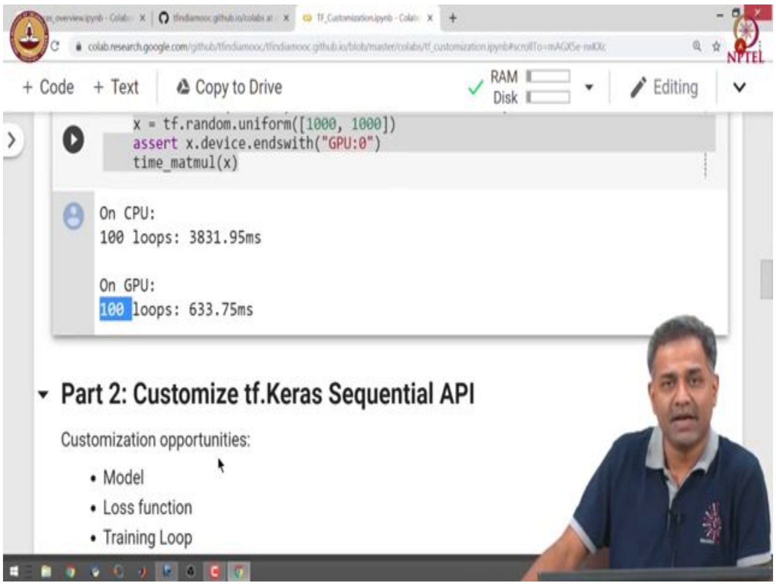
Practical Machine Learning with Tensorflow
Dr. Ashish Tendulkar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 36
Customizing tf.keras – Part 1

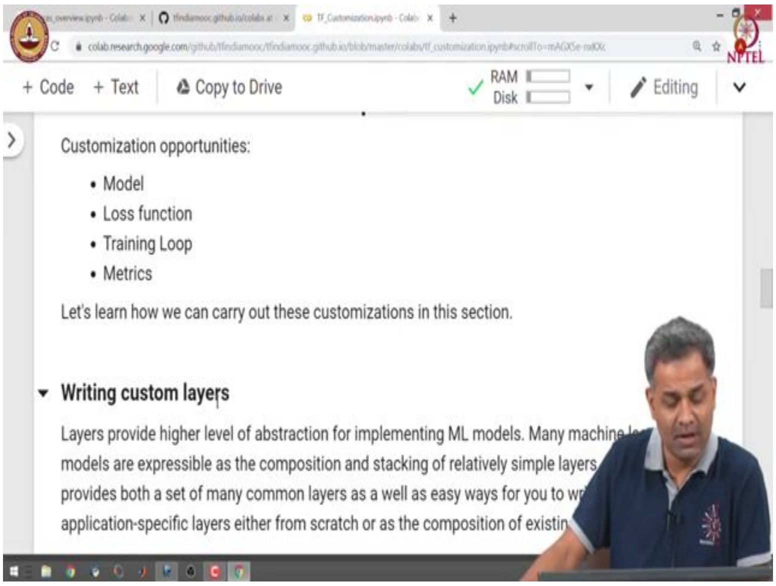
We just learnt how to perform a specific operation on the device that we intend to use.

(Refer Slide Time: 00:20)



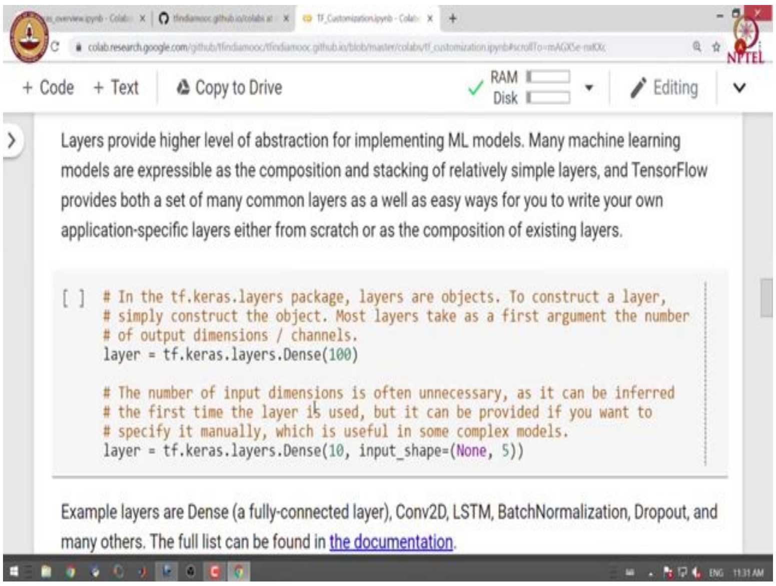
Let us move on to understand how to customize tf.keras sequential API.

(Refer Slide Time: 00:28)



There are customization opportunities in model, in writing a new loss function or training loop or using a new metric for measuring the performance of the model. Let us learn how to carry out this customization in this section.

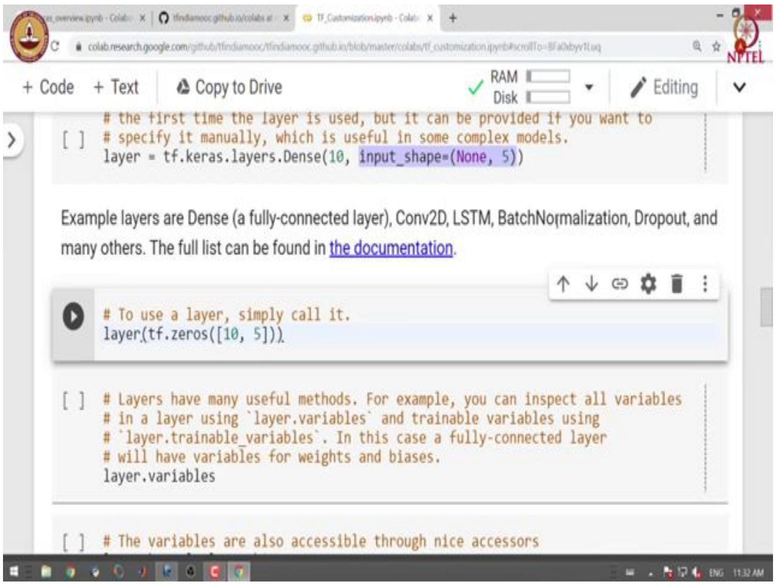
(Refer Slide Time: 00:54)



We will first study how to write custom layers. As you know layer provides high level abstraction for implementing machine learning models. Many machine learning models are expressed as composition and stacking of relatively simple layers. Tensor flow provides both the set of many common layers as well as easy ways to write your own application specific layer either from scratch or as a composition of existing layers. We have already used layers in various machine learning operations.

These are typical statement that we have seen many times in this course. So, this particular statement defines a dense layer with 100 units. We can also specify sometimes the input shape along with the number of units in the layer.

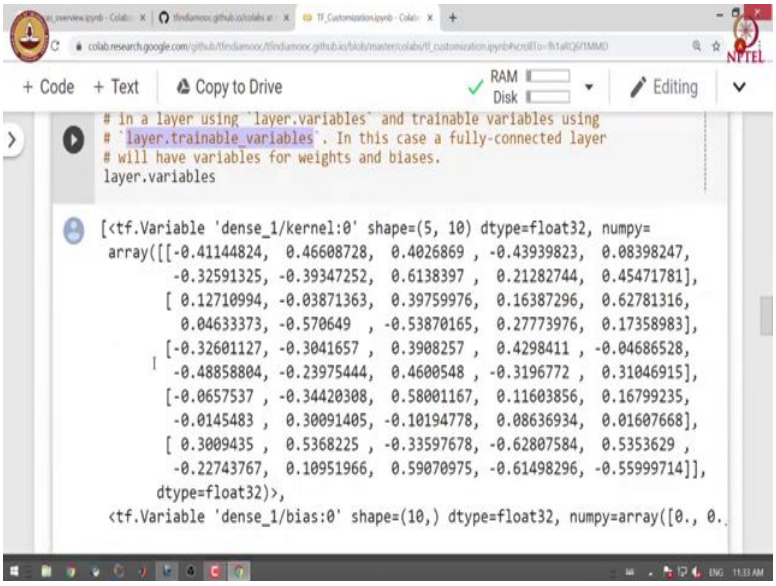
(Refer Slide Time: 02:01)



So, there are layers like dense layer, conv 2D layer, LSTM, batch, normalization, dropout and many other layers are already defined by keras. In order to use layer, we simply call layer something like this. Here we call layer with a tensor which is a matrix which is 10x5 matrix of zeros. Layers have many useful methods, we can inspect all variables in the layer using layer.variables.

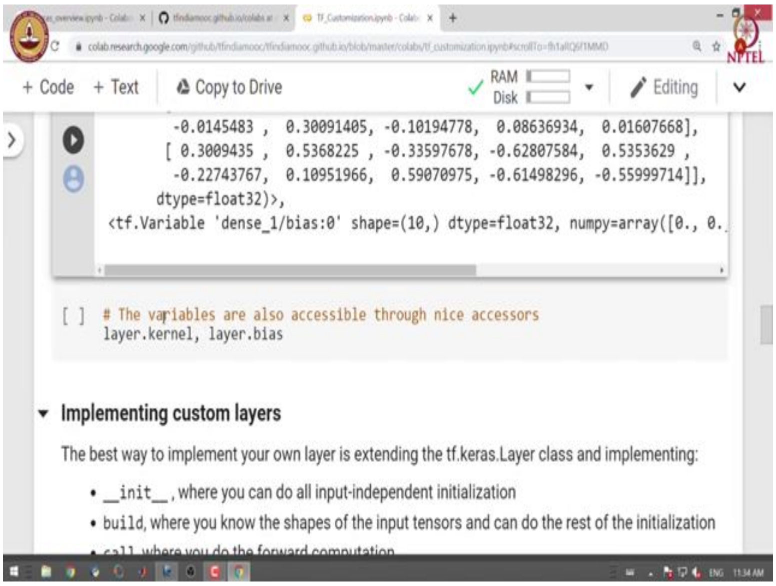
A trainable variables can be checked using layer.trainable_variable. The variables have weights and biases.

(Refer Slide Time: 03:23)



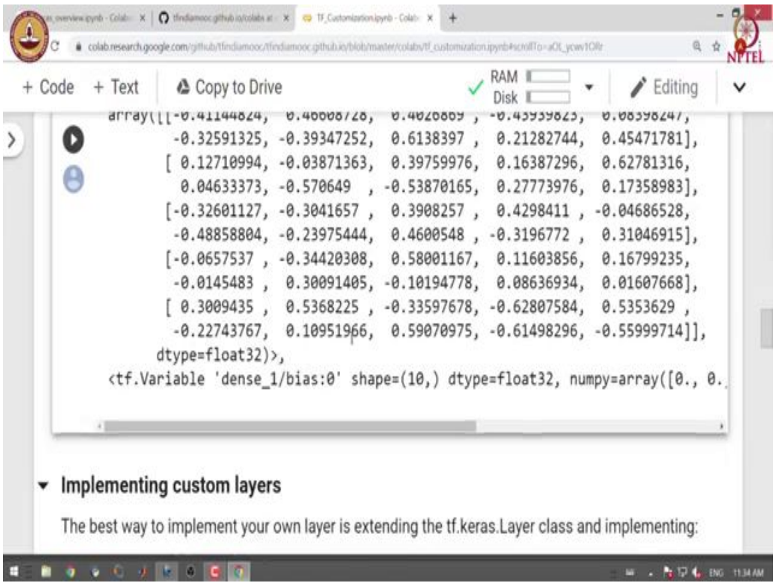
We can see that there are variables and biases. So, we have a variable tensor that shape of 5x10 and we have 10 biases.

(Refer Slide Time: 03:57)



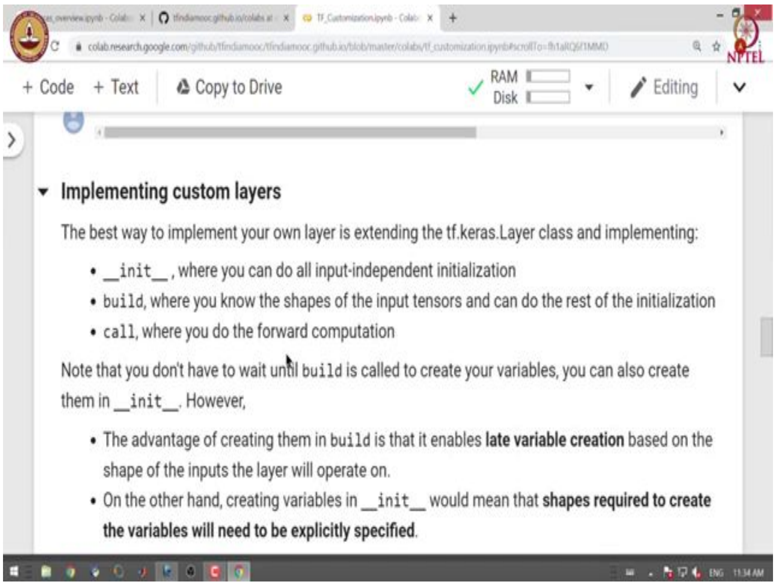
We can also access variables separately; for example, all the variables can be assessed using layer.kernel and biases can be accessed with layer.bias.

(Refer Slide Time: 04:12)



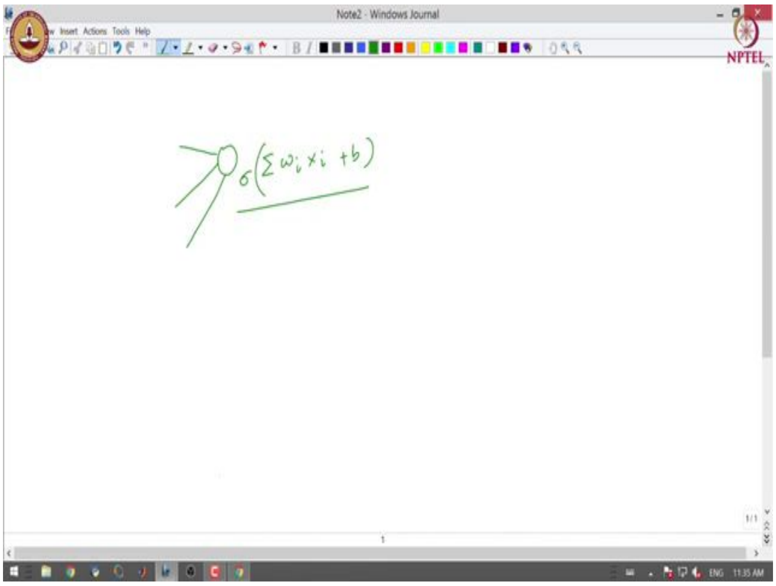
We can see that we get the same output. The difference is that when we use .variables, it gives us both kernel and bias. And we can separately ask for kernel and bias using .kernel and .bias accessors.

(Refer Slide Time: 04:38)



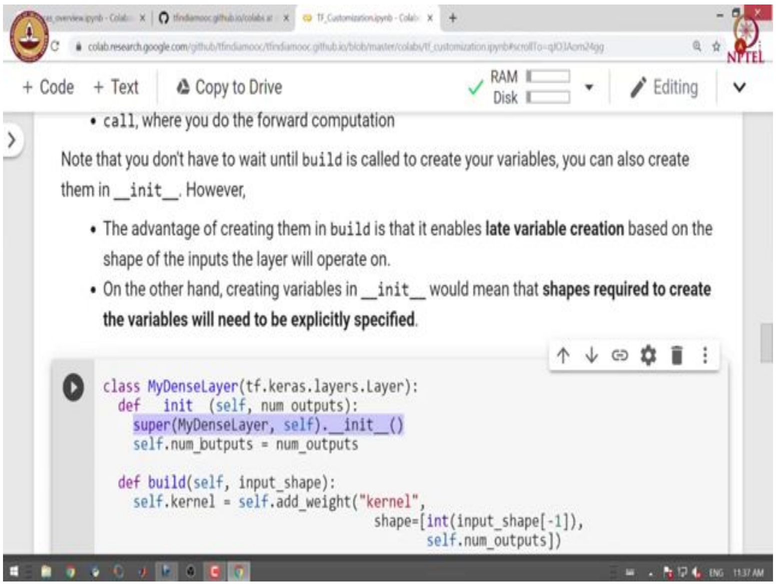
We can also implement our custom layer. In order to implement custom layer, we extend tf.keras.layers class and we implement the constructor. We implement build and a call function. The call function does the forward computations.

(Refer Slide Time: 05:19)



So, in order to give you an example, a unit in a dense layer takes input. So, essentially it does two operations; one is linear combination followed by non-linear activation. So, this is an example of a forward computation which is done in the call method.

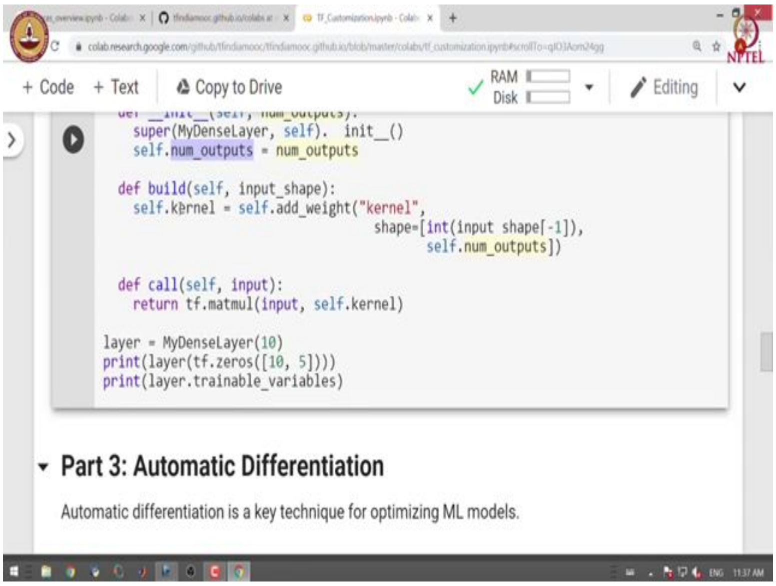
(Refer Slide Time: 05:52)



We do not have to wait until the build function is called to create variables, we can also create them in the constructor. However, the advantage of creating them in build is that it enables late variable creation based on the shape of input that layer will operate on. On the other hand creating variables in the constructor would mean that shape required to create the variable will have to be specified explicitly.

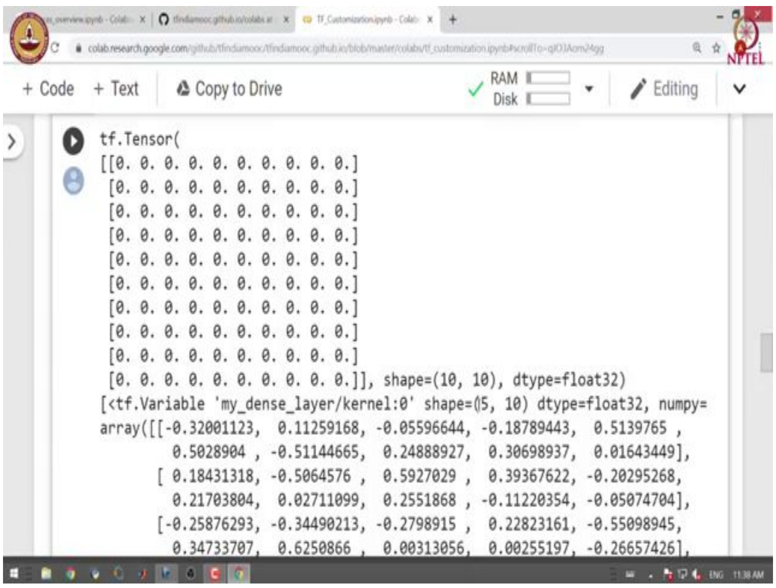
So, in this particular case we are defining MyDenseLayer which extends the layer class. And you can see that we have implemented three methods. One is the constructor; in constructor, we first call the constructor of the base class, we specified a number of outputs with a variable num_outputs.

(Refer Slide Time: 07:00)



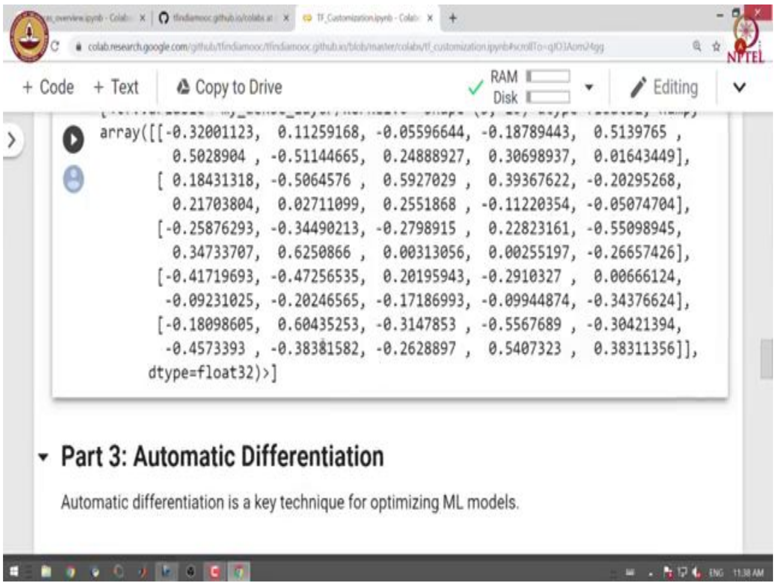
Then in the build stage, we specified the kernel. And in kernel, we have added weights in the kernel, which has the desired shape. And in the call function, we simply perform matrix multiplication of the input with the kernel or the weight vector. Here we define MyDenseLayer with 10 units; we pass the input shape of 10x5 to the layer. Let us print the layer and a trainable variables.

(Refer Slide Time: 07:53)



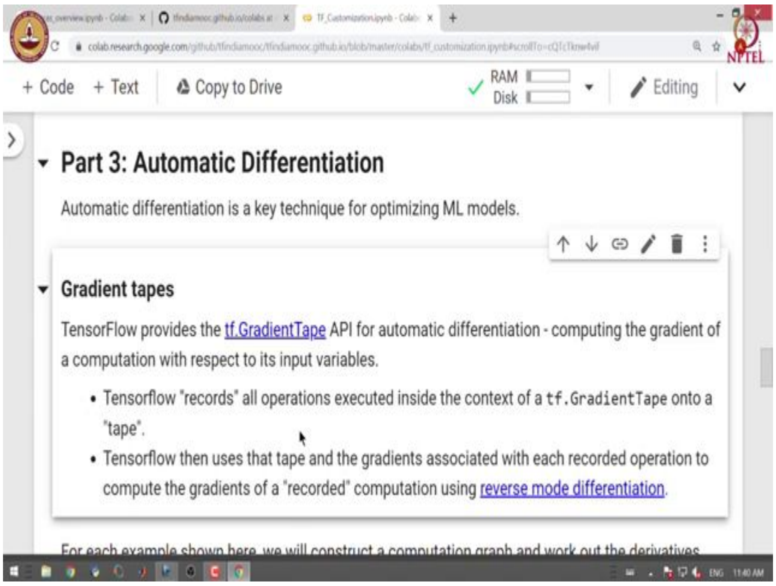
So, we can see that the layer was called with input shape of 10 cross 10.

(Refer Slide Time: 07:59)



You can see that your variable tensor by calling .trainable_variables on layer.

(Refer Slide Time: 08:29)

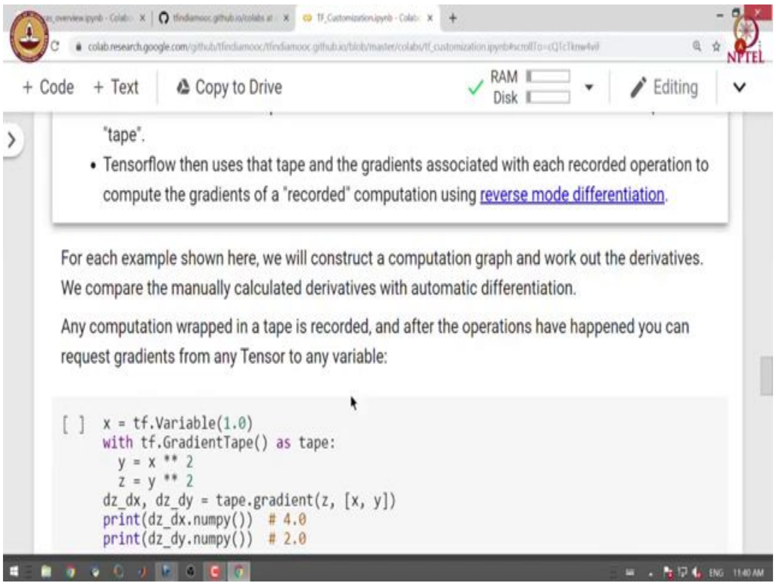


Having defined the custom layer; the next opportunity for customization lies in defining our own training loop. As you will remember from earlier classes; the main operation in training is to calculate the gradient and perform parameter update based on the gradient value.

Usually, these gradients are hand calculated, but tensorflow provides a way to automatically calculate these gradients based on the forward computation.

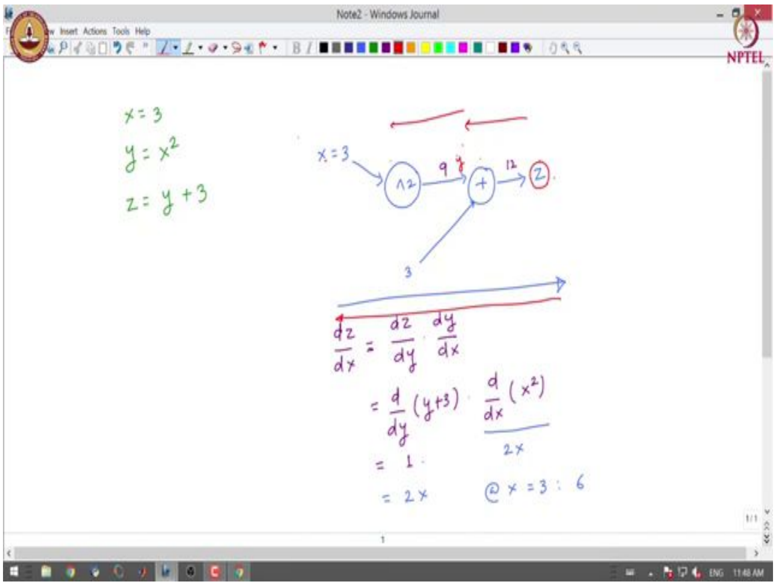
Tensorflow provides tf.GradientTape API for automatic differentiation. It computes the gradients of computation with respect to its own variable. Tensorflow records all the operations executed in the context of tf.GradientTape onto a tape.

(Refer Slide Time: 09:27)



Then it uses that tape and the gradients associated with each recorded operation to compute the gradients of a recorded computation using reverse mode differentiation. Let us take an example of how this particular thing operates.

(Refer Slide Time: 09:55)



So, let us say, we have $x=3$, $y=x^2$ and $z=y+3$. Let us use computation graph to represent the relationship between x, y and z.

The computation graph corresponding to the relationship between three variables x, y and z can be seen above. We perform forward computation, in the forward computation what we do is; we pass the values of the variable through this particular graph to obtain the value of z. It is concretely in this graph we set x to 3, we raise the power of the value of x by 2, we get the output of 9 over here. We add this 9 and 3 to get 12 at z.

Now, in machine learning we are interested in calculating the derivative of the loss that we normally see in the final step of neural network with respect to each of the input variables. So, we are modeling that particular situation with this toy example. In the context of this example we are interested in finding derivative of z with respect to x.

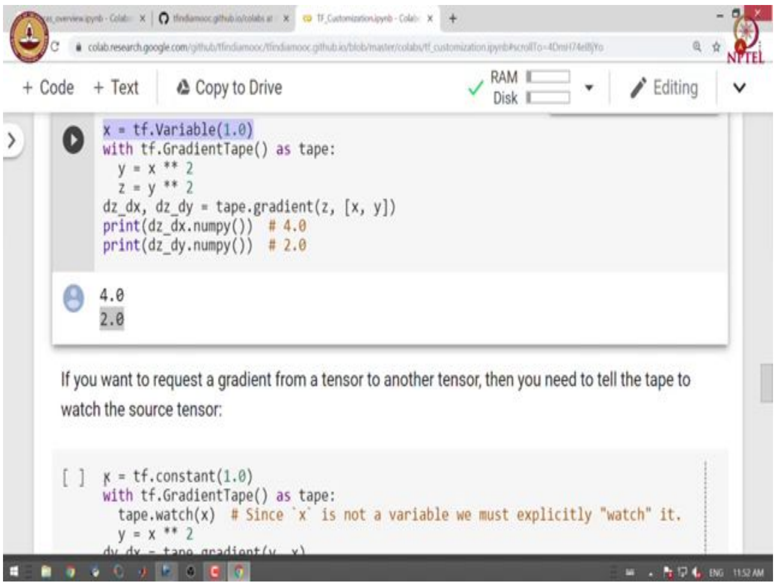
And in order to calculate this particular derivative, we use the chain rule of derivative; where we will take derivative of z with respect to y, which is known and derivative of y with respect to x. And with this chain rule of calculus, we calculate the derivative of z with respect to x. So, derivative of z with respect to x is calculated as the derivative of z with respect to y and derivative of y with respect to x.

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

So, we can now, we can see that $\frac{dz}{dy}$ is 1, because the derivative of y with respect to y is 1 and derivative of constant with respect to y is 0. From $\frac{dy}{dx}$ we get 2x. So, x equal to 3, this value this value is 6. So, this derivative computation is done in a reverse mode differentiation where, we start this variable we calculate the derivative of this with respect to y. And then we calculate the derivative of y with respect to x to obtain derivative of z with respect to x.

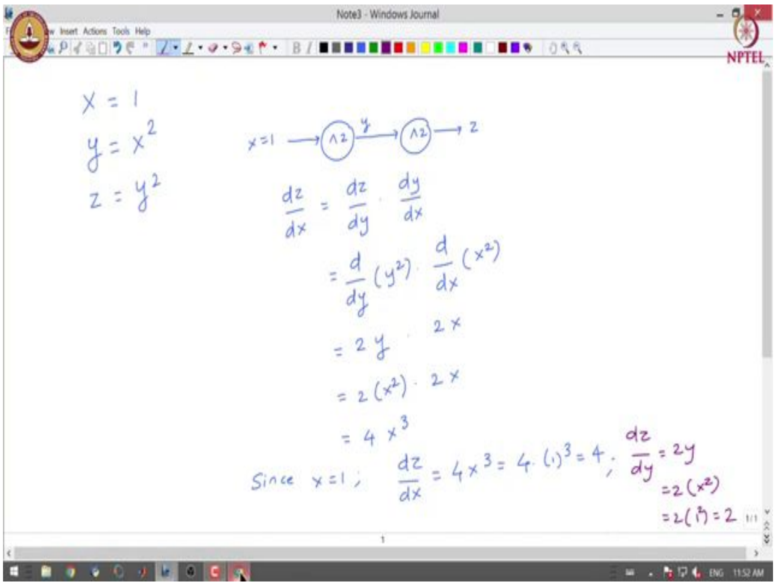
So, in the backward pass, we use reverse mode differentiation to calculate the derivative of z with respect to x. So, gradient tape is used to record the forward operations and then it uses the gradients associated with each recorded operations to compute the gradient of a recorded computation.

(Refer Slide Time: 15:28)



Let us take a complete example; we define x to be a variable and in the context of gradient tape we perform two operations. First we raise the power of x by 2 and then we get y as a result and again we raise the power of y to 2 and we get z. So, here we have the following situation.

(Refer Slide Time: 15:57)



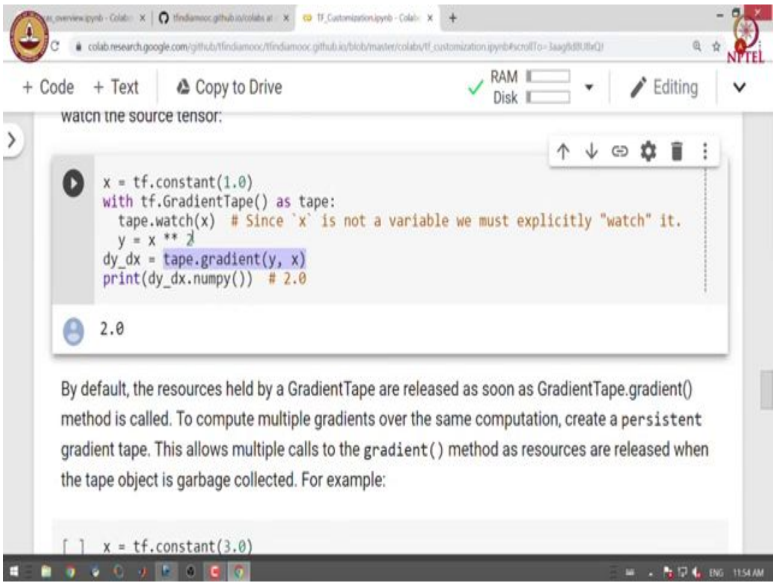
So, $x=1$, $y=x^2$ and $z=y^2$. So, we started with x equal to 1, we raise the power to 2, we get y we again apply the power operator to get the value of z is where we get y. So, this is the forward computation.

And we can calculate the derivative of the recorded computation with respect to the input variable with reverse mode differentiation. So, in this case $\frac{dz}{dx}$ is equal to $\frac{dz}{dy}$ into $\frac{dy}{dx}$, by the chain rule. So, $\frac{dz}{dy}$ is 2y and $\frac{dy}{dy}$ is 2x. So, y as we know is x^2. So, this derivative is $4x^3$.

So, you can see that, since x is equal to 1, we get $\frac{dz}{dx}$ to be 4 and $\frac{dz}{dy}$ to be 2. So, since x is equal to 1 $\frac{dz}{dx}$ is equal to $4x^3 = 4$. And $\frac{dz}{dy}$ is equal to $2y = 2x^2$ which is 2 into 1 squared is equal to 2. So, $\frac{dz}{dx}$ is 4 and $\frac{dz}{dy}$ is 2.

Let us run this to verify. We can see that $\frac{dz}{dx}$ is 4 and $\frac{dz}{dy}$ is 2. So, note that here x was defined as a variable. We can also request a gradient from a tensor to another tensor.

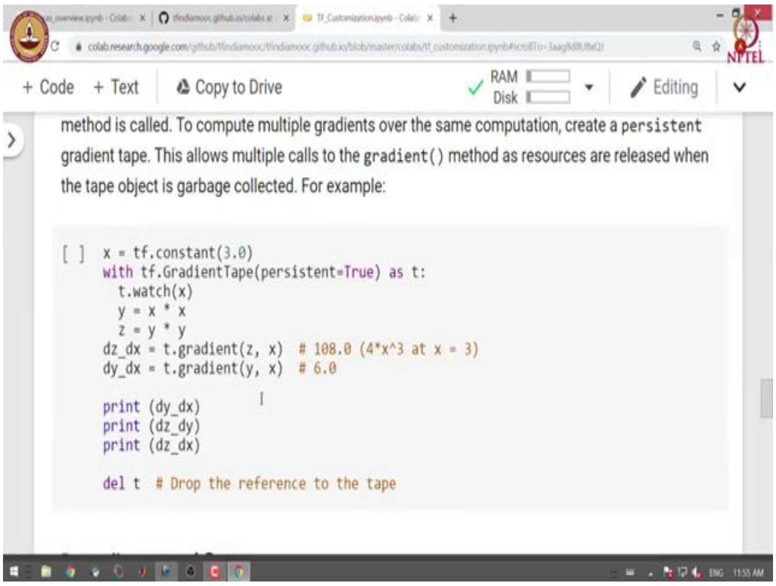
(Refer Slide Time: 19:12)



Let us look at the example. Here x is a tensor and we raise the power of x by 2 to get y. And now we are interested in calculating derivative of y with respect to x. Note that x is a tensor and y is also a tensor.

Now, since you are interested in calculating derivative of y with respect to x, which is a tensor, we add tape.watch and add tensor x to the watch list. By doing this we are able to calculate the derivative of y which respect to x with the gradient tape. And we perform both this operation in the context of gradient tape and then calculate the gradient using tape.gradient method. Let us run it and check it out. And you know that y is x square, so derivative $\frac{dy}{dx}$ will be 2x and x is equal to 1 that is why the value that we see here is 2.

(Refer Slide Time: 20:31)



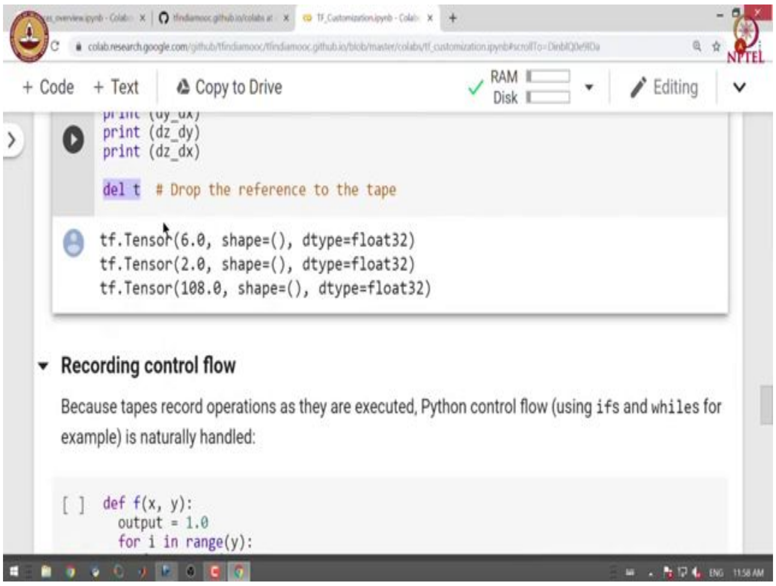
So, by default the resources held by gradient tape are released as soon as the gradient tape.gradient method is called. To compute multiple gradients over the same computation, we create a persistent gradient tape. We create a persistent gradient tape, this allows multiple calls to the gradient method as resources are released when the tape object is garbage collected. Let us look at the example of using a persistent tape.

So, we simply add persistent equal to true to gradient tape method and then perform the forward computation in the context of this persistent gradient tape. Since, we are interested in calculating derivative of z and y with respect to x, we first add the tensor x to the watch list and then perform the remaining forward operations. So, here we obtained y by multiplying x to itself or in other words, by squaring the value of x and z is obtained by squaring the value of y.

Let us look at what is the gradient at x is equal to 3. So, we know that the gradient of z with respect to x is $4x^3$. So, that is why at value of $x=3$, $x^3=27$, and 27 time 4 gets us 108 as the value of gradient.

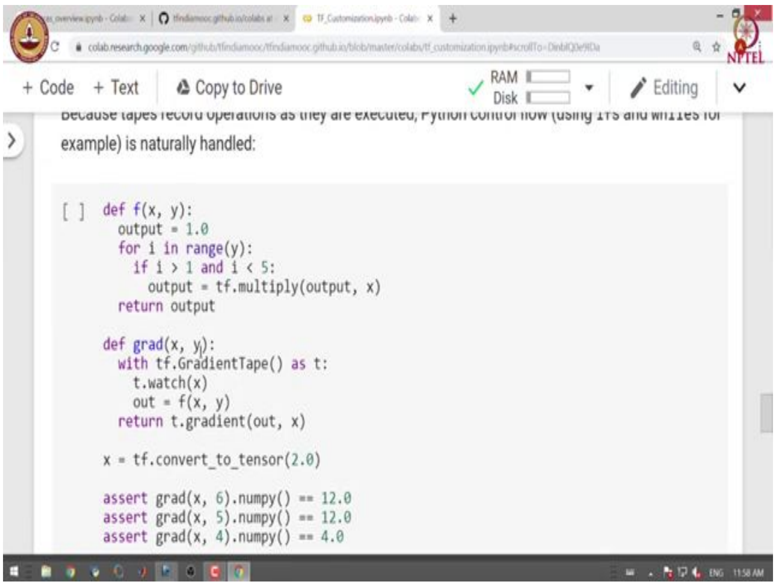
We know that the gradient of y with respect to x is essentially 2x that is why at x equal to 3, we get the value of 6. Let us print all the three gradients, which is the derivative of y with respect to x, derivative of z with respect to y and derivative of z with respect to x. And finally, we delete the reference to the tape using del command.

(Refer Slide Time: 23:28)



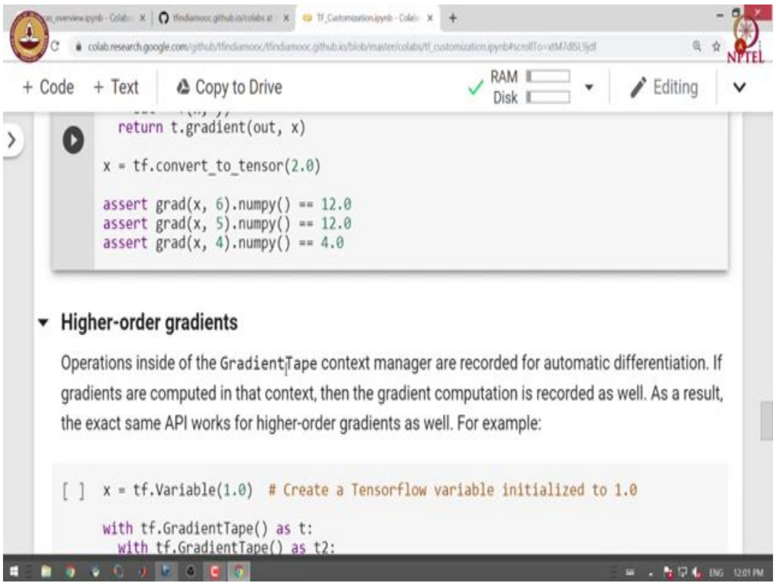
So, we obtained results as per our expectations.

(Refer Slide Time: 23:35)



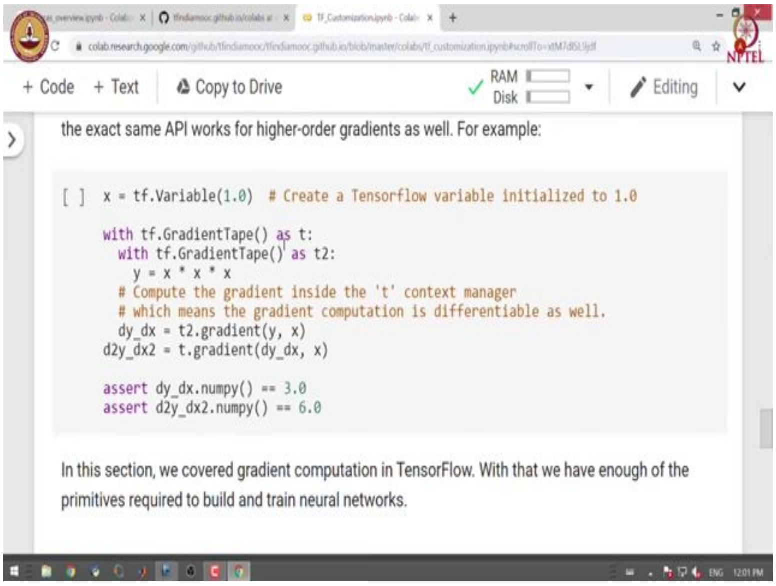
We can also record forward operations even in the presence of the python control statements or loops. So, this is an example where we first define a tensor containing value 2. And then we define a function called grad; the grad function essentially defines a couple of forward computations in the context of gradient tape and the computation that we carry out is defined in function f. f runs the computation in a loop that repeats y times and if the value of i is greater than 1 and less than 5. It performs the multiplication operation. It multiplies the output with x.

(Refer Slide Time: 25:12)



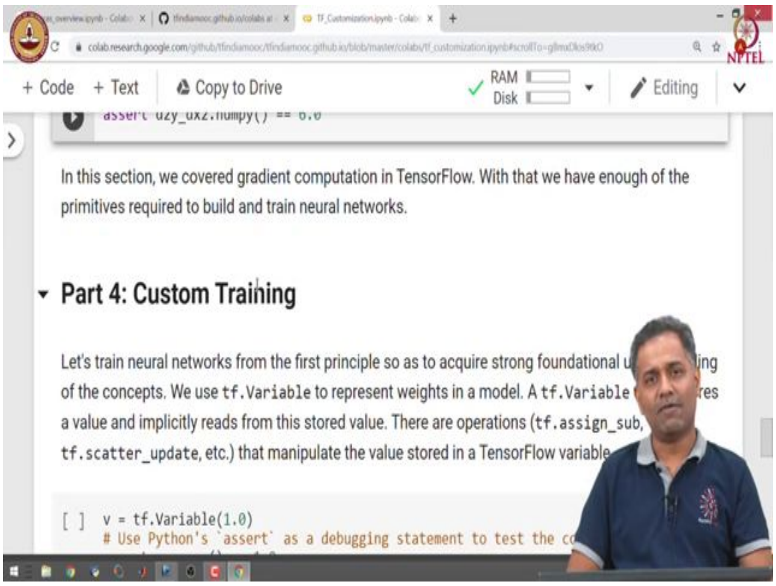
The gradient tape can also be used in the presence of loops or control statements. We can also use operations inside gradient tape to calculate higher order gradients.

(Refer Slide Time: 25:24)



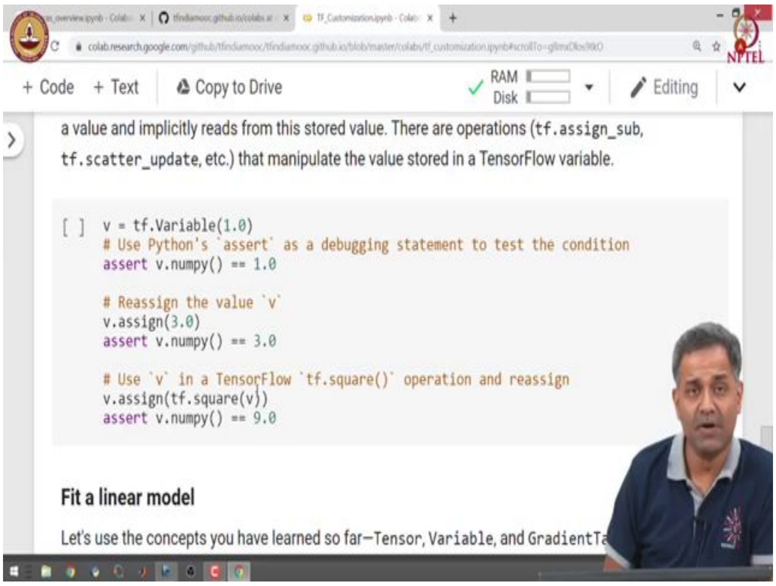
In this case, we are defining a nested gradient tape. So, there is an outer gradient tape in the context of which there is another gradient tape and in the context of both these gradient tapes; we are performing x^3 operation to obtain the value of y. In the context of inner gradient tape, we can get derivative of y with respect to x. And in the context of outer gradient tape, we can calculate the second derivative of y with respect to x.

(Refer Slide Time: 26:08)



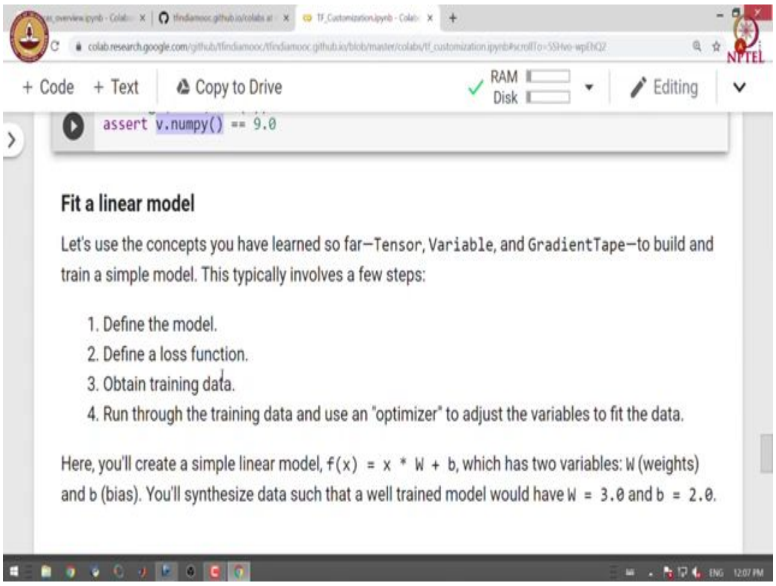
So far, we studied how to use specific devices to carry out tensorflow operations. Then we studied how to write custom layers and we also studied how to perform automatic differentiation to obtain gradients of loss functions with respect to input variables through automatic differentiation. With these three concepts, we have some tools available to us for writing custom machine learning algorithms. Let us use the concepts that we learnt in automatic differentiation to write our custom training loop.

(Refer Slide Time: 27:06)



Here, we will train tensorflow model from first principles. So, we use variables for storing weights of tensorflow model and then we use functions like assign to assign values to the variable. In this case, we are assigning the value of 3 to variable v and here we are assigning the value of square of v to v itself. So, let us run this particular code, we can see that the value of v was initialized to 1 here, then we assigned a value of 3 and here we assigned a value of 9 to v. And note that we are using v.numpy function to obtain the value present in the tensor.

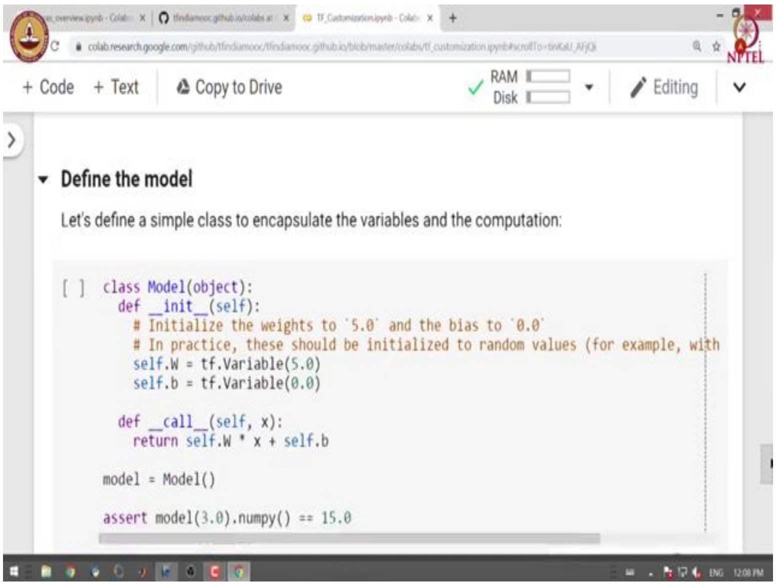
(Refer Slide Time: 28:06)



What we will do is, we will define a linear model using the concepts that we have learnt so far.

So, there are four different steps; we have to define the mode,l then loss function, then obtain the training data and train the model. And how do you train the model? We use a specific optimization algorithm for training the model. So, here in linear regression, we have a model which is a linear combination of weights and the input along with a bias term to it. So, if x into W plus b it has got two variables - W and bias.

(Refer Slide Time: 29:19)

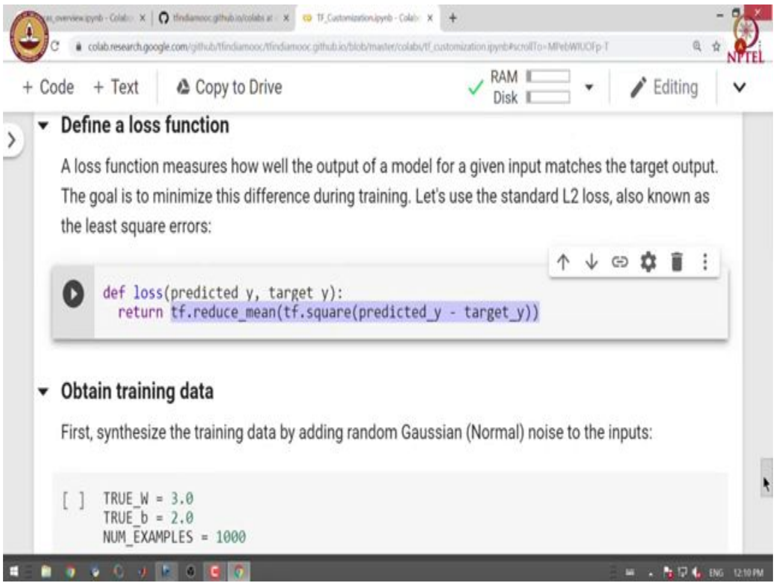


So, what we will do is; we will first obtain synthetic data with W=2 and b=2. We will define a simple class to encapsulate the variables in the computations.

So, in the constructor, we define variables W and b and we have set them to 5 and 0, in real life or in practice, we randomly initialize these values. But for the sake of simplicity in this example, we have set this variables to some fixed numbers. And in the call method we are performing the forward computation where we are multiplying the input by the weights and adding the bias term to it.

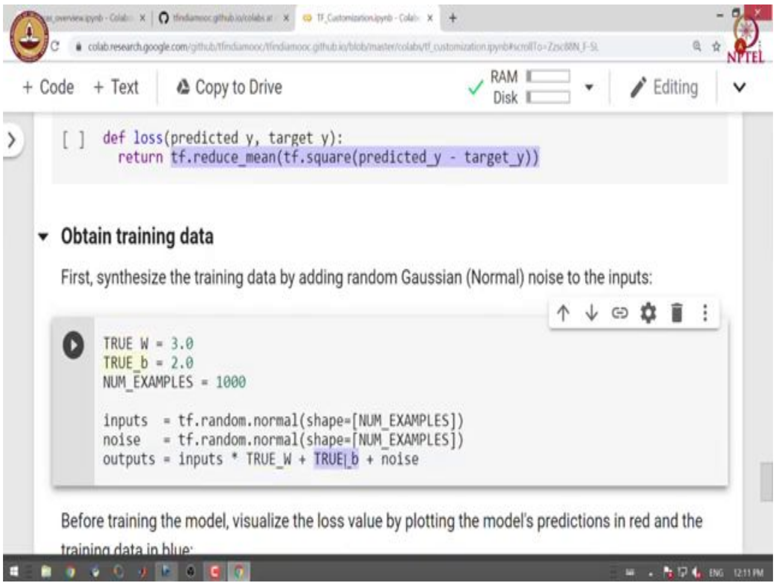
So, we have model over here. So, for value 3 we get value for input 3, we get output of 15. We can see it above, the value of W is 5, x is 3. So, 5 into 3 is 15 and bias is 0. So, 15 plus 0 is 15, that is how we get the value of 15 when we pass 3 as the input to the model.

(Refer Slide Time: 31:01)



The next task is to define a loss function. Here we use standard L2 loss or a least square error. And the way we define the least square error is we calculate the square of the difference between the predicted value and the actual value. And we sum up this difference across all the points in the training data. And this is how we define our loss function.

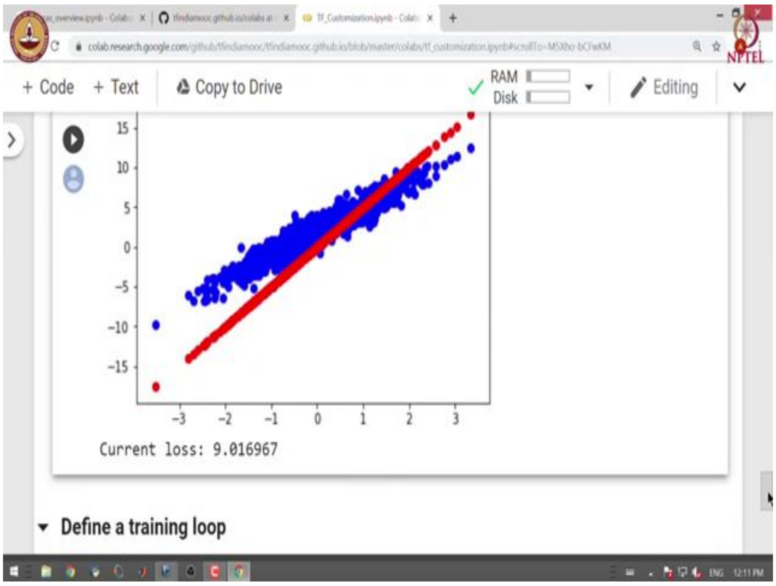
(Refer Slide Time: 31:35)



The next step is to obtain the training data; here we synthetically generate our training data by setting W to 3 and b to 2 and we generate 1000 examples by adding some noise to the regression calculation. So, here we first define input, which is drawn from a normal distribution, and then we have a noise which is again drawn randomly from a normal distribution. And we obtain output by multiplying input by the true weight and add true bias to it, that gives us output.

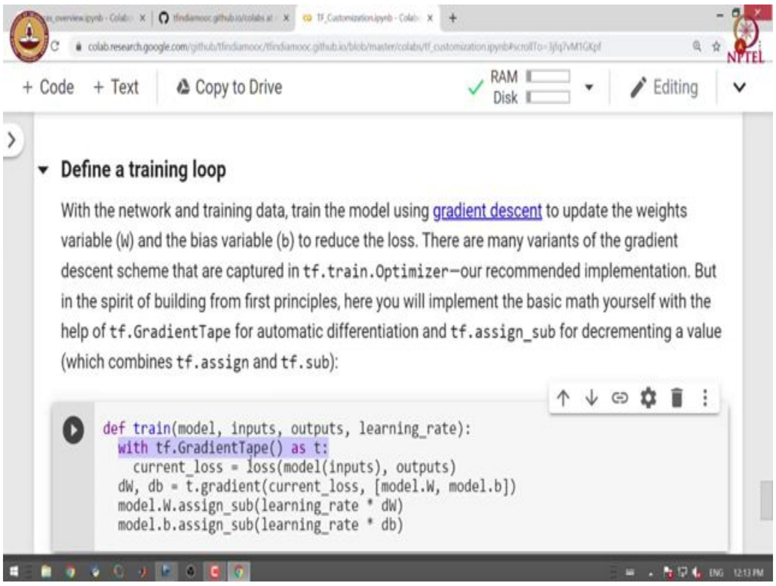
So, let us run these steps, let us define the model; let us define the loss, let us generate the training data and let us visualize the training data before building the model.

(Refer Slide Time: 32:54)



So, you can see that for the chosen parameter values, we have loss of 9.01 and you can see that, the points in the training that we generated are in blue whereas the red line or the points represented with red lines are the predicted points. Having obtained the training data, the next step is to train the model itself.

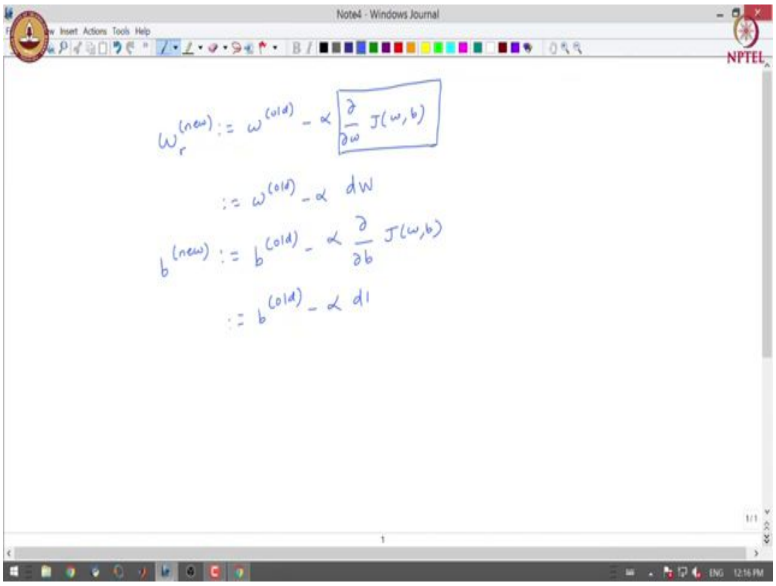
(Refer Slide Time: 33:38)



Let us define our custom training rule. We are going to use gradient descent to update the weights. Normally in real life applications or examples that we have seen so far, we use one of the optimizers from tf.train.optimizer. Gradient descent itself is available in the standard keras package, but here we want to train the model from first principles.

So, we will be using gradient tape to calculate the derivative of the loss function with respect to the input variables, let us see how do we do that. So, we calculate the loss in the context of a gradient tape. So, since we calculate loss in the context of the gradient tape, it records all the operation in the forward computation and when we call a gradient method on the tape, it gives us the gradients. For example, here we have gradients of loss with respect to W and b. Having calculated gradient you must remember to update value of W.

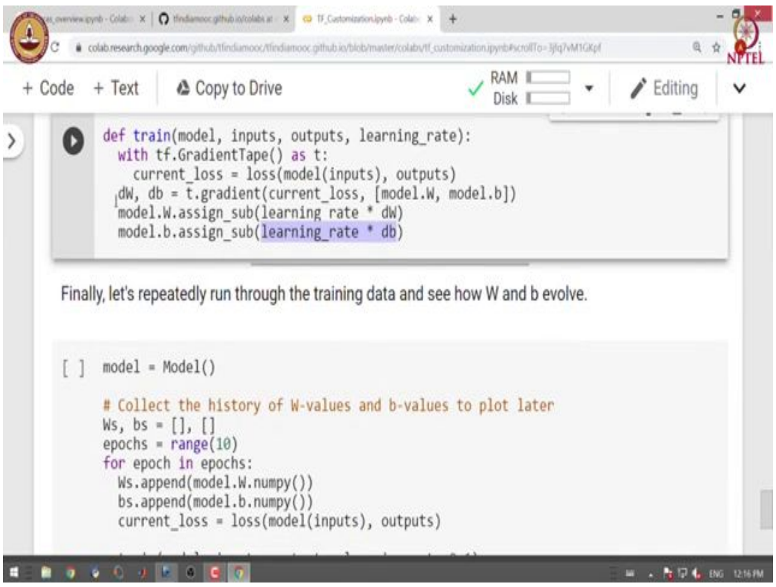
(Refer Slide Time: 35:02)



W$^{(new)}$ is said to W$^{(old)}$ minus learning rate times the gradient of the loss function with respect to the variables. So, the gradient part, you get from the gradient tape. We call the gradient method on the tape to obtain it. We do the same for b.

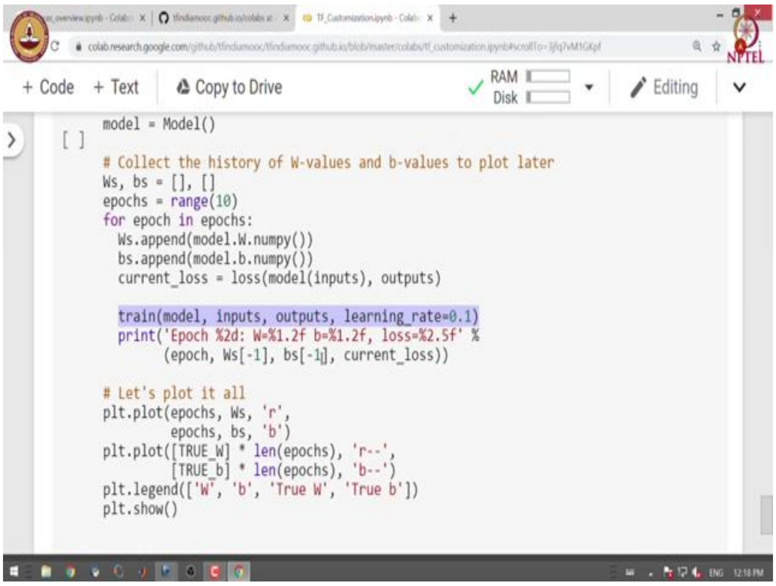
So, we assign W$^{(new)}$ by subtracting from W, the learning rate into the gradient, and we obtain b$^{(new)}$ by subtracting from b, the learning rate into the gradient with respect to b.

(Refer Slide Time: 37:03)



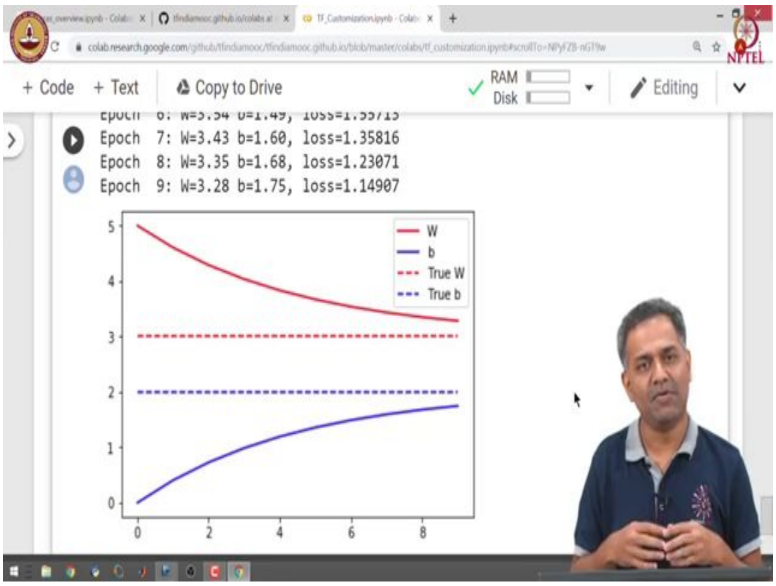
This is how we define a training loop, this is how we define the basic training operation; we have to run this function repeatedly.

(Refer Slide Time: 37:15)



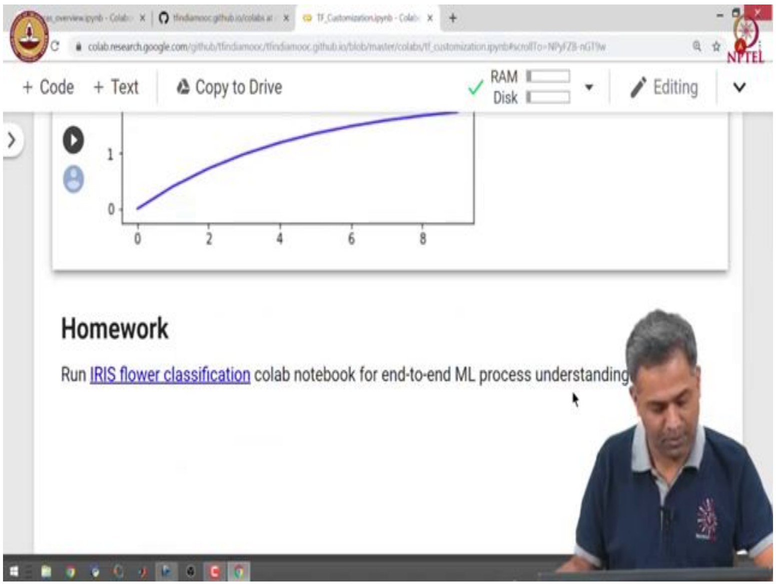
We have output the value of W, b and value of current loss. Finally, we also plot how W and b changed and also plot the true value of W and b. Let us run this.

(Refer Slide Time: 37:45)



We started W at 5 and b at 0 and as we get to 10 epochs, both W and b are getting closer to the actual value of W and b. Here we define our own model, we also defined our own training loop and then we also implemented the gradient calculation using gradient tape.

(Refer Slide Time: 38:23)



As a next step I would strongly encourage you to go through the iris flower classification notebook for end to end ML process. So, in this session, we learn a number of concepts that will help us in customizing the tensorflow functionality. We started with how to force operations on accelerated devices, how to write custom layers, how to perform automatic differentiation using gradient tapes, and we use the concepts learnt in automatic differentiation to define our own custom training.