Deep Learning for Visual Computing

Prof. Debdoot Sheet
Department of Electrical Engineering
Indian Institute of Technology, Kharagpur

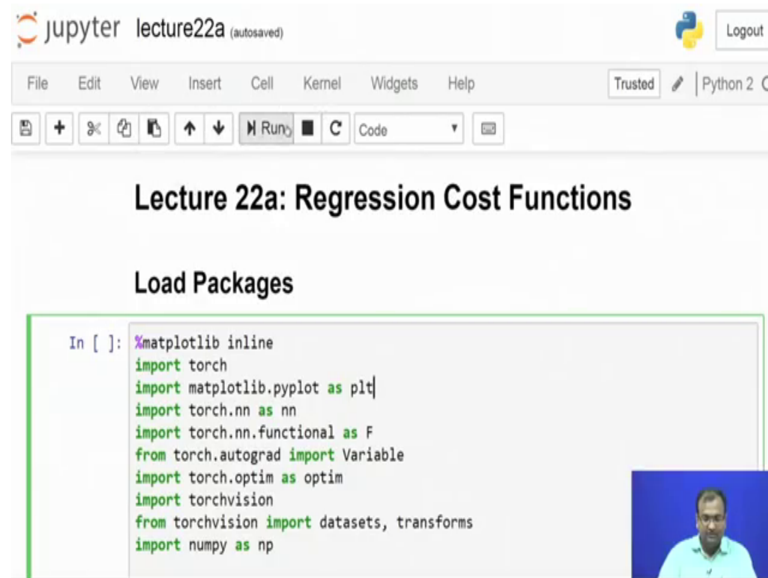
Lecture - 22
Classification Cost Functions

Welcome. Today we are going to do our understanding of different kind of Cost Functions. So, while in the previous lecture I have spoken about the 2 different families of learning problems one of them is (Refer Time: 00:26) the other one is classification and whenever you are trying to measure down performances, you use something which is called as the cost function and for degration you have one family of cost functions coming into play; for classification you have another family of cost functions.

Now, in typical autoencoding case you had seen that your in order to understand features through a reconstruction engine. As in an autoencoder you were using down a simple cross function like a mean square error loss or l 2 norm over there. So, this is not just the only one which you can use actually in order to just check out, whether whatever you have reconstructed is pretty much similar to whatever was given in the input.

You can also use something called as a l one loss or just an absolute error between 2 or absolute sum of error or absolute mean of error these kind of matrix as well. Now while we have studied one part which is just to do along with the theoretical aspects and the mathematics of how they work around.

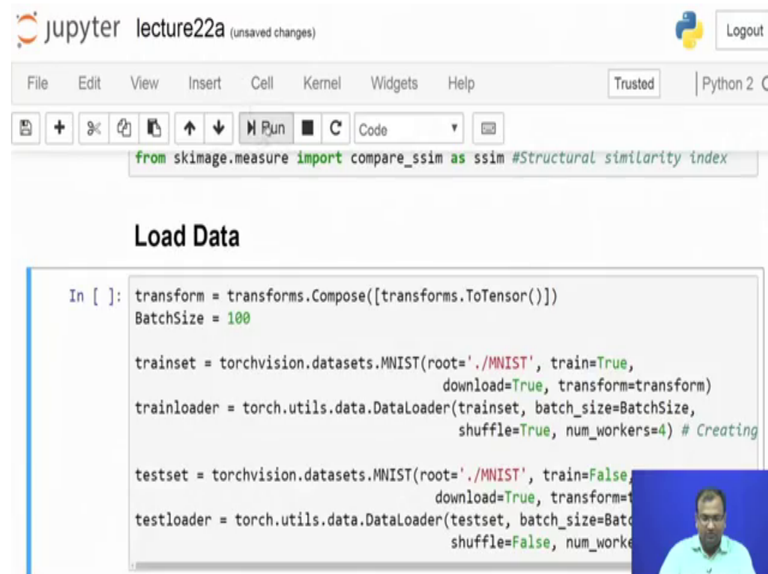
(Refer Slide Time: 01:18).



Today I would be focusing on how they play a different role while being used in your learning problem.

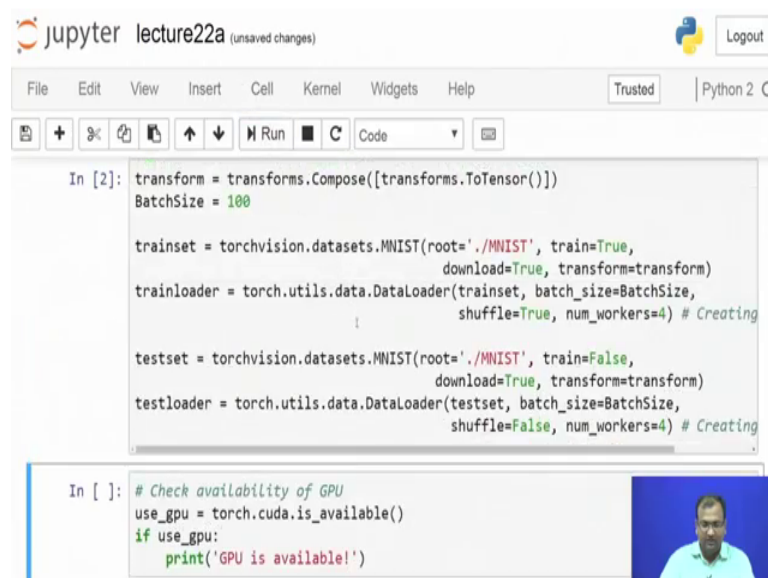
So, for the first part which is on regression cost function I am just going to demonstrate this one using autoencoders as a typical strategy and without much of saying, where it helps out is that while you have been learning on all auto encoders till now with just mscdos or mean square error. You should not be thinking yourself to restrict it only to an MSC loss in any way and that is where we make a change over here. So, let us get down on with it.

(Refer Slide Time: 01:51)



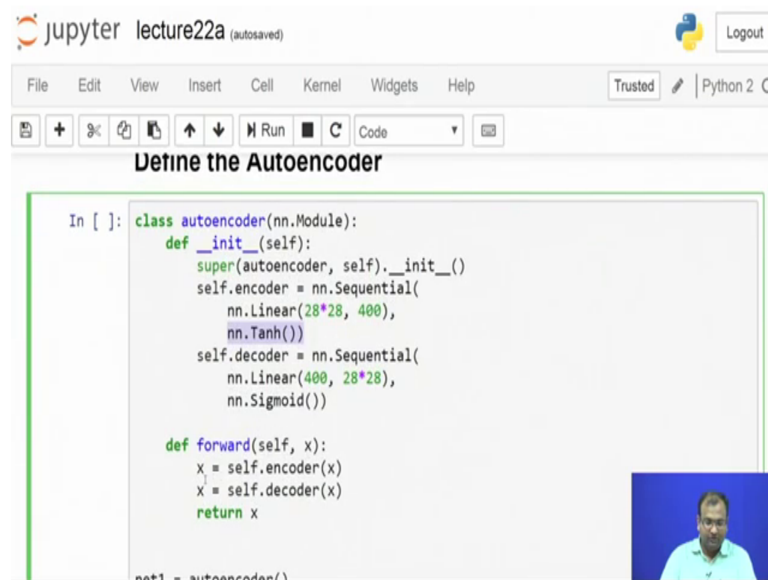
So, I load down my packages whichever I need over there and then I would be starting with my data and over here.

(Refer Slide Time: 02:00)



We stick down to our standard MNIST example as a (Refer Time: 02:01) example as we have been doing for all the other ones. Now check out for your gpus, that is available great and now I can start by defining my autoencoder.

(Refer Slide Time: 02:13)



Now, this is a very simple autoencoder as I have been using in the earlier cases as well. So, it just connects down your batch of 28 cross 28 or 784 neurons onto 400 neurons, which is present on the hidden layer. So, similarly on your decoder side you just map down 4 hundred neurons onto 700 and 84 neurons which forms a 28 cross 28 square patch the over there.

Now once your definition is complete, what happens is that your non-linearity in case of the mapping down to the first hidden layer is a tan hyperbolic and the non-linearity when mapping out to the output layer is a sigmoid over there. So, this tan hyperbolic will give you a dynamic range of all the outputs of your hidden layer within the range of minus one to plus one and your sigmoid is going to give that in the range of 0 to 1.

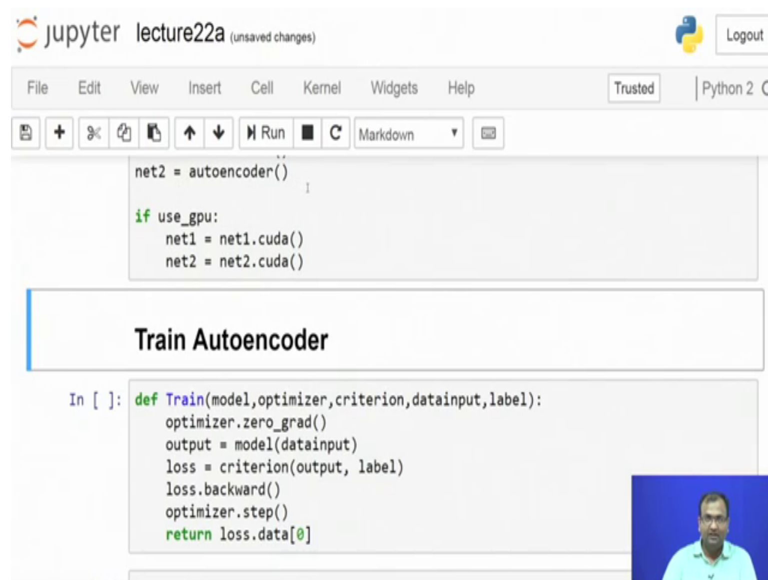
So, now, on the forward pass definition you just do a forward pass over the encoder get it is output then do a forward pass over the decoder and then you have your network defined over there.

Now, since we are going to use them to different cost functions. So, gradients of cost functions and everything in an effect will have it is own role. So, instead of trying to change the cost function every time and show you them and I am actually going to demonstrate training it down in one single epoch. So, what we do is we create 2 replica networks over there net 1 and net 2.

So, identically these 2 networks are quite identical to each other they are just made out of this autoencoder and nothing different over there.

However we will train down one of these networks with MSC cost function and the other one with an l 1 norm; cost function and that is the only difference which we have over here. Now if you have your code available your gpu available then just convert into cuda and then see it over there.

(Refer Slide Time: 03:52)



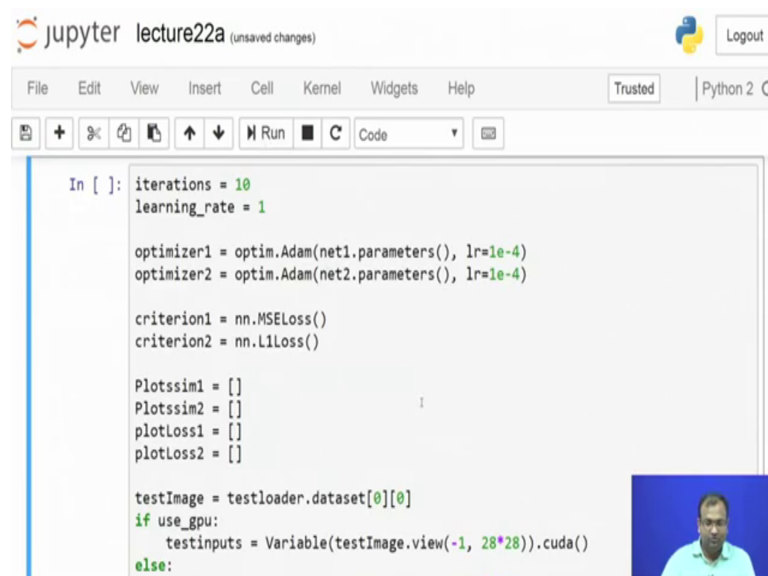
Now, that your network has been defined completely the next point is actually to start with training of an auto encoder. So, what we do over here is something clever. So, the idea was to write down a trainer function itself, where you can give as arguments to this network, which you would like to train the kind of optimizer, which you will be, which you would like to use for training the whole thing. So, you can use from your vanilla gradient descent to Adams stochastic radian descent anything whatever you write down, the criterion or the cost function which you would like to have the data to be given down over there and the labels which are associated with it.

So, this the whole reason for bringing this down as a function was later on when I write down my whole training routine over there I do not want to keep on using these internally over there.

So, one problem is that I will have to create down multiple instances of the same variable name and everything coming up; the other problem is that that just cumulates out on the lines of code. So, today I am just doing it I am just solving out integration learning with just 2 different cost functions maybe I would like to see what is the performance over some 10 different cost functions 11 different cost functions or something over there.

So, that would otherwise equivocally replicate to copying this same piece of code by changing down variables over that long period of time. So, the same the basic reasoning why we choose to go with functions is why I just chose to do it that way.

(Refer Slide Time: 05:16)



So, now once this function for training the model is defined next we enter into our trainer over here. Now optimizes in both the cases is chosen at the same optimizer; however, we just keep that as optimizer 1 and optimizer 2, because these are certain pointers which can define to the actual variables which are useful in the optimization parameter and each of them is specific to each network.

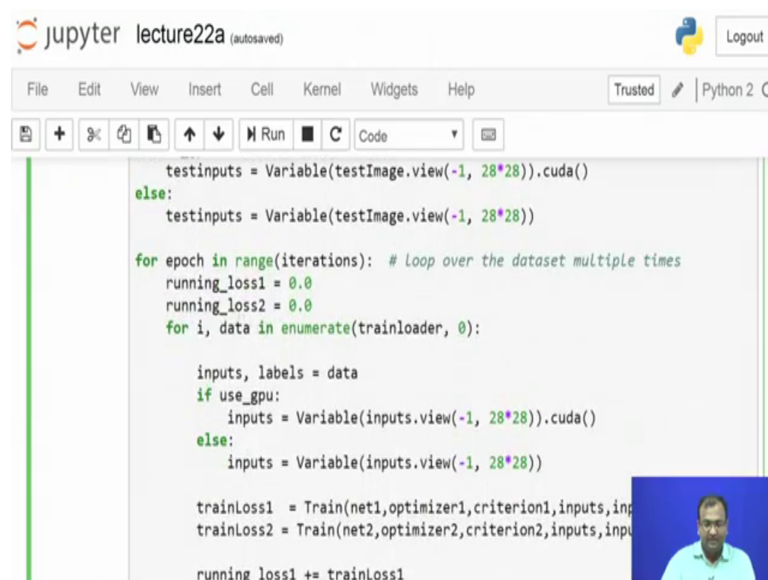
So, for network one there is a set of container variables network 2 there is a different set of container variables and the only difference between network 1 and network 2 is that they are trained with 2 different loss functions. Criterion 1 is what is used with a network 1 criterion 2 is what will be used for network 2 as we would go down over there.

Now, in order to measure down the performance we use 2 different measures on of them is called as s s i n or structural similarity index, which is more from taken up as a concept from image processing and quality assessment for reconstruction of images.

So, structural similarity index is a measure which is sort of like if you are close to one it means; that both the images are very similar to each other. So, it means that your deconstruction has been great. If you are on the negative or minus 1 then that would mean that 1 image is inverse of the other image or negatively correlated to the other image over there. If you have 0 then; that means, that one of the images just went on 0. So, that is that is what happens within s s i n.

Now, last over here is just the standard absolute mean absolute error over the total difference which is taken down over there. Now within my training what I do is I set up my data loader and then would just be invoking if there is a gpu then convert them into a gpu equivalent thing for my invoke.

(Refer Slide Time: 07:02)

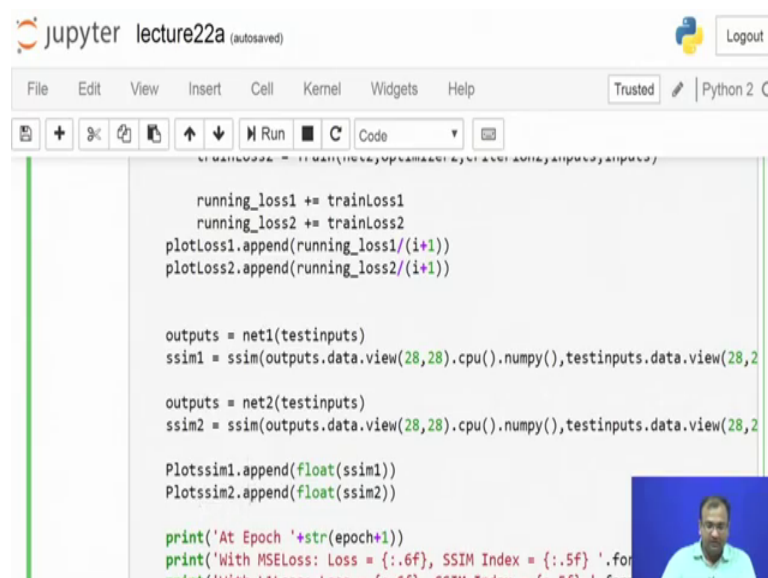


And then I start iterating over my epochs. Now inside what I am technically doing over there is that I try to train my model, which is net one using an optimizer optimizer 1 and the criterion function which is criterion 1. The whole reason is see I could have used the same network in both the places, but then you know that within every different call. So, when within an epoch it is going down for this one, then the output of that network is what is stored in the container variable or the pointer called as net.

Now, if I use the same net in the next call. So, it is basically going to modify my same network once again and that is the reason why I choose to have 2 different networks defined. So, within each epoch you will have each network getting modified by itself you can choose to have 2 different optimizers, but I since I have the same optimizer I can still make up with the same both of them can also be optimizer one without much of a problem, but then criterion are different for each of them and your test data and output data both of them are basically the input pattern which you have over here.

So, taking all of that just create out to your losses and then the final part is once your training is over you keep on accumulating, the ssin scores across each epoch as it keeps on going as well as the losses over there.

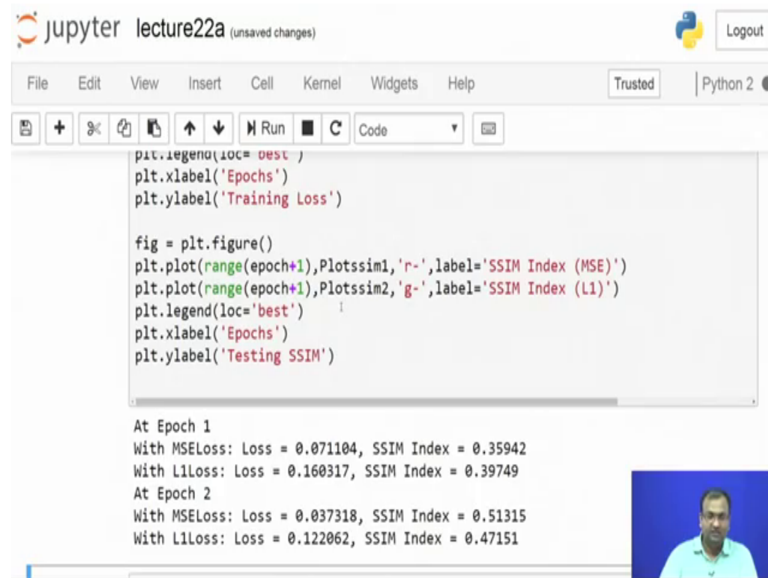
(Refer Slide Time: 08:17)



And then finally, we decide to just plot it out.

So, let me run it over. So, I just write over the tan epochs.

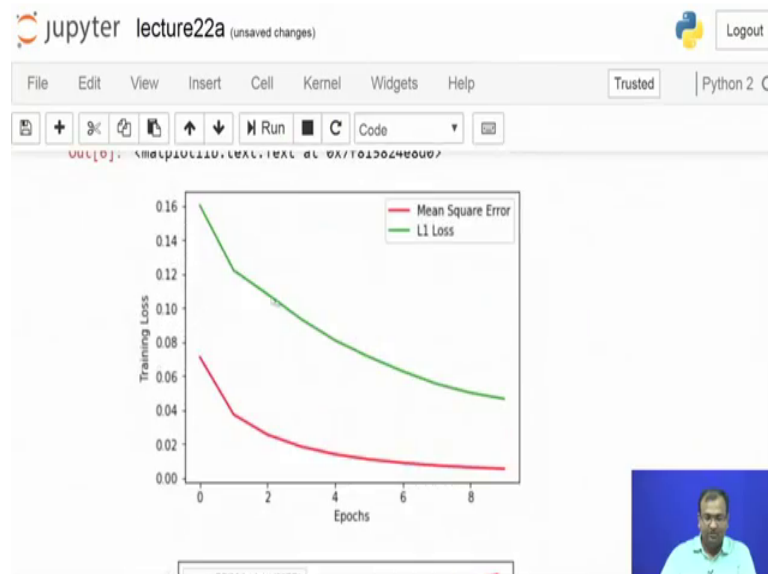
(Refer Slide Time: 08:26)



So, let us let us see how this works out. Now one thing which you keep in mind is as I have been saying with the earlier lectures as well that is that the dynamics at, which your error is going to fall down or your accuracy or your reconstruction efficacy increases; that is very much dependent on the data itself and it does not technically have anything to do around with what kind of an optimizer or you cannot even see it at the start of the whole problem that. This is where I will start from and this is where it would actually decay and go at the end of it.

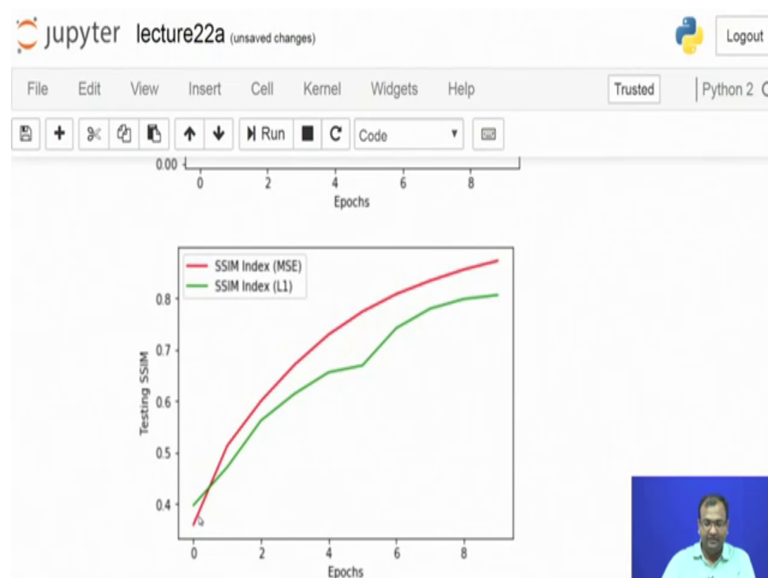
So, that is that is definitely a challenge which comes down over here. So, now, if we look into this particular form over here you would see that it has trained out for 10 epochs and then subsequently you see that the training loss has significantly gone down.

(Refer Slide Time: 09:13)



And we decide to plot down both of these losses or both the errors 1 of them is the l 1 loss the other one is the mean square error loss which comes down over there.

(Refer Slide Time: 09:24)



The other case is that as the images which are getting reconstructed as the loss comes down. So, you basically have the total quality of the image going better and better and that is what you see down in your MSE.

So, you see it starts somewhere in around 0.4 in both the cases which was a random point and then it goes up. Now 1 question you might ask is that I see that both have 2 different

points of starting and that is for the reason that they were randomly initialised. So, each has a different wait which come down over there. So, there is a slight change in how they behave; however, the objective is actually to look down at the trend over there and not exactly how they behave?

Now one more point which we need to understand as well when you looking down at l one loss which is mean absolute loss over there, now the mean absolute loss is sort of which is linearly dependant on the change of gray levels whereas, mean square error which is as a square law proportion or exponentially proportional on the difference between the gray levels which comes down.

And this kind of a loss higher weighted loss as in a mean square error is something which has a higher propensity of coming to a convergence, because the more the error it would. In fact, significantly amplify the whole error over there. So, now, if I try to take down some generalized l norm over there, which is so say raise to the third power or fourth power or fifth power, you would see even this curve going down much steeper and coming down to a much better convergence than in most of the cases.

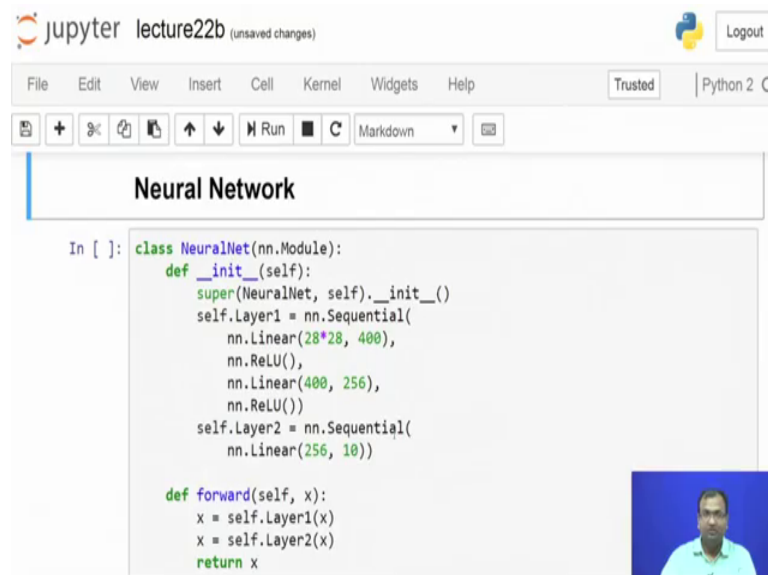
However 1 important thing is that whichever norm you take you need to see that it is differentiable. Now any anything which is higher than the quadratic order is always differentiable. So, it is not a major issue over there the next point is that since you are raising all of these differences through a polynomial order. And also subsequent to that after summing up everything you are going to take a square a route of that particular order to which the polynomial was raised.

So, if this becomes an l 3 norm it becomes the input x i minus the output x o raised to the an absolute value of that raised to the power of 3 and summed up over all the values and then a cube root over it. So, this cube root computation as well as raising this value to a number order of 3 is what is computationally complex? So, while you have your distinct advantages at the cost of this being a bit slow a tan bit slow not much of it.

So, this is one simple example of trying to do around with a simple autoencoder and a regression loss the other kind of a loss, which we had studied was actually to do with a classification loss function and that is what I would be doing in this next lecture over here which is on 22 b. So, the next part is to identify in different cost functions. So, let us get how we are doing. So, initially I just load down my packages over there and then you

have your data being loaded. And this data over here is still the standard MNIST and go down and get your g p u available and then you start by defining your network over here.

(Refer Slide Time: 12:23)

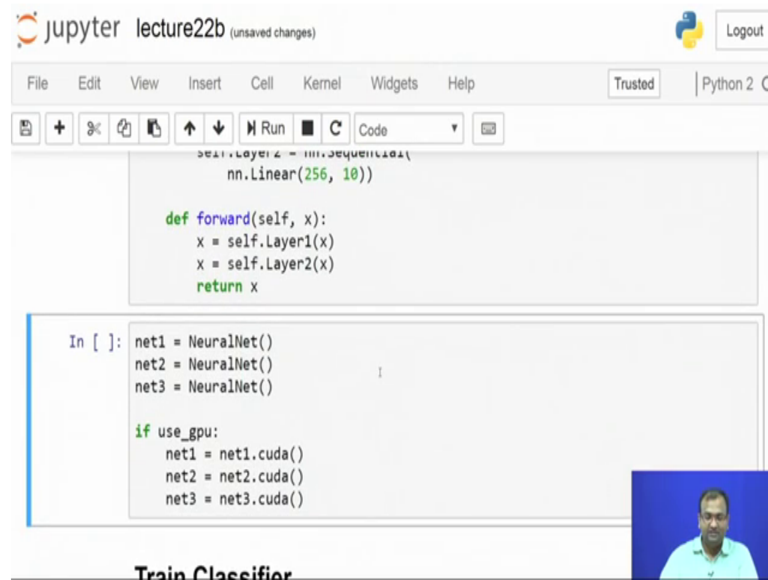


Now, the network which we do is a pretty simple network this is quite similar to a network, which you had seen earlier in in the initial few lectures is where you take your input image which is 700 and 84 neurons or 28 cross 28 sized. You map them to 400 neurons these 400 neurons are mapped onto 256 neurons and they are subsequently mapped down to just mere 10 neurons.

There is no autoencoder there is no autoencoding way of earning down features or anything, which is taken care of over here and this is a simple single feed neural network which is being defined. So, the your only objective is the given a image of size 28 cross 28 I just need to know what class it belongs to .

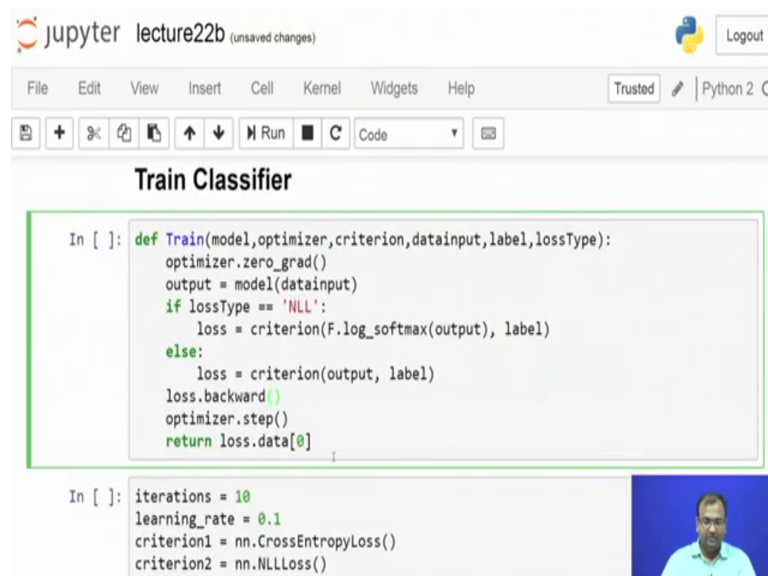
So, now, the only point is this part which is called as layer one this part of the network is what is your sort of what is your. So, sort of representation learning network and this is which is your classification network over there? And then it is just very straight forward to do it infact you might not even need to add this extra, but because you could just add a nn dot linear 256 to 10 as well from the standard definition point of view and then you find the forward pass over this network and that sets my network going.

(Refer Slide Time: 13:31)



Now, for classification we are going to use 3 different cost functions and for the same reason. So, that they do not have any future confusion effects we just define then as 3 different points to 3 different network and they are net 1, net 2, and net 3 the same logic as we had 2 different cos functions, we were doing it for the earlier demonstration with regression. So, here since we are having 3 different cost functions we just define it as 3 different networks. So, now, that is defined and loaded onto my coder now I can start my training over here.

(Refer Slide Time: 14:01)

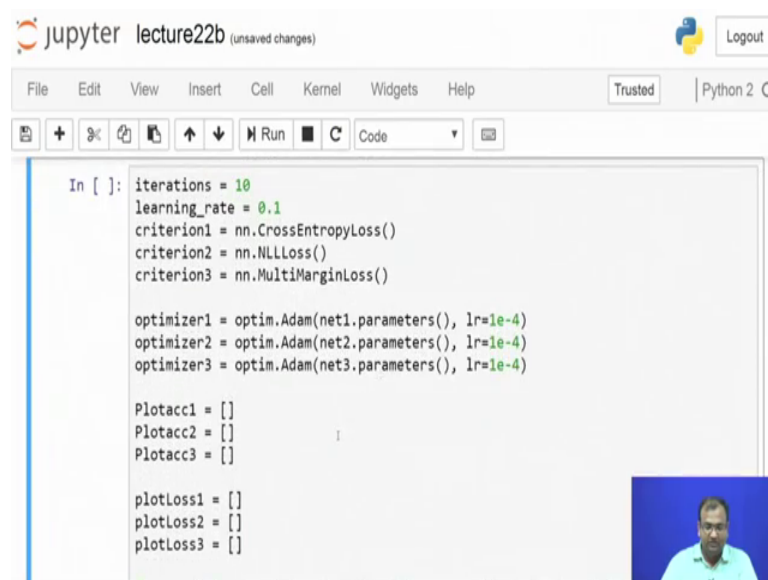


Now, within my training of this classifier what I do is big critical over there.

So, I would be using 3 different kinds of cos functions over there for (Refer Time: 14:11) and now for each cost function your output of the network needs to be in a particular dynamic range. And it can take only a certain specific types of nonlinearities over there. Now that has to be modelled down appropriately and if the need be then we need to modify it out. So, for that reason what we do is if say that one of my criterion function over sorry yeah one of my criterion functions over here is actually a negative lock likelihood criterion function.

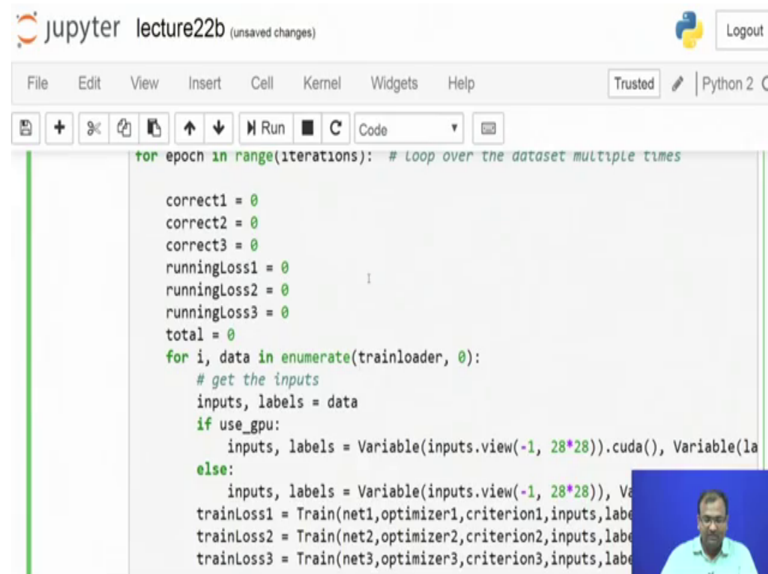
Now in that case I am actually going to change out my output over there and I need to have a log soft max of my output taken down. Whereas for others I will not need to have a log soft max of my output taken down. So, is you can just go back to the previous lecture and revise out why you needed that log soft max which I have mentioned out clearly.

(Refer Slide Time: 15:07)



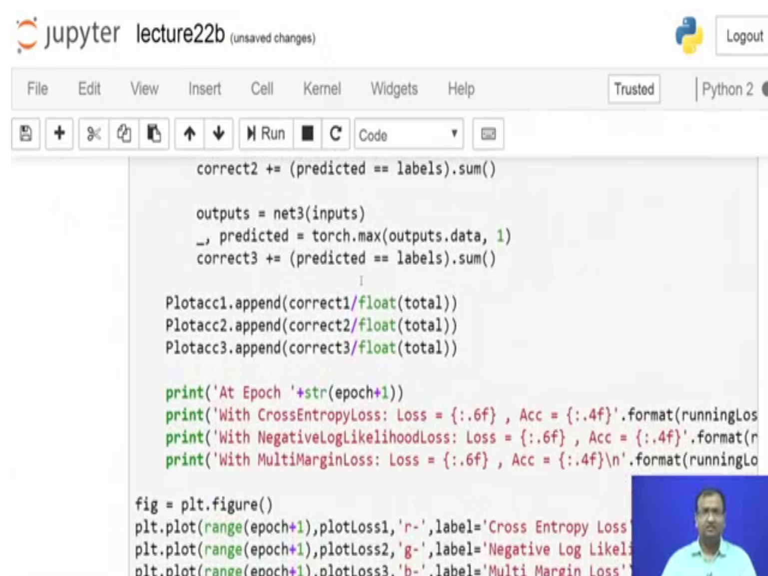
Now once that is done, next is to start with your training over there. Now for our purpose we are just going to stick down to an Adam optimizer for the purpose of it and with a learning rate of 10 power of minus 4 and then subsequently all of these things get defined, and we start our learning over here.

(Refer Slide Time: 15:18)



By the same argument as if we had for the example in the regression case here also for classification sense I am going to use 3 different cost functions over there. So, for each of them I will have a different network, which is rained out and for the 3 different cost functions which I do. So, one of my cost functions is cross entropy loss the other cost function is negative log likelihood loss and the other one is a Multi Margin Loss criteria which I use.

(Refer Slide Time: 15:49)



So, now once that is done I can just set this one running over there. Now here as in the earlier cases I had a regression problem to be solved and I was trying to reconstruct an image and actually try to find out how good my reconstruction is now here I do not need to do that. So, here what I am technically doing is you get a patch of an image and you just classify, whether this belongs to whatever class the ground route label was given on. So, if it is an image of 100 and digit 0 then whether you are classifying it as 0 if it was a 100 and digit 1 then whether you are classifying it as one or you mal classified it as say the number 7 or not.

So, my final way of cross validation one is your you are going to look down at a loss and that is going to come down, but then the loss dynamics and everything that does not have a direct relationship with the complete package of how it is running down and what is the final performance over there?

The final performance for whatever you wanted to do was actually to get down on understanding, whether I am able to solve a particular task which I am scheduled to actually out. So, for us the task which we wanted to solve out was actually classification so; that means that I need to see whether it is able to accurately classify or not. So, the number of times number which was say 0 got classified as 0 the number of times a number which was written as one got classified as one and so on so forth.

So, the total number of correct classifications divided by the total number of samples over which I am classifying so, that is what is called as an accuracy and that rather measure which I am going to use over here. So, now let us have a look over here.

(Refer Slide Time: 17:20)

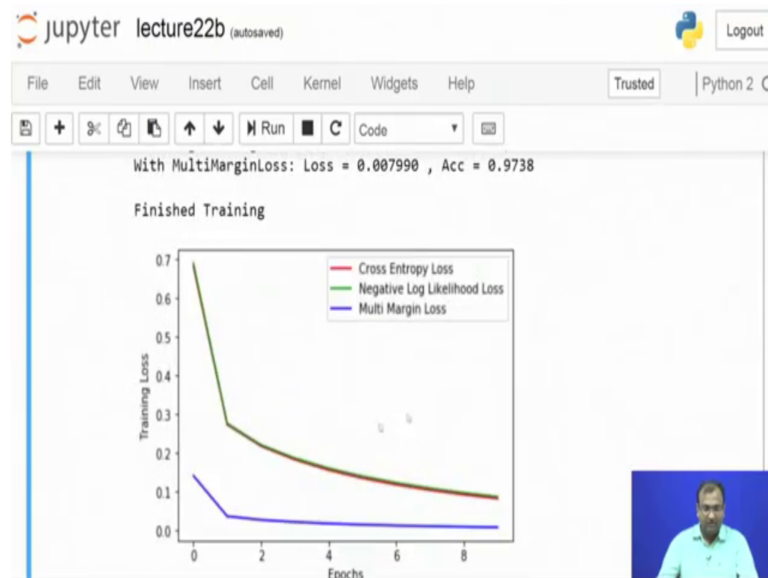
```
At Epoch 1
With CrossEntropyLoss: Loss = 0.682775 , Acc = 0.9144
With NegativeLogLikelihoodLoss: Loss = 0.690303 , Acc = 0.9126
With MultiMarginLoss: Loss = 0.141309 , Acc = 0.9154

At Epoch 2
With CrossEntropyLoss: Loss = 0.273348 , Acc = 0.9314
With NegativeLogLikelihoodLoss: Loss = 0.276065 , Acc = 0.9320
With MultiMarginLoss: Loss = 0.036570 , Acc = 0.9337

At Epoch 3
With CrossEntropyLoss: Loss = 0.218161 , Acc = 0.9439
With NegativeLogLikelihoodLoss: Loss = 0.220812 , Acc = 0.9430
With MultiMarginLoss: Loss = 0.027033 , Acc = 0.9446

At Epoch 4
With CrossEntropyLoss: Loss = 0.183081 , Acc = 0.9497
```

So, you can see as it goes down the epoch. So, for each of these different cost functions for classification I just plot down the loss as well as I measure down the accuracy at the end of that particular epoch itself. Now you can see that for 3 different cost functions there are 3 different losses; while each of them is just at the first epoch or whatever it is initialized with a wait at the start of it.

However, since 3 different networks just got initialized randomly, and they were not replica copies of one to the other so, these networks have some randomized waits which are taken down over there and that is why their starting points are a bit different on accuracy, but then the wait update happens after this one and then you would be seeing the change coming down across each of them.

So, you can see pretty much it starts from around 91 percent and then keeps on going pretty steadily to come down to about 97 percent for most of these cases.

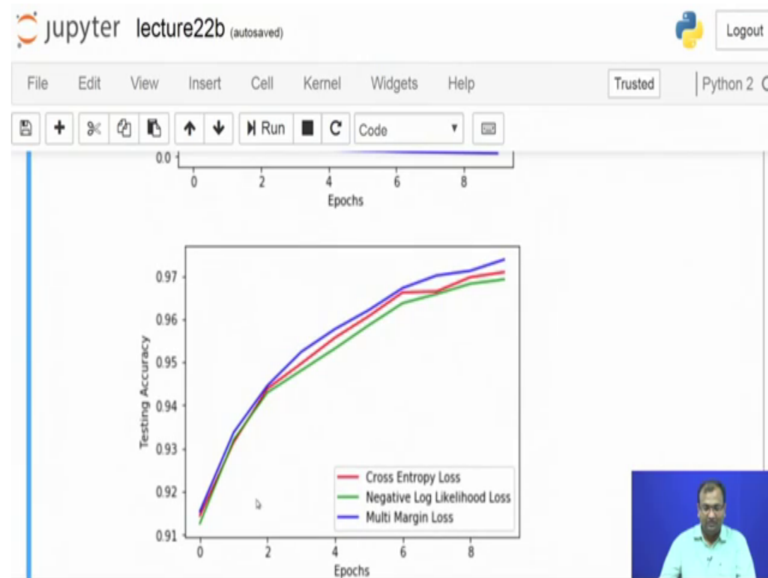
(Refer Slide Time: 18:11).



So, if you look in to this training loss, which comes down you can pretty much have a distinct curve for multi margin loss; however, negative log likelihood and cross entropy if you carefully look into the equations over there they seem to be quite inter moulded and quite similar to each other in like if it is in a perfect dynamic range, then actually they boil down to the same cost function and then that is what exactly happens over here for which you see that one curve is below the other.

So, if you look carefully the red curve over here is what is lying underneath and the green curve, which corresponds to negative log likelihood is what is superposed on top of it and. So, you cannot make out any 2 different of these curves.

(Refer Slide Time: 18:50)



Now, on the other side of it when we are looking down and actually see you see the accuracy keeps n steadily increasing and this is how for each of them it was moving on the dynamic side of it.

Now one clear understand is it does not def definitely depend on what data you are using and be whatever cost functions you use it is steadily going to do. However, the dynamics for each cost function and what will be the base error it would reach down id pretty different; having said that it does not necessarily mean that if you have the least error which you are received that you would be having the highest accuracy.

Because say all 3 of them have almost a comparable accuracy at the stop of it whereas, just by using a Multi Margin Loss, I was getting down a lower loss as compared to that the only reason being that multi margin loss has a different dynamic range in which the losses would come down as compared to cross entropy or negatively log likelihood.

And actually being just an absolute measure it does not count on what is the dynamic range of the signals or what can be the dynamic range of my output, which are coming down it just measures down low many number of correct classifications. I have done over the total number of samples which are present over there and provided to me. So, this this makes it much more easier to actually understand.

Now, at this point I would leave it up to you and your imagination, you can actually make up your own cost functions, you can find out their forward passes and backward passes make their own definitions as we had done in the earlier; classes for when we were trying to define other networks and their different parameter properties. And in fact, like the moment you have a function whose forward and backward is defined, you actually can either use it as a network layer or you can use it as a cost function in any of these and then mould and go on.

So, these are topics where you can do an advanced experimentation on your side you can find out how these things behave on an advanced level and pretty much.

So, this brings us to an end of our discussion with trying to understand different kind of cross functions and their effect on the learning rate dynamics, which were this case hasn't been that significant in in like using any cost function pretty much gives you, but then that is not always the case there will be complicated ones which we will encounter down the line when we start working towards convolution neural networks and then also in order to do multiple class classification or multiple class classification over there.

So, there your loss functions have a significant impact because sometimes certain loss functions do not work for a certain number of cases then you might have to choose down a very curated out loss function over there. So, with that we come to an end on this discussion with different kind of loss functions. So, stay tuned on the, onto the next lecture to understand about learning a dynamics and different kind of learning rules as well.

Thanks.