working ipython notebook and this presentation have to be submitted as a zip file (only one zip file per team). Name the zip file appropriately. You can either upload in Google drive or mail to bhaskarjyoti01@gmail.com
3. The final marks awarded by the evaluator **will be based on the following factors.**
   a) Working code that delivers result : 12
   b) Visualization, metric, comparison : 3
   c) Nature of dataset : 2
   d) Novelty of idea / publishability if completed : 3

ABOUT THE PROJECT

Project idea: To create a system which, given input as a part of a complete English sentence, can predict the next word in the sentence or generate the next sequence of words using Markov Chains and RNNs.

Why you think this can be turned into a publishable work:

This can be turned into a publishable work because analyzing the context of any text for a given situation is done unbiased without any human intervention. Usually this is done character by character but we are doing it word by word , which implies the dependency on previous words is captured than the previous character, which helps in remembering the context better because we train the RNN on the words directly. We will use this model in future to predict musical compositions of a particular singer.(This can be acheived even by training it on note by note in music as character by character)

LITERATURE SURVEY AND EXISTING WORK

Serial No	Title, author, web URL (if any)	What was useful as a reference
1	https://en.wikipedia.org/wiki/Language_model	We came to know what language models are exactly
2	https://en.wikipedia.org/wiki/Markov_property	Markovian property
3	https://en.wikipedia.org/wiki/Markov_chain	Markov chains
4	https://en.wikipedia.org/wiki/Recurrent_neural_network	RNN basics
5	Bengio, Yoshua, Simard, Patrice, and Frasconi, Paolo. Learning long-term dependencies with gradient descent is difficult. Neural Networks, IEEE Transactions on, 5(2):157–166, 1994.	Short term memory to train gradient descent
6	Dauphin, Yann N, de Vries, Harm, Chung, Junyoung, and Bengio, Yoshua. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. arXiv preprint arXiv:1502.04390, 2015.	Rmsprop optimization though we used adam
7	Graves, Alex. Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850, 2013.	Basic idea on how to generate sequences
8	http://cs229.stanford.edu/proj2015/054_report.pdf	Adam optimization and normalizations

DATASET SOURCE AND PREPROCESSING DONE

1. Dataset source :
   - For markov chains :
     1. Corpus of Contemporary American English (COCA) from here

        - About Dataset: The Corpus of Contemporary American English (COCA) is the largest freely-available corpus of English, and the only large and balanced corpus of American English.
        - Attributes: We will be using a subset of COCA mainly from here. Hence, we will be using two, three, four and five-gram inputs to train the model each time.
        - Instance count: Each of the n-gram datasets contains the following number of n-grams:
          - 2-gram: 1,020,386
          - 3-gram: 1,020,010
          - 4-gram: 1,034,308
          - 5-gram: 1,044,269
        - This was the primary dataset for testing the goodness of fit for every other dataset as it was exhaustive. We also trained using this dataset for next-word prediction as well as generating text.

1. Wikipedia Corpus
   - About dataset: This corpus contains the full text of Wikipedia, and it contains 1.9 billion words in more than 4.4 million articles.
   - Attributes: Wikipedia Corpus contains Wikipedia pages scraped and dumped into text files. Since it is only for testing, we used random articles with a total of 2286765 words.

2. RNN and LSTM:
   1. Narendra Modi speech in US : This has around 1000+ lines and is taken from a news website
   2. Narendra Modi speech on gst: This has around 2500+ lines and is also taken from a blog on web.
   3. Harry Potter Novel : It is the first book of the Harry potter series with has around 7000 lines and is has around 57 distinct words.

DATASET SOURCE AND PREPROCESSING DONE(Cont.)

1. Pre-processing steps performed
    1. Wikipedia Corpus:
        - Read all the lines of the input file.
        - Remove all unwanted characters and punctuations.
        - Remove lines with very few words.
        - Convert everything to lower-case.
        - Run a sliding window on the words

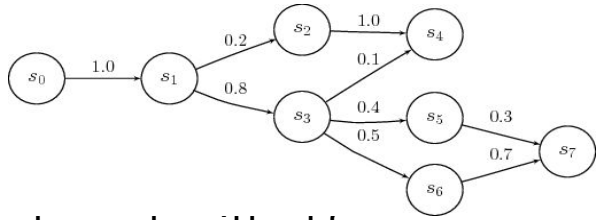
    2. Narendra Modi Speeches:
        - Same as for Wikipedia but further,
        - Distinct number of words are found
        - All words are represented as numbers
        - All sentences are represented as a sequence of numbers
        - All the sequences are padded to the largest sequence size wth zeroes

MARKOV CHAINS

HIGH LEVEL DESIGN OF OUR IMPLEMENTATION

MARKOV CHAIN APPROACH:

- We create the training set by creating (n-gram , n+1 word , count) pairs and save these pairs in the SQL database
- Then as said in pre-processing we also use the Wikipedia corpus, run. A sliding window and save the input,output pairs , which can also be used as training dataset but we are just using to test our model
- So , in order to predict the next word , we give a simple SQL query to the database and return an ordered dictionary of word , count pairs which the client.py file will help us calculate the probability(if needed).
- We have used the databse approach just because , it is very fast compared to vanilla lists , and practically impossible when the size of the text moves into GBs.



Note : Both the dataset and latest jupyter notebook (python 3.0) have to be submitted tor evaluation

PSEUDO-CODE

Create bi-grams:

```
>>> s = "I am Sam. Sam I am. I do not like green eggs and ham."
>>> tokens = s.split(" ")
>>> bigrams = [(tokens[i], tokens[i + 1]) for i in range(0, len(tokens) - 1)]
>>> bigrams[('I', 'am'), ('am', 'Sam.'), ('Sam.', 'Sam'), ('Sam', 'I'), ('I', 'am.'), ('am.', 'I'), ('I', 'do'), ('do', 'not'), ('not', 'like'), ('like', 'green'), ('green', 'eggs'), ('eggs', 'and'), ('and', 'ham.')]
```

Markov Chain: `I am Sam.`

```
{
    'I': ['am'],
    'am': ['Sam.'],
}
```

Add `Sam I am.`

```
{
    'I': ['am', 'am.'],
    'am': ['Sam.'],
    'Sam': ['I'],
}
```

Thus, give an input `'am'`, we get `'Sam.'`

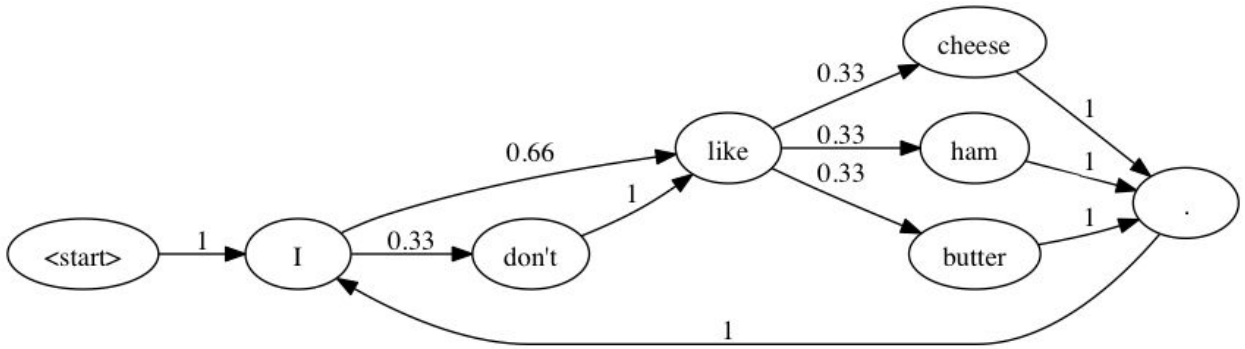


Figure 2 An example of fully modeled Markov Chain

RNN AND LSTM

RNN basics

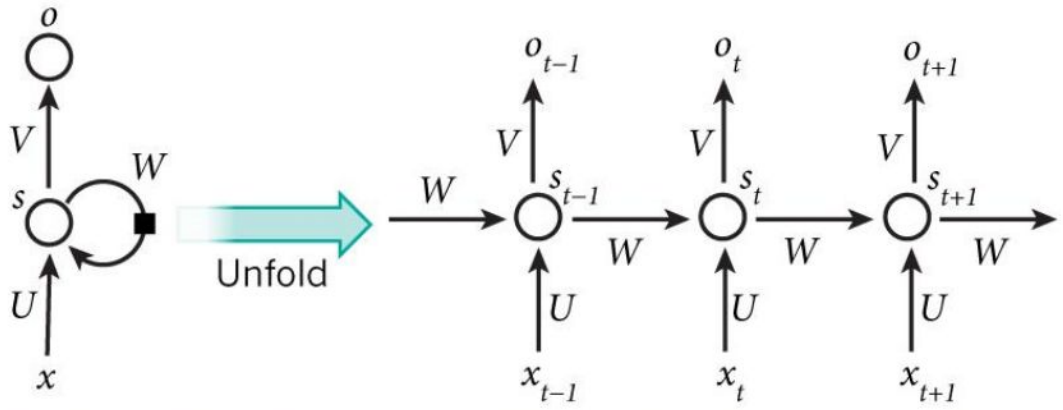


Figure 2 A typical RNN looks like one shown above. A RNN is unrolled (or unfolded) over time into a full network.

RNNs make use of sequential information. A traditional neural network assumes that all inputs (and outputs) are independent of each other.

Since our project is to predict text, it is crucial to know the information from previous states, i.e. the previous words to predict the next word.

RNNs are called recurrent because they perform the same task for every element of a sequence, with the output depending on the previous computations.

RNNs thus have a “memory” element to capture information from previous calculations.

Need for LSTM and its advantages over RNN:

Two major problems with RNN: Vanishing and Exploding gradients.

- Vanishing Gradients: In traditional RNNs the gradient signal can be multiplied a large number of times by the weight matrix. Thus, the magnitude of the weights of the transition matrix can play an important role. If the weights in the matrix are small, the gradient signal becomes smaller at every training step, thus making learning very slow or completely stops it. This is called vanishing gradient.
- Exploding Gradients: refers to the weights in this matrix being so large that it can cause learning to diverge.

LSTM stands for Long Short Term Memory, is a special kind of RNN that learns long-term dependencies. The memory cell of LSTM is composed of four elements: input, forget and output gates and a neuron that connects to itself.

CHARACTER MODEL

Design For the Character Model

In order to train our recurrent neural network, we use a large text file (we have used the Harry Potter books) as input. This large text file undergoes preprocessing and is stored in the form of two arrays X and Y defined as:

X:

which is a three dimensional array of the form X[i, t, char_indices[char]] where i is the index of each sentence, t is the index of each letter in each sentence and char_indices[char] indicates the letter encountered. Therefore, X is like the given input matrix.

Y:

which is a two dimensional array of the form Y[i, char_indices[next_chars[i]]] where i is the index of each sentence and char_indices[next_chars[i]] indicates the letter which is encountered after encountering SEQUENCE_LENGTH (set to 40) characters of the sentence. Therefore, Y is like the expected output matrix.

We iterate through the text file at a step size of 3 characters, scanning through all characters, till SEQUENCE_LENGTH (store in list of sentences) and storing the next expected character after SEQUENCE_LENGTH (store in list of next_char). This is then used in the creation of the X and Y matrices.

Model Architecture

We train the following model on the input format specified before:

```
model = Sequential()
model.add(LSTM(128, input_shape=(SEQUENCE_LENGTH, len(chars))))
model.add(Dense(len(chars)))
model.add(Activation('softmax'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 128)               86528
_________________________________________________________________
dense_2 (Dense)              (None, 40)                5160
_________________________________________________________________
activation_2 (Activation)    (None, 40)                0
=================================================================
Total params: 91,688
Trainable params: 91,688
Non-trainable params: 0
_________________________________________________________________
```

Our model is trained for many number of epochs till 120 using RMSProp optimizer and uses 5% of the data for validation.

WORD BY WORD MODEL

LSTM APPROACH FOR WORD BY WORD

1. After all the pre-processing as mentioned in the pre-processing slide.
2. Then we create a RNN model which is actually an LSTM preserved model with following architecture and train/use this with two approaches.

```
In [12]: model = Sequential()
         model.add(Embedding(vocab_size,128, input_length=max_length-1))
         model.add(LSTM(200, recurrent_dropout = 0.2, dropout = 0.2))
         model.add(Dense(vocab_size, activation='softmax'))
         print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 57, 128)           64640
_________________________________________________________________
lstm_1 (LSTM)                (None, 200)               263200
_________________________________________________________________
dense_1 (Dense)              (None, 505)               101505
=================================================================
Total params: 429,345
Trainable params: 429,345
Non-trainable params: 0
_________________________________________________________________
None
```

APPROACHES

1. Whole sentence is used as X (input) and the next word is the output (y). This approach is better than just the previous word. That is how the data to train is created.

```
# split into input and output elements
sequences = array(sequences)
X, y = sequences[:,:-1],sequences[:,-1]
y = to_categorical(y, num_classes=vocab_size)
```

2. In this approach we create the dataset , by training it as X which has the previous two words and the y has the next output word.

```
sequences = list()
for i in range(2, len(encoded)):
    sequence = encoded[i-2:i+1]
    sequences.append(sequence)
print('Total Sequences: %d' % len(sequences))
print("\n",sequences)
```

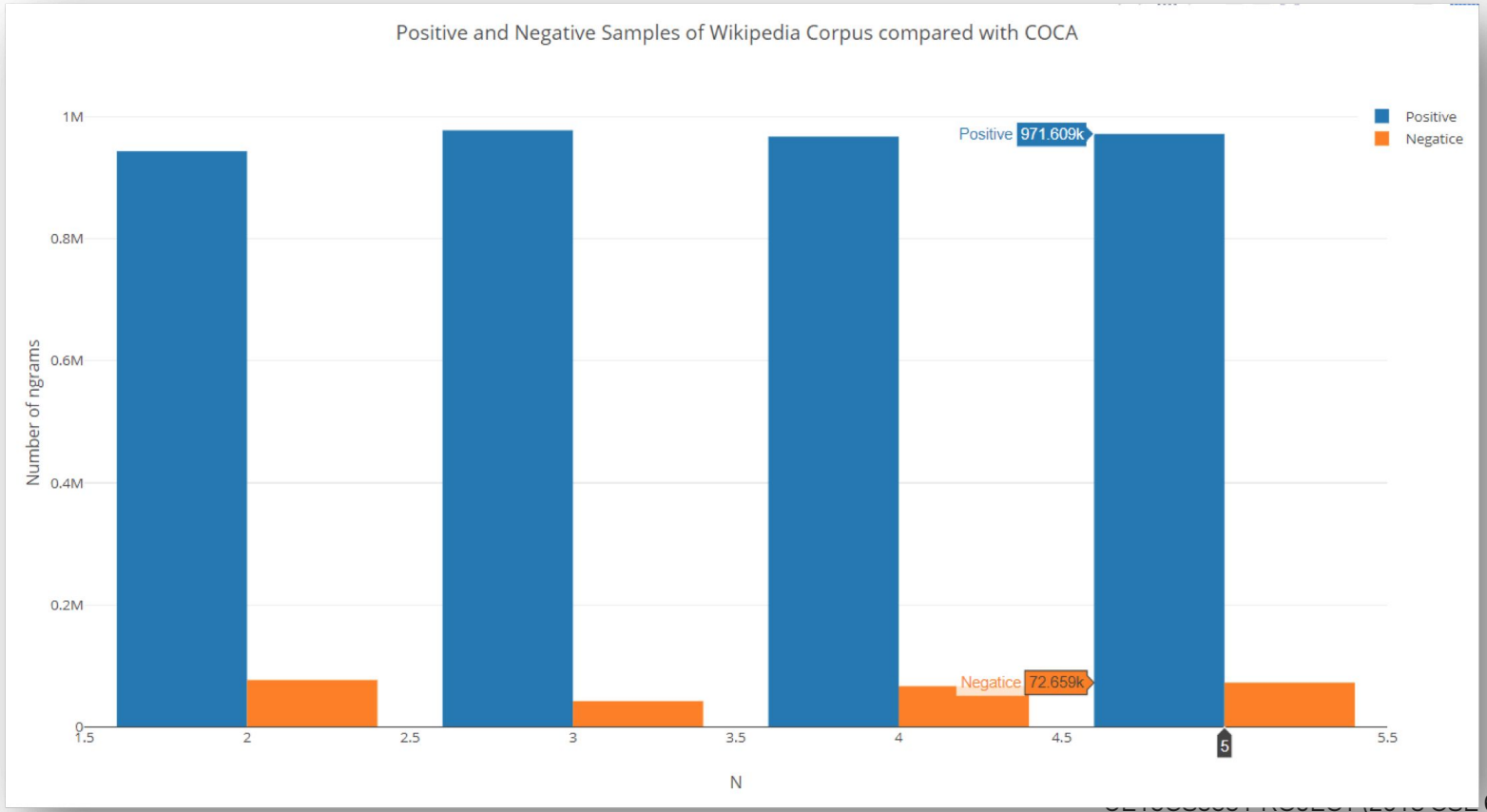
```
Total Sequences: 1192

 [[148, 149, 150], [149, 150, 151], [150, 151, 152], [15
 33, 155], [33, 155, 77], [155, 77, 1], [77, 1, 15], [1,
 [34, 156, 157], [156, 157, 50], [157, 50, 51], [50, 51,
```

RESULTS

RESULTS, VISUALIZATION, METRIC

Markov Chain

Positive vs Negative samples of Wikipedia with COCA

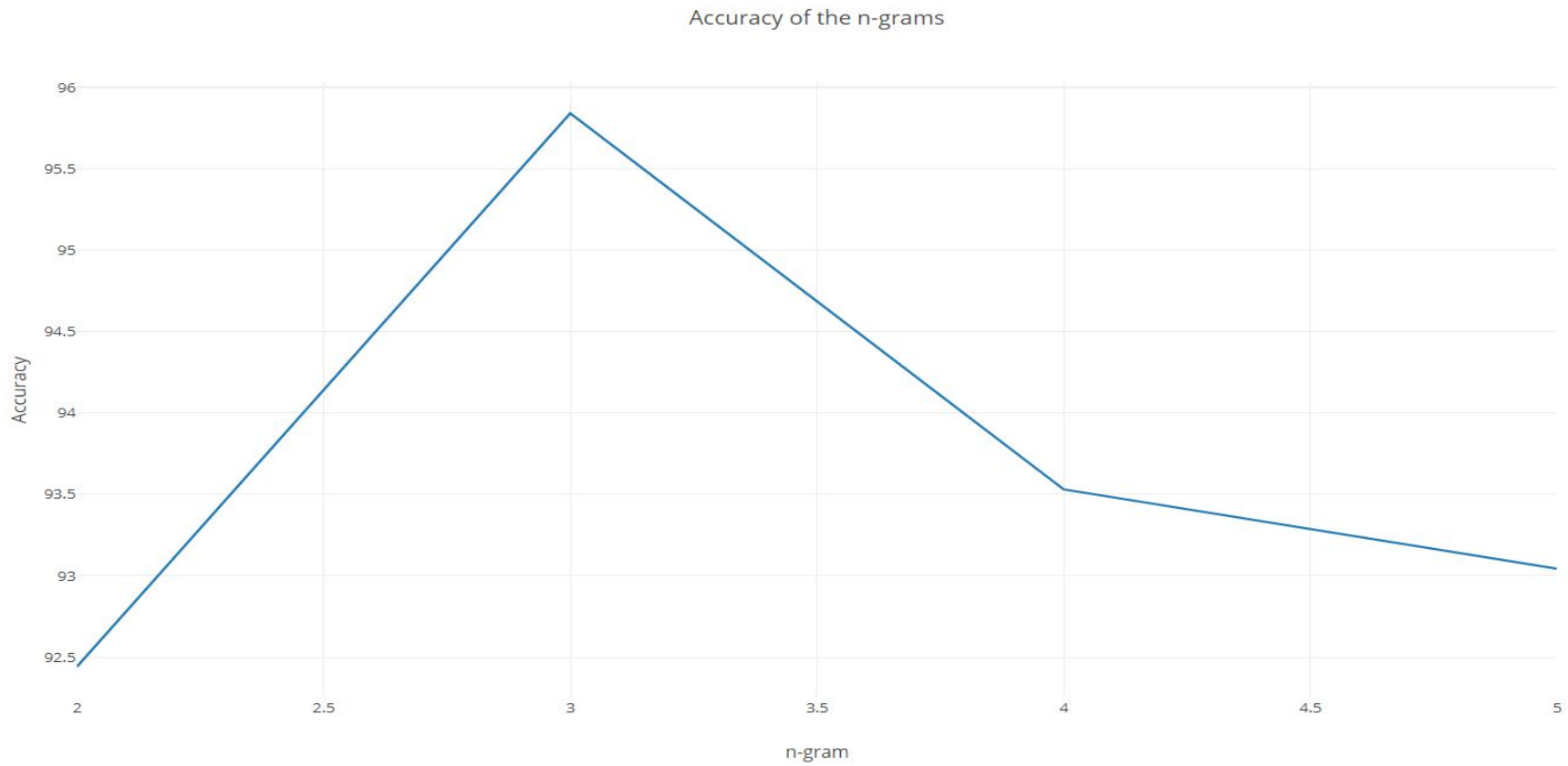


Coca tested over Wikipedia

Table 1 Classification Accuracy and Error Rate for Wikipedia Corpus on COCA Dataset

N-gram	classification accuracy	Error Rate
2	92.44	7.6
3	95.84	4.16
4	93.53	6.47
5	93.042	6.958

n-gram vs accuracy



Spoken and Written American English dataset accuracy

Table 2 Spoken and Written American English Dataset Accuracy

N-gram	w_acad	w_fic	w_mag	w_news	w_spok
2	67.89	62.83	66.74	68.91	67.84
3	72.97	69.27	68.76	71.98	70.29
4	71.67	66.54	67.45	70.27	69.92
5	70.23	67.78	67.23	69.89	69.02

```
@sameerg-Inspiron-3543: ~/SEM_6/NLP/Project/Markov
             ('for', 23),
             ('here', 20),
             ('this', 19),
             ('touch', 17),
             ('such', 16),
             ('or', 15),
             ('any', 14)])
favor of a new york city-based
sameerg@sameerg-Inspiron-3543:~/SEM_6/NLP/Project/Markov$ python3 markov.py -w database/big/w4.db -n 4 -words 100 --predict are you in
OrderedDict([('the', 114),
             ('a', 81),
             ('favor', 67),
             ('love', 54),
             ('there', 48),
             ('trouble', 30),
             ('pain', 25),
             ('your', 24),
             ('for', 23),
             ('here', 20),
             ('this', 19),
             ('touch', 17),
             ('such', 16),
             ('or', 15),
             ('any', 14)])
a minute and a
sameerg@sameerg-Inspiron-3543:~/SEM_6/NLP/Project/Markov$ python3 markov.py -w database/big/w4.db -n 4 -words 100 --predict are you in
OrderedDict([('the', 114),
             ('a', 81),
             ('favor', 67),
             ('love', 54),
             ('there', 48),
             ('trouble', 30),
             ('pain', 25),
             ('your', 24),
             ('for', 23),
             ('here', 20),
             ('this', 19),
             ('touch', 17),
             ('such', 16),
             ('or', 15),
             ('any', 14)])
the face and neck
sameerg@sameerg-Inspiron-3543:~/SEM_6/NLP/Project/Markov$
```

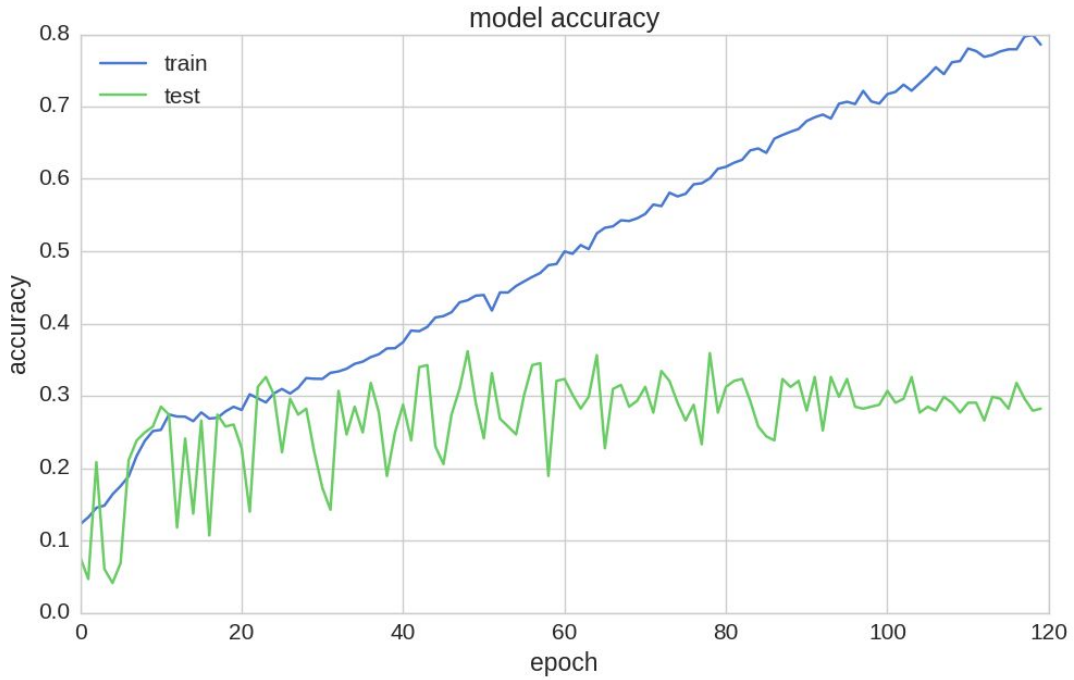
Outputs:

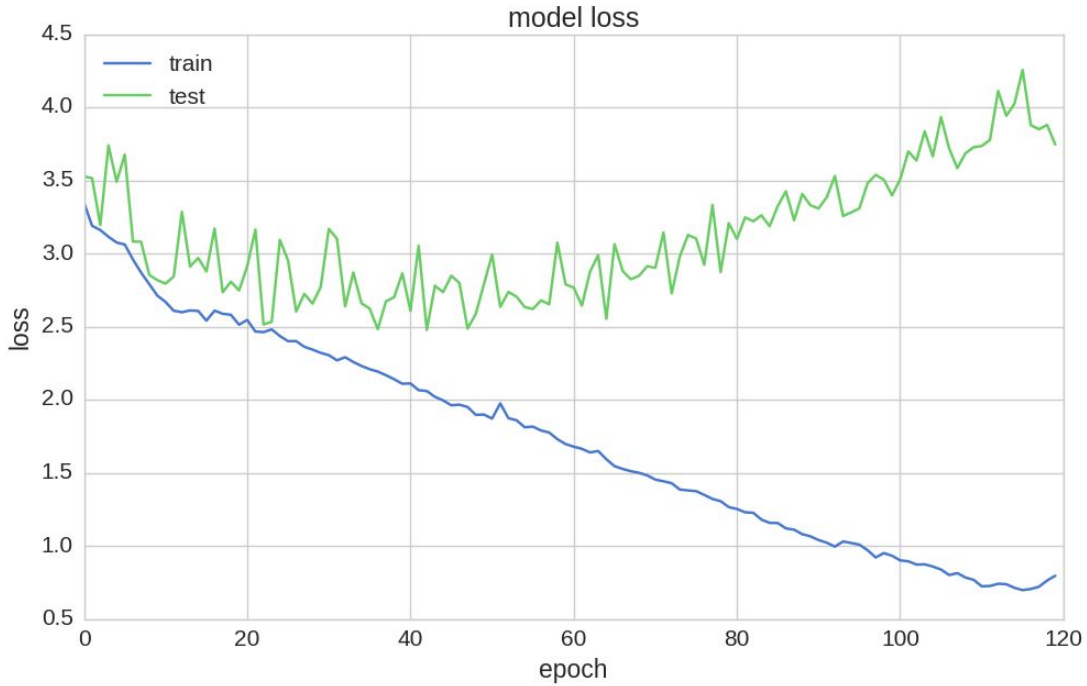
```
g@sameerg-Inspiron-3543: ~/SEM_6/NLP/Project/Markov
             ('committed', 26),
             ('denying', 26),
             ('friends', 26),
             ('talkin', 26),
             ('already', 25),
             ('any', 25),
             ('proposing', 25),
             ('tonight', 25)])
saying you have the same thing with me and my father had a chance for the last time i saw a man in the middle east peace conference on the floo
r of the most part they
sameerg@sameerg-Inspiron-3543:~/SEM_6/NLP/Project/Markov$ python3 markov.py -w database/big/w3.db -n 3 -words 100 --predict are you
OrderedDict([('going', 7993),
             ('doing', 6047),
             ('saying', 2096),
             ('talking', 2036),
             ('sure', 1942),
             ('a', 1524),
             ('all', 1235),
             ('ready', 1150),
             ('in', 924),
             ('still', 858),
             ('kidding', 824),
             ('okay', 729),
             ('looking', 709),
             ('trying', 699),
             ('there', 654),
             ('feeling', 613),
             ('getting', 562),
             ('committed', 26),
             ('denying', 26),
             ('friends', 26),
             ('talkin', 26),
             ('already', 25),
             ('any', 25),
             ('proposing', 25),
             ('tonight', 25)])
saying the words that are in a moment of silence in which the united states is the most important things to do it again and again to the point
of view and the united nations and other things to say that the government to the united kingdom and the united kingdom and the other day that
i have n't had the same way i was a very different than what you want me to do it for you in your life in the world and to the point is that th
e president and ceo john
sameerg@sameerg-Inspiron-3543:~/SEM_6/NLP/Project/Markov$
```

output for 100 words given a bigram , and trigram ,database w3.db is used, with starting by predicting the 3 gram , then last two words of the tri gram is used to predict the next 3-gram till 100 words are produced.

RNN using character approach:

First we trained by predicting character by character and achieved an accuracy of 78.57% over 120 epochs but then, this didn't suit our application and results weren't that satisfying





As we can see , here the results aren't that pleasing when seen with the actual sentence in the quotes list

In [42]:
```
quotes = ["Mr. Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly
"At half past eight, Mr. Dursley picked up his briefcase, pecked Mrs. Dursley on the cheek, and tried to kiss Dudley good-
,"he'd gotten into terrible trouble for being found on the roof of the school kitchens"]
```

In [43]:
```
for q in quotes:
    seq = q[:40].lower()
    print(seq)
    print(predict_completions(seq, 5))
    print()
```

```
mr. dursley was the director of a firm c
['lout... ', 'er ', 'as ', 'houlded ', 'it ']

at half past eight, mr. dursley picked u
['itew ', 'phow ', 'she ', 'moffed ', 'whordind ']

he'd gotten into terrible trouble for be
['chouddare. ', 'lhed ', 'theded ', 'eked ', 'kne ']
```

In [ ]:

Results of LSTM for word by word:

Approach / Dataset	US Speech	GST Speech
3 Word approach	87.28%	80.33%
Whole sentence approach	96.28%	95.60%

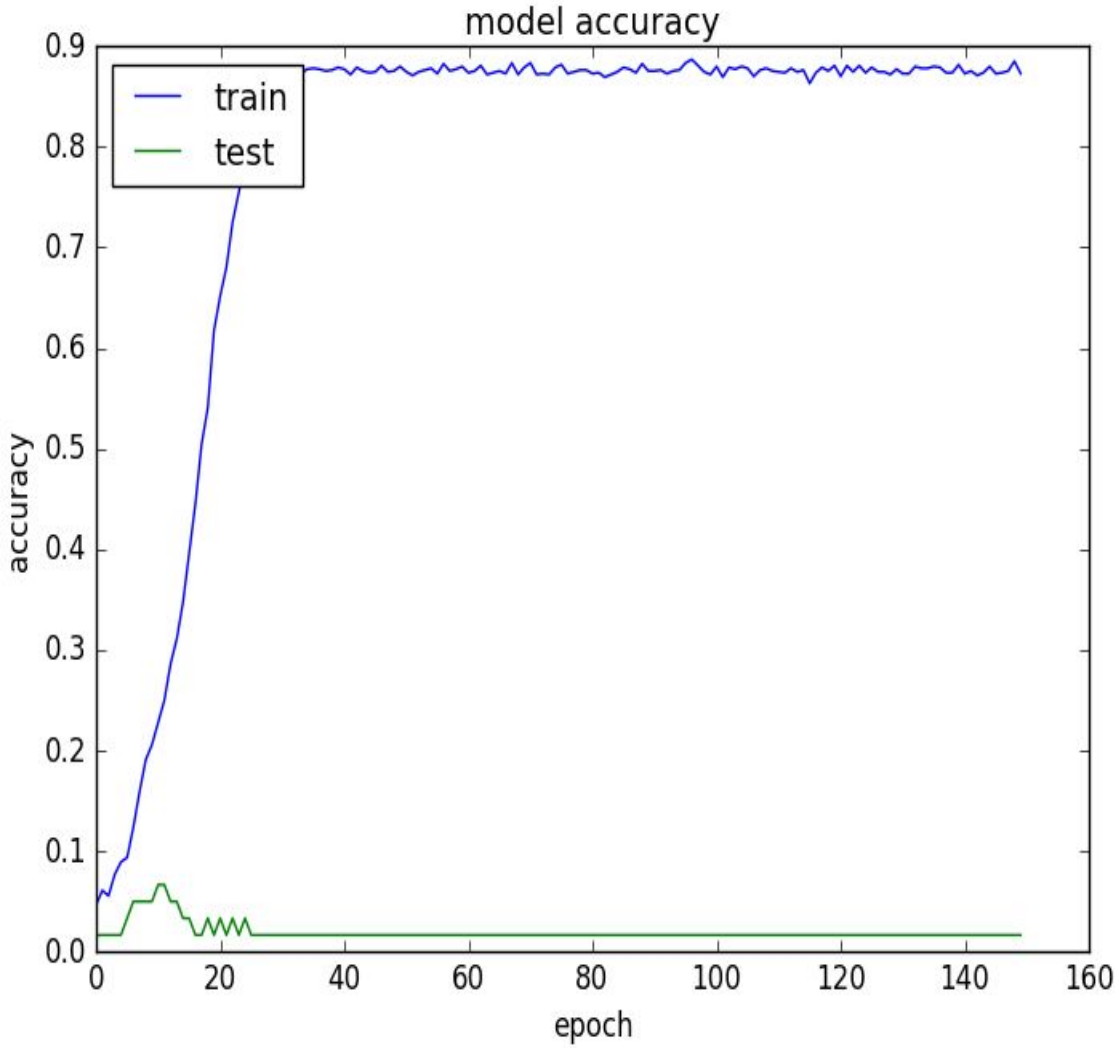
Model Accuracy on Modi's Speeches after 150 epochs

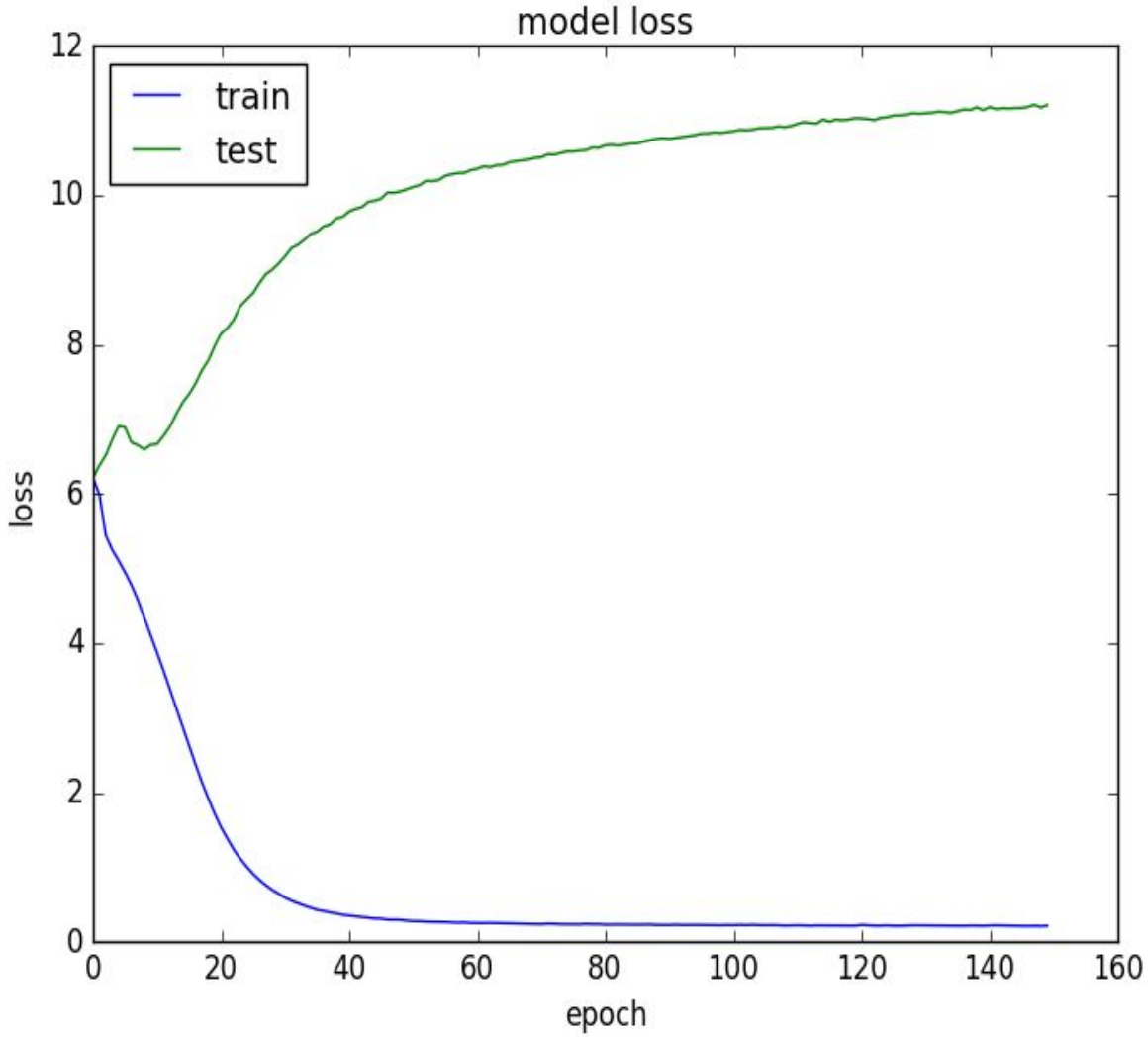
Model Accuracies (Adam VS RMSprop) for Harry Potter Novel after 120 epochs

Approach / Optimizer	Adam	RMSprop
3 Word approach	56.44%	37.74%

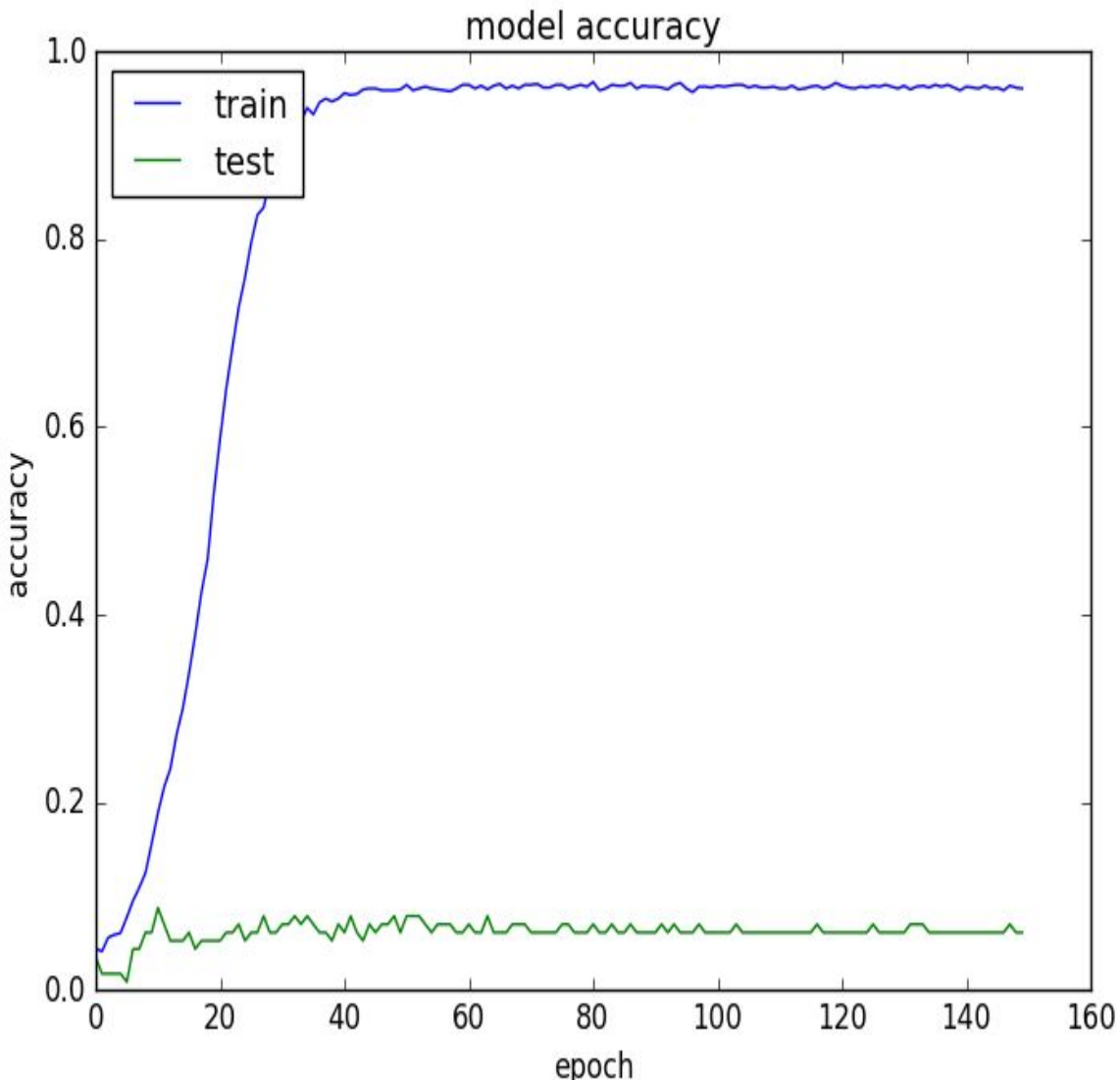
Epochs vs Accuracy/loss :

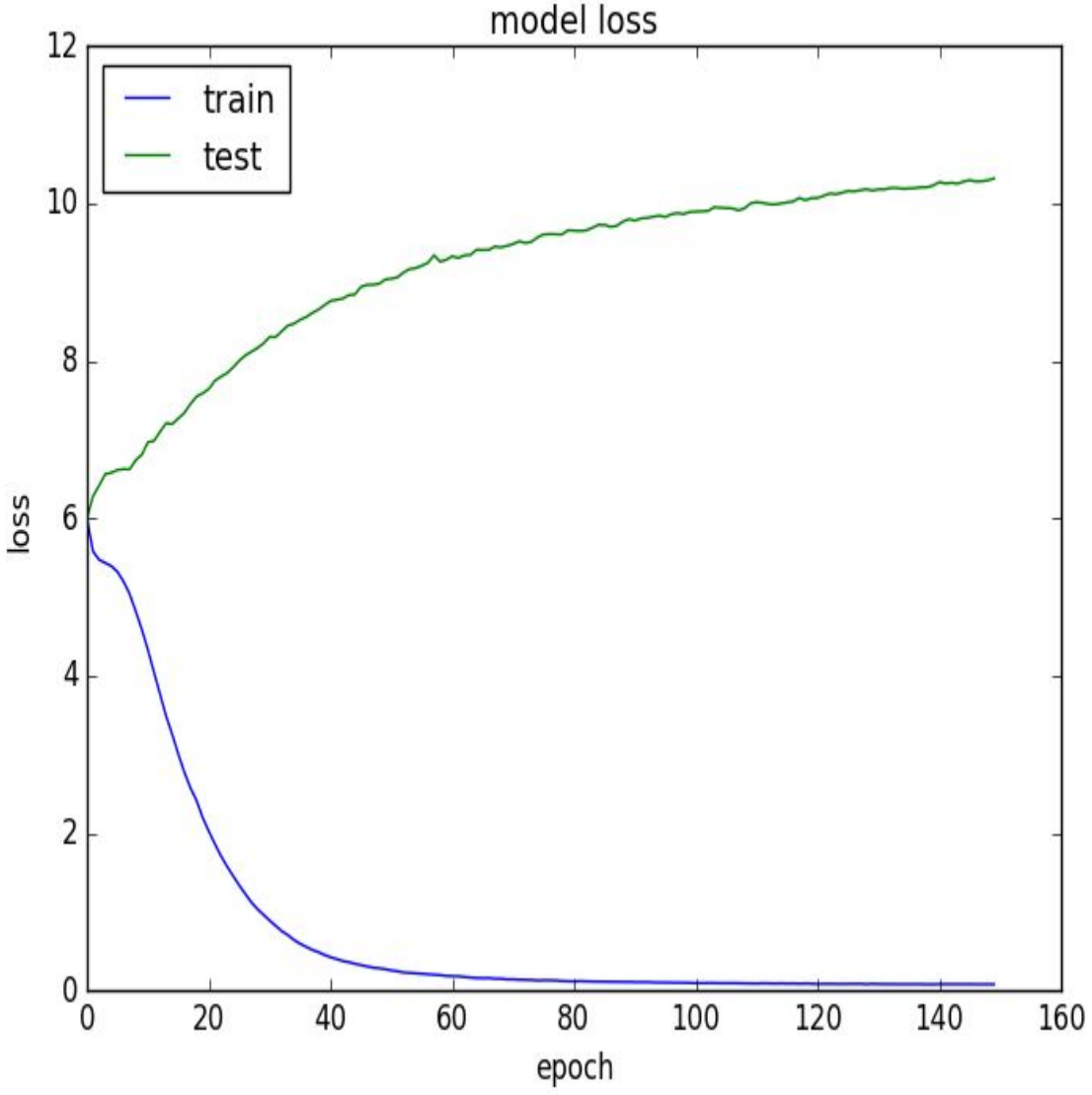
3 Word - US Speech





Sentence - US Speech





```
print(generate_seq(loaded_model, tokenizer, 2, 'people', 2))
```

```
people in the
```

```
print(generate_seq(loaded_model, tokenizer, 2, 'United Nations', 50))
```

```
United Nations including its security council so that it carries greater credibility and legitimacy and will b
e more representative and effective in achieving our goals are comprehensive it gives priority to the future o
f the island states i speak about blue revolution which includes the prosperity sustainable use of marine weal
th and
```

Fig: Predictions on 3 Word Model for US Speech

```
print(generate_seq(loaded_model, tokenizer, 57, 'People', 2))
```

```
People in the
```

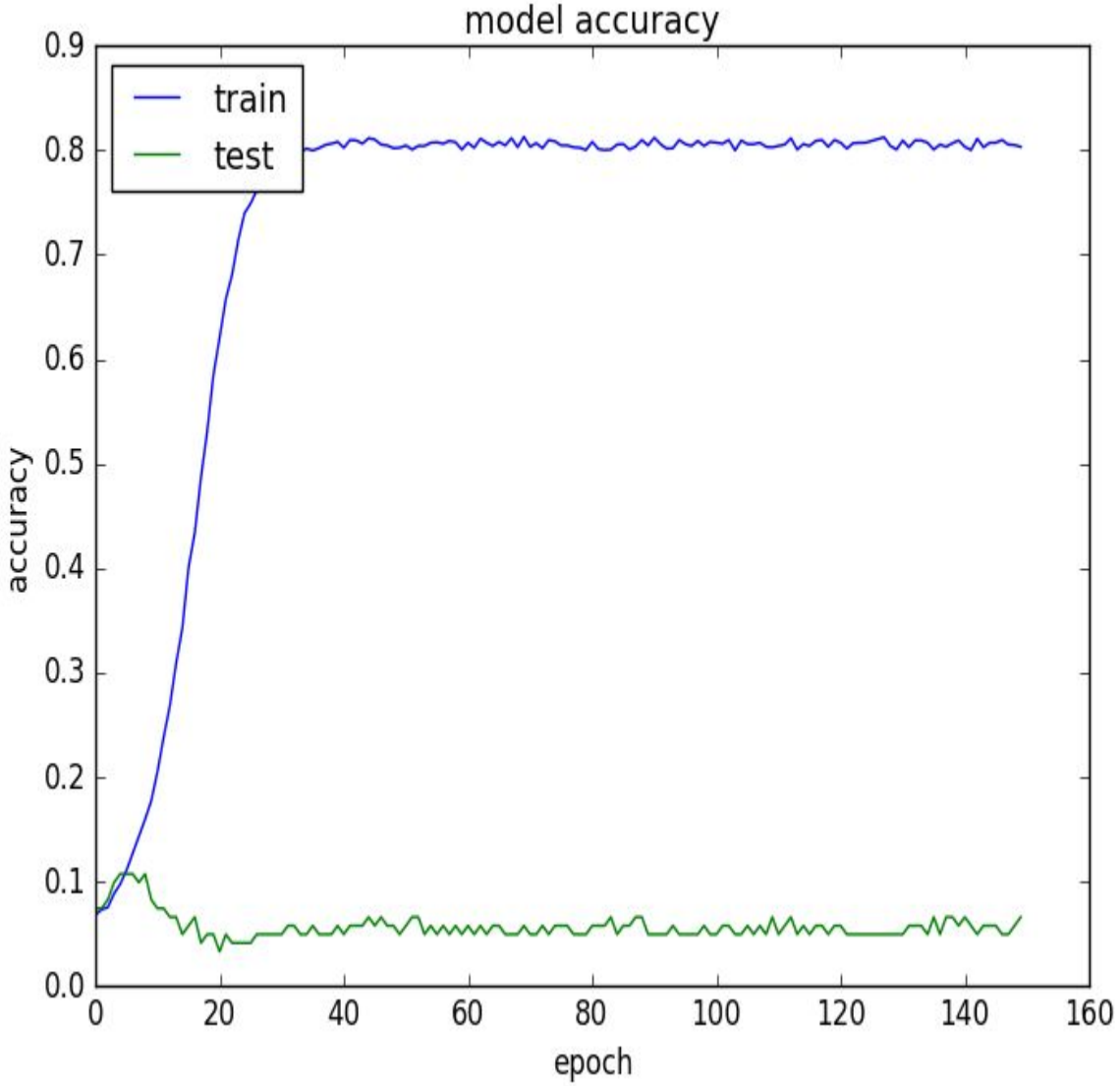
```
print(generate_seq(loaded_model, tokenizer, 57, 'United Nations', 50))
```

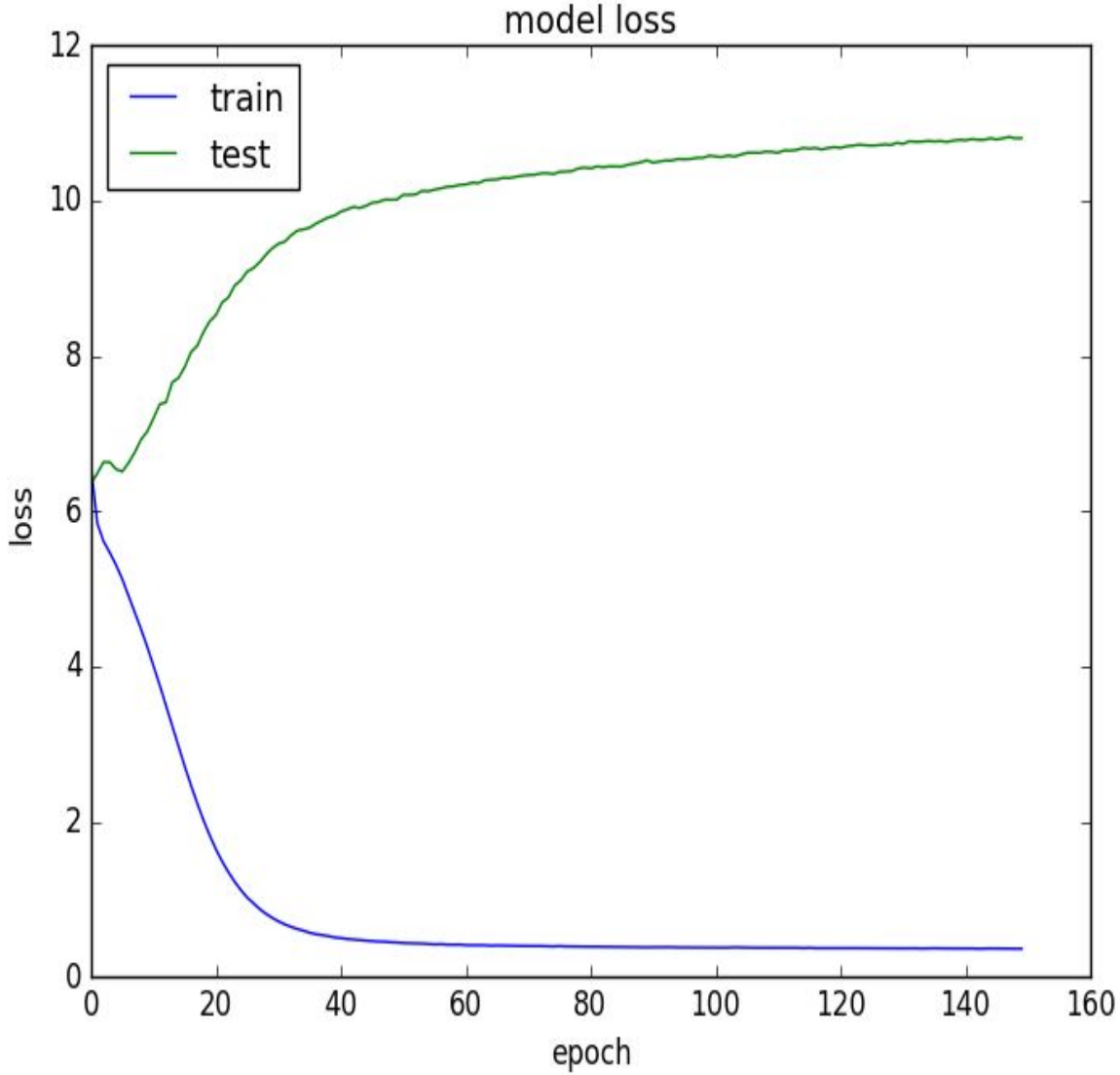
```
United Nations see now distance is no insulation from challenges and privations from distant lands states from
 the pacific to the atlantic of natural disasters and pension for everyone's sunset years energy efficiency a
tax on coal a huge afforestation programme reforming our transportation and cleaning up our cities and rivers
up
```

Fig: Predictions on Sentence Model for US Speech

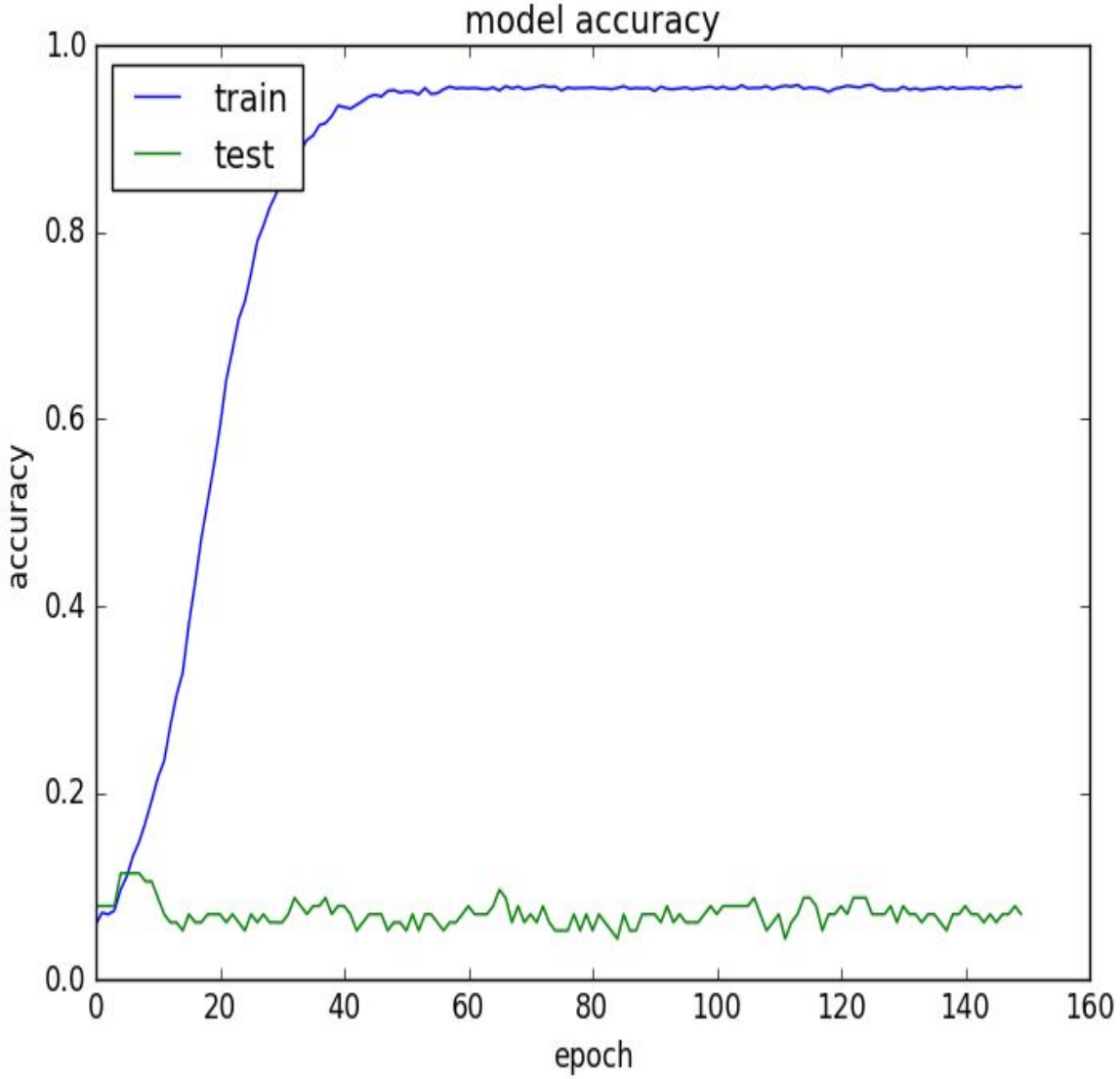
We see that both the models have pretty much given a meaningful set of words as output. But the seentence approach is close the actual data but the three word trade off looks very meaningful too.

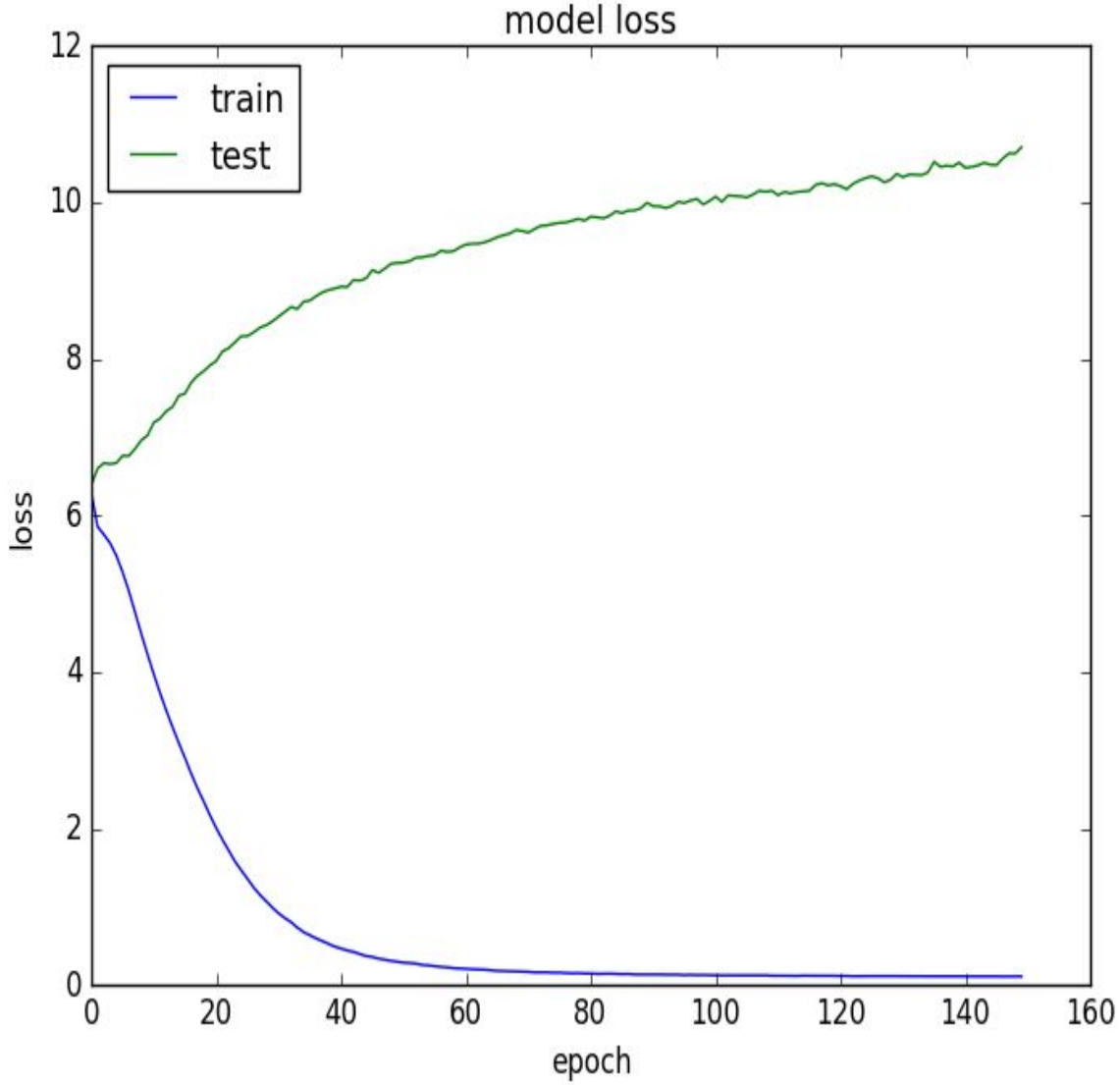
3 Word - GST


model accuracy
train
test
accuracy
epoch
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
20
40
60
80
100
120
140
160


model loss
train
test
loss
epoch
0
2
4
6
8
10
12
0
20
40
60
80
100
120
140
160

Sentence - GST





```
print(generate_seq(loaded_model, tokenizer, 2, 'important', 50))
```

```
important when the constitution was framed it introduced a framework of equal opportunity for development gst
is the dream of our country's poor as we expand horizontally we need to ensure vertical growth as well the wor
k in unison for india under gst for the first time the central service officers
```

```
print(generate_seq(loaded_model, tokenizer, 2, 'State Governments', 100))
```

```
State Governments there have been done for this new system but those arrangements are at basic level and the e
steemed conglomerate gathered here representing diverse sectors in the same product will have different prices
 in say delhi gurugram and noida which are 25 30 kilometres apart this is a landmark step towards economic ref
orms beyond the taxation revamp it is a landmark step towards economic reforms beyond the taxation revamp it i
s a landmark step towards economic reforms beyond the taxation revamp it is a landmark step towards economic r
eforms beyond the taxation revamp it is a landmark step towards economic reforms
```

Fig: Predictions on 3 Word Model for Speech on GST

```
print(generate_seq(loaded_model, tokenizer, 66, 'important', 50))
```

```
important when the constitution was framed it introduced a framework of equal opportunity and rights for all t
hose troubles as the poor so that the poor is benefitted just in a india of one of the biggest strength of fed
eral structure the gst reform be on the nation and accepted by
```

```
print(generate_seq(loaded_model, tokenizer, 66, 'State Governments', 100))
```

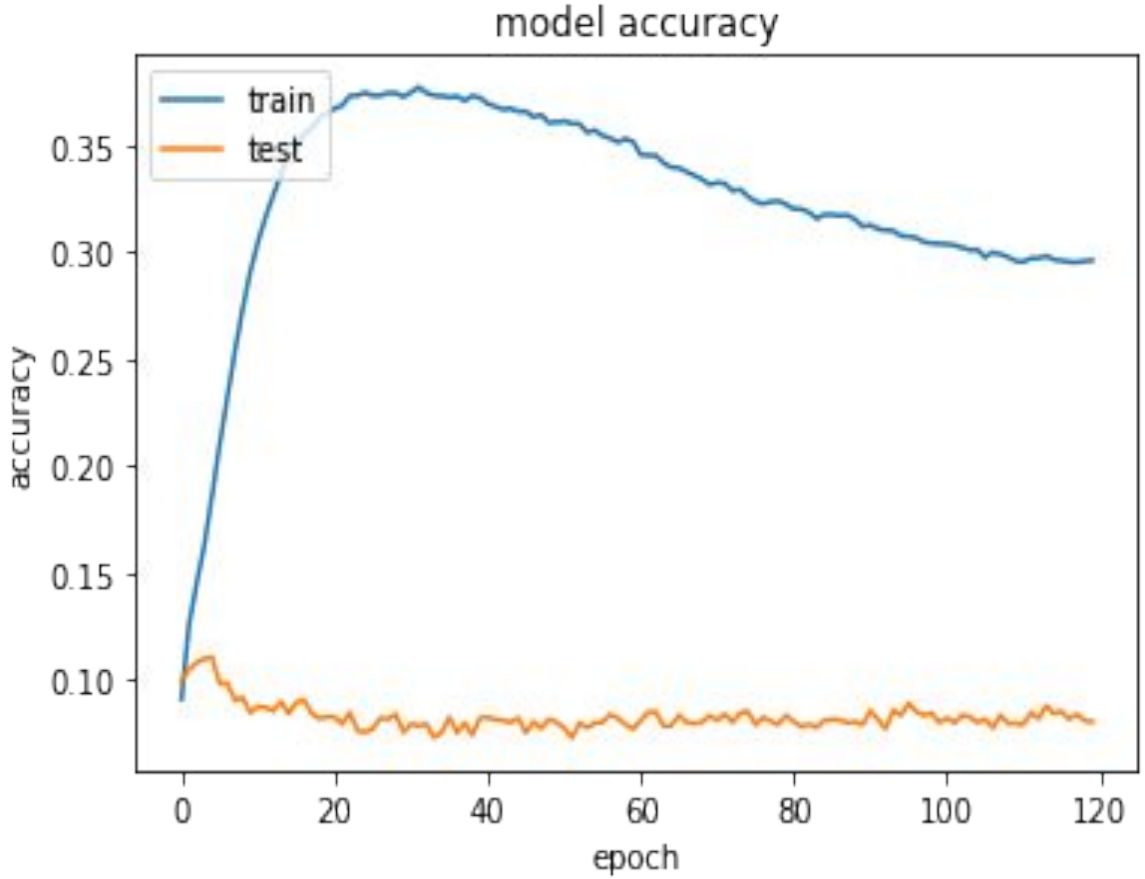
```
State Governments will be one nation one tax and tax system shall be executed in a standard manner in all the
states and they are slated to gain immensely as they shall now get equal opportunities of development and the
constitution was accepted by the nation and they will be shred off to have one nation one tax right from ganga
nagar to itanagar and from leh to enriched take take it resulted in take the example of the country participat
ed in the country's constitution was accepted by the nation attaining freedom on 26th november 1949 this house
 stands as a testimony to
```

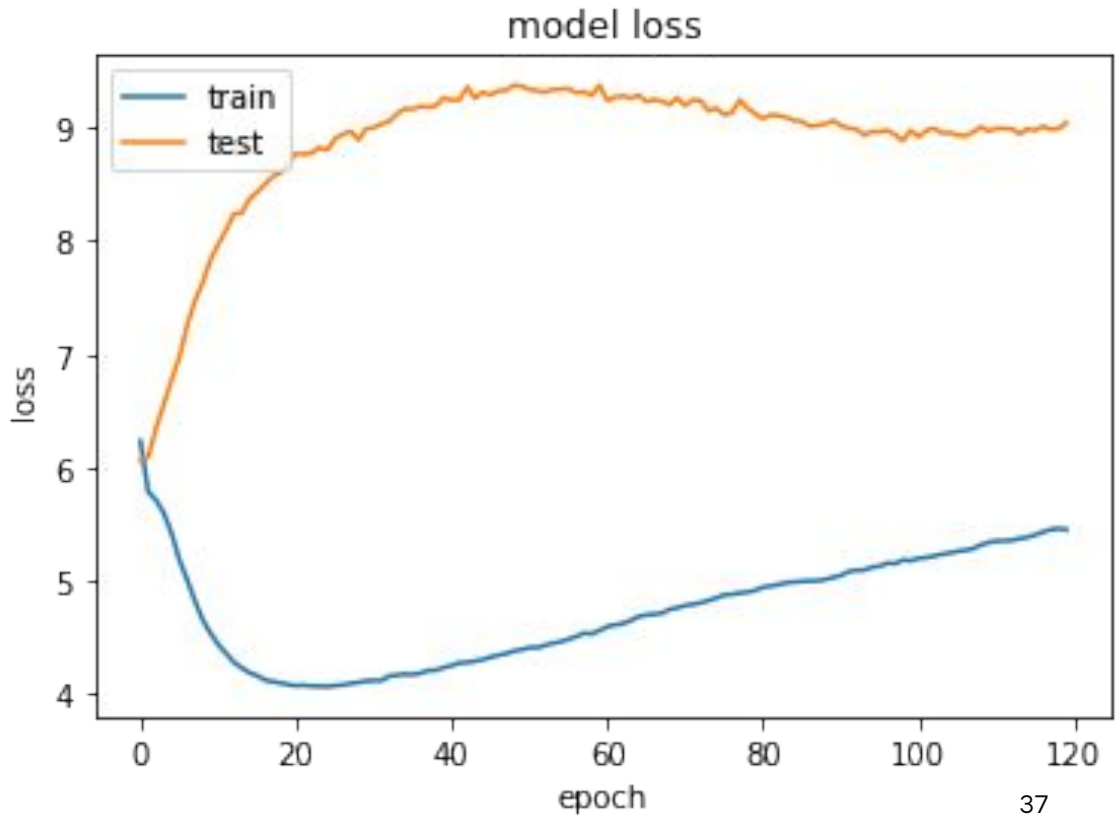
Fig: Predictions on Sentence Model for Speech on GST

Even here , we see that both the models have pretty much given a meaningful set of words as output. But the seentence approach is close the actual data but the three word trade off looks very meaningful too.

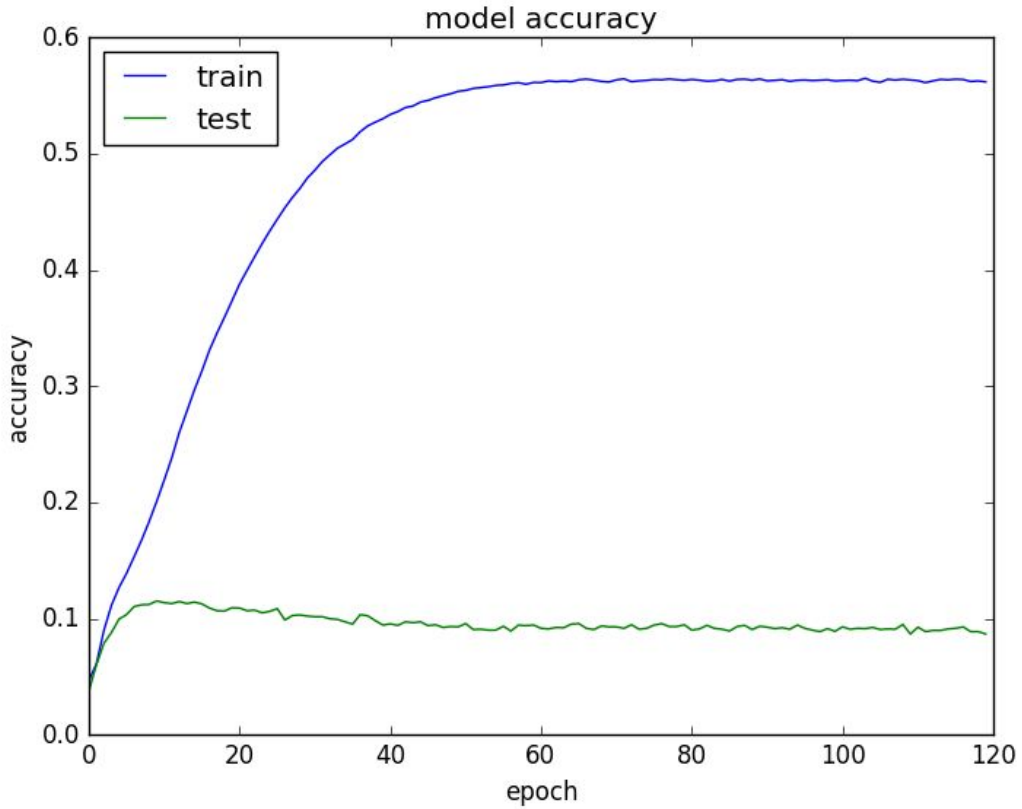
RMSprop V/s Adam for Harry Potter Novel

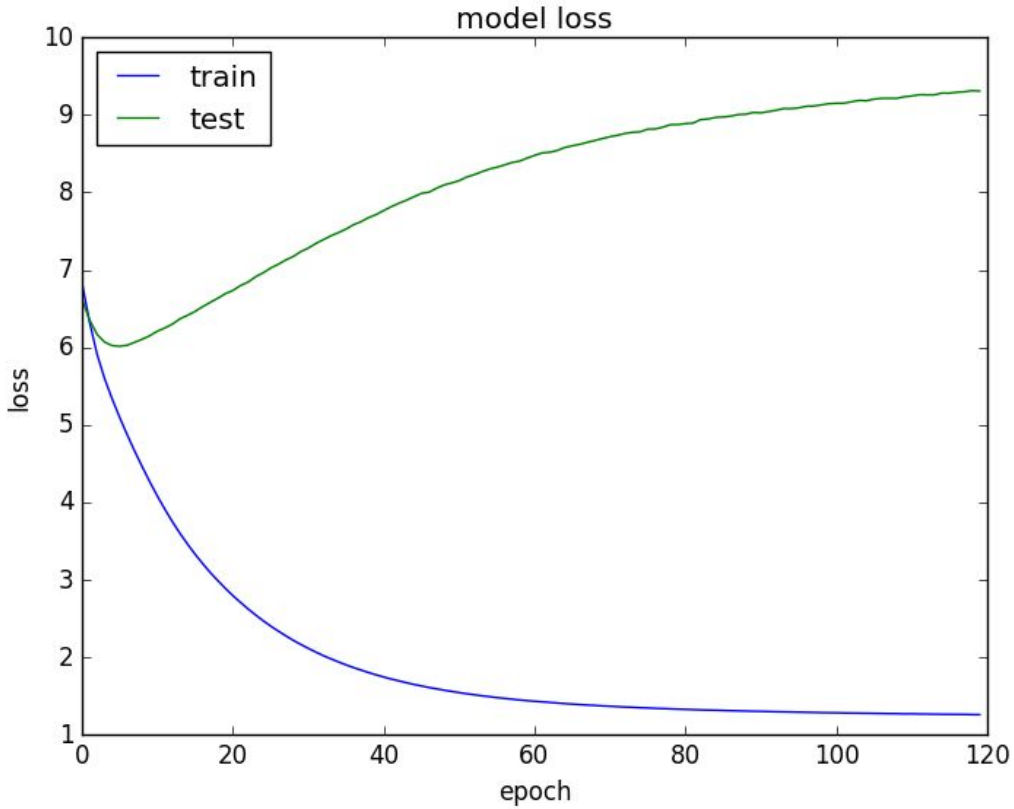
RMSprop optimization fetched an best accuracy of 37.74%





Adam fetched an accuracy of 56.44%, which even has a better validation accuracy when compared to RMSprop





```
print(generate_seq(loaded_model, tokenizer, 2, 'hogwarts woke', 50))
```

```
hogwarts woke to go and once and harry was there he was going to be a back to the dursleys had been a back to the dursley
s had been a back to the dursleys had been a back to the dursleys had been a back to the dursleys had been a back
```

```
print(generate_seq(loaded_model, tokenizer, 2, 'It took quite a while for them all to get off', 10))
```

```
It took quite a while for them all to get off the morning it was a very good term said dumbledore
```

FIG: Test Outputs on Harry Potter Novel using RMSprop optimizer

```
print(generate_seq(model, tokenizer, max_length-1, 'hogwarts woke', 50))
```

```
hogwarts woke to find out how to get past fluffy aren't you wearing yours ron george demanded come on harry an
d ron thought see professor snape disliked him by his proper name voldemort professor mcgonagall was out of th
e way to the door and said she had a long time he was
```

```
print(generate_seq(model, tokenizer, max_length-1, 'It took quite a while for them all to get off', 10))
```

```
It took quite a while for them all to get off neville for neville had been a mistake said uncle vernon
```

FIG: Test Outputs on Harry Potter Novel using Adam optimizer

We can see clearly that adam optimization performed well than RMSprop which has repeated words, than adam which actually produced a paragraph with enough sense

Markov Chains V/s RNN

- The RNN is more accurate than the n-gram model because it can retain more history than just the previous 2 tokens and using word vectors instead of discrete tokens for each word allows it to generalize and make predictions from similar words, not just exact matches.

- The RNN takes longer to train (e.g. 8 hours vs 8 seconds, about 4000x longer), but would easily be able to fit into memory for applications like phone text entry - it also performs word predictions a bit slower than the n-gram, but not enough to be noticeable to an end-user.

Character by Character V/s Word by Word LSTM

1. As we can see that the outputs of word by word model were very very better than character by character.

2. And the training time is also less for the word by word model.

3. The two tradeoffs that is the sentence and the three word approaches in word by word models were compared in the previous slides accordingly to the dataset.

Training Times

Dataset/Approach	3 word	Full sentence
US Speech	2 secs/epoch (GEFORCE 920M)	6 secs/epoch (GEFORCE 920M)
GST Speech	~3 secs/epoch(GEFORCE 920M)	18 secs/epoch(GEFORCE 920 M)

Dataset/Approach	3 word/Adam	3 word/RmsProp
HarryPotter	250 secs/epoch (GEFORCE 860M)	194 secs/epoch (GEFORCE 860M)

MILESTONES

Serial no	Milestone description	Status (% complete)	Comments
1	Markov Chain	100%	Was easy to implement
2	Integrating SQL databases to Markov chain	100%	Was a little tough to maintain the .db files but querying was very fast
3	Writing a simple LSTM network	100%	Initial results were bad
4	Fine tuning the network	80%	We could get accuracy above 90% for NM speech but validation accuracy is less around 15%. If no validation set is given then the results are very good .
5	Have to train the network over large text like novels	100%	We got around 60-70 % accuracy which has to be improved
6	Fine tuning for large text	80%	Still figuring out on the optimizers other than adam to improve the accuracy
7	Dataset for Music compostions	-	Further Work
8	Creating the application for keyboard and music generation -	-	Further work

TOP CHALLENGES UNRESOLVED SO FAR

1. Achieving accuracy above 75% over the large text corpus

2. Improving validation accuracy above 30 % for small text corpus

3. Training time isn't reduced less than 260 sec per epoch on different batch sizes for large text corpus

OUR GOING FORWARD PLAN (OPTIONAL)

1. Step 1 : Creating better architectures to attain better validation accuracy

2. Step 2 : Making an keyboard application to android phones and Survey with people on how good this application is performing with their phones.

3. Step 3: Training the architecture on musical compostions of a particular singer and try to generate predictive musical notes which can uniquely represent his compositions.

OUR TOP THREE LEARNING IN THIS PROJECT

1. Learning # 1 : LSTM property to remember sequences and and produce the next output . This helped us a lot in creating better architectures.

2. Learning # 2 : Markov Chains to predict sequences with their unique property.

3. Learning # 3 : Working with different libraries like nltk , keras ,tensorflow, matplotlib ….