Deep Learning HW03

Siffi Singh – 0660819

Program to build a recurrent neural network model for character-level language model. In this report, I will be explaining:

Part 1: Preprocessing of Text

a) Download dataset
b) Encode each character into one-hot vector

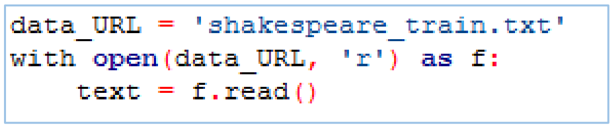
Part 2: Recurrent Neural Network

a) Showing for a standard **RNN**:
    1. Network architecture
    2. Learning curve
    3. Training error rate and Validation error rate
b) 5 breakpoints during training process
c) Training loss V/s different parameters
d) Showing for a standard **LSTM**:
    1. Network architecture
    2. Learning curve
    3. Training error rate and Validation error rate
    4. 5 breakpoints during training process
    5. Training loss V/s different parameters
    6. Comparison of results with RNN
e) Priming the model

Part 1: Preprocessing of Text

a) Download dataset

Downloaded the given Shakespeare dataset, which has **4321640 sequences**. This training set is large enough to train the model with the structure of the dataset.

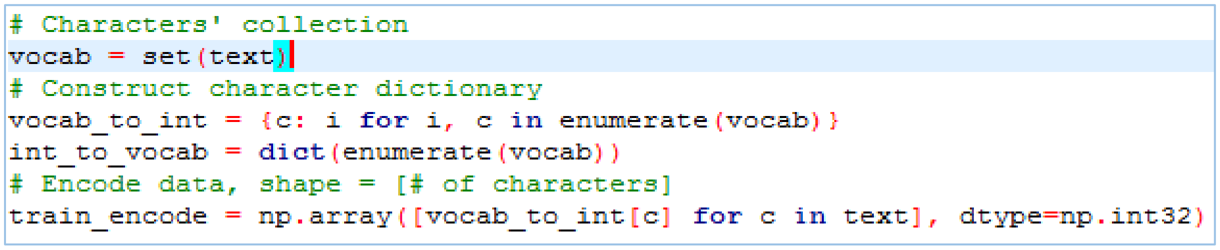


```
data_URL = 'shakespeare_train.txt'
with open(data_URL, 'r') as f:
    text = f.read()
```

This dataset could be replaced with datasets of codes of any language or with probably a math book function, and with the RNN, it learns the syntax of writing a given dataset. The goal here is to build a **character-level language model** that takes in some input text, such as in our case, takes Shakespeare dataset and learn how to write like Shakespeare. It means that in the output for a given small text, we see a new text in a similar style/syntax of the inputted text.

b) Encode each character into one-hot vector

There are 68 different vocabulary in the given training dataset of Shakespeare, and foe each unique character we store a vector of 68, and this vector will be **one-hot encoded** for all vocabularies.



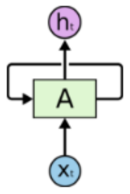
```
# Characters' collection
vocab = set(text)
# Construct character dictionary
vocab_to_int = {c: i for i, c in enumerate(vocab)}
int_to_vocab = dict(enumerate(vocab))
# Encode data, shape = [# of characters]
train_encode = np.array([vocab_to_int[c] for c in text], dtype=np.int32)
```

This allows the representation of categorical data to be more expressive. Specially in Recurrent Neural Network, it is necessary for the proper representation of the distinct elements of the variable.

Part 2: Recurrent Neural Network

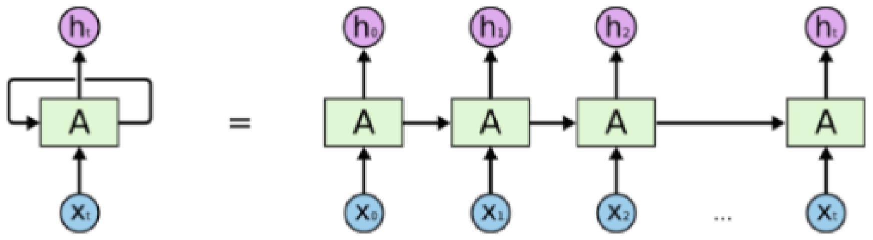
a) SHOWING FOR A STANDARD RNN:

Recurrent Neural Networks try to mimic the human behavior of understanding by persisting previous information. Traditional neural networks cannot do this and hence to overcome this drawback we have Recurrent Neural Network. We try to use the reasoning of previous events in order to inform later ones. Therefore, recurrent neural networks have loops:



Recurrent Neural Networks have loops.

This represents a small part of neural network; the network looks at some part of input text and outputs a value at time step ‘h’. The loop helps in passing the information from one time step to another. The major difference between traditional convolutional neural networks and recurrent neural network is that you have multiple inputs and multiple output units, and that the hidden states are updated at each time step.



An unrolled recurrent neural network.

Following is its implementation:

```
def get_rnn_cells(num_hidden, prob=1.0):
    cells = [tf.contrib.rnn.BasicRNNCell(n) for n in num_hidden]
    dropcells = [tf.contrib.rnn.DropoutWrapper(c, input_keep_prob=prob) for c in cells]
    stacked_cells = tf.contrib.rnn.MultiRNNCell(dropcells)
    stacked_cells = tf.contrib.rnn.DropoutWrapper(stacked_cells, output_keep_prob=prob)
    return stacked_cells
```

a) 1. Network architecture

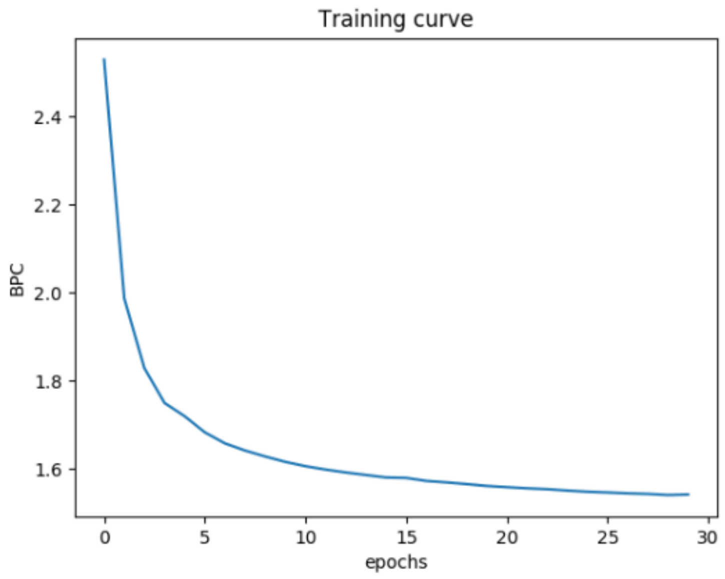
Our network architecture for the RNN based character-level language model
Total training examples: **4321640**
Loss Function Used: **BPC (Bits-per-character)**
Evaluation Function: **Accuracy**
Input layer: **68 one-hot encoded vector**
No. of layers: **2-layer RNN**
Layer Dimensions: **[512, 256]**
Learning rate: **0.1**
Optimization Strategy: **Stochastic Gradient descent with Momentum**
Sequence length: **50**

```
num_steps = 50 #sequence length
num_hidden = [512, 256]
num_class = len(vocab)
learning_rate = 0.1
```

Process: The one hot encoded vector is passed to the input layer. For this RNN model, the first layer has dimensions of 512, so the first layer has 512 hidden units. The second layer of RNN has 256 hidden units and the output layer will have 68 units same as the input vocabulary giving the prediction confidence via a softmax function for each character.

a) 2. Learning curve

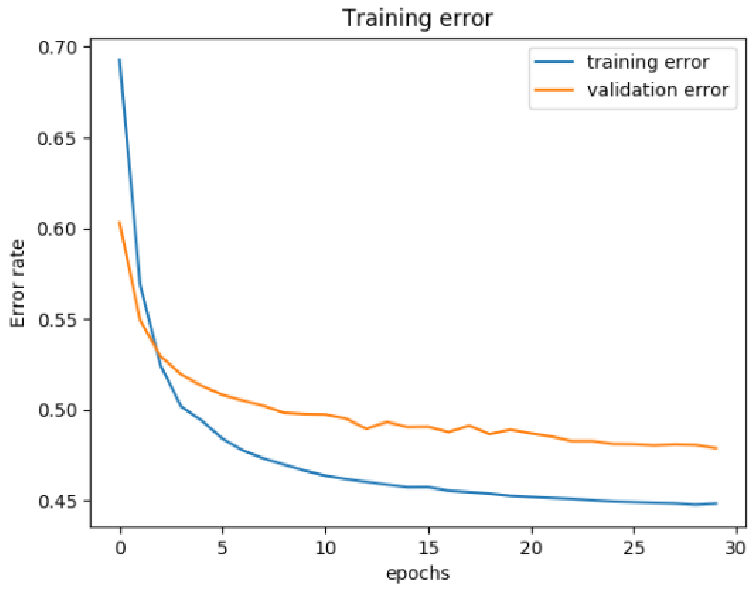
The learning rate was kept at 0.1 and following is the loss curve during the training process. The training time was ~3hours for 30 epochs.



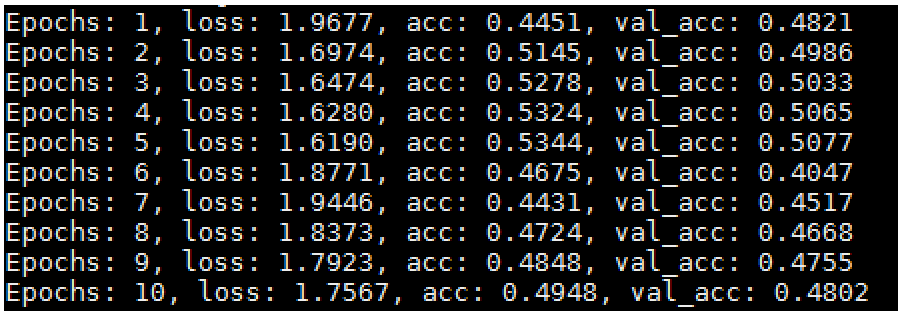
Using SGD with momentum, after 30 epochs, the loss curve converges, and we know that the model is trained.

a) 3. Training error rate and Validation error rate

The training error rate and validation error rate for RNN has been plotted below.



As it can be seen in the graph, the training accuracy reaches a max of ~**55%** and a validation accuracy of ~**49%** using Recurrent Neural Network. We will see in the next part, how the accuracy improves with LSTM.
First ten epochs, have been shown here:



```
Epochs: 1, loss: 1.9677, acc: 0.4451, val_acc: 0.4821
Epochs: 2, loss: 1.6974, acc: 0.5145, val_acc: 0.4986
Epochs: 3, loss: 1.6474, acc: 0.5278, val_acc: 0.5033
Epochs: 4, loss: 1.6280, acc: 0.5324, val_acc: 0.5065
Epochs: 5, loss: 1.6190, acc: 0.5344, val_acc: 0.5077
Epochs: 6, loss: 1.8771, acc: 0.4675, val_acc: 0.4047
Epochs: 7, loss: 1.9446, acc: 0.4431, val_acc: 0.4517
Epochs: 8, loss: 1.8373, acc: 0.4724, val_acc: 0.4668
Epochs: 9, loss: 1.7923, acc: 0.4848, val_acc: 0.4755
Epochs: 10, loss: 1.7567, acc: 0.4948, val_acc: 0.4802
```

b) 5 breakpoints during training process

During the training process, we allowed the code to print intermediate outputs at five breakpoints among the 30 epochs. This helped us in the visualizing the process of learning of the network step-by-step. Following are the five epochs and the corresponding result:

After 1st epoch:
ennn m't ectiedtoohd iyheode lCaocturCt rhU,i egKt
de mulok ?ase,ttuslaguw tia

After 6th epoch:
GTPRAGRF HFFENTANRT:
I first down, the carver art had whose old that petrous, mane cobblers.
wimp on an
Fill me in would so telethon, her tee, lo thou home taint dams after.

After 11th epoch:
My Warps the in shall the feud, daffier story
That, congas have to fear do is be there list

After 16th epoch:
IARO:
That twisty, that spark good she may to the prayer and not than me.
Thou parity, as themes.

After 21st epoch:
Now sod you. And would wiry, yakking
What strut had long! I'll and think, beech and hoist! Out'sole;

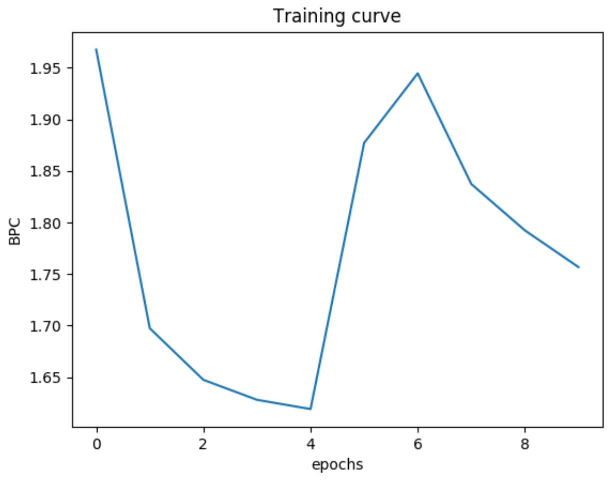
After 30th epoch:

Ay, to this day, I shall set them to the face of honor
For thou shall know, the righteous being.

For the first few epochs we see some gibberish text, but by the later epochs, it starts to learn some syntax of writing from the Shakespeare text, and words start to have more meaning too. We see that by the 30th epoch, the network has learnt well and is able to place commas and full stops as Shakespeare text does.

c) Training loss V/s different parameters

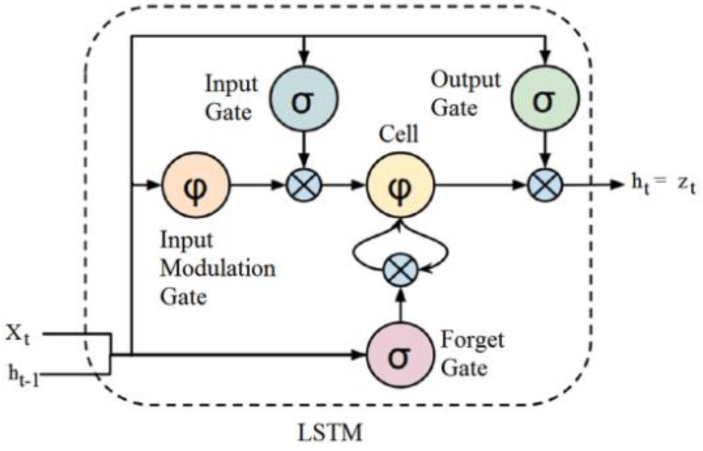
For this homework, we are asked to change the:
1) Different size of hidden states:
I changed the no. of units in the hidden states. The layer dimensions I gave was 256 for the first layer and 256 for the second layer. I ran the code for 10 epochs and following is the training loss curve.



After several changes and observing the training loss curve, I set the value to 512 units for the first hidden layer and 256 for the second hidden layer for 100-sequence length.
2) Sequence Length:
The sequence length for this code is 50, but I changed it to 80 and 100, and the result improved. Although the time taken, also increased accordingly, and I could only note the result for few epochs. From the observations, I conclude that as the "sequence length" increases the accuracy of the network increases, and the network learns better.

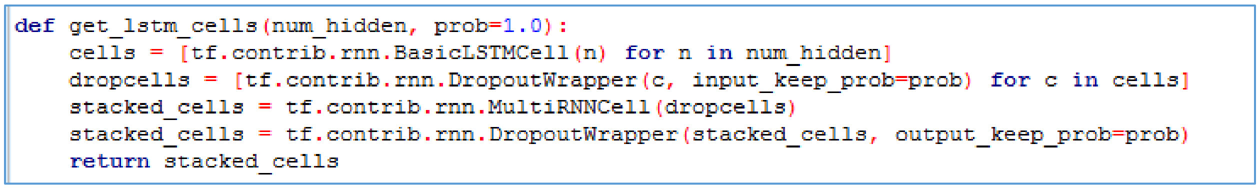
d) SHOWING FOR A STANDARD LSTM:

Theoretically, the conventional recurrent neural network can work, but in practice, it suffers the main two problems that makes it unstable: Vanishing gradient and Exploding gradient.
LSTM was made to solve this drawback of RNN, by proposing a memory unit, also known as the cell of the network. More specifically, it remembers values over arbitrary intervals. It is best known for classification, processing and predicting time series given time lags of unknown duration.



LSTM Cell

The LSTM is called cell state, it has a recursive nature of cell that allows information from previous intervals to be stored in LSTM cell. They have a forget gate that forgets a cell state by multiplying with forget gate and adds new information to the input gates.
Here is the implementation:



```
def get_lstm_cells(num_hidden, prob=1.0):
    cells = [tf.contrib.rnn.BasicLSTMCell(n) for n in num_hidden]
    dropcells = [tf.contrib.rnn.DropoutWrapper(c, input_keep_prob=prob) for c in cells]
    stacked_cells = tf.contrib.rnn.MultiRNNCell(dropcells)
    stacked_cells = tf.contrib.rnn.DropoutWrapper(stacked_cells, output_keep_prob=prob)
    return stacked_cells
```

d) 1. Network architecture

Our network architecture for the LSTM based character-level language model
Total training examples: **4321640**
Loss Function Used: **BPC (Bits-per-character)**
Evaluation Function: **Accuracy/Error rate**

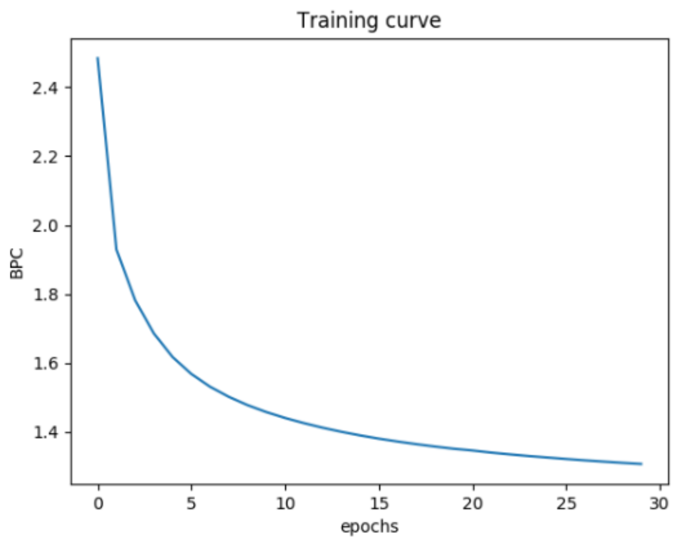
Input layer: **68 one-hot encoded vector**
No. of layers: **2-layer LSTM**
Layer Dimensions: **[512, 256]**
Learning rate: **0.1**
Optimization Strategy: **Stochastic Gradient descent with Momentum**
Sequence length: **50**
Process: The input to the network is the same as RNN i.e. one-hot encoded vector, but the processing is different, we make use of the LSTM cells instead of RNN cells this time. We make use of the 'input gate', 'forget gate', 'output gate' and 'hidden gate'. Rest of the parameters are kept same for the fairness of comparison.

```
num_steps = 50 #sequence length
num_hidden = [512, 256]
num_class = len(vocab)
learning_rate = 0.1
```

The training time for both the networks was ~3-4 hours, sequence length and hidden units are also kept same.

d) 2. Learning curve

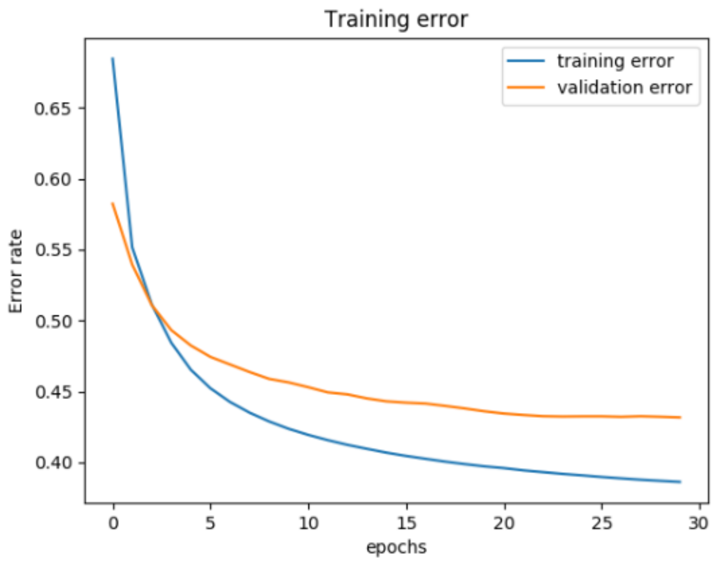
The learning rate for the entire training duration is kept at 0.1. Following is the training curve for LSTM.



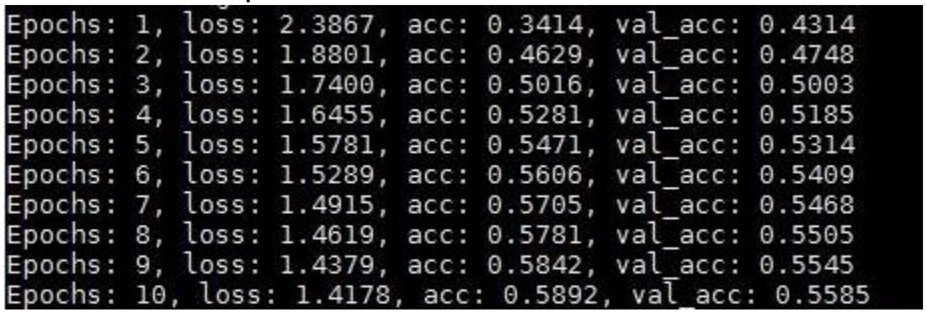
Number of epochs for training was 30, after 30 epochs we see a convergence region. The current value of 0.1 was chosen after verifying with a range of values of learning rate and the value for which the best result was seen was selected for the training process.

d) 3. Training error rate and Validation error rate

The graph for the training and validation error rate at different epochs has been shown in the graph below. As can be seen in the graph for the given parameters the training accuracy reaches a maximum of ~**62%** and the validation accuracy reaches a maximum of ~**55%.** This is clearly better accuracy than the previously observed RNN.



The result for the first 10 epochs is shown in the screenshot below:



```
Epochs: 1, loss: 2.3867, acc: 0.3414, val_acc: 0.4314
Epochs: 2, loss: 1.8801, acc: 0.4629, val_acc: 0.4748
Epochs: 3, loss: 1.7400, acc: 0.5016, val_acc: 0.5003
Epochs: 4, loss: 1.6455, acc: 0.5281, val_acc: 0.5185
Epochs: 5, loss: 1.5781, acc: 0.5471, val_acc: 0.5314
Epochs: 6, loss: 1.5289, acc: 0.5606, val_acc: 0.5409
Epochs: 7, loss: 1.4915, acc: 0.5705, val_acc: 0.5468
Epochs: 8, loss: 1.4619, acc: 0.5781, val_acc: 0.5505
Epochs: 9, loss: 1.4379, acc: 0.5842, val_acc: 0.5545
Epochs: 10, loss: 1.4178, acc: 0.5892, val_acc: 0.5585
```

d) 5. Five breakpoints during training process

Also from the above plot we learn that somewhere between 2500-4000 epoch the learning curve reaches its saturation point and the data is separated properly.

Epoch 1:

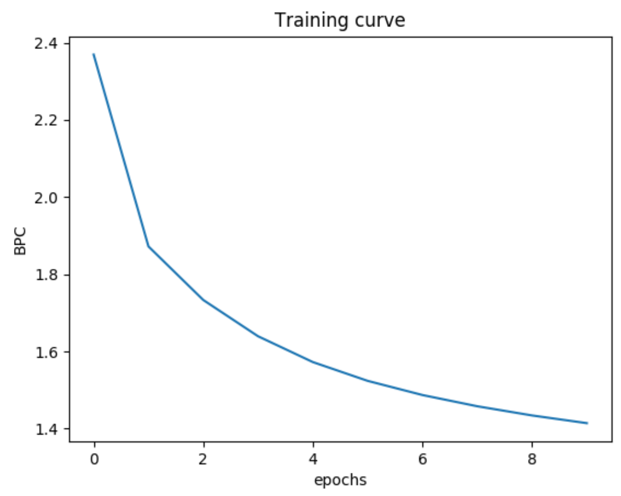
I are Heres,
Why, and he's Selmer. Providing the shortly bears, the spearase
Toet I but susut, at ratulet have

<u>Epoch 6:</u>
Be lordly-and the netsuke the grain for tell our logged batter heat a born more mimed ham your gouts.
Shat warm,
<u>Epoch 11th:</u>
The will, and mine of my son, and show to the sick and make a sir,
And this with the man thee we shall see thee, sir, and sent man me to the sour the sounting of me to the marrish a son to the man a man
<u>Epoch 16th:</u>
I am a great sight so may be so far assured
And that the wind of the suns and the thing.
PROSPERO:
I am a third in treason shall to the charge,
And to their seats and their angers and the service
That the stares of the world of this a country.
<u>Epoch 21:</u>
PANOLICE:
The more of the streets of an earth and soul,
And the such state of this the state of the court.
We have see the word of her that were an enemy. I am a thing
of my chaste will the streeks of the stars as the
<u>Epoch 30:</u>
VALENTICE:
A man a promition to me to the such and should so shall see him
and moored to my marring and so see and shall he see and see there
Mother to my more,
And should never send to true of my moneys as the with the souling

d) 6. Training loss V/s different parameters

Training loss varied with sequence length and number of hidden states:
1) Sequence Length: In the simulations, I changed the value to 80 and 100, similar to the RNN network, and observed the change in the accuracy values. As expected the accuracy values went up. Hence concluded that as we increase the sequence length the accuracy of the network increases, which means it, is able to learn the context better. However, the maximum sequence length should not be above a threshold value where the network stops learning the relevant structure.

2) Hidden states: The number of hidden states was initially set to be 256 and 128, but changing it to 512 and 256 gives better accuracy. The tuning of these hyper-parameters play a key role in the performance of the network.



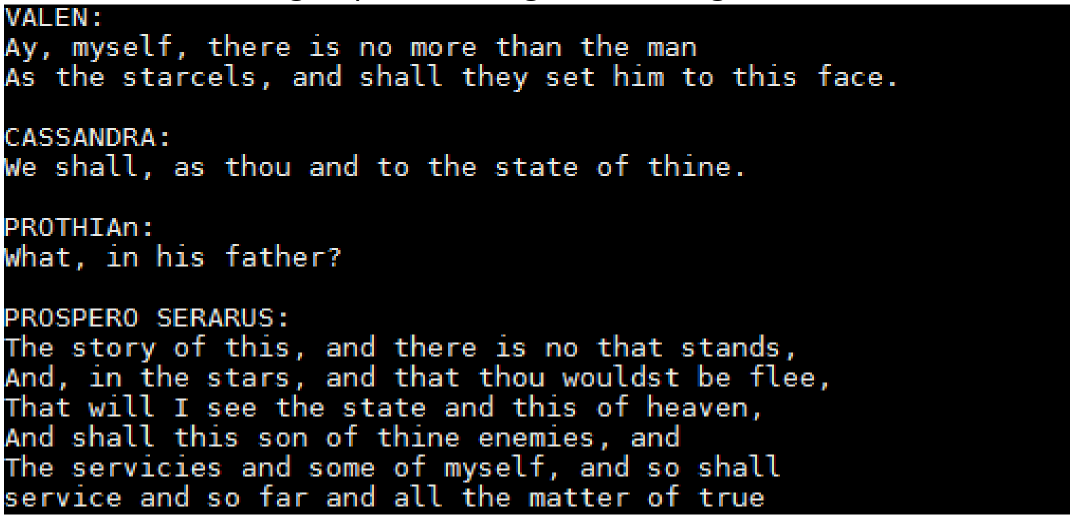
The above graph is for 256 hidden states in first layer of LSTM and 256 hidden states in second layer of LSTM for 10 epochs, learning rate of 0.1.

d) 7. Comparison of results with RNN

LSTM	RNN
1) LSTM has both cell states and hidden states. The cell has the ability to remove or add information to cell, regulated by 'gates'.	1) RNN does not have cell state; they only have hidden states that serve as memory for RNNs.
2) LSTM deal with the problems of vanishing gradient and exploding gradients and can handle long-term dependencies.	2) RNN suffers the problem of vanishing gradients and exploding gradients.
3) The accuracy on the Shakespeare dataset for the same set of parameters is better than RNN. Training Accuracy: ~62% Validation Accuracy: ~55%	3) The accuracy value of RNN is less as compared to LSTM, even though the parameters are same and takes the same time for training. Training Accuracy: ~55% Validation Accuracy: ~49%

e) Priming the model

I gave it a word "Valen" as a prime, and observed the output, after the model has been trained. In all cases, the sentence started with Valen, which was as expected, and a range of characters can be specified. The new sentence will include the number of characters specified in the range. Following is the output for "Valen" as a prime. For the following ouput the range of words given was 500.



```
VALEN:
Ay, myself, there is no more than the man
As the starcels, and shall they set him to this face.

CASSANDRA:
We shall, as thou and to the state of thine.

PROTHIAn:
What, in his father?

PROSPERO SERARUS:
The story of this, and there is no that stands,
And, in the stars, and that thou wouldst be flee,
That will I see the state and this of heaven,
And shall this son of thine enemies, and
The servicies and some of myself, and so shall
service and so far and all the matter of true
```

After 30th epochs, LSTM gives appropriate results capturing the significant parts of the Shakespeare train dataset that it has been trained on, In terms of putting "thou" and putting commas and in terms of specifying grammar correctly.

Inferences from the result

From this homework, I have learnt:

1) Implementation, evaluation and deeper understanding of RNN and LSTM network.
2) The application areas of RNN and LSTM including but not limited to Image Captioning, Sentiment Classification, and Machine Translation etc.
3) Impact of selection of no. of hidden units and sequence length on the network.