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

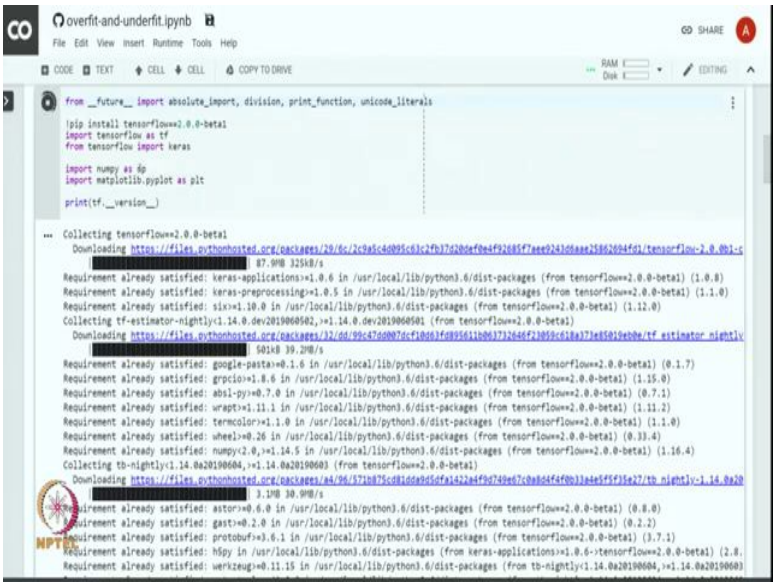
Lecture - 18
Underfitting and Overfitting

Welcome to the next module of our course in this module, we will understand Underfitting and Overfitting through the code. You know that overfitting happens when our model has excess capacity to memorize the entire training data. So, what happens if we look at the learning curves we observe that the training error and validation error both reduce to begin with.

After a point the training error goes down, but validation error starts climbing up, if we are seeing that those kinds of learning curves we infer that the model is suffering from overfitting. On the other hand if your model is so simple that it does not have enough capacity to learn the model or to learn the patterns in the training data, then we then our model is suffered from underfitting.

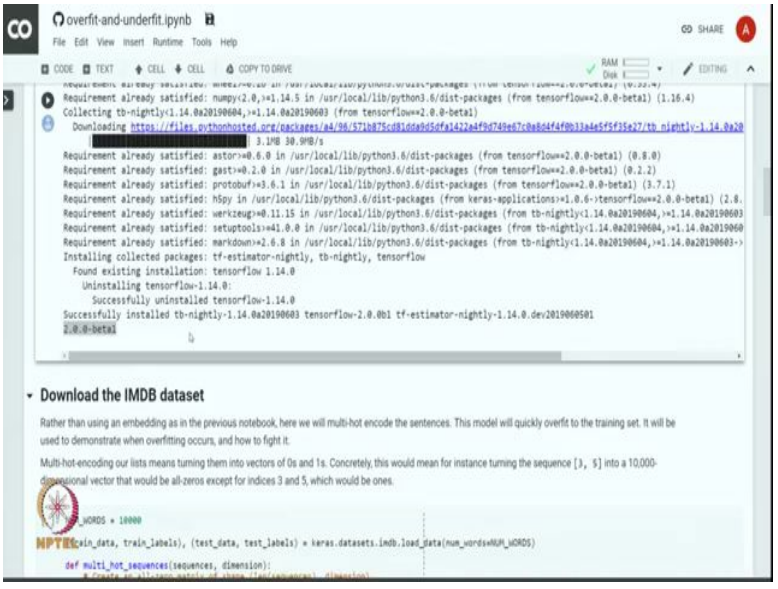
In case of underfitting both training and test error are high. In this lab we will use IMDB movie review dataset. To demonstrate underfitting and overfitting we will initially build a baseline model, then we will build a model to underfit the data and overfit the data. We will first build a baseline model and then we will build a couple of models, so that our model underfits and overfits to the training data. Later in the lab we will demonstrate some of the techniques that help us to overcome the underfitting and overfitting problems.

(Refer Slide Time: 02:02)



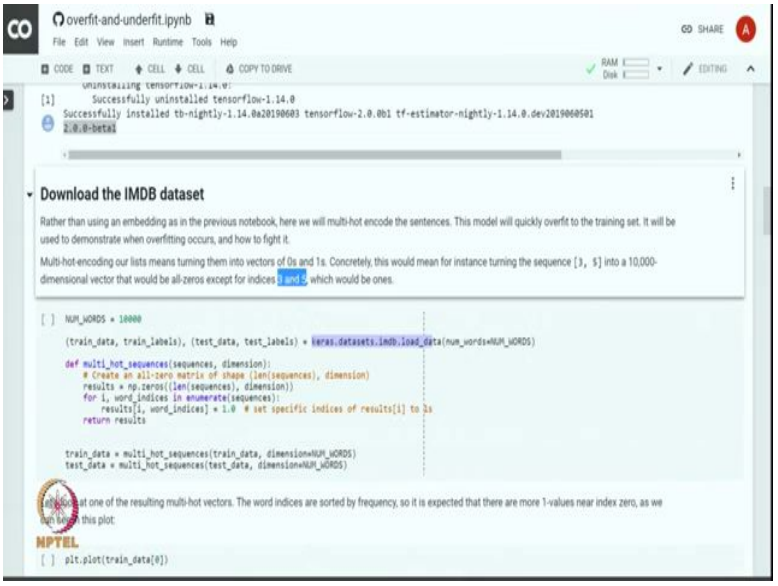
Let us first connect to the colab collabrant time, install and import tensorflow 2.0, numpy and matplotlib.

(Refer Slide Time: 02:44)



So, you can see that we have successfully installed tensorflow 2.0 through the print command.

(Refer Slide Time: 03:04)

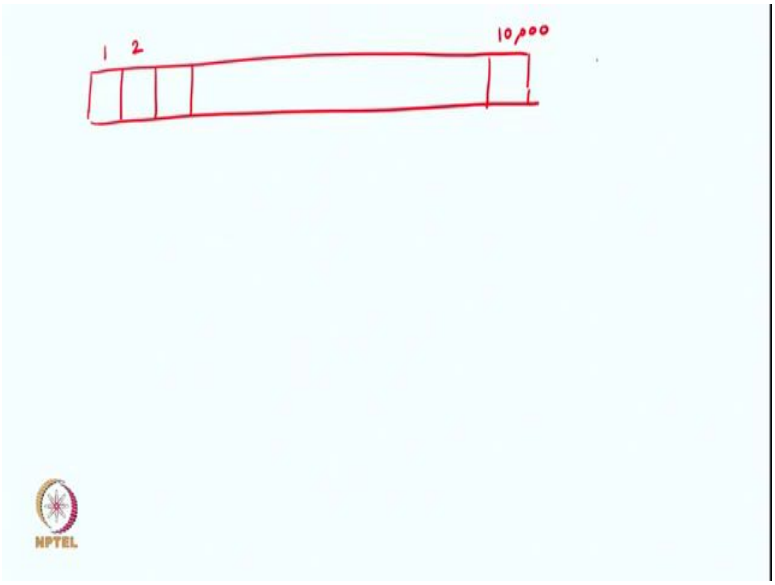


Next step is to setup the training data. IMDB dataset has movie reviews and each of the movie reviews is tagged with a label 0 or 1, 0 meaning the movie review is a negative and 1 meaning the movie review is positive.

So, it is a problem of identifying whether the movie review is positive or not. IMDB dataset is available in the keras.datasets. So, we do not have to write a lot of code to load the IMDB dataset. We can easily load IMDB datasets and we will be using a multi-hot encoding, where we turn the words into vectors of 0 and 1 in 10000 dimensional vector space.

So, if the word is present at a specific position that we will see one over there. Concretely, if we have a word whose index is 3 and 5 present in the in the in the document it gets converted into a 10000 dimensional vector, where all the entries are 0s except for indices 3 and 5.

(Refer Slide Time: 04:55)

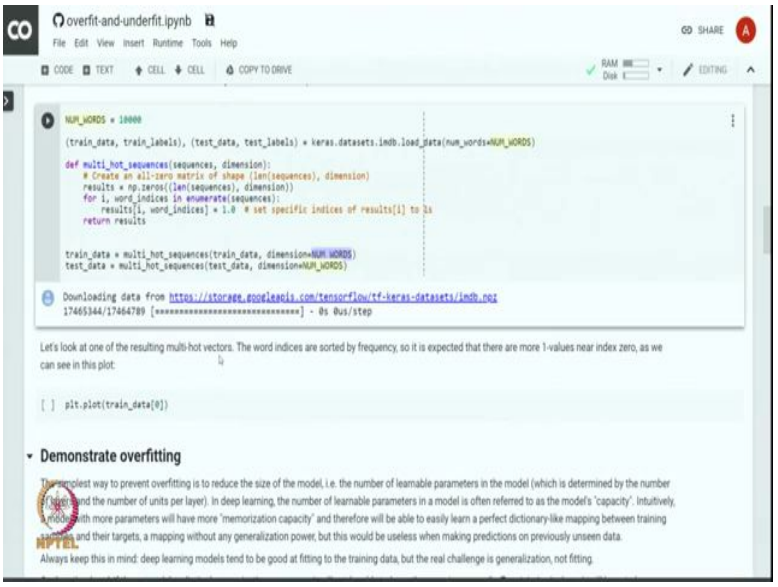


So, let us try to understand that every review is converted into a multi-hot encoding, where we have we have a vector with 10000 positions. So, initially we convert each movie review into a sequence of word indices. So, the IMDB dataset is maintained as a sequence of numbers in the keras, we convert each review into a multi-hot encoding.

That means, if there are two words with indices 3 and 5 present in a review only the position corresponding to those words will have 1 in the list, rest of the other elements will have 0 in their list. So, that is what that is how we convert the review into multi-hot encoding. So, you can as you can see in the code we use a multi-hot encoding with 10000 words, we use the load_data command to a load training and test data along with their labels and then we define a function for converting the sequence into multi-hot sequences.

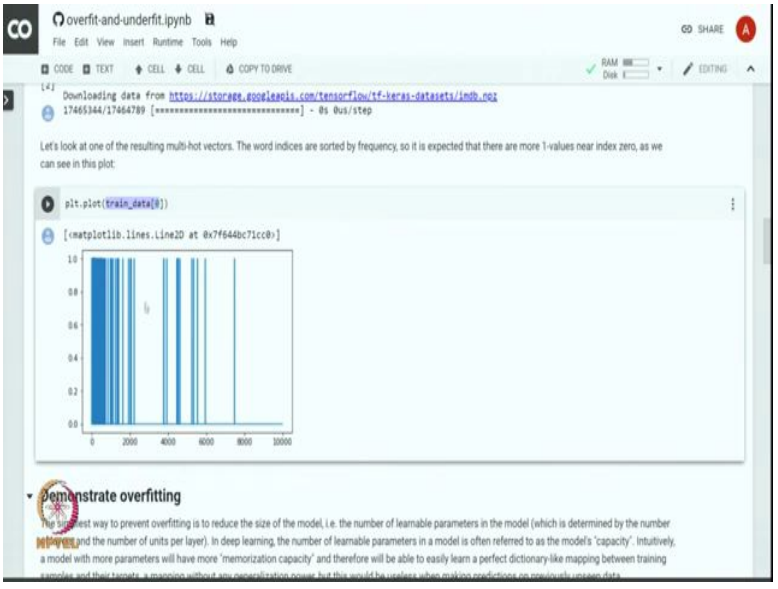
So, this particular function takes into takes sequence and dimension as argument and it essentially defines or it essentially tries to fill a vector of dimension 10000. We first convert the training data into multi-hot encoding and test data into multi-hot encoding using multi-hot sequences function and we are going to use the number of dimensions to be 10000.

(Refer Slide Time: 07:18)



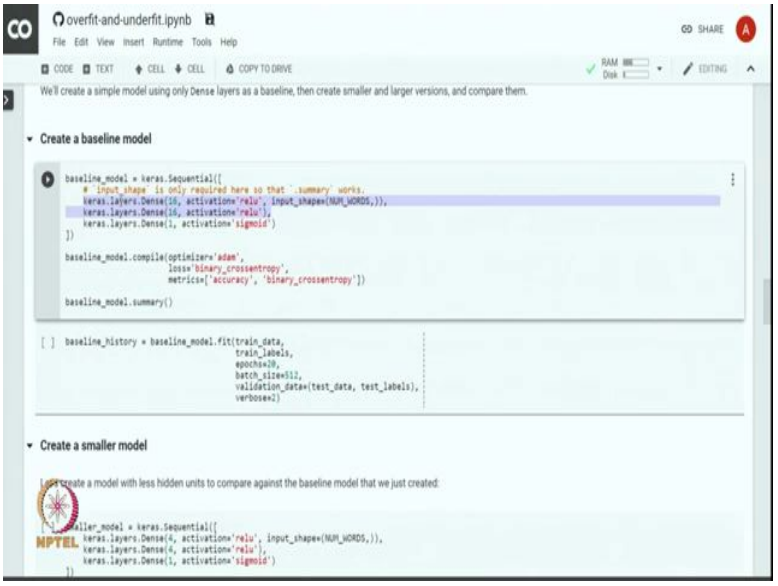
So, let us run this and convert training and test data into multi-hot encoding. Now you can see that the data is being downloaded and getting loaded into memory and the data is converted into multi-hot encoding here. Let us look at some of the resulting multi-hot vectors.

(Refer Slide Time: 07:38)



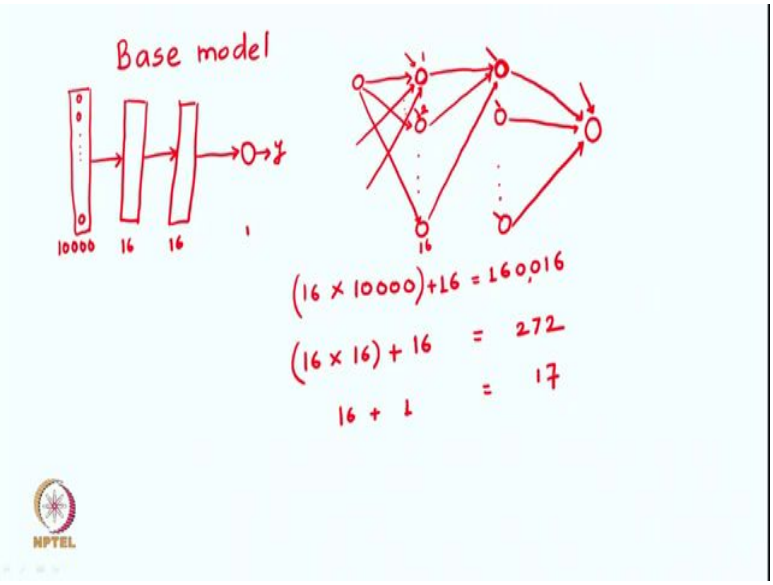
So, here we have plotted the multi-hot vector for the first training example and we can see that the index where the word is present is 1 everything else is 0. So, you can see that there are a lot of words from the initial index are present in this particular example and fewer words from letter indices are present here.

(Refer Slide Time: 08:07)



Now, that data is loaded let us try to build the model. We will first build a basic model where we will use where we will use a neural network with two hidden layers each layer having 16 units each. So, let us look at how this neural network looks like.

(Refer Slide Time: 08:26)

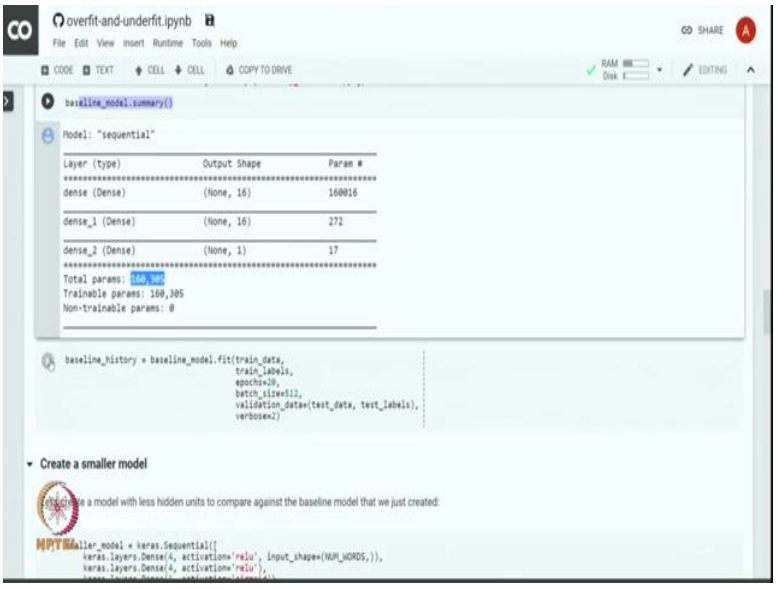


In base model we have neural network which has two layers of 16 units and then there is a single output unit and we have a multi-hot encoded vectors. So, the number of inputs is 10000. So, each of the 10000 inputs are fed into the first unilayer and the second unilayer and

in the output layer which gives us the probability of the review belonging or review being a positive review. So, let us see how to set this up in the code.

So, we build a keras.Sequential model, where we are going to stack layers one over the other. We are going to use two dense layer each with 16 units with activation of relu. The output layer has one unit with sigmoid activation. We use adam as an optimizer and binary cross entropy as a loss because we are trying to solve a binary classification problem and we will track accuracy and binary cross entropy both during the training and this particular command will summarize the model.

(Refer Slide Time: 10:23)



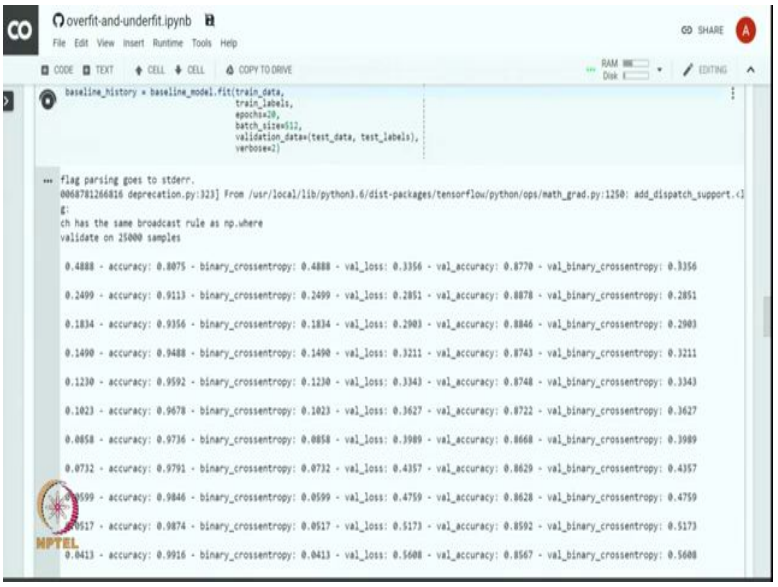
The model summary is a useful way to understand what is going on in the model and to check whether model is setup as per our expectation. So, you can see that there are there are three layers in the model. So, the first hidden layer has 16 units, the second one also has 16 units and then the output layer has one unit.

Each unit from the first hidden layer has 10,001 parameters each because there are 10000 inputs plus a bias uni. This adds up to 160016 parameters. Similarly, we have 272 parameters for the second hidden layer and 17 parameters for the output layer.

So, after setting the model let us fit the model with the training data and validate it on the test data. So, we will train the model with 20 epochs with a batch size of 512 and we are going to

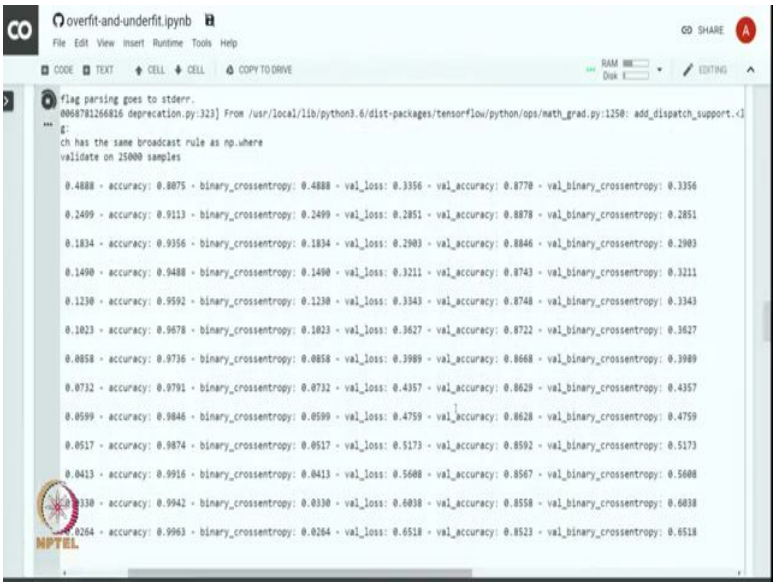
use adam optimizer for training the model and use binary cross entropy as a loss that will be optimized.

(Refer Slide Time: 14:40)



So, we can see that as we train the model the cross entropy loss is reducing, the validation loss is also reducing but after third layer it seems that the validation loss is going up while training loss is still going down and we can see the same thing happening for the cross entropy loss it initially went down then it started going up.

(Refer Slide Time: 15:08)



This points to some kind of an overfitting problem. So, we see that after 20 epochs our accuracy is 1 we got a perfect classifier. In validation, we only get 85 percent accuracy. Our job is to build a model that works well on the future data and not the one that works well under training data. So, there is a problem which we need to fix.

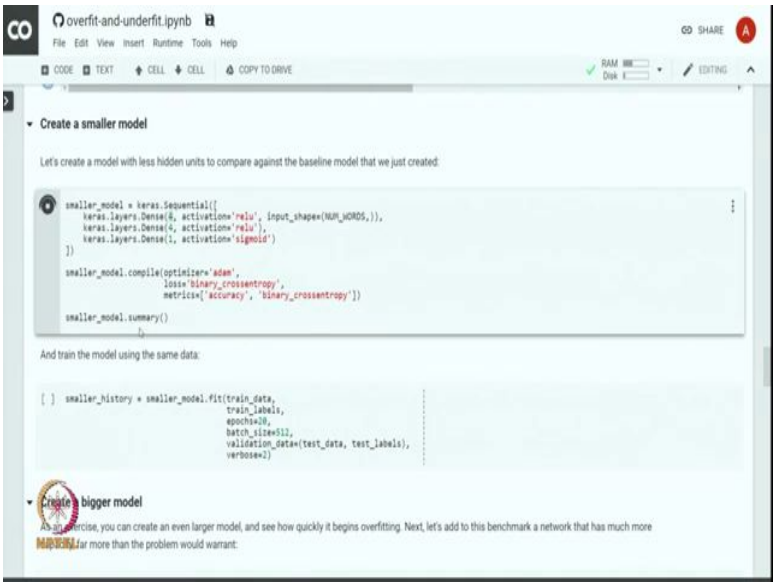
So, one way to one way to address the problem of overfitting is by reducing the number of parameters in the model. In neural network it is easy to reduce the complexity - we can either reduce the number of units in each of the layer or we altogether remove a layer that will result in lesser number of parameters. Another way to reduce the overfitting is by getting more of training data. If training data is not available then we have regularization techniques that we can use. In regularization, we can use either L1 or L2 regularization or a dropout regularization that is used in the context of neural networks.

In case of L1 and L2 regularization we add penalties that are proportional to the weights of the model. In case of L1 regularization, we add penalty that is proportional to the sum of absolute values of the parameter. In case of in case of L2 regularization, we had a penalty that is proportional to the sum of square of the value of the parameters.

In case of dropout we decide to randomly drop certain nodes from the hidden layer or input layer of a neural network. We normally said dropout between 20 percent to 50 percent that means 20 percent to 50 percent nodes will be randomly dropped out in each of the iteration in the neural network training. The dropout is only used only used while training, so in the same

manner even regularizations or dropout are used during training in the test we simply apply the test data on the model.

(Refer Slide Time: 17:51)

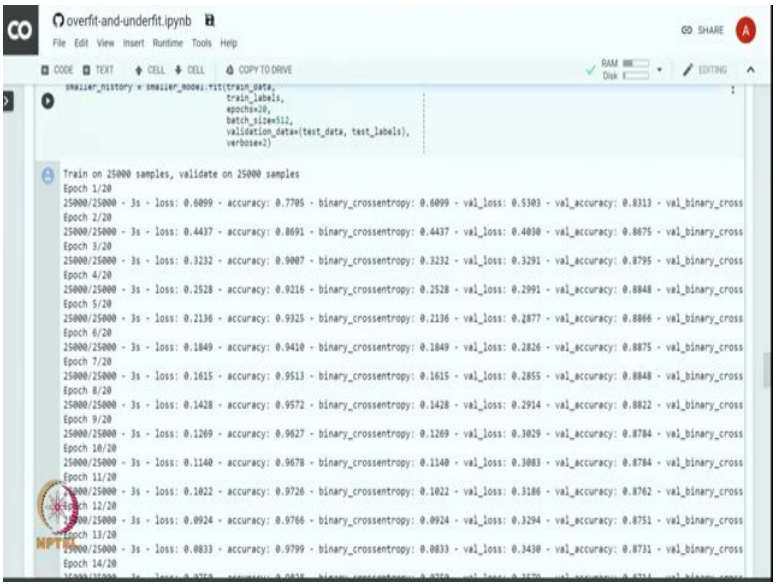


So, in the first strategy let us create a smaller model here. So, instead of using 16 units we used 4 units, so naturally the number of parameters will go down that we just have 4 units. So, let us run and see how many parameters are there in this.

(Refer Slide Time: 18:09)

So, you can see that the number of parameters have come down at least four folds. So, just by reducing the number of units in each of the layer we got it down to 40000 parameters. Let us train this smaller model and see what happens.

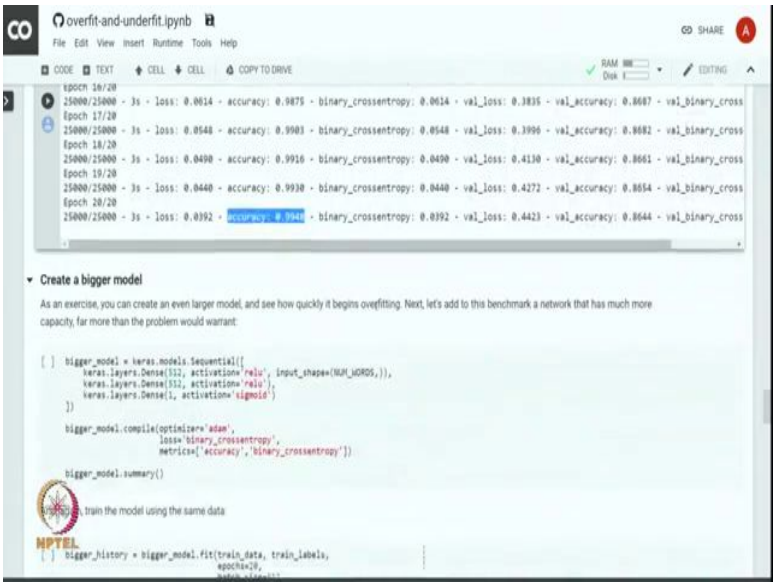
(Refer Slide Time: 18:44)



So, you can see that even this smaller model seems to be overfitting the training accuracy is close to 1 but validation accuracy is quite low. That means, the model is probably memorizing the training data it has not enough capacity to memorize the training data. But going back you can see that the model started overfitting somewhere around, we can look at the validation loss is going down and epoch 7, it started overfitting where validation data validation loss started going up.

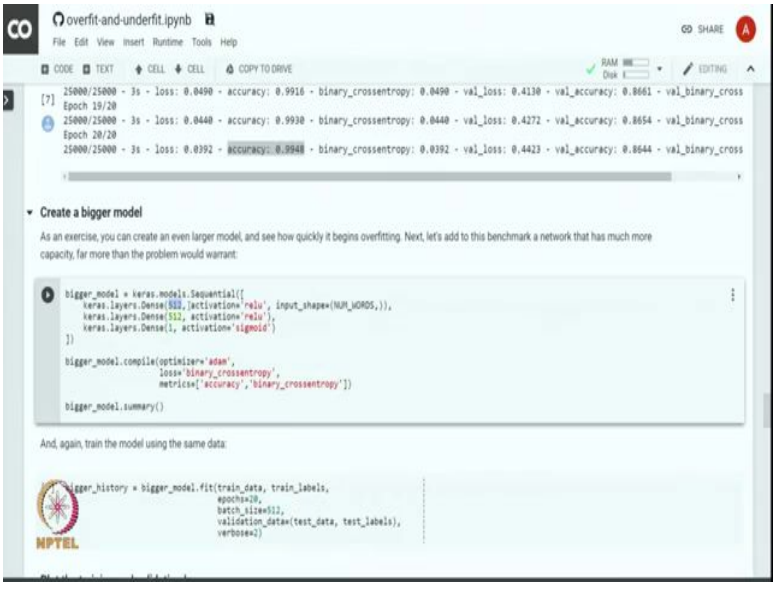
If we go back and check when our baseline model started overfitting our baseline model started overfitting right from the third epoch. So, you know by creating smaller model we are able to we are able to delay the overfitting of the model. So, if we use smaller model and if you stop training around fifth epoch we should still be fine.

(Refer Slide Time: 20:09)



If we stop our training after fifth epoch we will not get the overfitting that we see after 20 epochs.

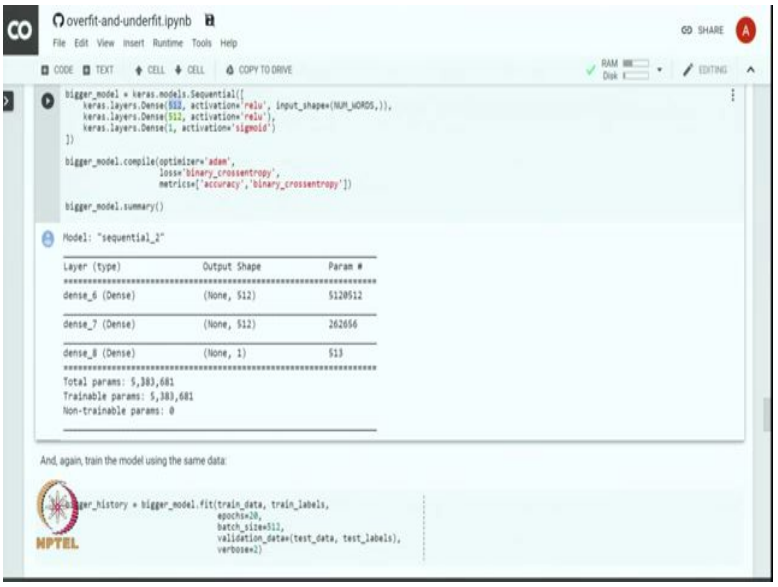
(Refer Slide Time: 20:10)



Let us go to the other extreme and create a bigger model and see how fast it overfits. We create a bigger model in neural network we can simply add more units in each of the layer or add more layer itself. So, that way we will have more capacity in the model or more parameters in the model and since we have more parameters in the model, if you do not have

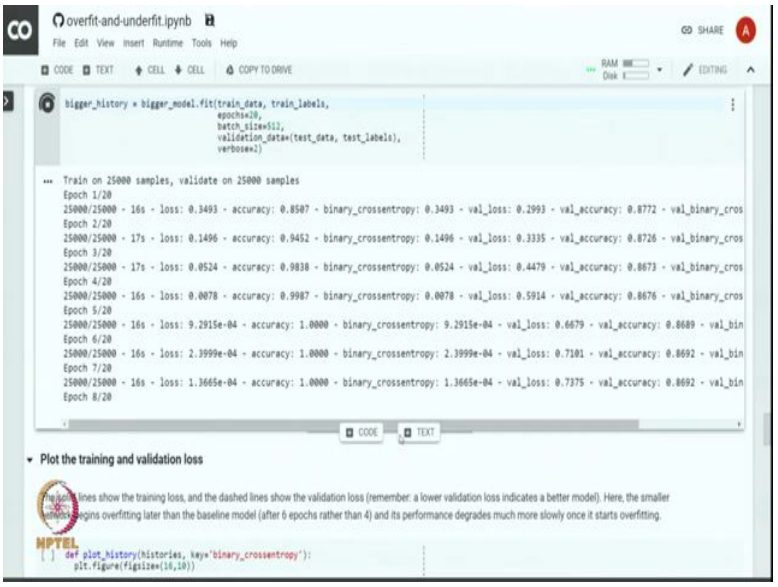
enough training data the model tries to memorize the training data and gives us perfect output on our training set but it did not perform well on unseen data.

(Refer Slide Time: 20:58)



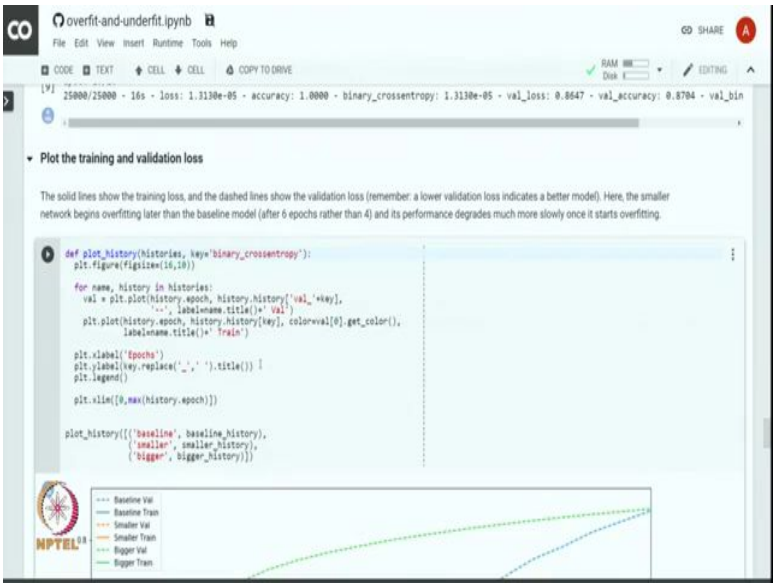
So, let us compile this model and you can see that now we have far more number of parameters. So, there are close to 5.3 million parameters as against 160k parameters that we saw in our baseline model.

(Refer Slide Time: 21:12)



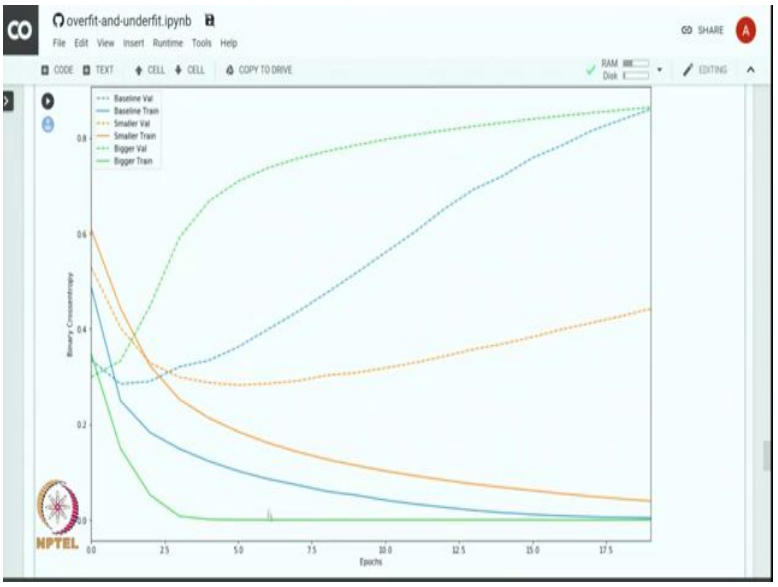
Let us try to train the model for 20 epochs with the same batch size of 512 you can see that after 4 epochs. So, the model right away started overfitting. So, you can see that the validation loss is increasing right after the second epoch. So, in the first epoch it was 0.29 it went up to 0.33 and then it went it kept going up all the way and at the same time you can see that the cross entropy loss or the training loss is coming down training accuracy is going up, but the validation accuracy is reducing. These are the signs that this is probably the model with quite a large capacity or excess capacity and it is proven to overfitting pretty quickly in the training cycle.

(Refer Slide Time: 22:25)



So, we can see that after training the model for 20th for 20 epochs you can see that the training loss is very small close to 0. But the validation loss is quite high and validation loss never reduced rather it started growing up right from the second epoch. So, this points to the fact that the model is overfitting pretty fast in the bigger model and we will compare how the size of the model affects the possibility of overfitting.

(Refer Slide Time: 22:58)



So, you can see that there are there are six lines in this particular plot. So, the solid lines are for the training loss and dotted lines are for the validation loss. So, all of them we will have will have the same number of iterations in one epoch and on the y axis we have got a binary cross entropy loss.

So, you can see that the green line corresponds to the bigger model the blue line corresponds to the baseline model whereas, the orange line corresponds to the smaller model. We will have to keep an eye on the dotted lines because those give us validation loss and we infer that there is a overfitting when validation loss starts going up while training loss keeps going down. So, for the smaller model, the validation and training both the losses are going down and around eighth iteration the validation loss started climbing up.

So, we can say that the smaller model started overfitting after 8 epochs. The baseline model started overfitting around third epoch. Whereas, in case of bigger model it overfit almost instantly right from the first iteration, training loss was coming down, but validation loss never came up. Hence, we see that the smaller the model the longer it will take to overfit.

(Refer Slide Time: 25:02)

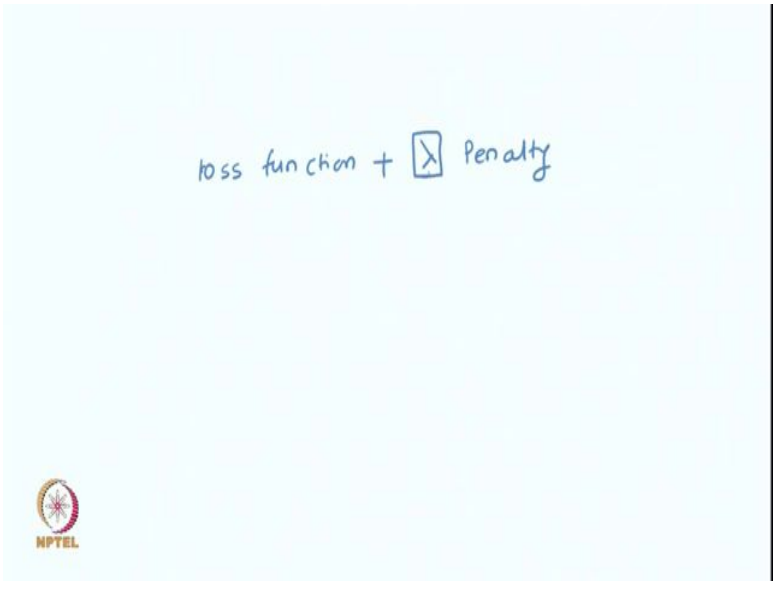
Now, that we have visualized the overfitting of the model, how can we prevent overfitting? So, there are three strategies - (1) you can get more data overfitting can be prevented. But getting more data is not always an option because getting more data is can be costly sometimes. (2) We have is to add regularization or a penalty term to the loss function. So, there are multiple ways of performing regularization couple of them are based on L1 and L2 penalty.

So, L1 regularization adds penalty proportional to the sum of absolute values of the weight, whereas L2 regularization adds penalty proportional to sum of square of the weights of the parameters.

(Refer Slide Time: 26:03)

Let us see how to add L1 and L2 regularization to the neural network through tf.keras.regularizers (Refer Time: 26:32). So, along with each of the layer dense we add along with each of the dense layer while defining the model we add a kernel_regularizer argument. So, kernel_regularizer we can either use l1 or l2. So, here we are using l2 and we are also specifying what is the regularization rate.

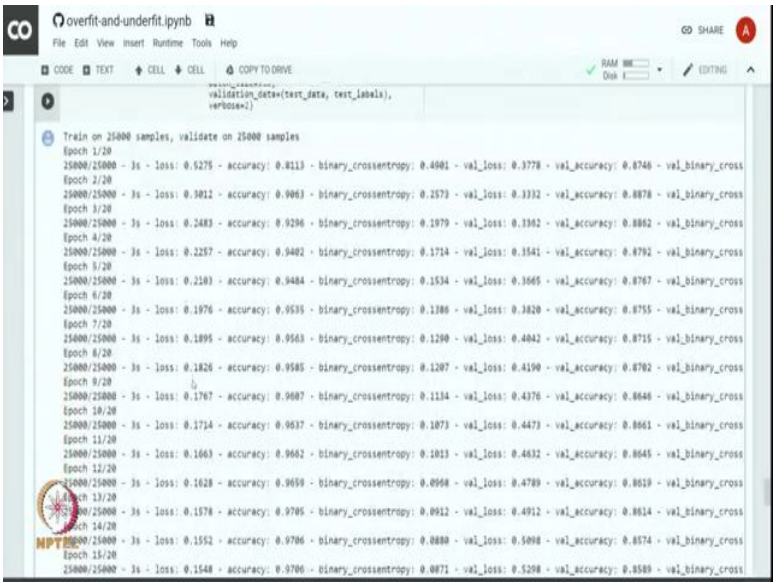
(Refer Slide Time: 26:57)



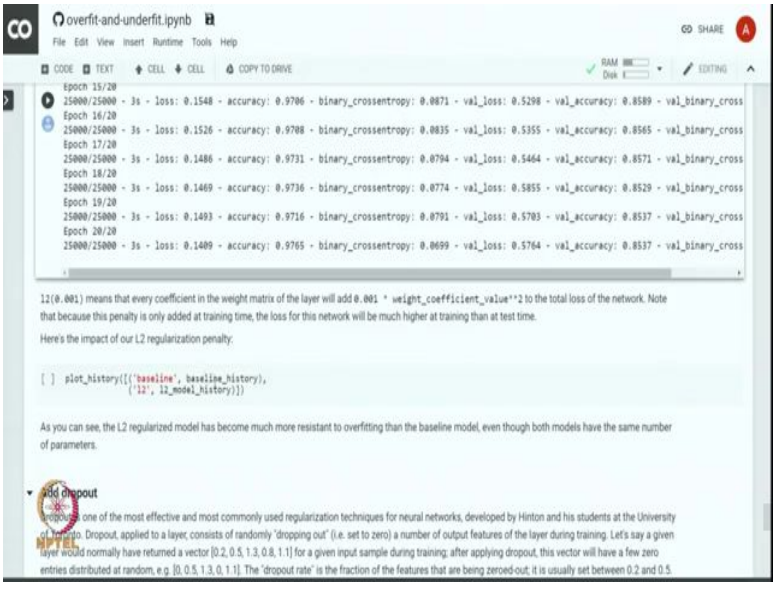
The regularization term for the loss function is added. Here, the regularization penalty is 0.001 and we are using l2 regularizer here. So, we use l2 regularizers in both layers in order

to prevent overfitting. So, this is how you can define your l2 regularizer and then we can compile the model and train it you can also add l1 regularization in the similar manner.

(Refer Slide Time: 27:39)

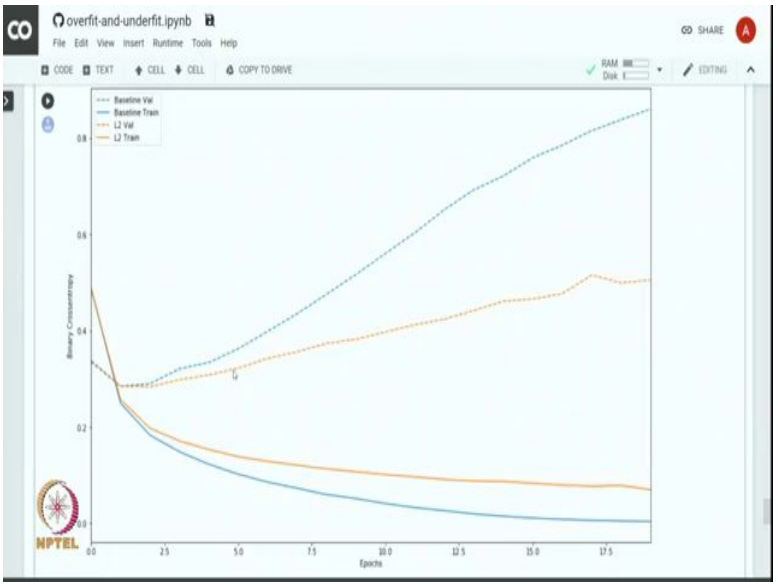


(Refer Slide Time: 27:56)



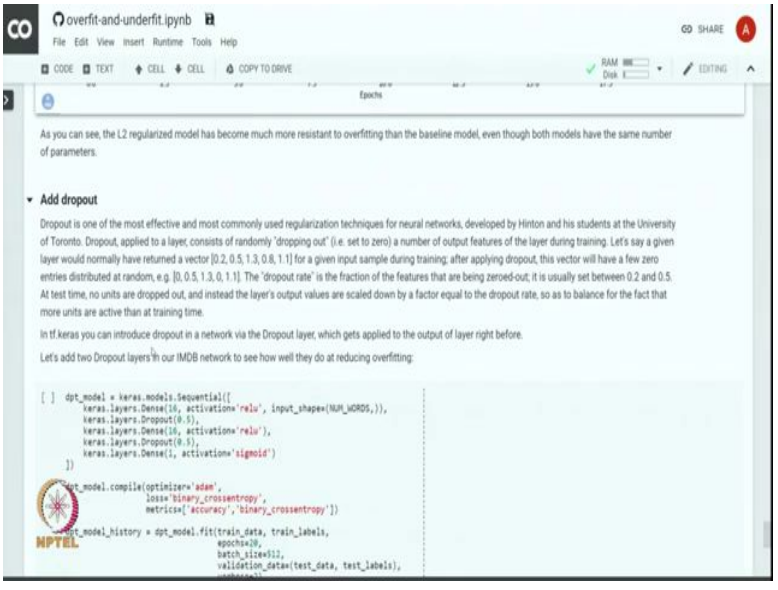
Now, let us compare how the training of regularized model compares with the baseline model.

(Refer Slide Time: 28:05)



Let us plot the loss with respect to the epoch. So, we can see that the blue model corresponds to the baseline model, whereas the orange model corresponds to the baseline model with L2 regularization. You can see that the blue line which is which is a baseline model started overfitting around third epoch, whereas the regularized model took slightly longer to start overfitting. We can see that the regularized model is resistant to overfitting for some more epochs than the unregularized model.

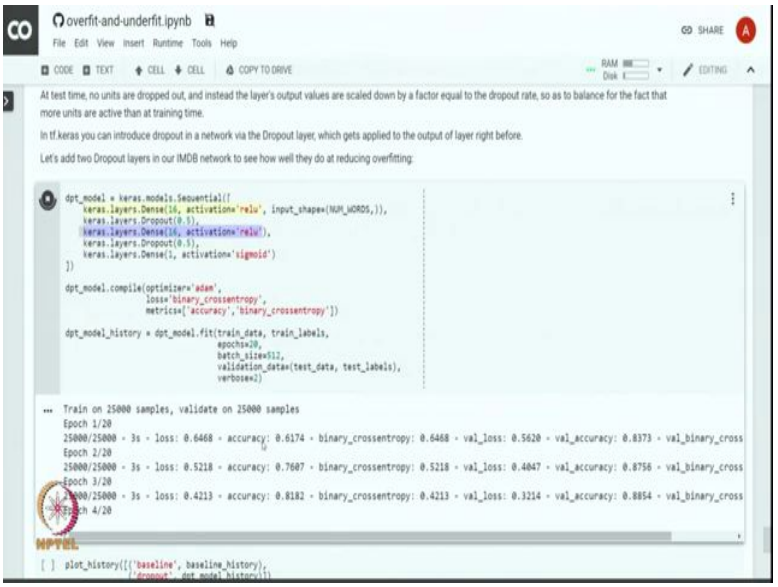
(Refer Slide Time: 28:43)



You can see the l2 regularized model has become much more resistant to overfitting than the baseline model, even though both models have the same number of parameters. The other way of adding regularization or other way of regularizing neural network is by adding dropouts. This results in randomly dropping out a number of output features in the layer during training, dropout is not applied at the test time. It is very important to note that at the test time, we do not drop any units, instead the layers output values are scaled down by a factor equal to a dropout rate.

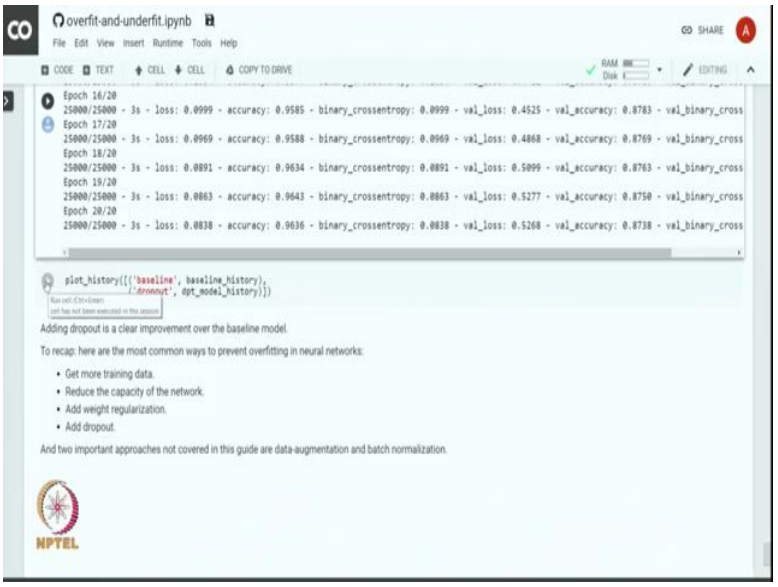
So, let us see how to use dropout in the context of neural networks, keras has a dropout layer we can use a dropout layer with dropout rate as it is parameter.

(Refer Slide Time: 29:24)



So, you can see that this particular dropout will be applied to a layer that is specified just before that. So, this dropout will be applied to the first hidden layer and then this dropout of point five will be applied to the next hidden layer and after specifying the dropout we can again retrain the model and compare its performance with the original baseline model.

(Refer Slide Time: 30:08)



So, let us compare the baseline model with the with the dropout added. So, now compare this also with the L2 regularization, we can see that after adding L2 regularization the model marginally improved its resistance to overfitting. But after applying dropout we can see that there is a substantial improvement in the resistance to the dropout and the model overfits after few more epochs than the original model.

So, to recap we what we did is we actually build a model on IMDB dataset, the model that we build a baseline model on IMDB dataset then we build a very big model on IMDB dataset that was overfitting. In order to prevent overfitting we first reduce the capacity of the model by building a smaller model and we also added regularization to it, we studied L1 and L2 regularization and then dropout regularization. So, these are the common things that are done to prevent overfitting in the neural network.

The first strategy is to get the training data if its possible. If training data more training data is not possible, we can reduce the capacity of the model or add L1 or L2 regularization or dropout. Dropout is the most commonly used regularization technique in neural network. Apart from that we can also use data augmentation to generate more training data from the available data and batch normalization to prevent overfitting. Hope you understood underfitting and overfitting through this coding exercise and you had fun learning these concepts.