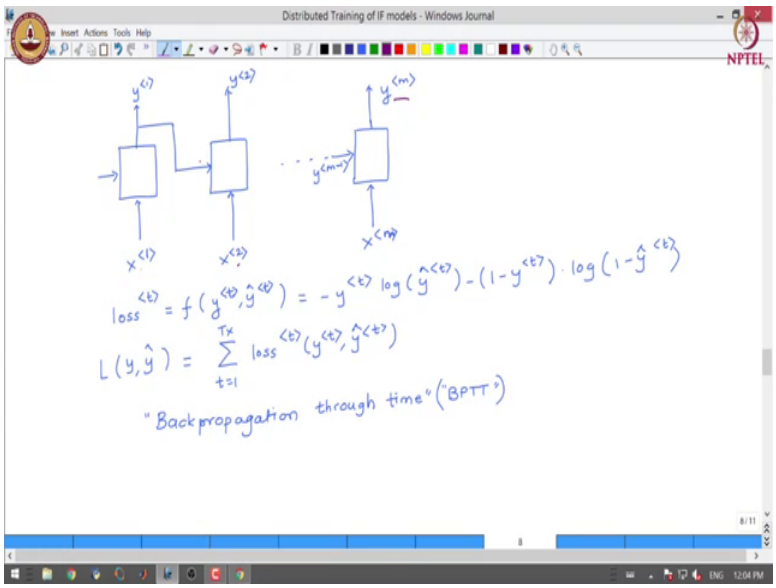
Practical Machine Learning
Dr. Ashish Tendulkar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 30
Recurrent Neural Networks (Part 2)

So, let us try to understand how the loss is computed in case of simple RNN.

(Refer Slide Time: 00:27)



We use the input, perform non-linear activation over linear combination to obtain the output for each position.

So, we get the error at every position. So, for example, the loss at t^{th} position is a function of the actual value and the predicted value.

$$loss^{<t>} = f(y^{<t>}, \hat{y}^{<t>}) = -y^{<t>} log(\hat{y}^{<t>}) - (1 - y^{<t>}) log(1 - \hat{y}^{<t>})$$

In case of classification we can use cross entropy loss. So this is the loss that we incur, this is a cross entropy loss that we incur at each position in the sequence. So, the total loss of the total loss due to the prediction can be obtained simply by summing across the sequence length.

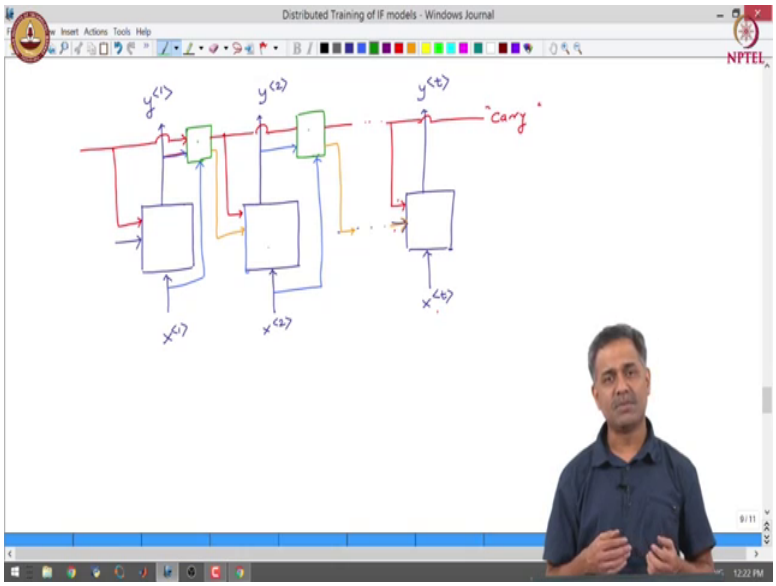
$$L(y, \hat{y}) = \Sigma_{t=1}^{T_x} loss^{<t>}(y^{<t>}, \hat{t}^{<t>})$$

So, we obtained this loss at each position and in order to calculate the gradient of the loss with respect to the input. We use an algorithm called back propagation. So, in case of RNN; it is called as back propagation through time or BPTT.

So, each of these outputs are through this link dependent on each of the input variables and the outputs of all the previous states. So, we use back propagation through time to calculate the weights that minimize this particular loss. We know that simple RNNs suffer from vanishing gradient problem and we also know that there is a neat trick that was used to make changes in the architecture so that vanishing gradient problem can be addressed.

So, let us look at the modified RNN architecture that takes care of vanishing gradient problem; it is called as LSTM or Long Short Term Memory.

(Refer Slide Time: 06:51)

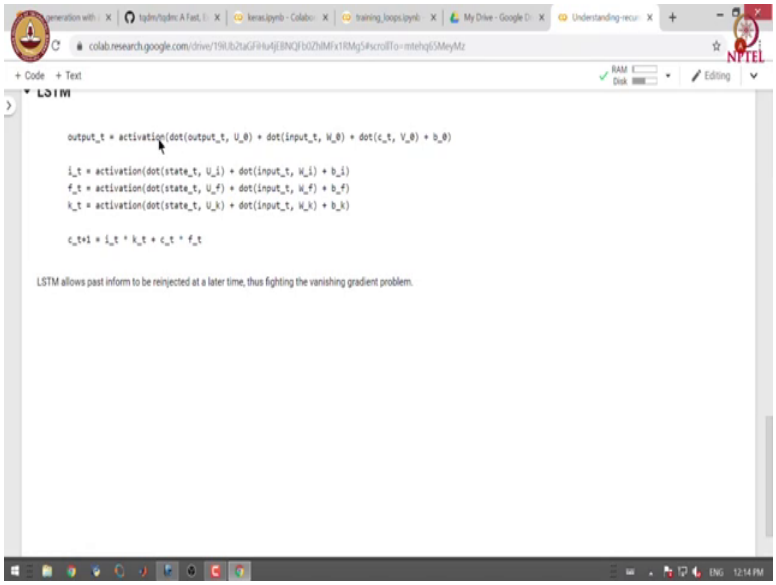


So, beginning with our simple RNN. We had this architecture for simple RNN where we gave a feature vector corresponding to each position as input. We obtain output for each layer and in addition to the input feature vector, we have a recurrent connection which is the output from the previous node that is also used as an additional input for calculating the output.

In order to solve the vanishing gradient problem faced by the simple RNNs because they cannot really remember information from some previous node. We add a provision to carry information from previous node and use that as an additional input while calculating the output.

So, now in case of simple RNN, we just had the input vector of the particular position and the recurrent connection. In case of LSTM, we have an additional input which is carry which is some information from previous states. Now we have to figure out how to decide which information to carry on the carry line. Let us try to understand how to decide that particular information. We will be using the same strategy as simple RNN for deciding what to carry in the next state. Let us understand that through a pseudo code.

(Refer Slide Time: 10:03)



So, the output will be computed based on the dot-products of the output from the previous state or the recurrent connection along with its weight; the input vector and its rate, the carry information in its weight and the bias term. We add all these dot-products and biases and apply non-linear activation on that to get the output at t^{th} position. In order to decide what to carry further, we define three different non-linear activations on some dot-products.

$$output\_t = activation(dot(output\_t, U\_0) + dot(input\_t, W\_0) + dot(c\_t, V\_0) + b\_0)$$

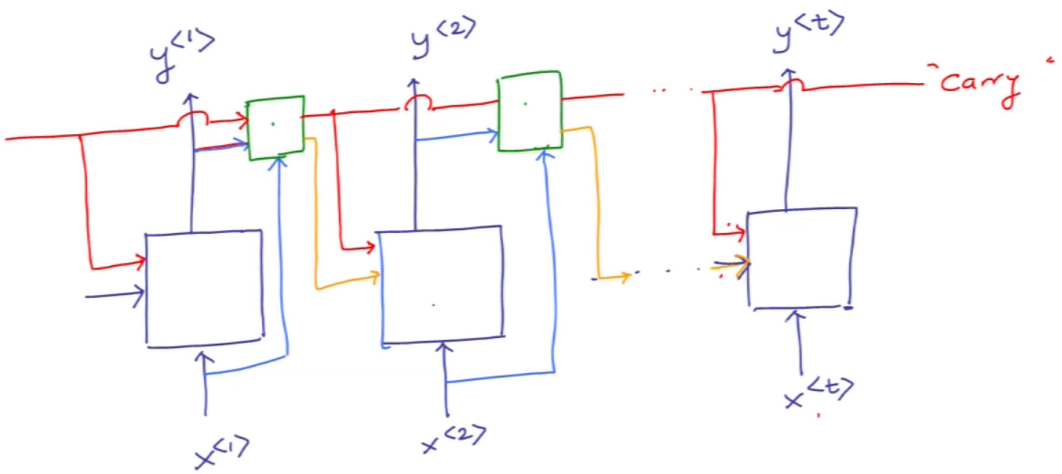
So, we define the first output i_t which sums up the output from the previous state or from previous timestamp and its feature vector. Then this is the linear combination or this is a dot-product between the feature vector of that position and its weight and the bias term; f_t is calculated by applying non-linear activation on the dot-product between the output of the previous state and its feature vector and dot-product between the feature vector and its weight vector and the bias, k_t is again calculated in the same manner.

$$i\_t = activation(dot(state\_t, U\_i) + dot(input\_t, W\_i) + b\_i)$$
$$f\_t = activation(dot(state\_t, U\_f) + dot(input\_t, W\_f) + b\_f)$$
$$k\_t = activation(dot(state\_t, U\_k) + dot(input\_t, W\_k) + b\_k)$$

So, all these three numbers are calculated based on activation applied to dot-product between the recurrent connection or the recurrent information and its weights; adding that into a dot-product between the feature vector and its weight, adding that to the bias term. And we decide a next carry is by multiplying i_t with k_t and c_t with f_t. c_t is the carry coming in and this is the formula for deciding the carry for the next stage.

$$c\_t + 1 = i\_t * k\_t + c\_t * f\_t$$

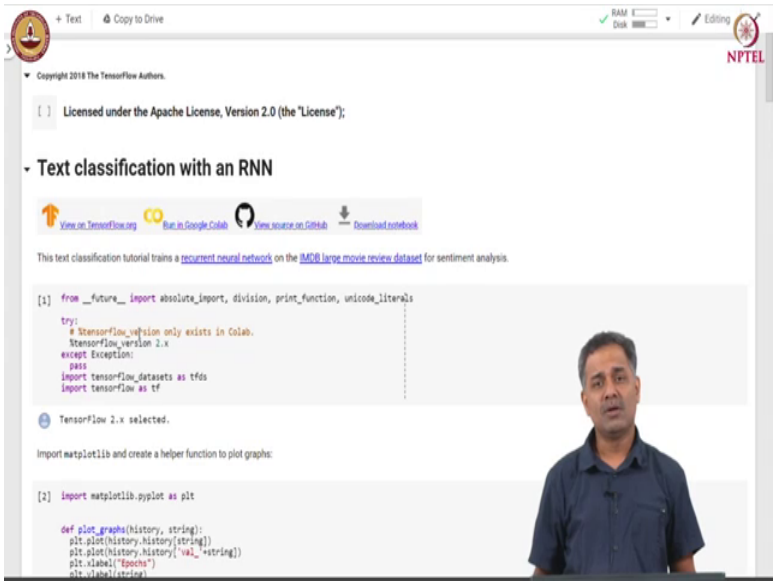
So, let us try to demonstrate this in a picture. We will insert a small block to calculate carry. So, here we calculate the new carry and that carry is passed on the carry line and we also pass the state information coming from this box to the next box. So, these are architectural details. Most of the time we find such kind of architectures by doing search in the architecture space.



We will take this LSTM model and try to use this in practice to solve some of the problems related to text and time series. So, this is the LSTM model; the only change that we make is we use a carry information and we perform additional computations to decide what should be carried to the next state. We learn the weights of feature vectors used in all this calculation through the training of these LSTM models.

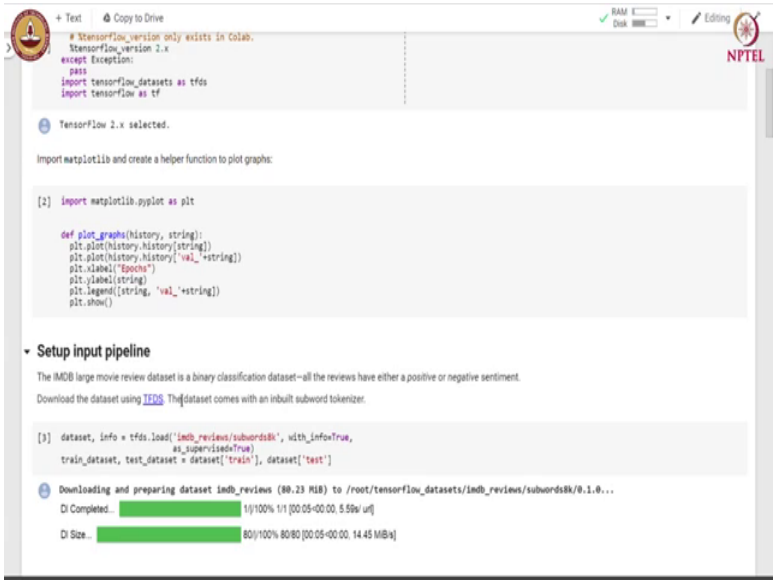
You can see that because of these different operations LSTM have far more number of parameters as compared to simple RNN. But LSTM models are quite powerful and they are showing state of the art results on lots of sequence learning tasks. Let us experience LSTMs in action by going through some of the practical examples.

(Refer Slide Time: 17:15)



So, let us use RNN models to obtain sentiment of movie reviews. In this example we will use movie reviews from IMDB and we will use RNN classifier to predict the sentiment of each of the movie reviews. The output here is binary; the review can either be positive or negative. We begin by importing the necessary libraries and downloading the movie review data set using tensorflow datasets.

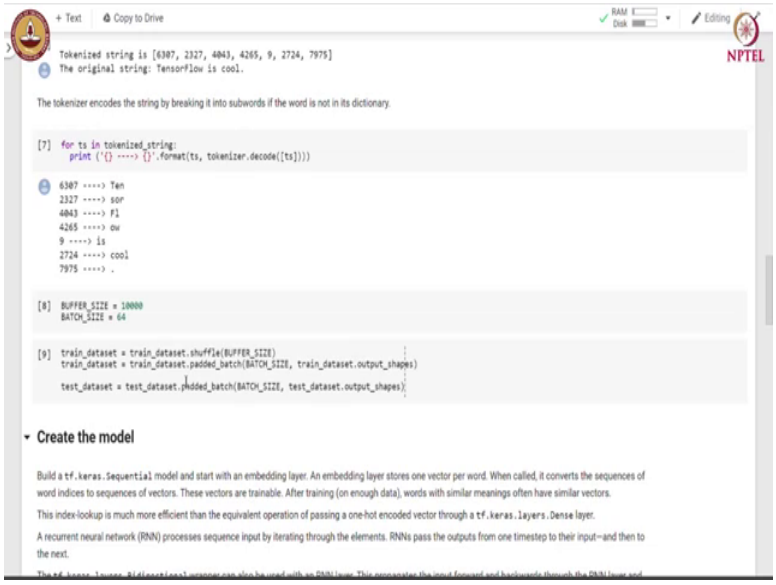
(Refer Slide Time: 18:15)



This data set comes with an inbuilt sub word tokenizer.

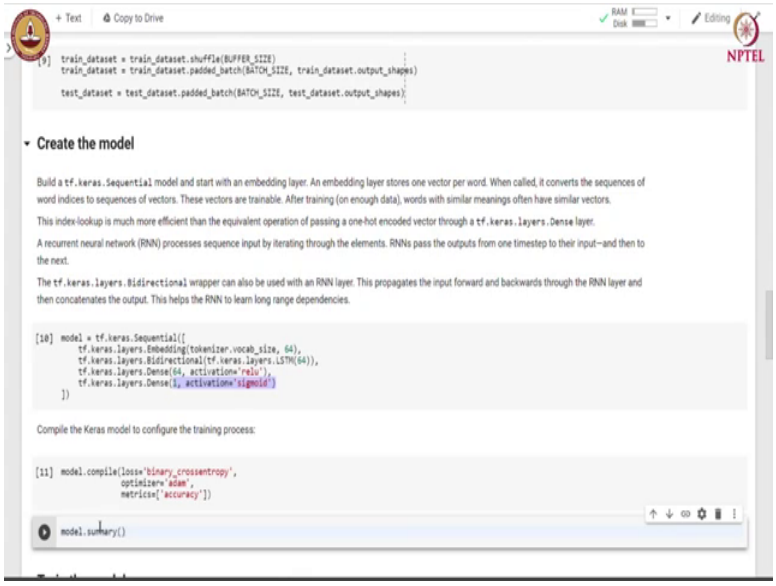
(Refer Slide Time: 18:24)

(Refer Slide Time: 18:27)



After downloading the data set, we will shuffle the data set and we use padded_batch function to obtain the training data set where each sequence has a fixed length. We also used pattern_batch method to obtain the test data set.

(Refer Slide Time: 19:01)

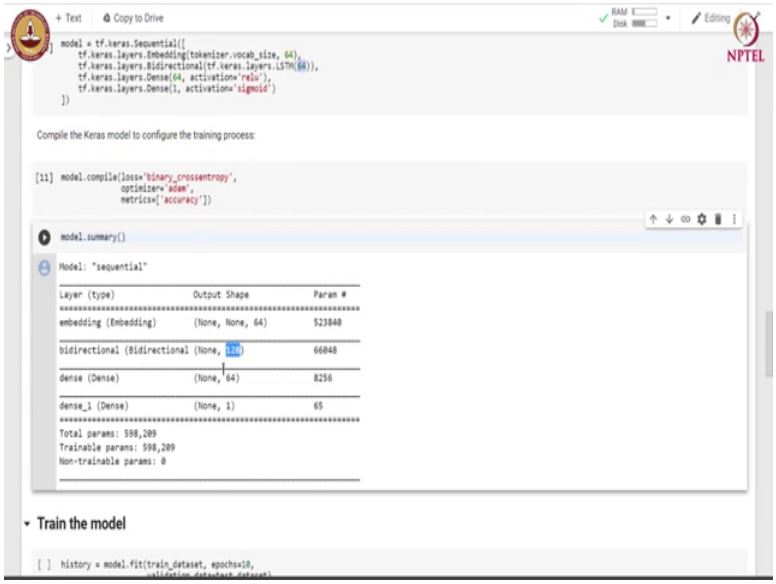


Let us come to the model creation part; we are going to use RNN models over here. Here we use a bi-directional model on top of LSTM; the bi-directional wrapper propagates the

input forward and backward through the RNN layers and then concatenates the output. This helps RNN to learn long range dependencies.

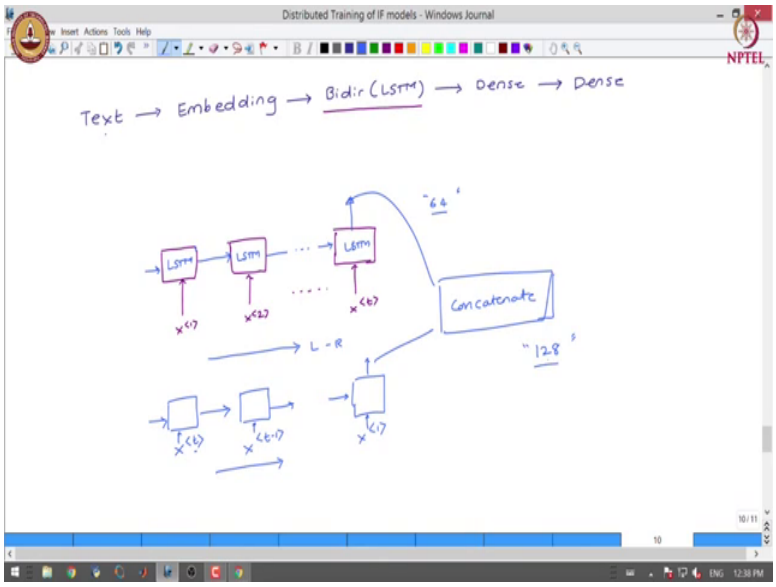
So, we take the input which is text and pass it through the embedding layer which gets us a vector for each word. We embed each word into 64 length vector. We pass the output of embedding to bi-directional layer. So, the output of bi-directional layer is passed to a dense layer with 64 units and we use radio as an activation function. The output of dense layer is passed through the output layer which is again a dense layer with a single unit. As we have binary classification problem here and we use sigmoid as an activation function here. Let us define the model, compile it and look at model summary.

(Refer Slide Time: 20:55)



So, you can see that embedding outputs 64 numbers and the bi-directional LSTM output 64 numbers for each direction: forward 64, backward 64. So, concatenation of that results into output containing 128 numbers. The dense layer output 64 numbers as we are using 64 units here and the final layer outputs a single number.

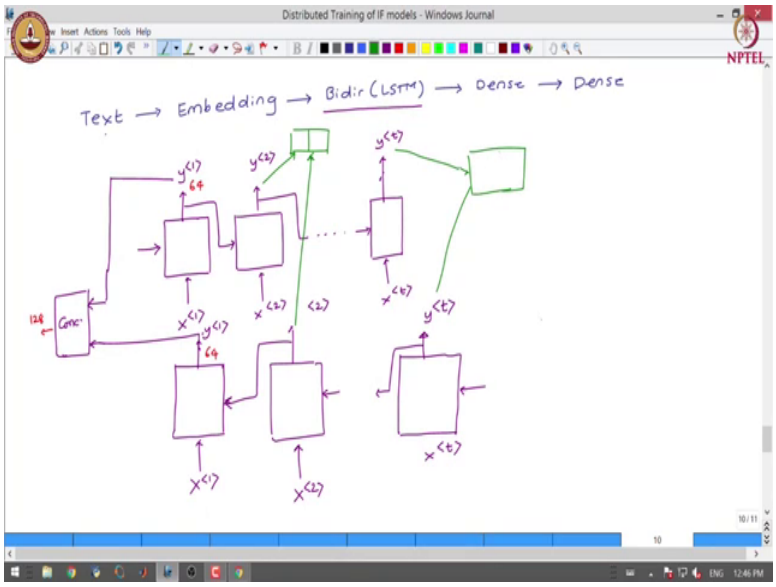
(Refer Slide Time: 21:46)



We start with text we have embedding layer followed by bi-directional LSTM followed by our dense layer and one more dense layer which is an output layer. We have let us say an input sequence of length *t*. So, let us say these are all LSTM units. So, here we are passing the recurrent connection in this particular direction from left to right; we actually get 64 numbers from the LSTM.

We are essentially passing that output to the next level and we are collecting the outputs only at the last layer. In the other pass, we start with last word and do LSTM calculations and pass the recurrence from right to left; in a sense because we started with the last word and we are going up to the first word and then both these outputs kind of concatenated and we get 128 units from this.

(Refer Slide Time: 25:07)

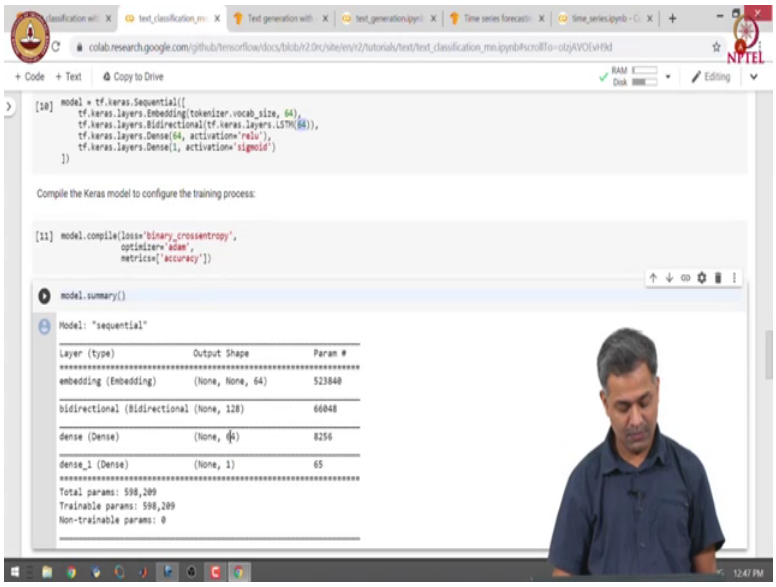


We start with LSTM model, then we take an input. So, we have LSTM model that outputs 64 number for input at each position. So, this is the forward pass because the output of t^{th} position or i^{th} position is being used as a recurrent connection in express i^{th} position. For example, the output of $x^{<1>}$ is being used as a recurrent connection for $x^{<2>}$.

So, this is called the forward pass in bi-directional one; we also define some kind of a backward pass, where you pass the result of i^{th} node to $i-1^{th}$ node as a recurrent connection. For example, for the second position; the output of the second position will be passed back to the first position and will be used as a recurrent input. And we concatenate the outputs of let us at first position; the forward pass and the backward pass.

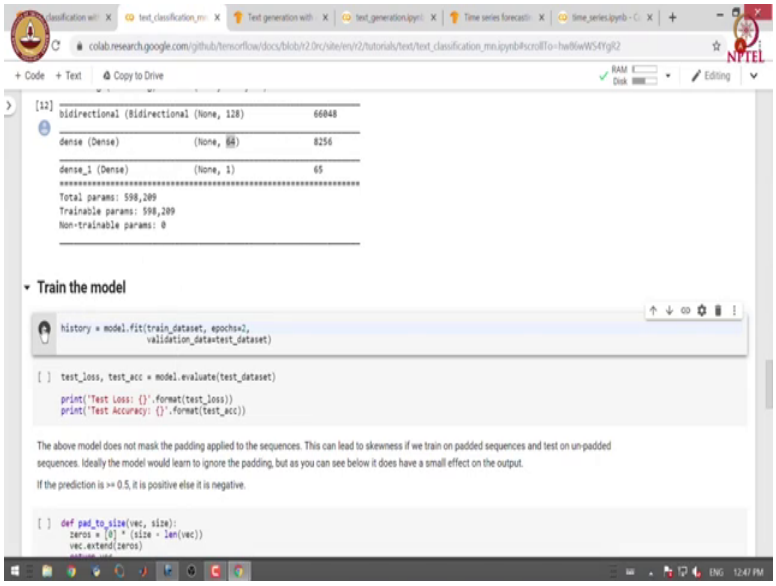
So, since here we are outputting 64 values each; the concatenation outputs 128 values. So, we perform this concatenation at each position for example, these two will get concatenated or these two will get concatenated and each one of them will output 128 numbers.

(Refer Slide Time: 29:19)



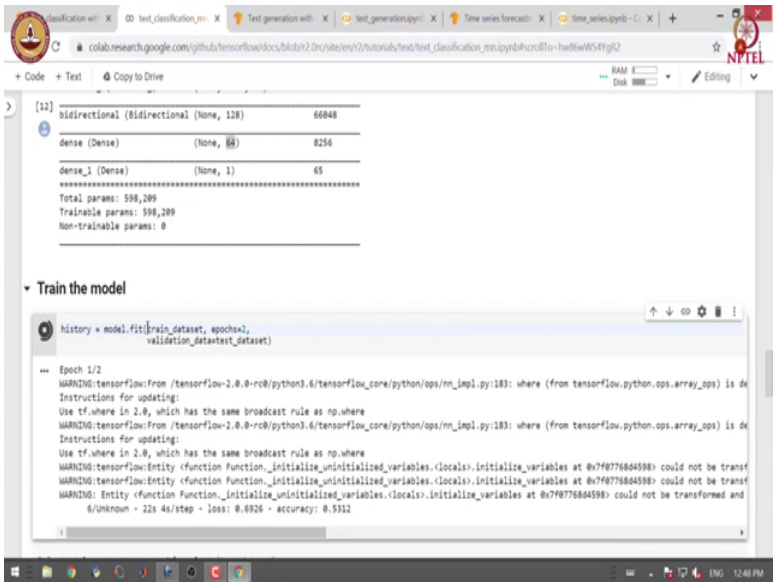
And is 128 numbers are passed to dense layer which has got 64 units.

(Refer Slide Time: 29:31)



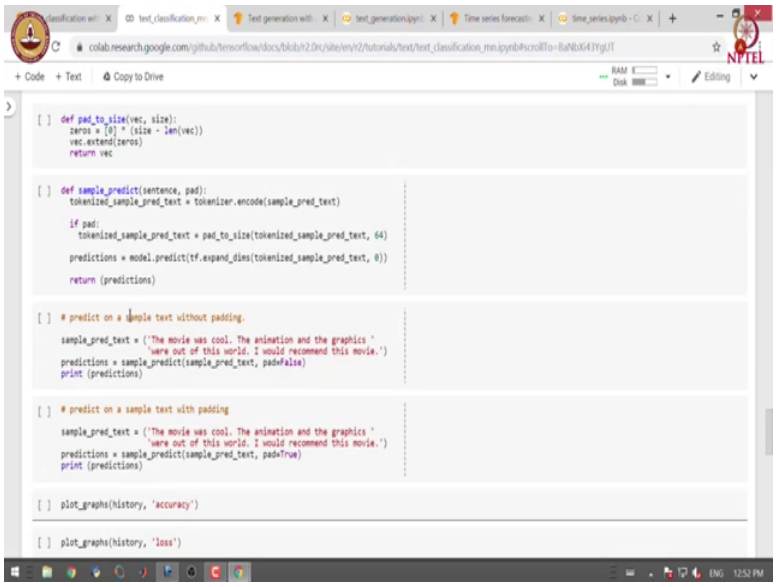
After setting the model; we will train the model for few epochs so that we get to experience output.

(Refer Slide Time: 29:46)



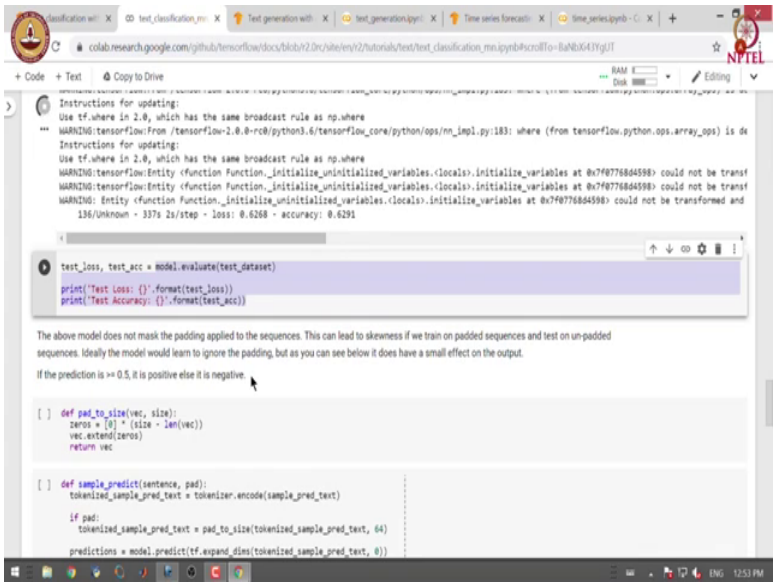
And we store the progress of the model in the history object, so that we can later plot how the training progressed.

(Refer Slide Time: 30:09)



The training usually takes longer to complete because we are trying to train on a large data set.

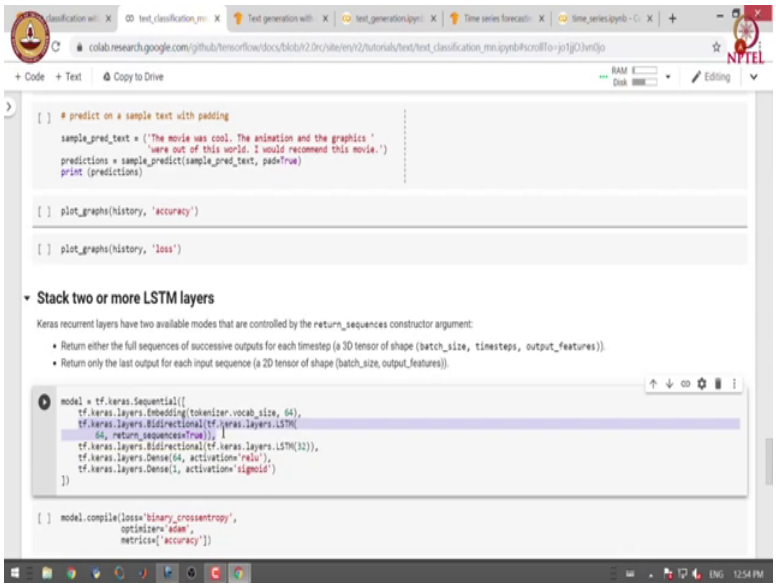
(Refer Slide Time: 30:31)



We can use the evaluate function on the model and obtain the test loss and the training loss. If the prediction has probability greater than 0.5, we mark it as a positive review; otherwise we mark it simply as a negative review.

We will also check how the model performance effects when we give sample text without padding and with padding. Ideally, the model should learn to ignore the padding, but you will experience that there is some effect of padding on the output.

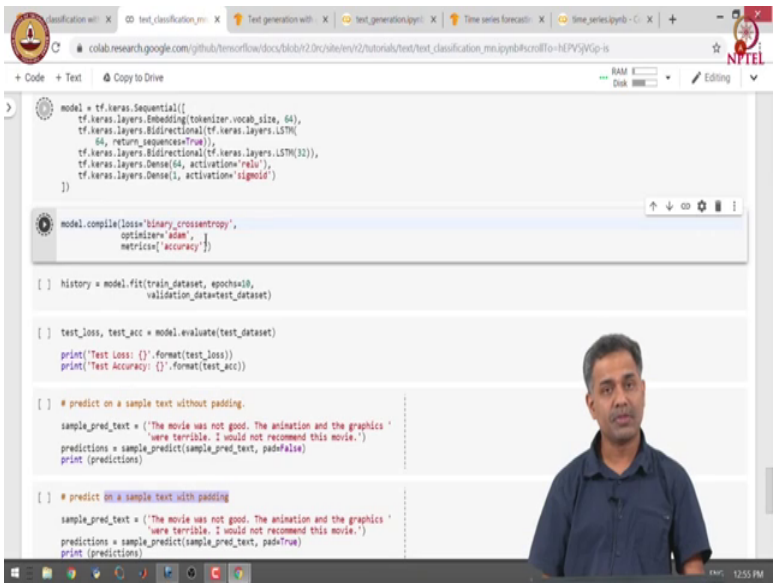
(Refer Slide Time: 31:39)



So, now that you have trained our first model with a bi-directional LSTM, we will try to stack up couple of bi-directional LSTM and get model with more complexity. Let us see how to stack up different LSTMs and obtain in the model with more capacity. So, here we define the first bi-directional LSTM model; here we put return_sequence = *true* so that we get output from each node.

Each LSTM outputs 64 numbers in each direction, so the concatenation that happens in bi-directional LSTM will result into 128 numbers coming out of this particular layer. The second bi-directional LSTM will contribute to 64 numbers that will be passed into another dense layer followed by an output layer.

(Refer Slide Time: 32:55)



You can compile the model and fit the model and use the train model to calculate the loss and accuracy on the test set.

Just in case of the earlier model where we used a single bi-directional LSTM. This is the first example where we used bi-directional LSTM for predicting sentiments of movie reviews. We will have couple of more examples of using LSTM models for time series forecasting and for text generation.