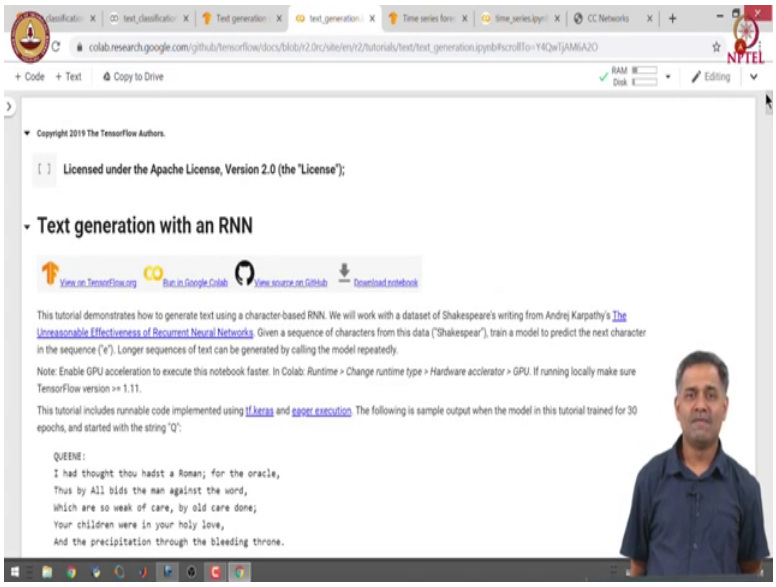
Practical Machine Learning
Dr. Ashish Tendulkar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 32
Text Generation with RNNs

[FL]. We will study how to use LSTM models for generating text. We will be training LSTM on the text written by Shakespeare character by character.

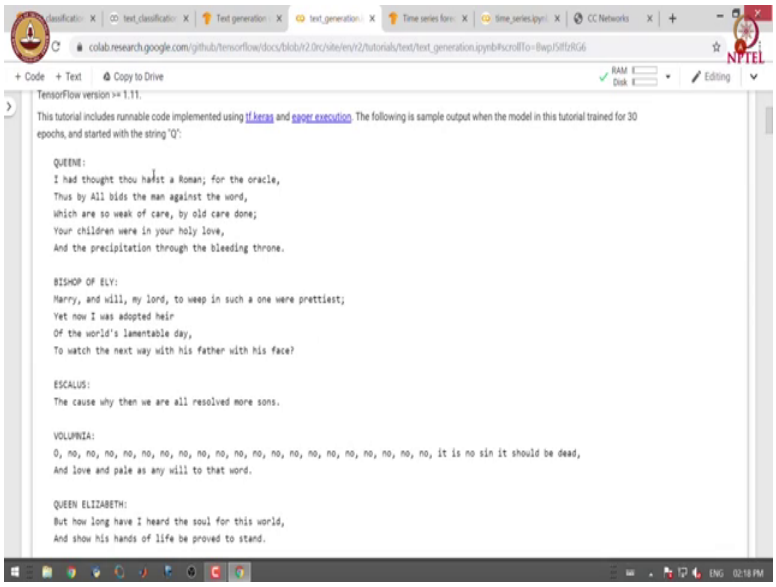
(Refer Slide Time: 00:35)



So, we are going to use the dataset from Andrej Karpathy's writing of Unreasonable Effectiveness of Recurrent Neural Networks. The problem that we are going to solve is given a bunch of characters. What is the most likely next character?

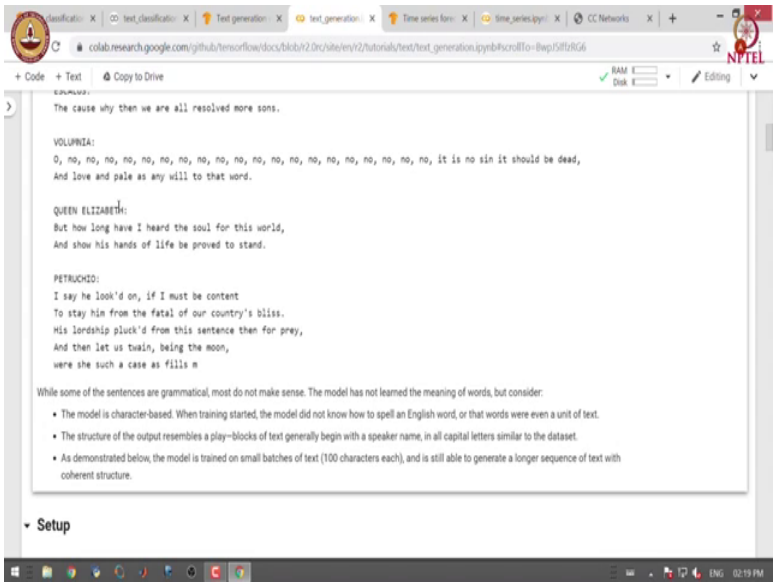
For example, if we give sequence of character the model has to predict what is the next character? So, in this case if model predicts "e" as the next character will have Shakespeare word completed. We can generate longer sequences by calling the model repeatedly.

(Refer Slide Time: 01:32)



So, just to give you an idea this is the sample output of the model trained for 30 epochs, we started the model with letter 'Q' and model went on.

(Refer Slide Time: 01:49)

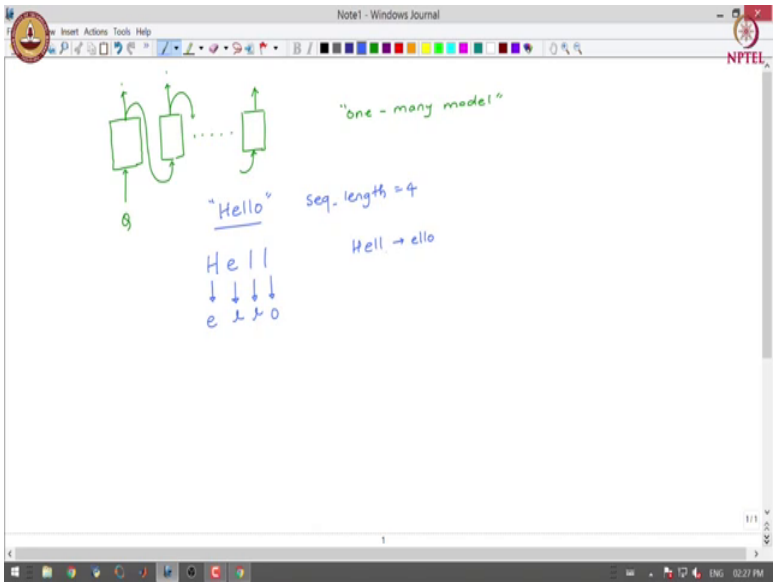


Why some of the sentences are grammatical most do not make sense? The model has not learned the meaning of words, but consider the model is character based when training started the model did not know how to spell an English word or that word where even a

unit of text. So, given this background, this is really an impressive performance. You can see that the structure of the output is very similar to the structure of the play.

The blocks generally begin with a speaker name in all capital letters followed by the dialogues. So, this is very similar to what you see in the Shakespeare plays. And the model is trained on a small batch of text 100 characters each and is able to still generate longer sequence of text with coherent structure.

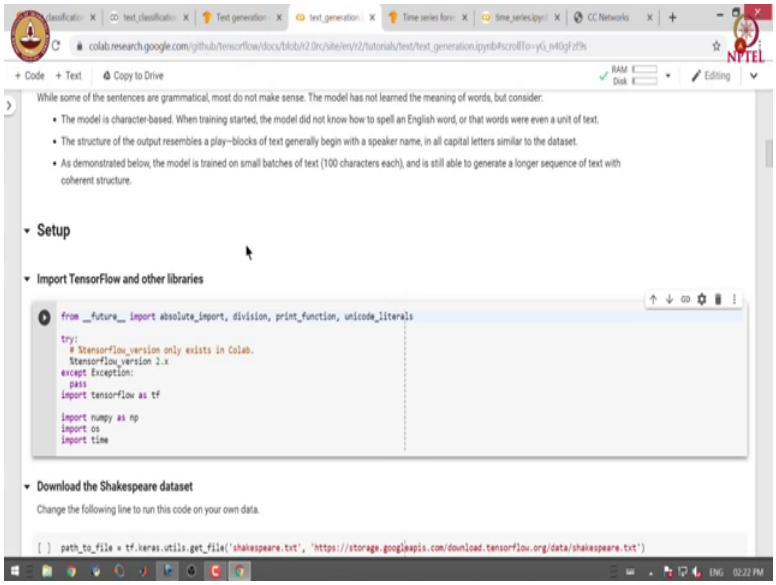
(Refer Slide Time: 03:12)



So, just to give you an idea here we are using the following RNN architecture. So, we are starting the model with a single letter 'Q' and you are asking model to generate the next letter. The model is generating output, the output is the next letter and that letter is again fed into the second node which you generating the next letter and that letter is fed into the next node and so on. So, these how we get the text generated from RNN.

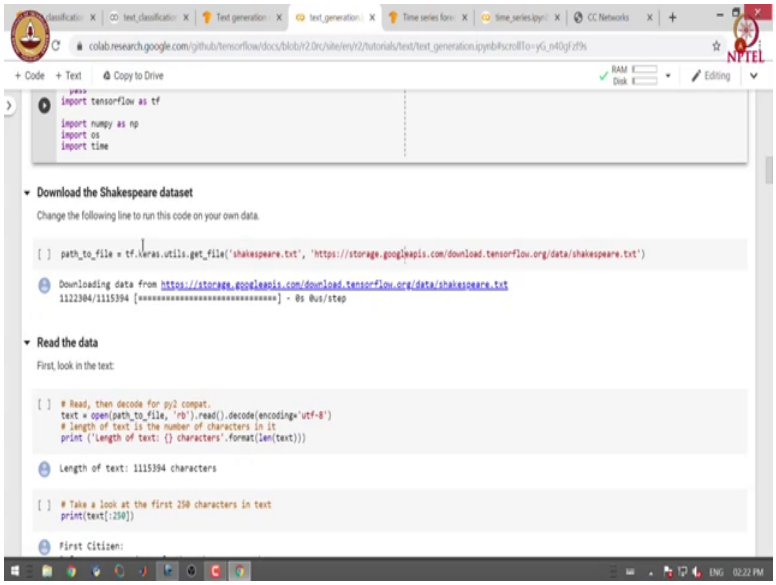
So, here we are giving a single input the so, this is one-to-many model architecture that we are using here.

(Refer Slide Time: 04:37)



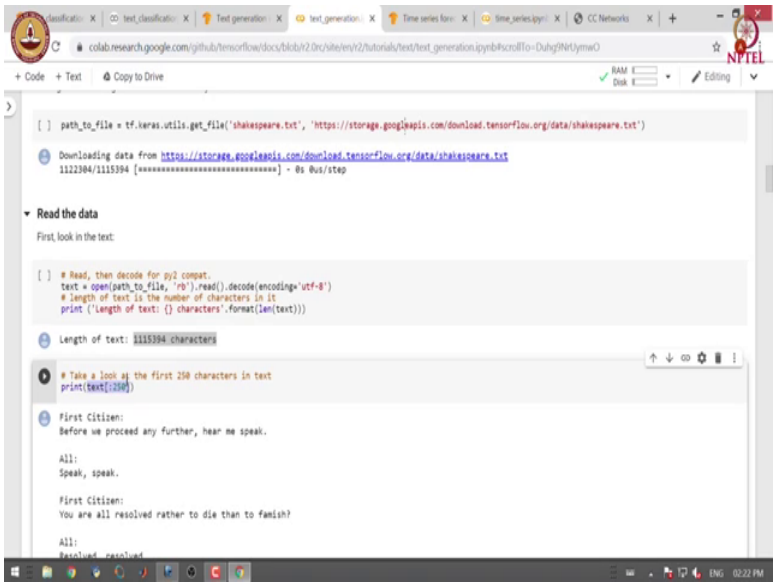
Let us import TensorFlow on other libraries; make sure you are using TensorFlow 2.0 for this example.

(Refer Slide Time: 04:51)



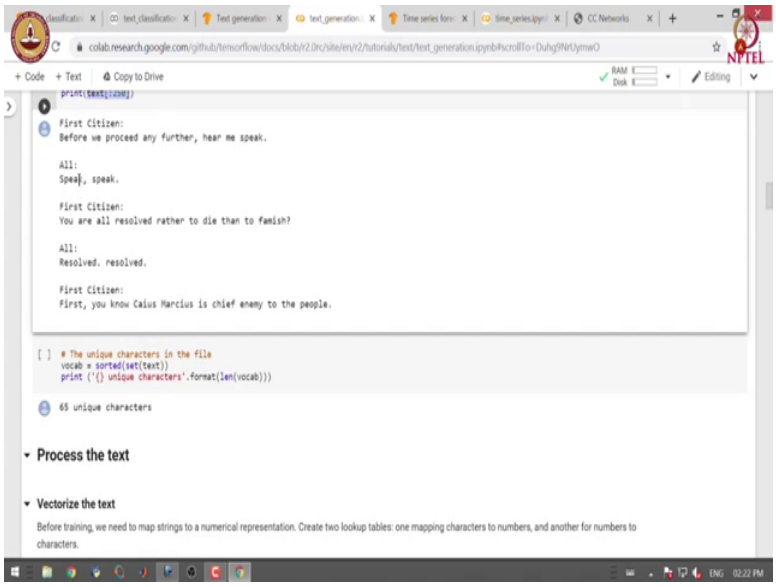
We will download the Shakespeare data set. We can look at the text; you can see that there are about 1.1 million characters in the data set.

(Refer Slide Time: 05:09)



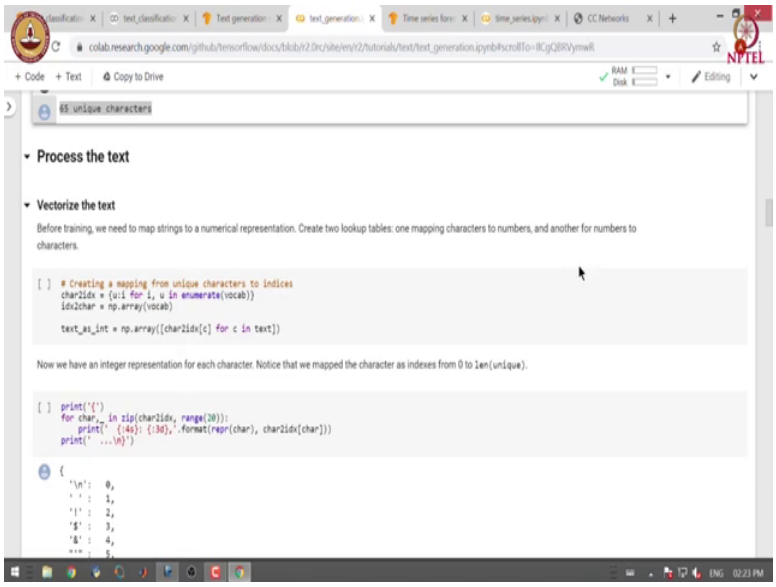
So, length of the text is about 1.1 million characters. We can look at first 250 characters with this particular access mechanism.

(Refer Slide Time: 05:33)



So, you can see that these are first 250 characters. So, the number of unique characters in the file are 65. So, our job is to predict one of the 65 characters for a given sequence of characters.

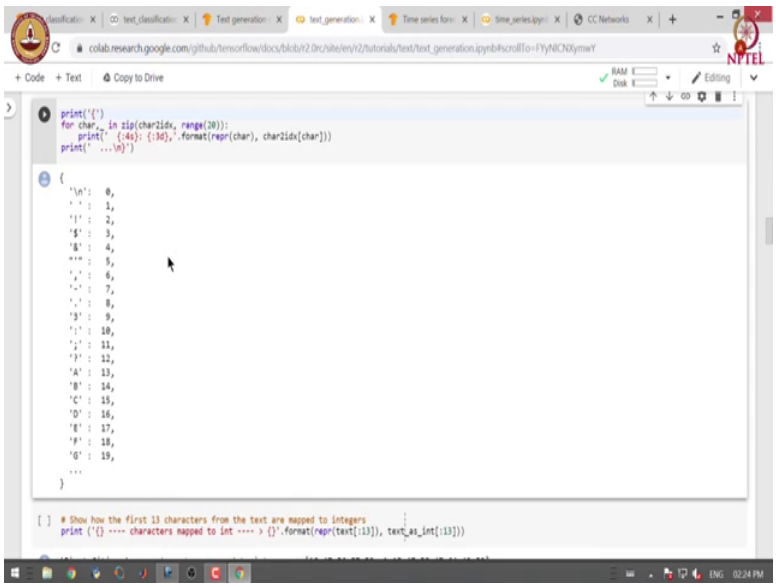
(Refer Slide Time: 05:55)



The predicted character is the most likely character to be following in the sequence.

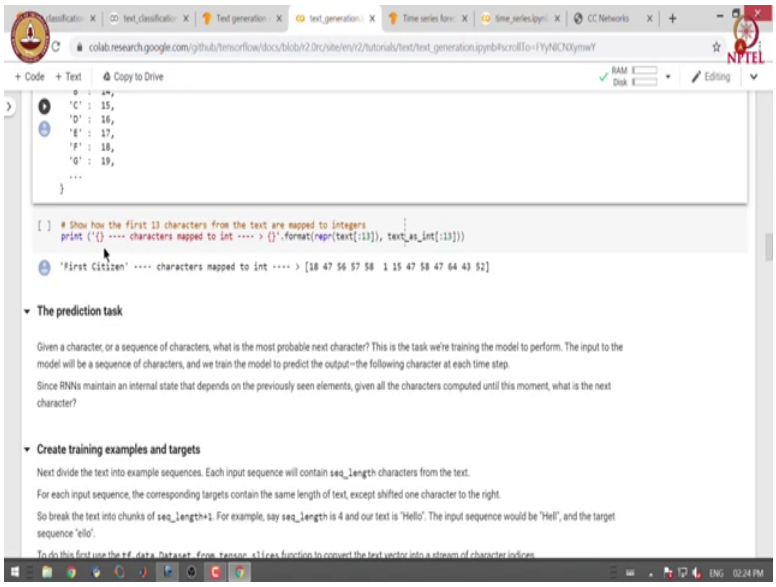
So, as we studied earlier we cannot process the string as it is. So, you have to convert string into a numeric representation. So, we use embeddings for converting text into numeric representation.

(Refer Slide Time: 06:26)



So, here we are defining few helper functions to assign a unique id to each of the character.

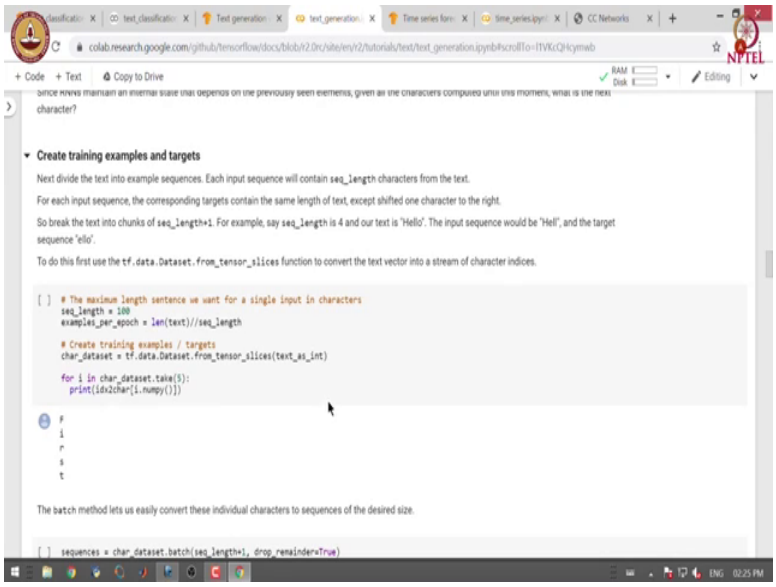
(Refer Slide Time: 06:33)



You can see that the first 13 characters around mapped to their integer representation. So, the prediction task here is given a character or a sequence of character what is the most probable next character? These are task you are training the model to perform.

The input to the model will be a sequence of character and we train the model to predict the output, and the output is going to be the following character at each time step. Since, RNNs maintain an internal state that depends on the previously seen element given all the characters computed until this moment, what is the next character?

(Refer Slide Time: 07:32)



So, let us try to understand how to create training data for this task. So, we will divide the text into example sequences each input sequence will contain, sequence_length number of characters from the text. For each input sequence the corresponding target contains the same length of text except, it is shifted by one character to the right. We break the text into chunks of sequence_length plus one character's.

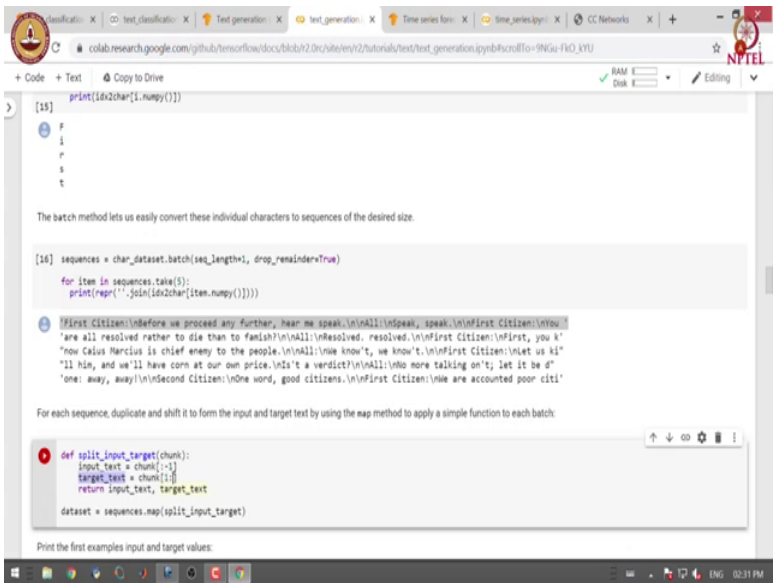
For example, if the sequence length is 4 and our text is "Hello". The input sequence would be "Hell" and the target will be "ello". Let us say we have word hello and a sequence length is equal to 4. So, we take a chunk off sequence length plus one letters. So, hello is exactly 5 words.

So, we take hello we construct the training data which has got first 4 characters 'h', 'e', 'l', 'l'. For 'h' we expect the model to predict 'e', for 'l' you want to predict 'l', for 'e you want to predict 'l', this 'l' we want to predict next 'l' and this 'l' we want to predict 'o'. So, 4 input 'h', 'e', 'l', 'l', we want to predict this particular output. So, thus for the input sequence "Hell", the target sequence is "ello", which you just shifted by one character to the right.

So, to do this first we use tf.data.Dataset.from_tensor_slice() function. We convert text vector into a stream of character indices. And then we use indexed to character mapping

to obtain the character corresponding to each index. So, you can see that first 5 characters are First here.

(Refer Slide Time: 10:27)

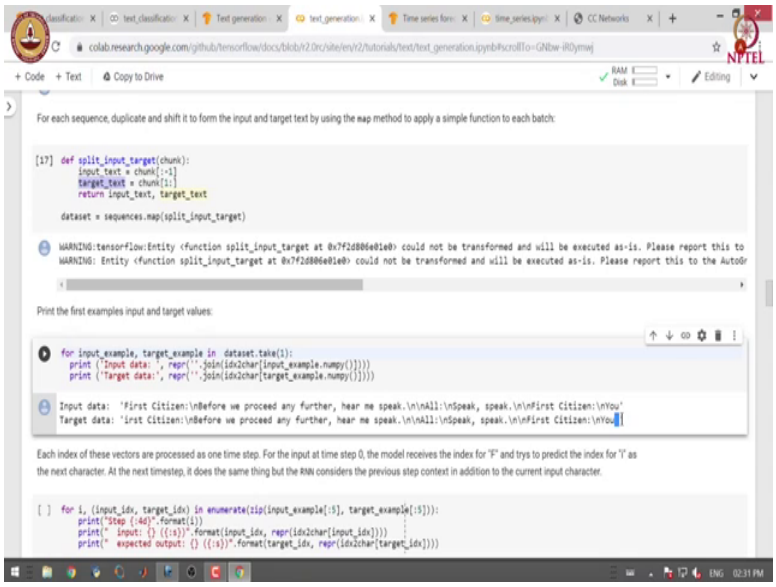


So, the batch method let us easily convert these individual characters to sequence of desired size. So, simply called the batch method provide the length which sequence_length + 1. So, let us run this; so, these are first five sequences, each with 100 characters. So, now, letter for each sequence we will duplicate it and shift it to form the input and the target text by using the map method to apply a simple function to each batch.

So, you want to take this sentence, we want to shift it by one letter. we am going to apply this particular transformation to each and every example in the data set. So, for that we use the map() method. So, in the sequences we use the map method and we call split underscored input under scored target function.

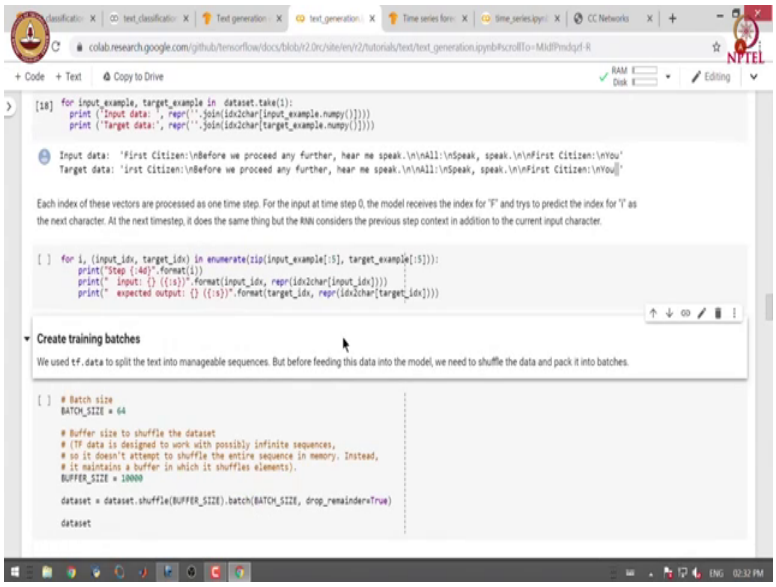
All that it does is it shifts it by one. So, we can see that we use in the input text you have everything, but the last character and target text starts from the second character onwards.

(Refer Slide Time: 12:10)



Let us print the first example of input and target value. So, you can see that these are the first example that you have a sentence and the target is copy of the same sentence except that the first character is missing. And one character has gotten added at the end of the sequence.

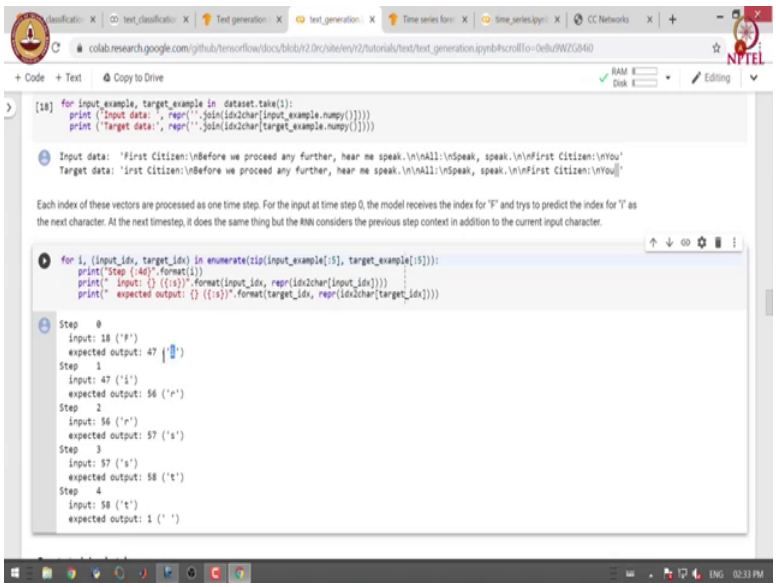
(Refer Slide Time: 13:06)



So, each index of these vectors are processed as one time step; for the input at times step zero the model receives the index of ‘f’ and tries to predict the index of ‘i’ as the next

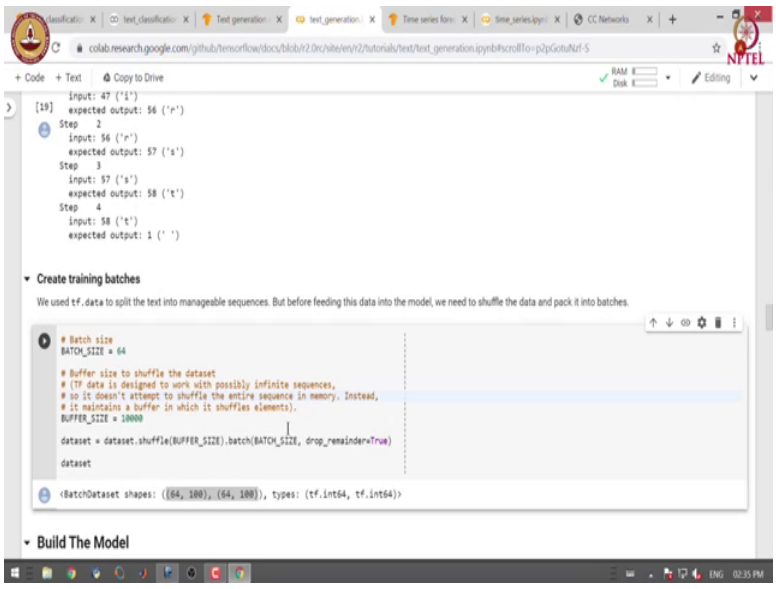
character. At the next time step it does the same thing, but RNN considers the previous step context in addition to the current input character.

(Refer Slide Time: 13:43)



So you can see that for letter ‘F' you want to predict ‘i'. So, we give the index of ‘F' as input and it is expected to produce the index of ‘i' as an output. Given index of ‘i' is and input is expected to predict index of ‘r’ as an output and so on.

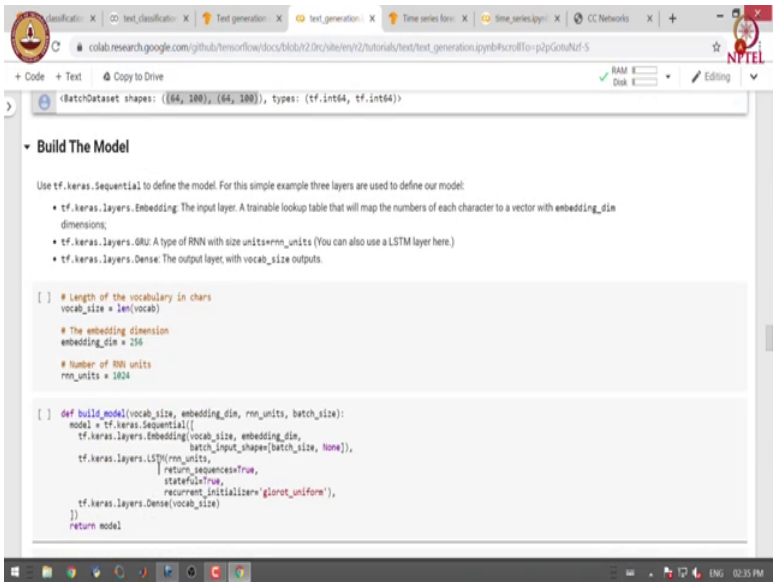
(Refer Slide Time: 14:09)



So, let us create training batches. So, far we used tf.data to split the text into manageable sequences. But before feeding this data into the model we need to shuffle the data and packaged into the batches. We use the batch size of 64. We use buffer size of 10000 for shuffling.

So, we shuffle the data set and then we batch it. Note that your setting drop_remainder is = *True* and this will essentially drop any of the elements that are left in the last batch. So, look at the shape of the data set. So, it is a tuple each with 64 x 100 dimension. So, these are essentially the input and the target sequences.

(Refer Slide Time: 15:40)

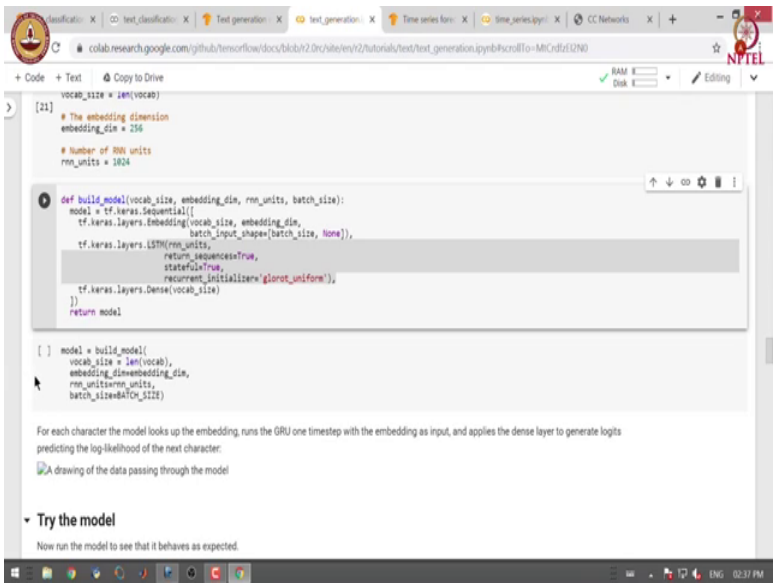


Let us build LSTM model for solving the problem. Here we use tf.keras.Sequential model, which has got three layers. The first layer is an embedding layer, which maps each character to a vector of embedding dimensions. In this case you are used LSTM with a fixed number of units you are set return sequence is equal to True; that means, you will not output from each node.

And we also said stateful = *True*; stateful essentially passes the output of the last character in the batch to the first character in the next batch. You are using a specific recurrent initializer called glorot_uniform. We pass the output of LSTM through a dense layer containing number of units equal to the vocabulary size. We are doing this because

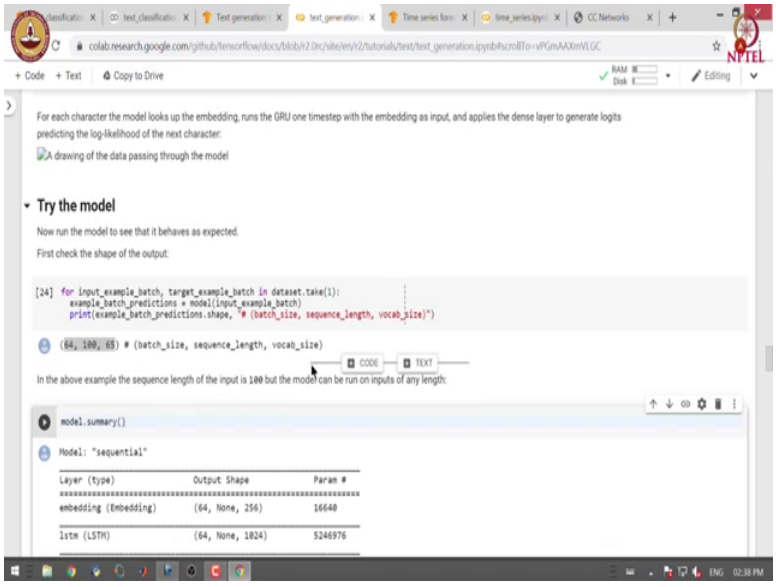
we want to predict one of the 65 characters which is the size of the vocabulary in this context. So, this is a function for building the model.

(Refer Slide Time: 17:31)



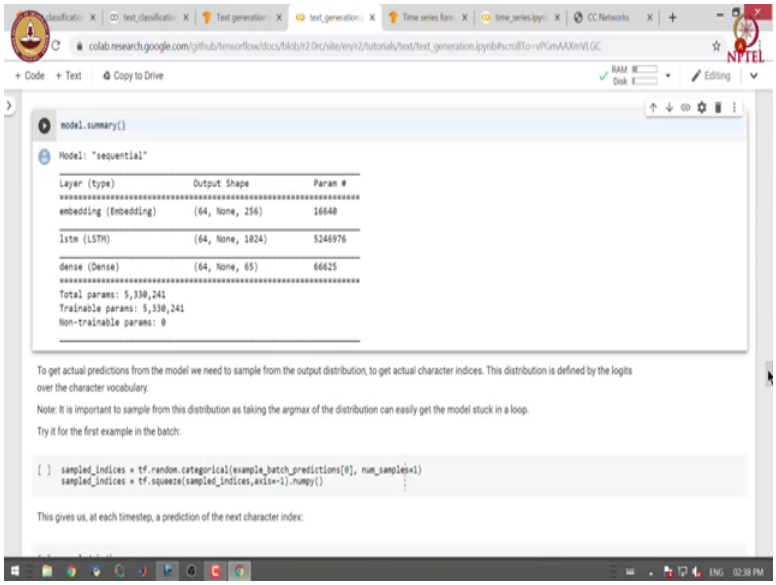
Let us build a model. For each character model looks up the embedding runs LSTM. One time run the LSTM, one time step with the embedding as input and applies dense layer to generate the next character.

(Refer Slide Time: 17:55)



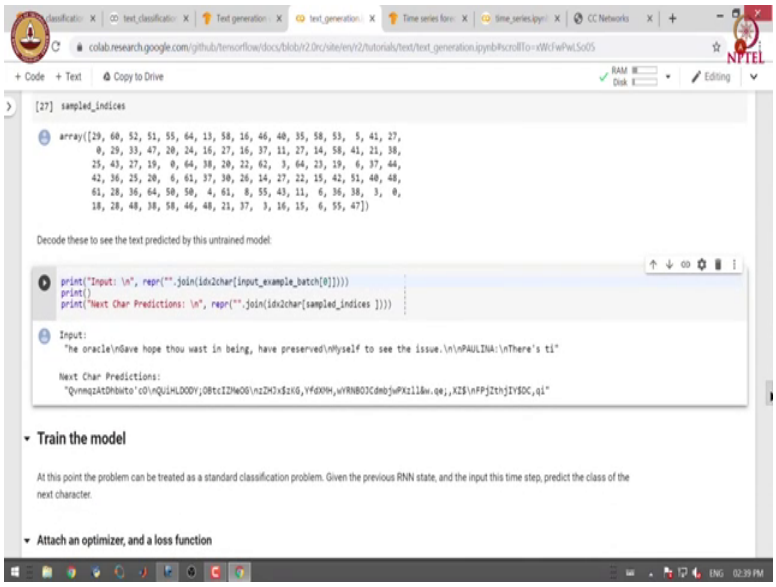
Let us try the model let us check the shape of the output. In this case you can see that we have a 3D tensor here, we checks 64 x 100 x 65; here 64 is a batch size 100 is a sequence length and 65 is the size of vocabulary. In the above example the sequence length of the input is a 100 but the model can be run on inputs of any length.

(Refer Slide Time: 18:47)



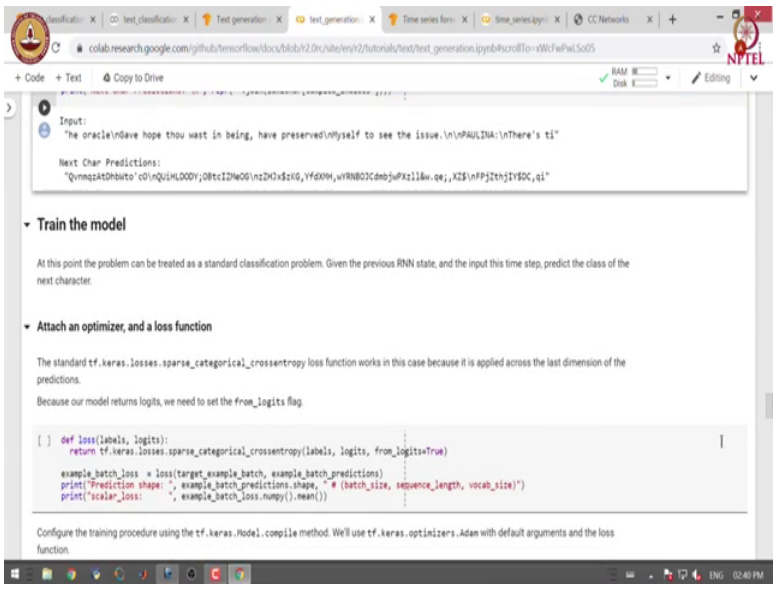
Let us look at the summary of the model; we can see that the model has 5.3 million parameters a very large number of parameters indeed. To get the actual prediction from the model we need samples from the output distribution to get the actual character indices. This distribution is defined by logits over the character vocabulary. Note that it is important to sample from this distribution as taking argmax of the distribution can easily get the model stuck in a loop. Let us try the model on first example of the batch.

(Refer Slide Time: 19:25)



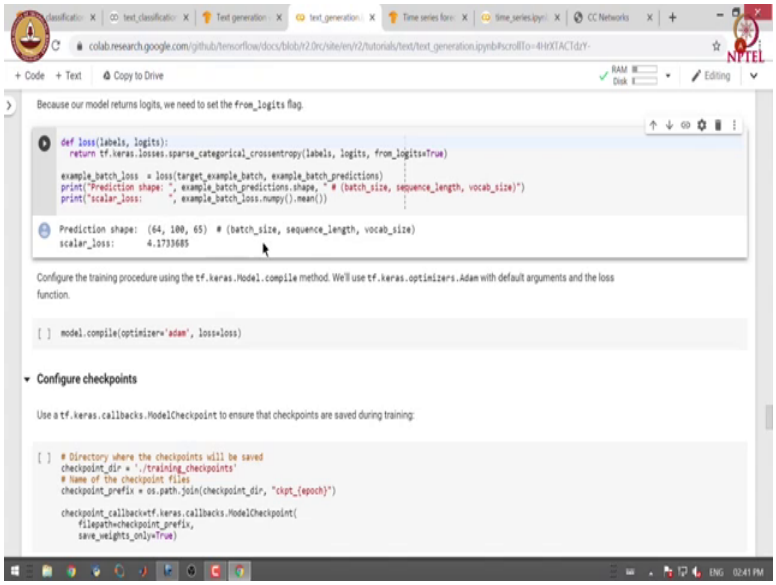
So, these are the sample indices, which is a prediction of the next character index at each timestep. Let us decode this to see what model has predicted. Note that you are not trained the model so, this is still an untrained model. So, you can see that currently the next character prediction is not really working that great we are getting quite random characters as next corrector. So, let us try to trained the model and see if we can get better results with the model.

(Refer Slide Time: 20:21)



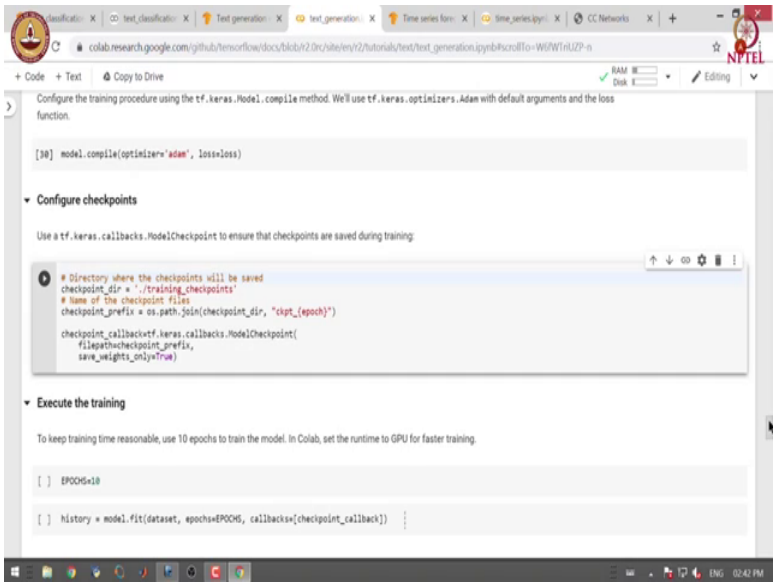
So, at this point the problem can be treated as a standard classification problem. Given the previous RNNs state and the input this time step we want to predict the class of the next character. We use *sparse_categoricalcrossentropy* loss, in this case because it is applied across the large dimension of the prediction because our model returns logits we need to set the from.logits flag. So, we define a loss() function and then calculate the loss.

(Refer Slide Time: 21:15)



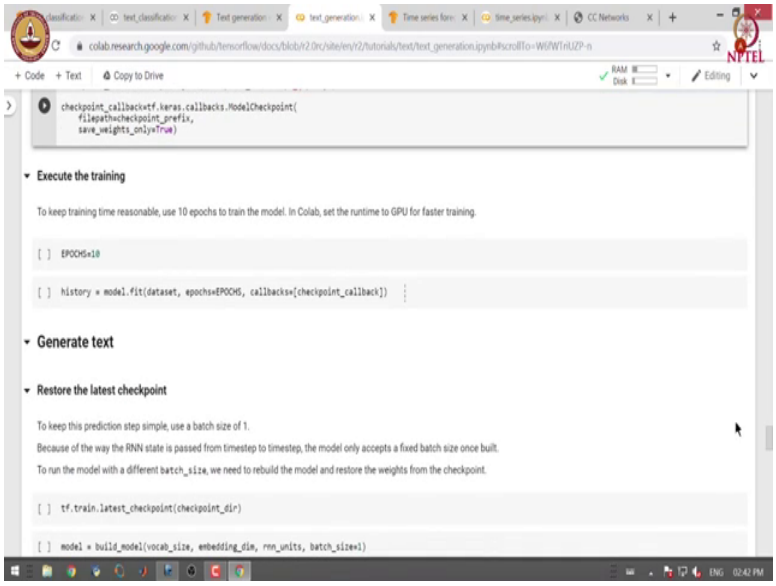
So, you can see that the prediction shape is 64 x 100 x 65, where 64 is a batch size 100 is a sequence length and 65 is the size of the vocabulary and the scalar loss is 4.17. Let us configure the training procedure using model.compile() method we will use Adam optimizer over here. We will use model checkpoints to ensure that the checkpoints are saved during the training.

(Refer Slide Time: 21:54)



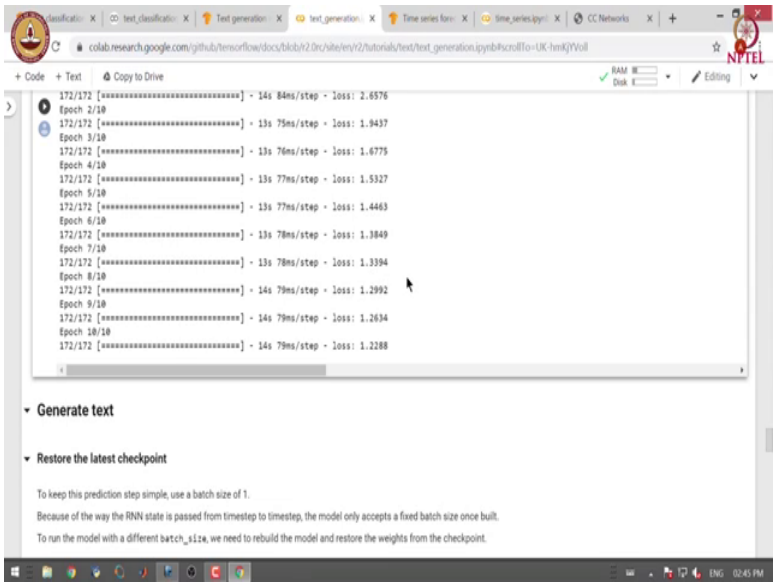
So, we give the checkpoint directory and provide the checkpoint prefix.

(Refer Slide Time: 22:07)



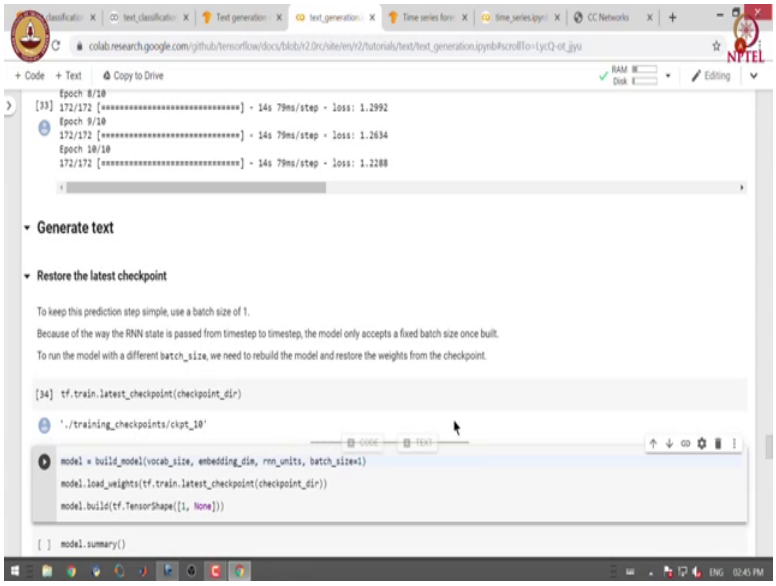
Let us train for 10 epochs and note that we are using GPUs here for faster training.

(Refer Slide Time: 22:19)



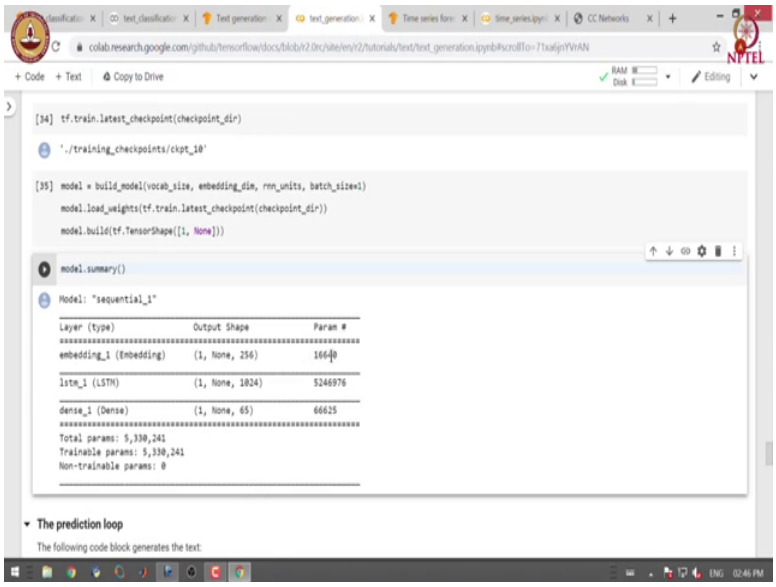
Now, that our model is trained let us look at how to generate that text from the model.

(Refer Slide Time: 22:31)



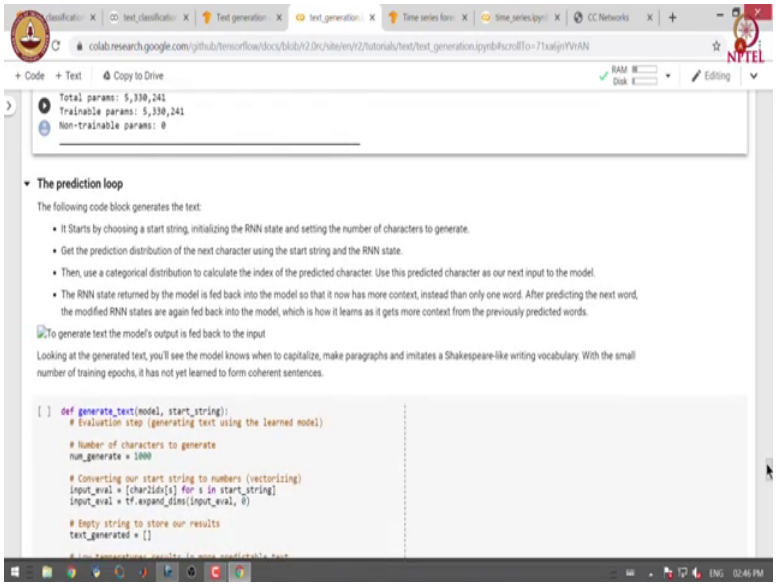
We will restore the latest checkpoint of the model.

(Refer Slide Time: 22:39)



And we will proceeding let us check the model summary. You can see that the model that is restored from the checkpoint is exactly same as the model that we built and the model that we trained a few minutes earlier

(Refer Slide Time: 23:01)



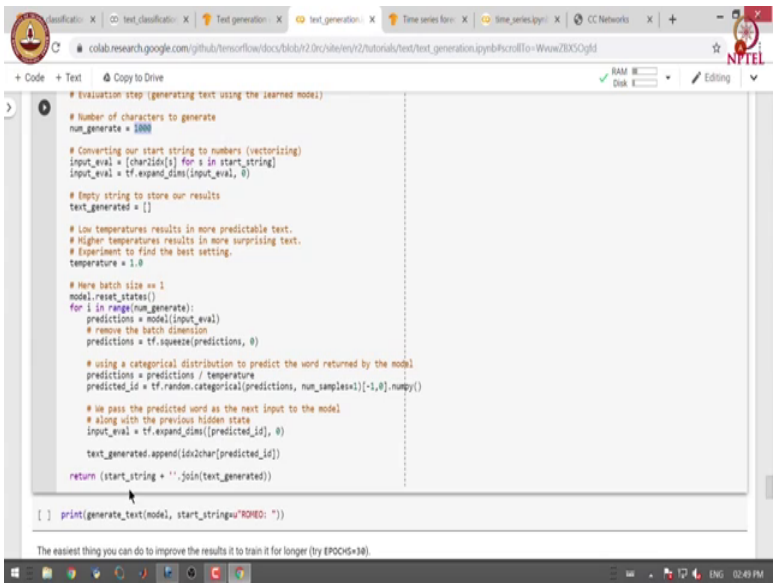
Let us try to understand how do we generate the text. So, we start by choosing the start string, we initialize the RNN state and we set the number of characters to generate. We get the prediction distribution of the next character using the start string; using the start

string and the RNN state then we use a categorical distribution to calculate the index of predicted character.

We use this predefined character as our next input to the model. The RNN state returned by the model its fed back into the model, so that it now has more context instead than only one word.

After predicting the next word, the modified RNN states are again feed back into the model, which is how it learns as to get more context from the previously predicted words. Looking at the generated text you will see the model knows when to capitalize make paragraphs and imitates a Shakespeare like writing vocabulary.

(Refer Slide Time: 24:19)



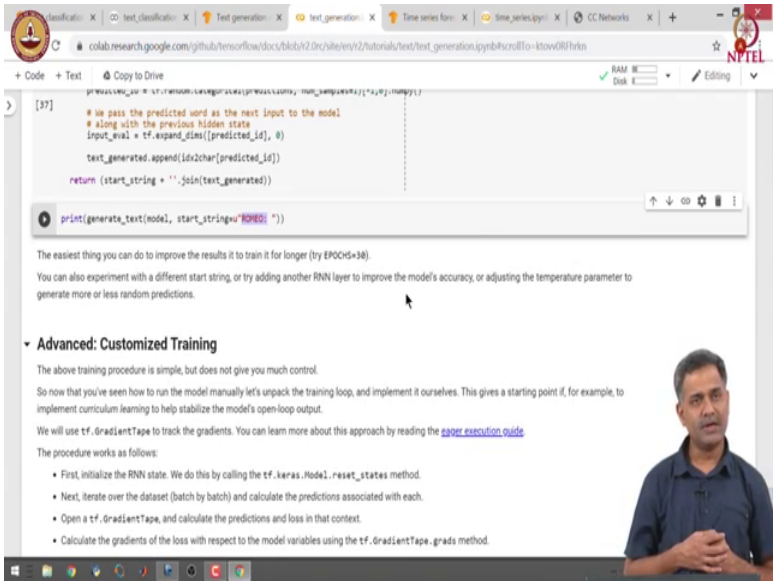
With a small number of training epochs, it has not yet learned to form coherent sentences.

So, let us look at how to code in python. So, we specify the number of characters to generate, we convert our start string to its number. This is the array that we will be storing our result; you specify the temperature value low temperature results in more predictable text. If you want more surprising text we need to set up higher temperatures. Temperatures we use batch size of 1, you first obtain the prediction from the model. We remove the batch size dimension we use categorical distribution to predict the word

written by the model. We get the prediction id; we pass the predicted word as the input to the model.

So, input_eval is get expanded by including the predicted word and we append the predicted character to do generated text. So, we run this in the loop to generate in this case about thousand characters. Finally, we return the generated text.

(Refer Slide Time: 26:16)



You can improve the results, if the results are not good you can improve them by increasing the number of EPOCHS. You can also expand with a different start string or try another RNN layer to improve the model accuracy. Other way to improve the model accuracy is by setting appropriate temperature parameter.

So, LSTMs or in general RNNs are very powerful models are being used extensively for sequence mining problems.