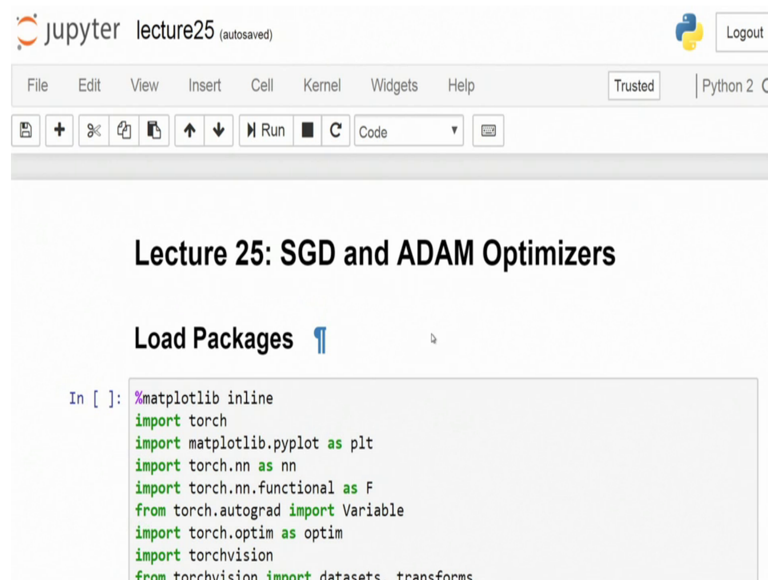
Deep Learning for Visual Computing
Prof. Debdoot Sheet
Department of Electrical Engineering
Indian Institute of Technology, Kharagpur

Lecture - 25
SGD and ADAM Learning Rules

Welcome back to today's lecture and this one which we are going to do is basically on using different kinds of optimizers.

(Refer Slide Time: 00:18)



So, as you have learnt on your update rules over there. So, one of the ways of doing it was using something called as a gradient descent or which is like for within every single epoch you find out what is your error then find out a gradient of the error and then use that gradient of the error multiplied by something called as learning rate and that error over there is what is used as an update over the waits itself. So, that that was the most simple part over there.

Now, along with that while there are advantages disadvantage. It is pretty simple to derive it out and was one of the first ways of doing any sort of an learning rule which was defined. Now, these whole problems as such are also within gambit of engineering solution which is called as optimization. Now, the reason why this is called as optimization is from the fact that you know that you are never able to go down to a 0

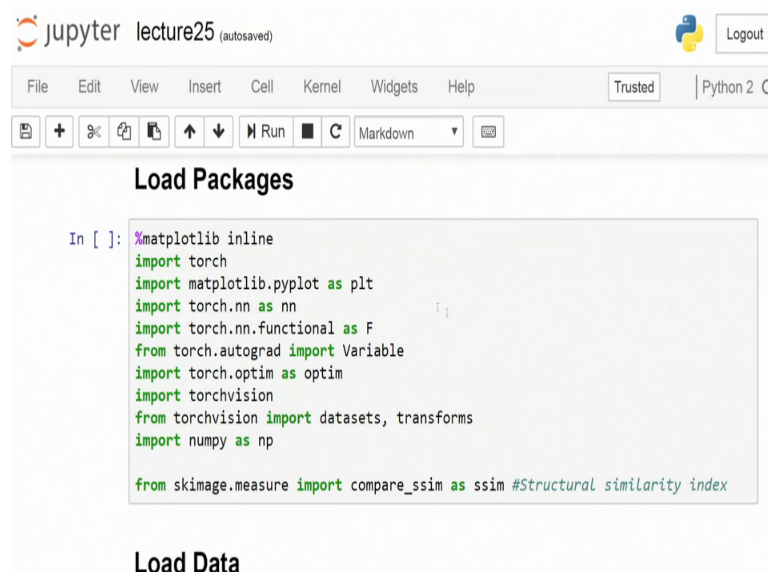
error technically you will most of the cases you will never be leaking down a 0 error, but you will be coming down to a very low error on the error surface over there.

Now, the whole purpose of optimization is like to find out and it is a agreeable solution of that minimum error point where it is good for you to go down. Now, as it turns out, gradient descent is not the only way of coming down over there, there are in fact, multiple different ways of that. So, one of them is called as stochastic gradient descent operator the other one is an learning rule using something called as an adaptive momentum or ADAM optimization technique.

So, today I am just going to while in the earlier lecture we have covered down on the theoretical aspects. So, this is about how do you actually create your whole network to use any of these optimizers and we will come down to using something generally within an optimizer package. And instead of trying to write down your vanilla gradient descent as an update rule every single time within an epoch here is where you will be using something called as an optimizer dot step as we have used in some earlier examples as well. So, here is just to show you what is the effect which happens down with two different optimizers.

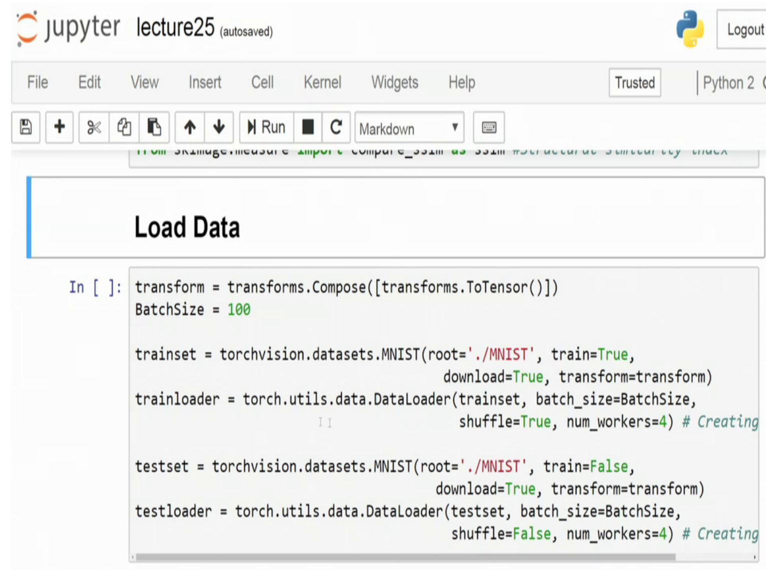
Now, we will stick down to just using stochastic gradient descent and adaptive momentum or ADAM optimizers.

(Refer Slide Time: 02:30)



So, as in for whatever we have been doing with the earlier classes as well the initial header is just a loader of all the important functions which are needed over there.

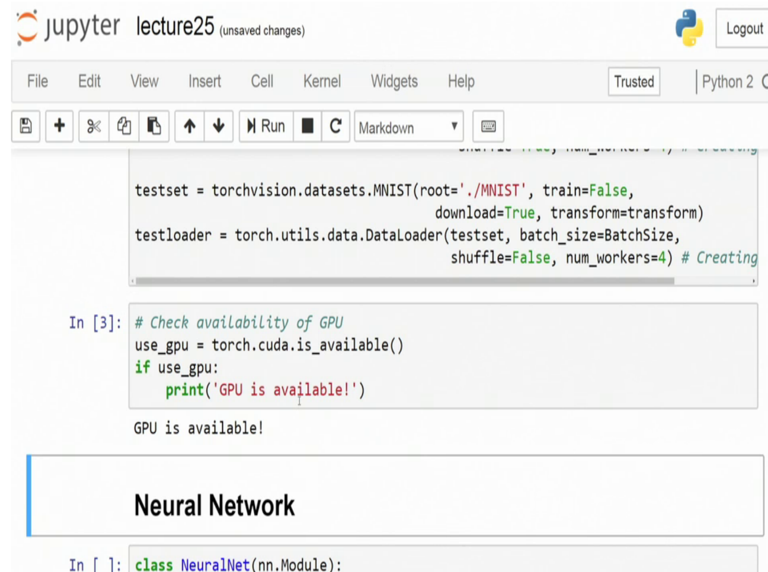
(Refer Slide Time: 02:42)



We start with the doing our data loader. So, the data which we are using is MNIST with a batch size of 100. So, it every single batch it is going to do 100 samples loaded from the hard drive and then use it for the whole purpose over there. Now, we would stick down to the best possible approach which we found out in the earlier case on different kind of update system. So, one was where you had done one update at the end of an epoch, the other one was there was an update every batch and the remaining was where there was an update for every single sample which goes into the network and every epoch.

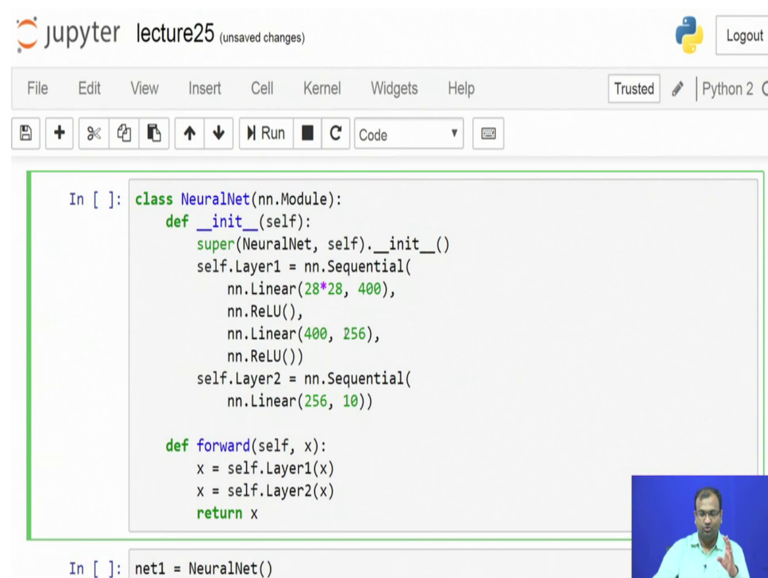
Now, while you did find out that in the first case it was taking the least amount of thing when you are updating it once within an epoch in the second case it was taking almost the same amount of time, but you are updating it multiple times within an epoch and as a result of this multiple times of revision within an epoch it was going to a much higher performance. So, your error was steadily much below then what you had in the earlier case and your accuracy is also high. So, we are going to stick down to this best possible approach which takes the least amount of time comes down to the best accuracy at the fastest rate and that is batch update.

(Refer Slide Time: 03:53)



So, we are just stick down to that have our batch training over here ready and then yes check out if for GPU is available and that is there and so we defined our network.

(Refer Slide Time: 03:58)

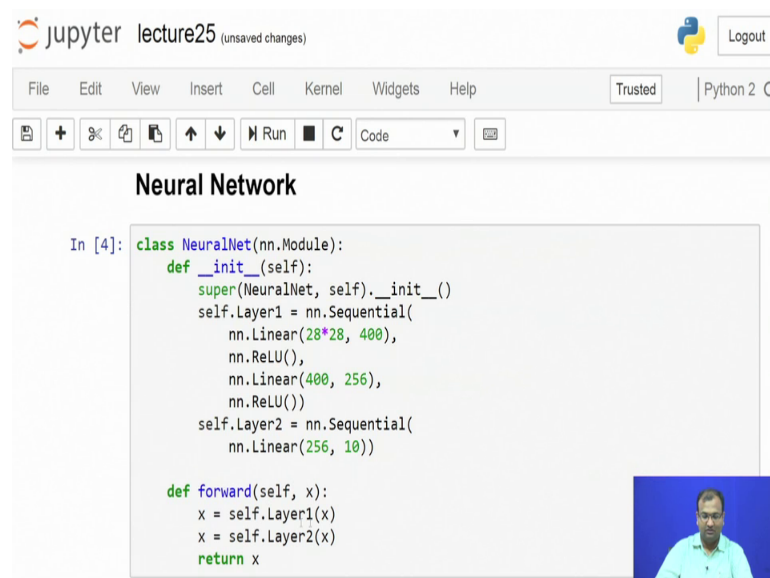


So, the network is a plain simple in neural network for this classification problems it takes in a MNIST of the size of a 28 cross 28 or 784 neurons which are connected down to 400 neurons in the first hidden layer that is connected down to 256 neurons in the second hidden layer then the immediate transfer functions for each of them is a rectified linear unit or a ReLU block which means that any activation which is negative is

clamped on to 0 level any activation which comes out of this is if the input to this ReLU layer is positive then it just preserves over there. So, it creates a positive only side of an output in a linear way and negatives are all 0. So, that that was ReLU.

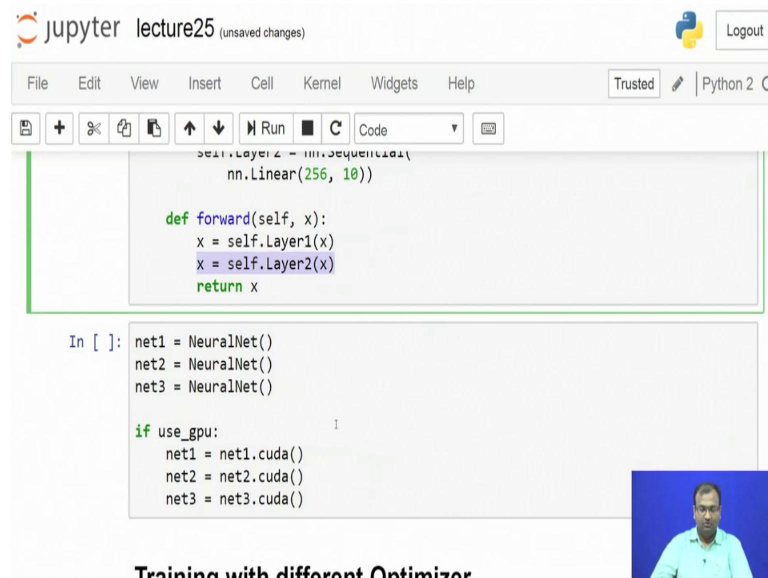
And then on the next part of the layer which is my, so while the previous part is what you have been defining as the feature discovery layer the next part of the layer is what we define as the classification part of the layer and this is where you take down from 256 neurons in your hidden layer and connect down to 10 neurons on your output side and they correspond to 10 classes in which your classification is going down. And finally, there is this forward pass of the definition or how what happens when you have the data given down.

(Refer Slide Time: 05:09)



So, first it dos does a forward pass over the layer one which is the representation learning layer. The second time it does a forward pass over the layer two which is my classification mapping layer.

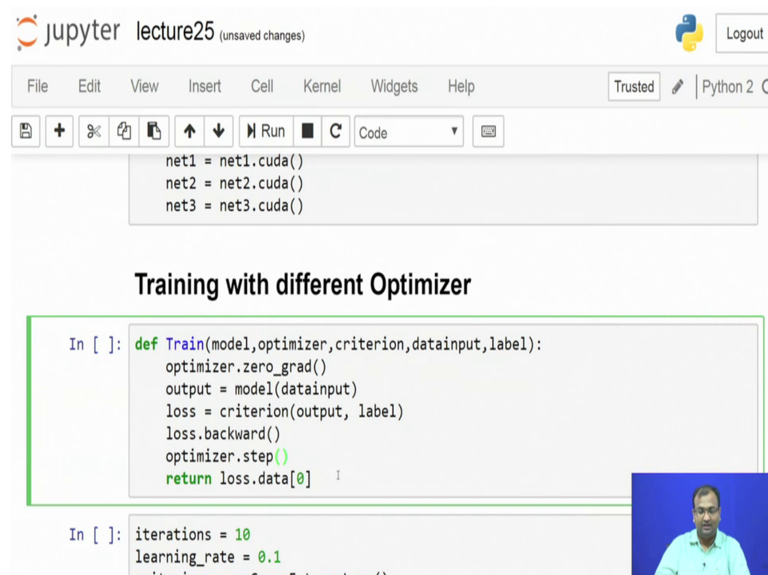
(Refer Slide Time: 05:20)



Now, what I choose to do over here is basically introduce three different kinds of networks. Now, one you have you have done this in the earlier case when we were doing down multiple kinds of cost functions in its way, but then the question over here would be we are just using to optimize and why do 3.

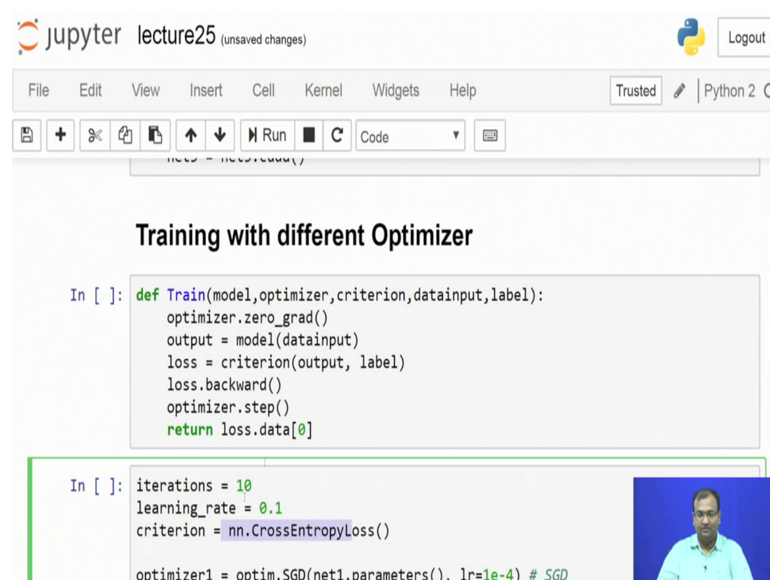
Now, when you go down with stochastic gradient descent you actually have two different kinds of SGDs which will be dealing with and for that we just try to introduce it in two different variants and that is where three different networks are defined over here.

(Refer Slide Time: 05:53)



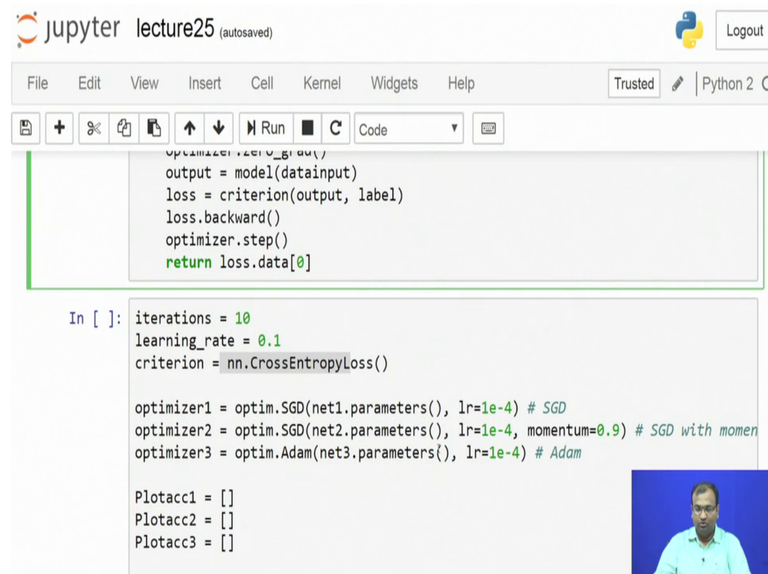
Now, going down with the same kind of an argument as we had in the earlier case when we were trying to use different kind of cost functions. So, over there the whole point was that this criterion was changing at everything. So, if it was a regression problem then you had two different regression criterion points either as an MSE loss or L 1 norm loss if it was a classification then we had done it with the cross entropy loss with a negative log likelihood loss and with a multi margin loss criteria. So, here we are just going to stick down to one of the classification losses which for us is going to be cross entropy. Just one of them.

(Refer Slide Time: 06:26)



The only thing which we are going to vary over here is this optimization technique over there. While in the earlier cases were just sitting down to plain vanila gradient descent over here we are going to have down to variance of SGD and one of the variants of ADAM taken off.

(Refer Slide Time: 06:42)



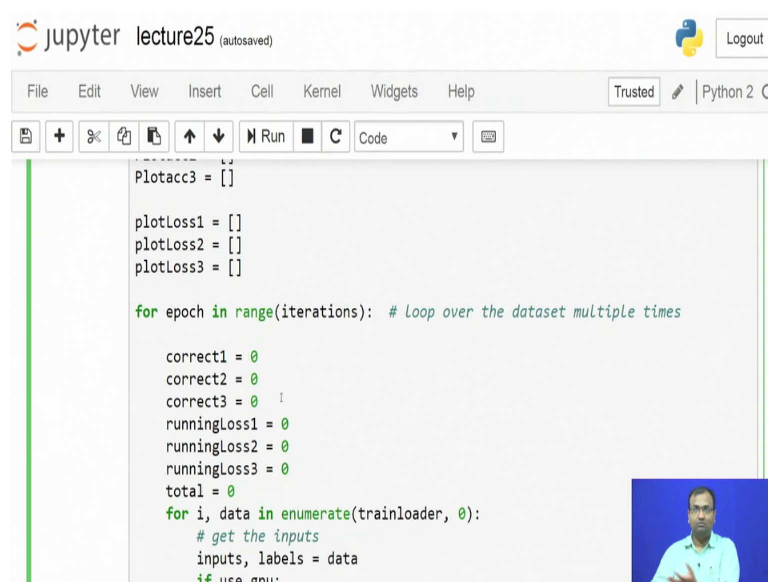
So, now if you go over here, you see that I have two different, I have the 3 different optimizers which are created. So, the first one is from my opt-in package, I called on my function called as SGD. Now this SGD, what it takes in is just a learning rate of ten power of minus 4. The second one is where I have an SGD call down along with a momentum of 0.9. So, where this actually significantly changes is that every single epoch after every single epoch that SGD has operated out it will add a significant amount of momentum to the gradient itself. So, well in the earlier case with vanilla gradient descent you had your learning rate which was eta. So, this eta is what gets multiplied on to that, but along with that we are just going to add down an extra momentum factor to the complete aspect over there. So, this is the whole eta times of nabla j plus a certain dc factor which is handed down and it has been found that with it adding this moment.

Now, typically what happens is if you are very close down to one of these saddle points. So, maybe your global minima is over here, but you are actually over here. Now, this being a local minima over there your error is still going to settle down to 0 and you will not be able to come out, whereas if we have a dc error added down over there so it knows that it is going to somehow climb up over there over a few epochs and then settle down to this exact point of my global minima over here. So, that is the point which goes on with adding momentum to my stochastic gradient descent.

The third one which we do is an adaptive momentum approach the difference which comes down from a stochastic gradient descent to an adaptive momentum approaches that, this momentum value which is given on as a constant it actually adapts itself over epoch looking at how many number of times I have the total number of epochs which has run down as well as the change in relative change in errors which have been occurring down while it is going down the number of epochs. And as it keeps on going lower and lower this moment of factor also keeps on adapting itself and coming down. So, this is what we had learnt in the theory and today we are just going to put that into practice.

So, for each of these three different networks which apparently get trained by 3 different optimizers I just have three different variables to put down my accumulate my accuracy as well as my loss because this is just a classification function.

(Refer Slide Time: 09:11)

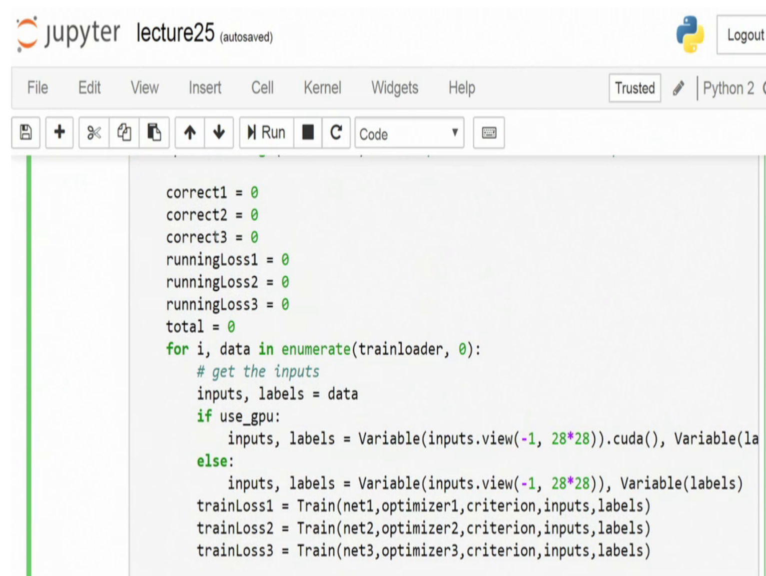


I am just taking inaccuracies and not similarity measures in terms of MSE or anything, but you can actually take all of these examples and plug it into your regression loss function based examples as well. So, if you are just training an autoencoder for representation learning or say for denoising you can use the same kind of a concept. So, say a denoising autoencoder or denoising sparse autoencoder can also be trained to using either an SGD or an ADAM or whatever you choose to like over there. So, we have trained it with an ADAM though, because it was coming down to much faster one, but by

now you have understood about what is the difference which comes down with batch updates versus single updates versus one epoch based objects. So, you can go back with the same kind of a learning option over there.

Now, today we will be looking into the difference what comes down between an SGD and an ADAM and while all the earlier experiments have been with the standard vanilla gradient descent, you can actually bring it up to that level and compare it up completely.

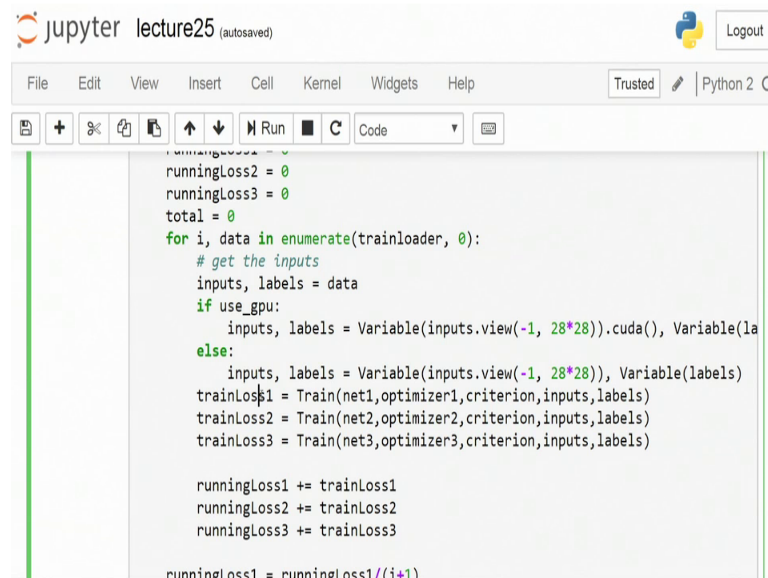
(Refer Slide Time: 10:07)



So, now within my epoch what I am going to do is since there are three networks and I need to compute out number of corrects and number of the loss over there. So, I just initialize these variables within every single epoch over there.

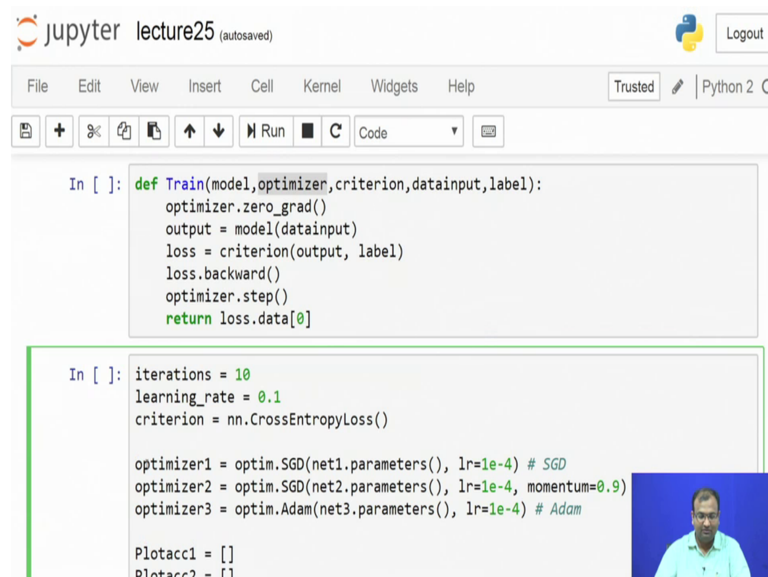
Now, within over there what I do is within an epoch it is going to load down in batches. So, my batches over here of a batch size of 100 because that is what we figured out as the optimal way of training down a network.

(Refer Slide Time: 10:34)



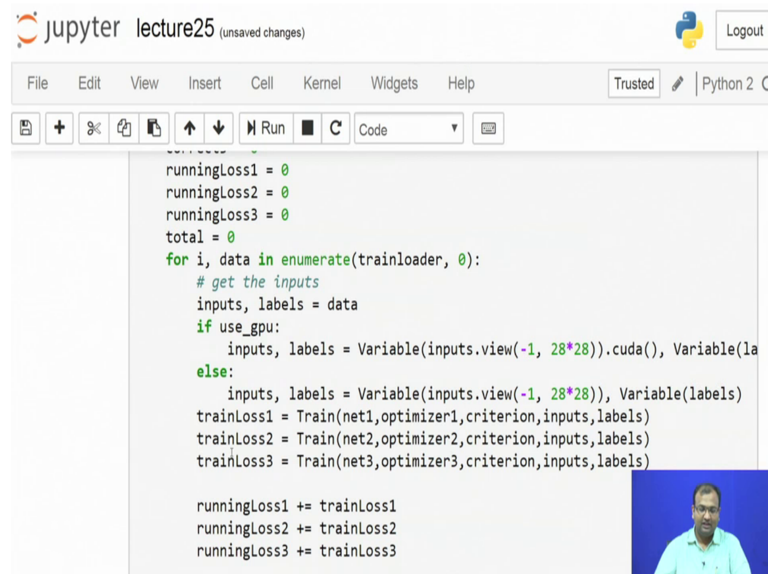
And then it starts its own function over there. Now, the training function within every epoch what it does is the input argument to it is a network then you have the optimizer whichever you are going to use. So, as an optimizer 1 2 3 is what gets defined over here, earlier and then you can keep on doing.

(Refer Slide Time: 10:49)



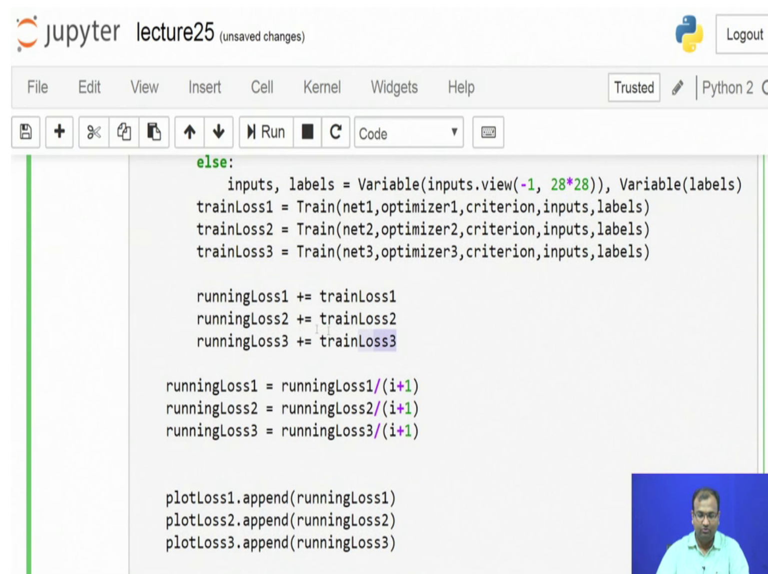
So, let us mean on this part of it which is the general function for training with the training function has a call and then we can start with our training within an epoch.

(Refer Slide Time: 11:02)



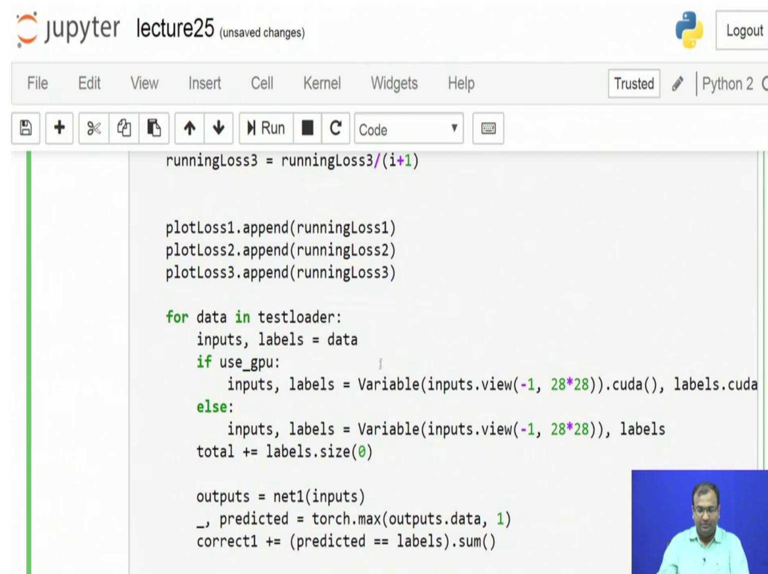
So, now I have my three losses coming down from my three different optimization functions for the same network though it is just different pointers. So, that you can track out which one is getting updated at what cross.

(Refer Slide Time: 11:19)



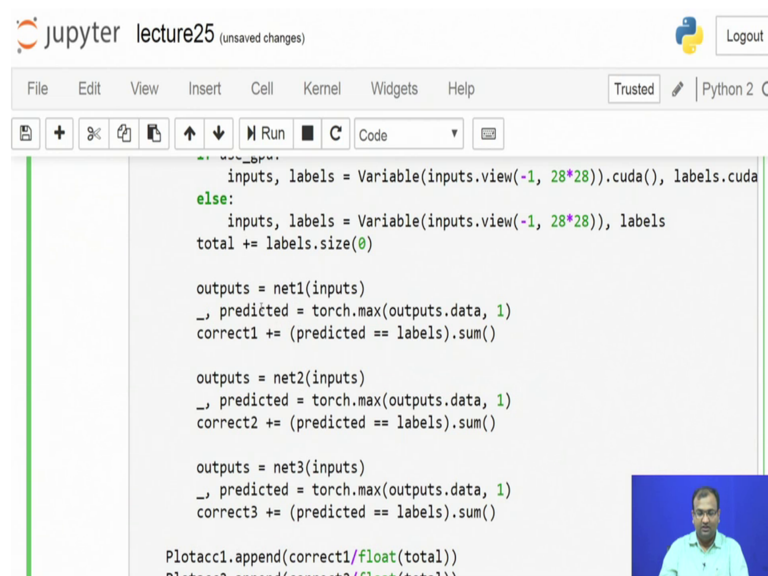
And you find out your total loss over that whole all the samples within one single batch and then cumulate it out over all the number of batches in your epoch and then you have the loss per epoch coming down to you.

(Refer Slide Time: 11:36)



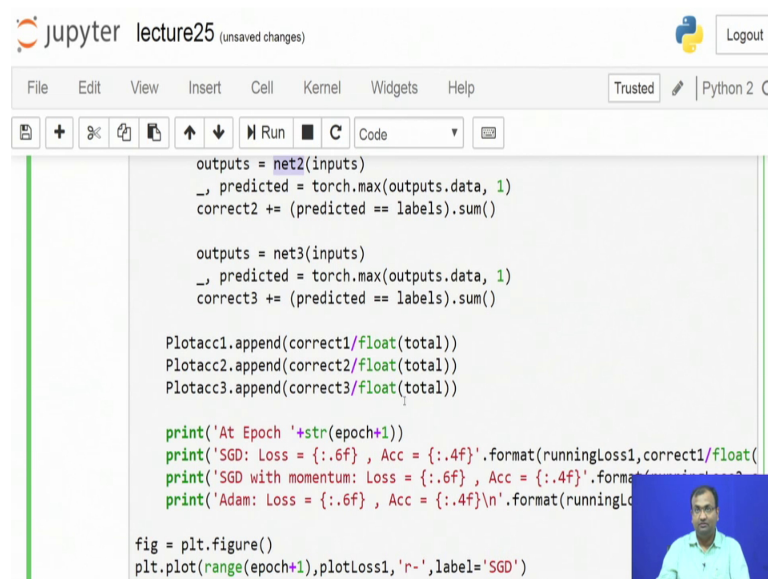
The next part is basically to find out what is the accuracy over there and for that we will be making use of the test data set and then making dominant use wire the test loader for this data set itself. And then for every single network we are just going to do a feed forward over the data which is present in my input and then get down what is my predicted value and then find out how many number of it are correct for say the network 1.

(Refer Slide Time: 11:50)



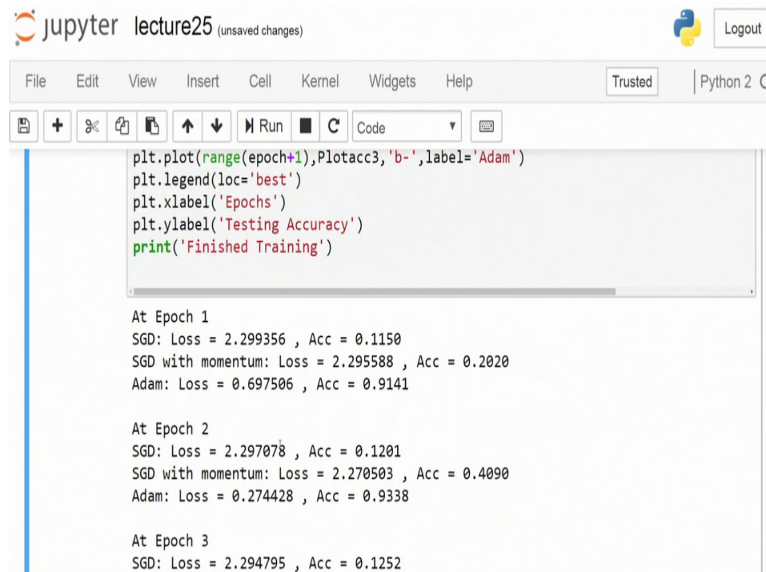
Then the total number of corrects for network 2 by doing the same thing except for that the network over here changes. And during testing it is just a feed forward inferencing engine it does not have to do anything with optimizers during testing itself. But this testing is there within every epoch so once it is updated within an epoch I just want to see what is the performance going down over there and finally, we just end up plotting this one.

(Refer Slide Time: 12:17)



So, this, these parts of the code are what is, what are more familiar with in the earlier examples and it says just a direct take away from them. The only reason we put it down here so that you have one distinct place where everything is written from scratch for every single thing which you are doing and you can make reuse over how you choose to do it and these can act as standard templates.

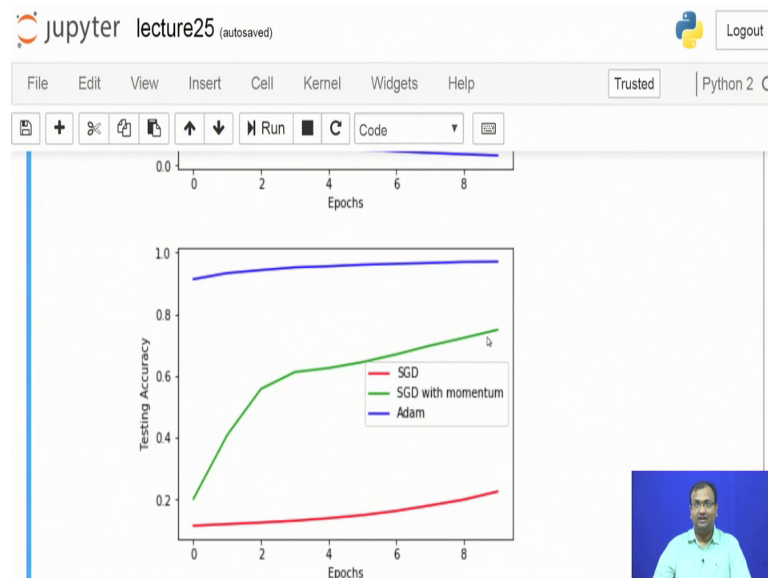
(Refer Slide Time: 12:47)



So, now let us look over here you see that typically an epoch does not take quite long to go although we did not put the timer over here, but sometimes you can actually put the timing engine over there. Now, when I am trying to do it with just SGD over here you see that I have somewhat of an accuracy of 11 percent with SGD with the momentum I did go down to an accuracy which is higher over there and one of the reasons is that, say it was hitting down this local minimum point within every single, within any of the batches with in an epoch. So, it was able to actually dislodge itself and go down towards other global minimum points or it is still continuing to do in that way.

Whereas, on other case you see with ADAM coming down the loss has decreased signify and the accuracy itself at the end of first epoch. So, the in the first epoch you just had 400 times of an update on to it which is the total number of batches sorry 600 times of an update you have back size of 100 and your total number of samples are 6000. So, 600 times is what is updating within every single epoch. Now, over there it already is at a point of 91 percent now that is large series seriously nice number which you have seen in the earlier cases and one of the reasons is the advantage of getting this adaptive momentum coming into it, here as compared to the other ones.

(Refer Slide Time: 14:06)



Now, this does bring you to a point that whenever you are trying to use an ADAM or whether one is that your losses are definitely much lower although it is the same loss function and I am not changing that different loss functions that they will have different dynamic ranges. It is the same loss function across which it is pitted down and it is being compared. Now, on the final performance on accuracies you would see that with the stochastic gradient descent, it finds really hard to actually pick up it will take a long time in order to go down over there whereas, with it is some sort of a momentum added over there it comes down closer to actually going down to the final accuracy whereas, with ADAM it really comes down to a sweet spot on the final convergence accuracies over there and that is an advantage which you get done with this adaptive momentum.

Now, the method is not very old it is actually a new it is this, see this decades itself one of the major contributions. If you look on the other side of it going down by plain vanilla gradient descent to a stochastic gradient descent, these were things which are 30 to 40 years older and in the last 10 years we have a method which is much more superior to that and something which is finding its own popularity and gaining its significance for all the different kind of applications which you are going down with. So, in the earlier cases you have used you have seen that we have been using ADAM for certain number of cases in order to just within 10 epochs come down to a good convergence and show you this convergence comes the reason is that if we flow down with this say stochastic gradient descent or vanilla gradient descent instead of just 10 epochs it is going to take

me really long I mean I would possibly have to continue for a thousand epochs or something. So, this curve to reach down somewhere over here which is closer to it.

Now, in order to just come down to a tractable time and adaptive momentum approach or an ADAM is something which is more appreciable and on this note I would definitely go down and suggest that for a lot of problems and them works out. However, you need to give one thing in mind that your batch sizes need to be significantly large. So, we are handling our batch sizes over about 100, the moment you come down to a smaller batch sizes say batch sizes of 10 or 12 or 16 and I might not significantly prove as a good option and in those cases rather than trying to go with ADAM its easier that you actually stick down to plain vanilla gradient descent or a stochastic gradient descent approach.

Now, this is up to you we are we have small patches of 28 cross 28. The moment you go down to large sized image say 2 to 4 cross 2 to 4 and in the next lectures we will starting with convolutional neural networks and that is where you will be dealing with significantly large sized images not those stamp size and thumbnail sized images in any way and that is where you will be seeing where the difference comes in.

Now, going with those large size images the amount of viable RAM which is available to you to run any of your codes becomes lesser and lesser over time and your batch sizes need to be appropriately decreased. So, that is where you would now start facing a problem that can I use ADAM at all if I cannot use ADAM then will my vanilla gradient descent or my stochastic gradient descent come down to that kind of a convergence in a much better way.

So, this as we keep on going down with those examples of large sized images you will be having complete feel about how it works out in fact, will be going down on with multi class classification problems as well and then also trying to do none temporal classifications from over videos and that is where you find these of great use.

So, with that we come to an end of understanding things with our auto encoders and using those auto encoders and just simple neural networks to understand what happens down with different cost functions, different kind of learning rules, different kind of optimizations and how they can be used intermittently, exchangeably between each other with a lot of scratch parts.

Next class, next week onwards which is where we start down with our CNS or convolutional neural networks, and then on I am not to get back anymore on to these kind of rules and updates and techniques because they just remain same for that you are using a fully connected versus whether you are using convolutional operator over there. The only thing which is going to change over there is the architecture is much more elegant it is much more expressive and the computer around that architecture is also something which is intriguing.

So, we will be getting into more details of those as we keep on discussing. So, till then stay tuned and then keep on enjoying for the next subsequent weeks.