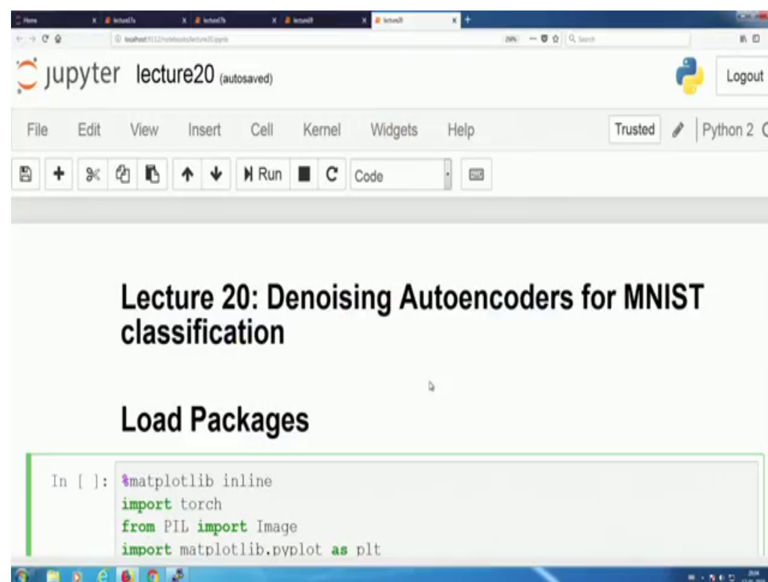
Deep Learning for Visual Computing
Prof. Debdoot Sheet
Department of Electrical Engineering
Indian Institute of Technology, Kharagpur

Lecture -20
Denoising Autoencoders for MNIST classification

So, welcome and I take over at this point from where I left in the, had left in the earlier one. So, in the earlier lecture you had seen about, how to train down a sparse auto encoder and a simple network, which had just, we were taking down the MNIST digits input over there and you had 28 cross 28 or 784 neuron, which connected down to just 400 hidden layers and then from there, for the auto encoder, you again reconstructed back to 784 and then for classification you just went down to 10 classes right good.

(Refer Slide Time: 00:32)

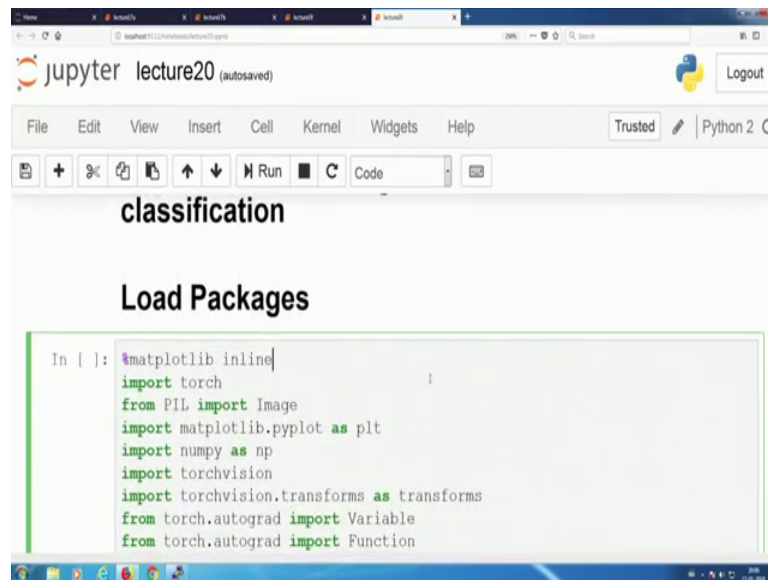


So, two lectures earlier, what we had done on lecture 18 was actually to get you introduced to two different dual concept; one was parse, another was denoising and then we said that these are independent of each other, while in the sparsity you actually learn to have over complete representation, in denoising you become more robust to removing out noise. So, one good aspect was that. Now, you can have a neural network which can cleanse an image, you can remove out noise from the image and we did also realize from some of the other papers that, while you are introducing more and more noise your network tends to learn better and better features over them and. In fact, the number of

unique features, it learns in an over compute representation and the better number of features. It lasts is much higher, when going down with a denoising auto encoder.

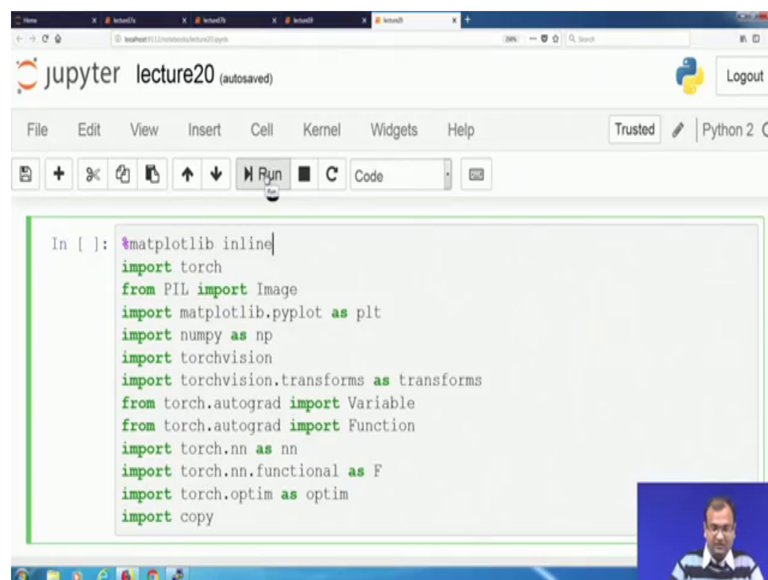
So, here we take the standard example building up on top of what we had finished off with a sparse encoder.

(Refer Slide Time: 01:45)

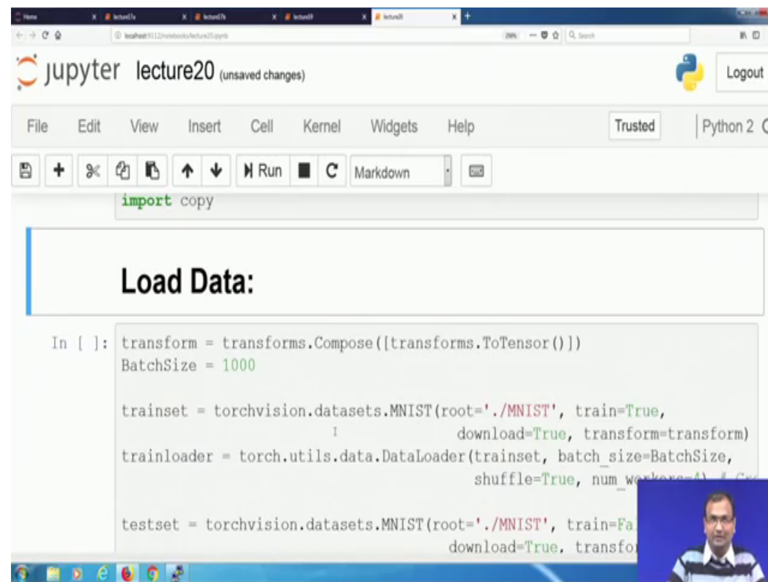


In order to get into creating something called as a denoising sparse, in auto encoder for MNIST classification. So, they are all standard, stand alone runs.

(Refer Slide Time: 01:52)

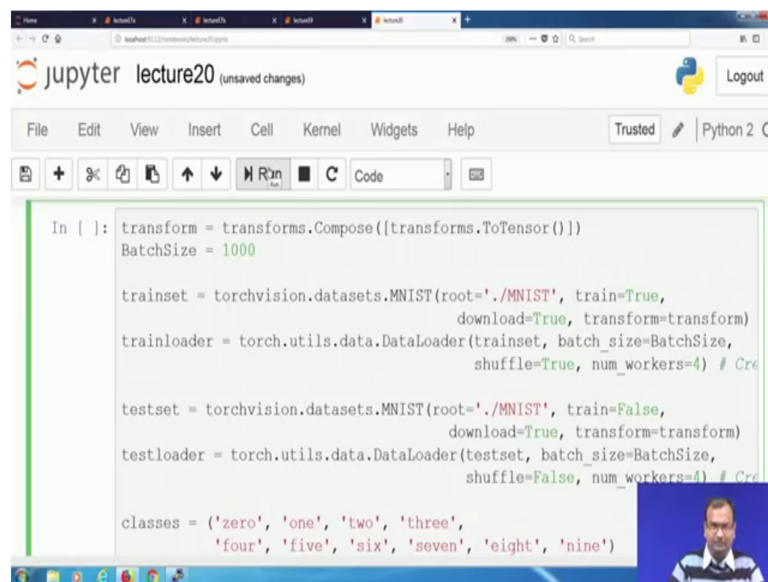


(Refer Slide Time: 01:57)



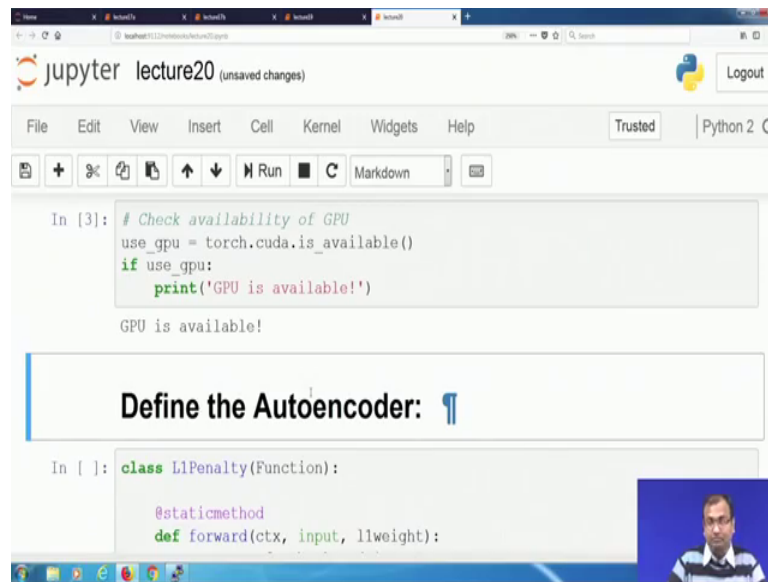
So, that you do not get confused between any of them and I am not reusing any of my. I am not going to ask you to reuse any of the codes, everything is a standard release.

(Refer Slide Time: 02:05)



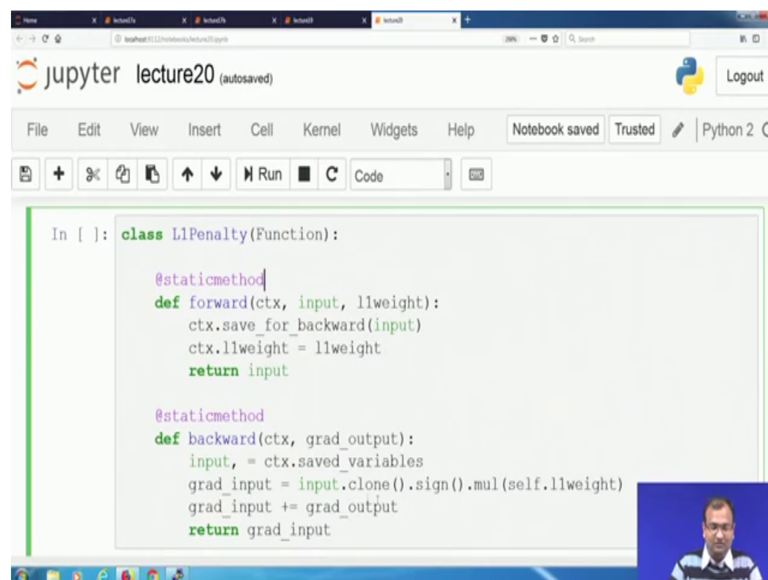
So, my header gets loaded. All the functions which I need to call then, I load down my data, set over here, my training and testing.

(Refer Slide Time: 02:10)



And then my gpu. So, by now, a lot of you guys might get bored around, because I keep on repeating the same thing and then yeah.

(Refer Slide Time: 02:19)

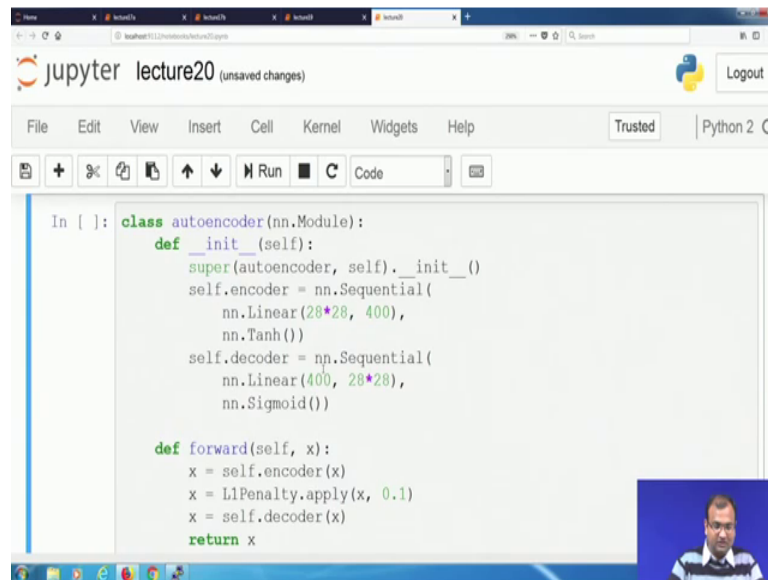


So, since you are watching a new video every day. So, let us keep on repeating the same thing. So, here comes down my first part, which was about the l1 penalty loss and the whole aspect was that I needed to have this extra weight, extra amount of loss incorporated in my whole loss and since my l1 penalty is not something which is predominantly defined in py torch, based on whatever releases we are recording with.

So, it is not yet defined, maybe in future releases or if you are viewing these videos, beyond the release of this mooc l1 penalty might come down as a standard release and you can use that as well.

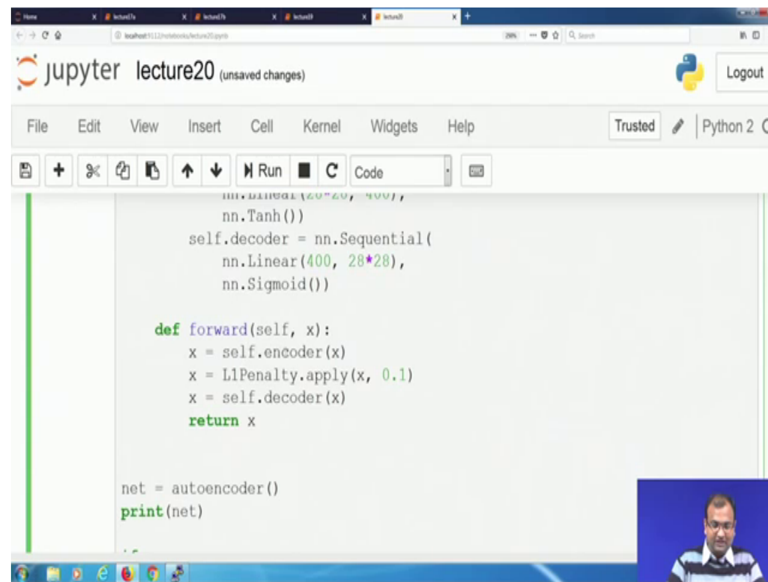
Now, till then what we do is, we have two parts which we need to define, one is the forward pass over the network and other is the backward pass over the network and these two methods have been explained in the earlier class and I am just going to reuse them.

(Refer Slide Time: 03:11)



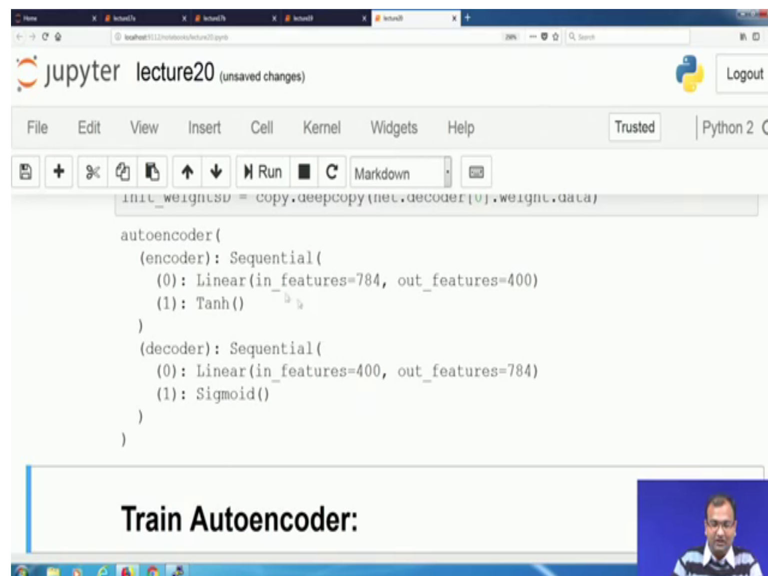
And then not go on to any further. Now, here comes down my auto encoder definition over here. So, what I do is my network, is defined as 784 neurons to 400 neurons and then 400 neurons 1 to 784 neurons.

(Refer Slide Time: 03:26)



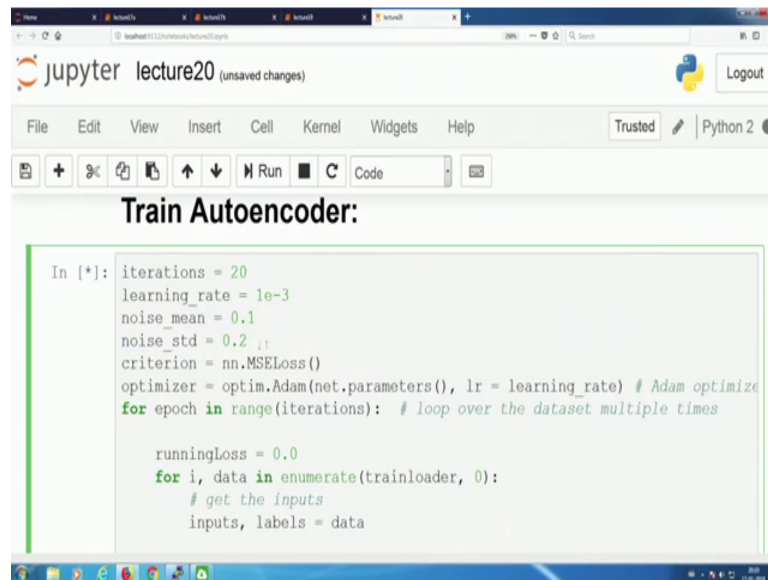
Forward pass of the data is defined as a forward pass over the encoder, then an L1 penalty with a row factor or the penalty factor or target sparsity factor, actually target sparsity factor set down to 10 percent or 0.1 and then you have your decoder coming down and then it returns down the output over there and then I print down my network, I just do a copy of all the weights available over there and this is what my final network looks like.

(Refer Slide Time: 03:54)

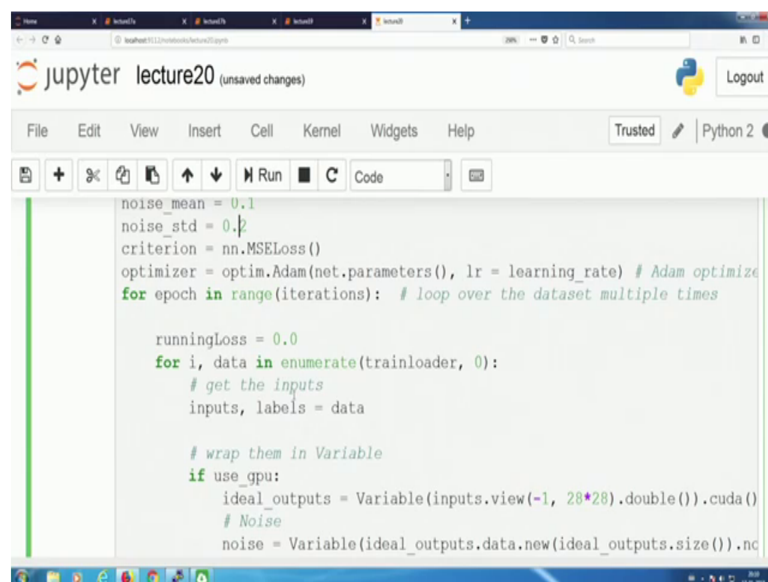


And since the L1 penalty over there is just a functional implementation. It is not a pretty layer, which comes down on the architecture that does not have any impact in terms of how this architecture looks like and how it is defined so.

(Refer Slide Time: 04:07)



(Refer Slide Time: 04:12)



Now, I will have to start by training my auto encoder. Now, here comes an interesting aspect one is we shifted over from the vanilla gradient descent on to an Adam optimizer. So, I did have used it in the earlier case as well, but I said that let us just hold on to a later on lecture, where I speak more details about optimizers and there is where I will be

explaining you what Adam is, but today the reason we took it out, because other than using an Adam optimizer, it becomes really tough with just a simple gradient decent, in order to show you the efficacy of denoising. And in fact, there is a good amount of relationship between why a batch learning with a denoising criteria works out really good with an Adam optimizer, as compared to a gradient descent optimizer.

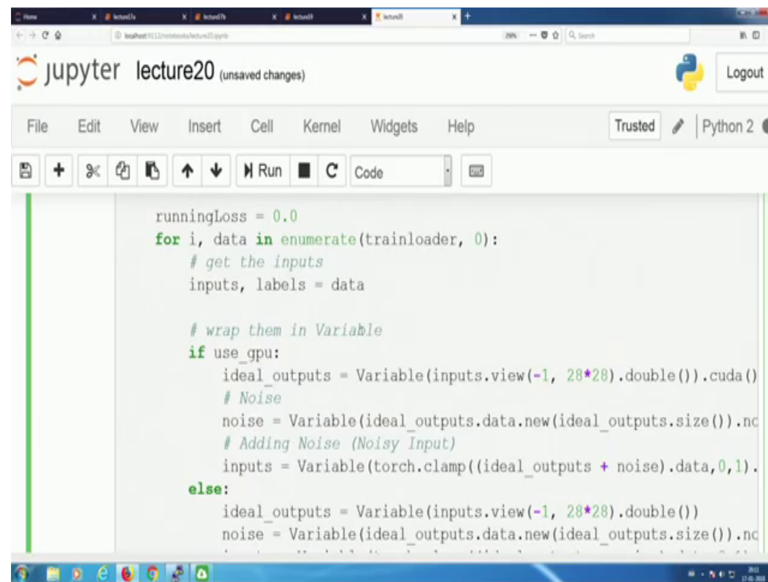
So, we come down to them in the later on lectures, when I would be going more details into the theory and linking it forward over there. Now, on the start part over there you see that I am going to iterate it over 20 iteration. So, let us just keep this one running and I can do the rest of the speaking and that time. So, if you guys are also watching this and at the same time executing the code and we just keep on set the whole thing to run and by the time I finish it off, you can actually see the results on your side of pc as well my learning rates. Over here, I changed down to 10 power of minus 3 and that is quite consistent with the kind of an optimizer, which I am using over here.

Now, what I need to do in my denoising is that you remember that for within an epoch, whenever I am taking a sample, I add some noise on to that sample and that is what is given to the network as an input, the version which is coming out of the network is compared down with the unknown or the original version of the sample together and then I get down my MSELoss. So, I need to define the property of noise and what I choose to do is just add standard white Gaussian noise, which is a drawn from a distribution from a Gaussian distribution, which has a mean of 0.1 and standard deviation of 0.2.

You have your standard relationship between your noise power to the mean and standard deviation of my. So, you can use any of your preferable, whatever you want to use your mean and standard deviation and still keep on doing except for keep one thing in mind that, the moment you start changing these noise powers over larger ranges and there are drifts and since your original images in a dynamic range of a 0 to 1. So, try to keep down your noise also in the similar dynamic range and do not just scatter out and come down to a much higher, change your value and that is why you do not keep a 0 mean, but we have kept a non zero mean a higher value over there of 0.1.

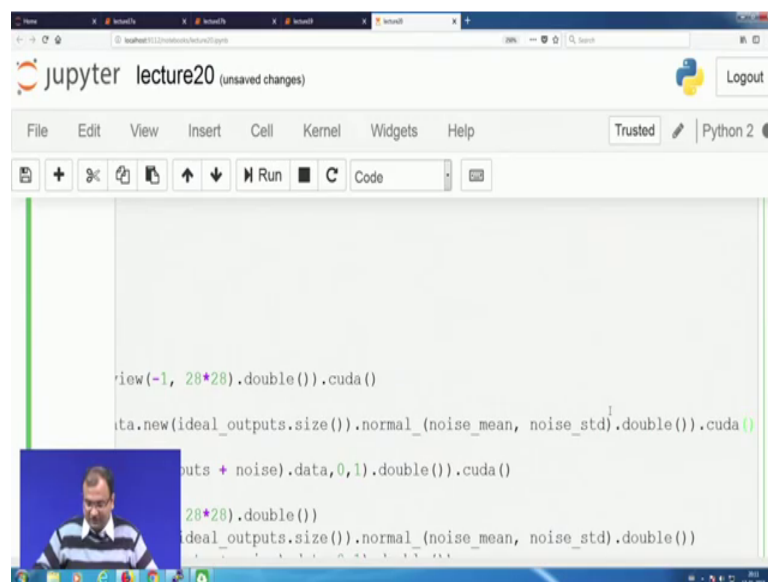
So, now within each epoch, as it keeps on ranging. So, what I do is,

(Refer Slide Time: 06:50)



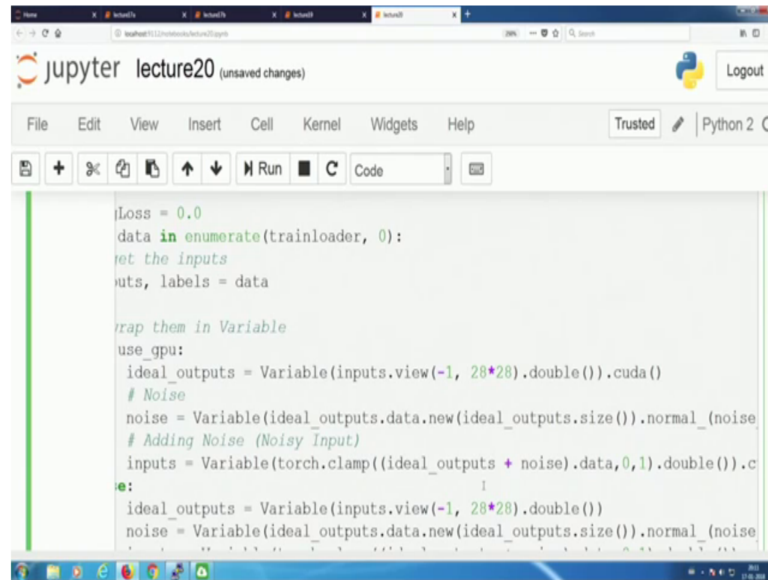
I will have to add the, some noise coming down over here. So, what we do is we create these noisy, very noisy and data over there and again keeping in the same tandem as we were.

(Refer Slide Time: 07:10)



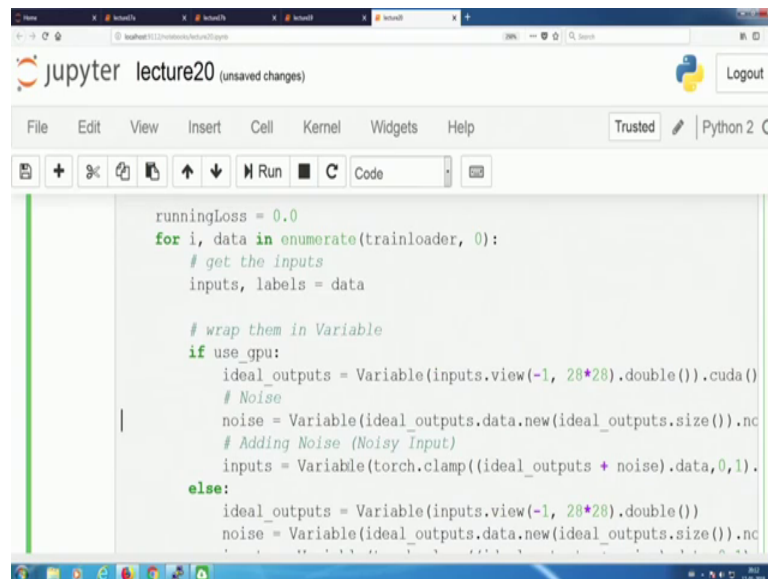
Now, if you see over here, what I am doing is basically, I am drawing down from a normal distribution with a particular mean and a variance; I convert that onto a double data type or double precision number system point over there.

(Refer Slide Time: 07:21)

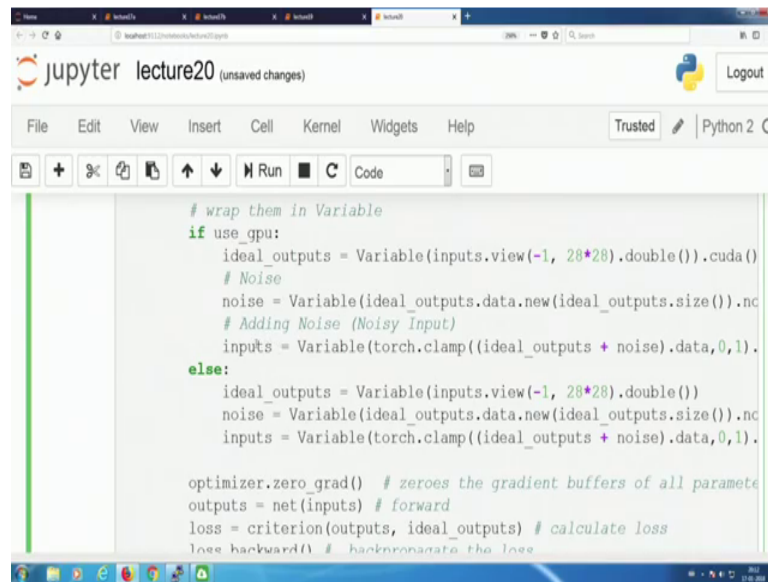


And then what I do is, I have that one, made into the same size as that of my input and that is also converted onto my standard container called as a variable. So, that is what we were doing down, in any of our examples, which we have been doing with pi torch to make it into a variable, otherwise it does not workout, it is just a static pointer.

(Refer Slide Time: 07:41)

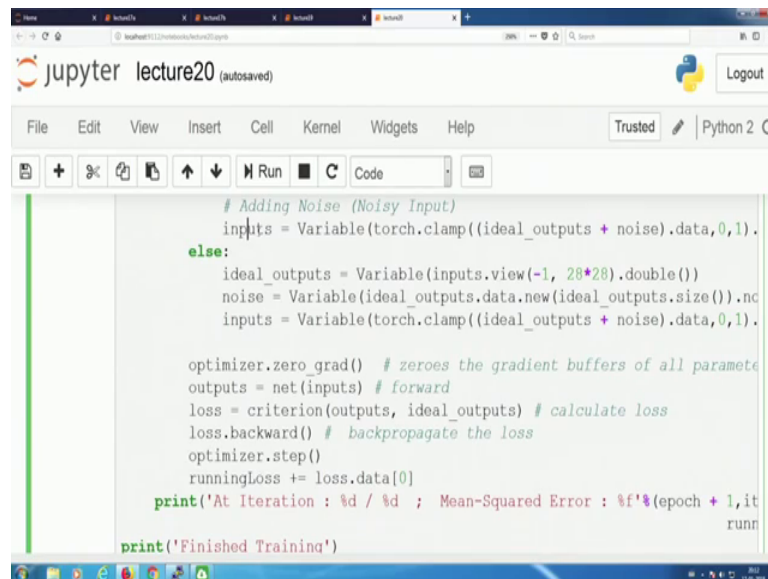


(Refer Slide Time: 07:48)



Now, what I do is that my inputs are modified as my outputs plus my noise. So, output is or what I call as ideal output is the unnoisy version of my input. And now, what I choose to do is, I would add down the noise and make that as my input over there. So, this is how my x plus that noise eta. This is getting generated over here as my input.

(Refer Slide Time: 08:15)

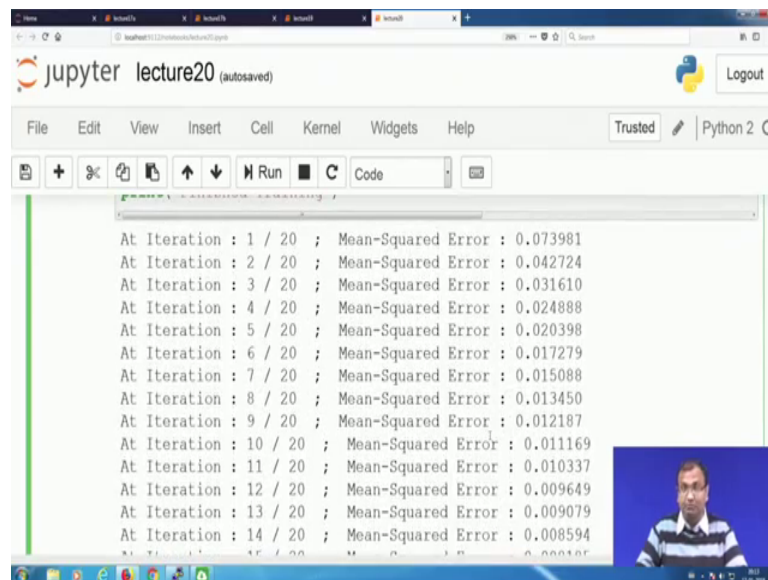


Now, what I do is, I have my optimizers gradients all 0 down. So, that I do not have any issues coming down and then I take my outputs coming down from my network. Now, you would see that what I do is on my loss function, which is my l2 norm or MSELoss

function, my difference is being taken from my outputs, which are produced by the network and the ideal outputs which are expected. So, this ideal output was the un noisy version. So, you had this ideal output, which was actually the input variable, which was given down over there whereas, your input is the noise added version over this ideal output. So, that is x plus noise, which is over there.

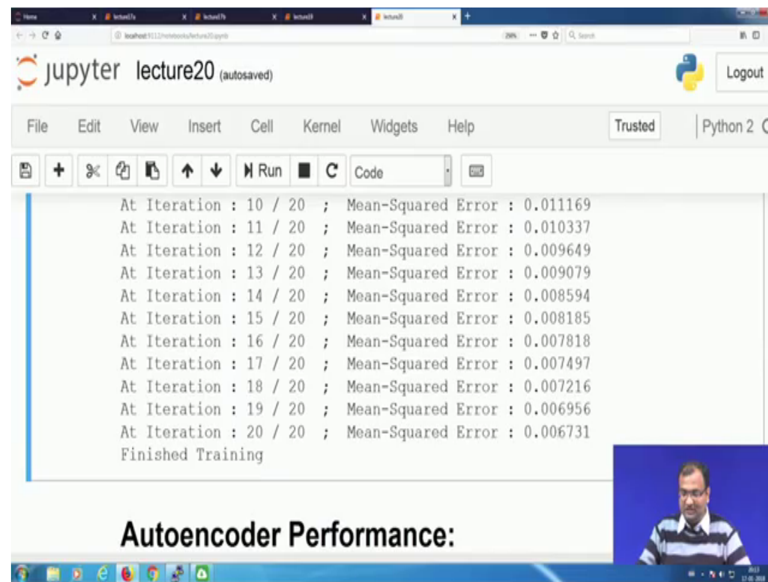
And then with this coming down over here, I have my derivative over the loss computed and then I set my optimizer running down. So, the whole update rules and everything is what happens within my, within 1 unit step operation of my optimizer and then I just look into my loss functions. So, this is typically, how you see the loss going down and typically you see that,

(Refer Slide Time: 09:18)



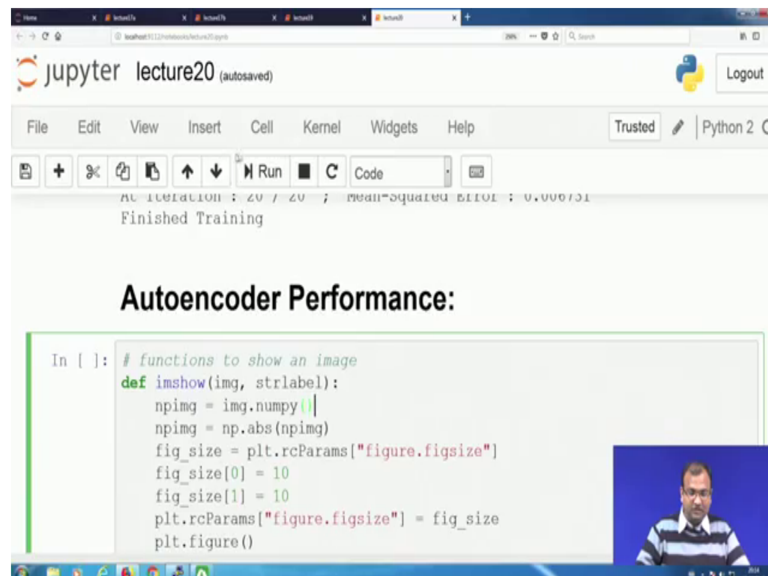
There has been a significant drop in my mean square error as compared to when I was using just a gradient descent. In fact, I have even not even able to reach down the third The decimal point over there, in terms of MSE when we are just using a gradient descent whereas, here.

(Refer Slide Time: 09:27)



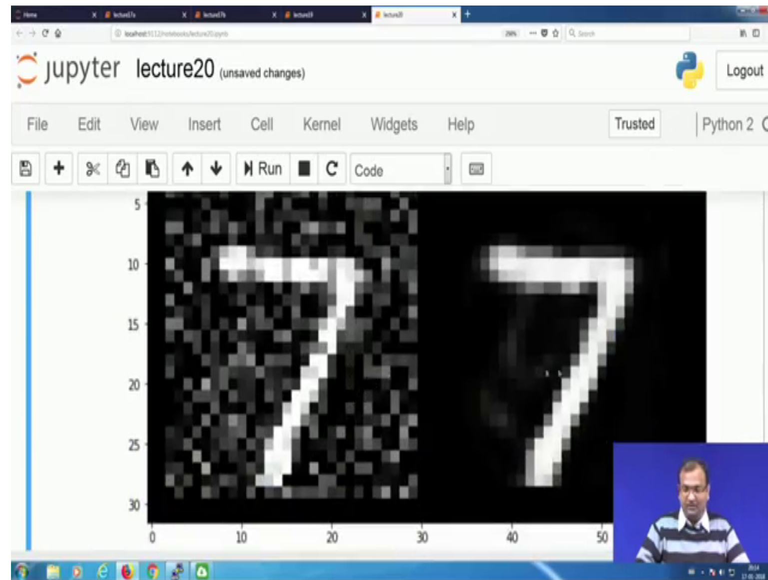
So, one is you have an Adam optimizer, which is running down. The next part is you, also have a denoising attribute given, done, which is making the whole system learn down, much robust features as compared to what it was learning when this noise was not there and that is one of the findings, which we had read in the paper as well.

(Refer Slide Time: 09:52)



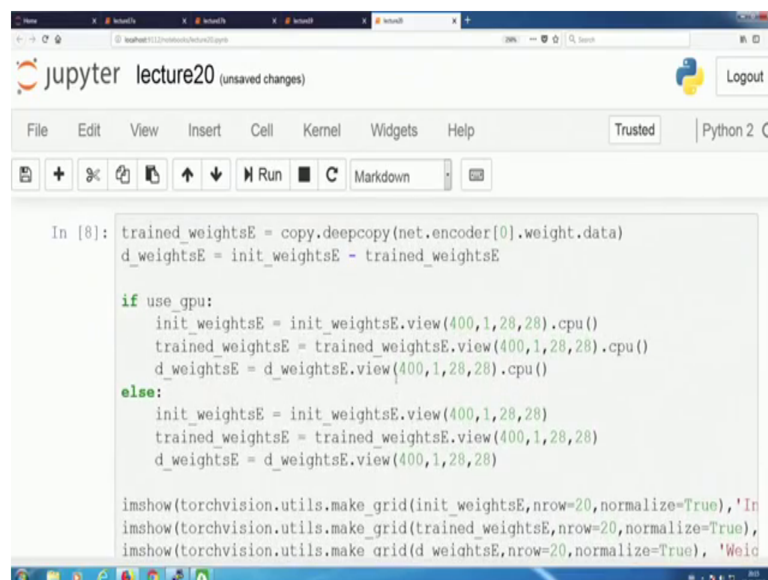
So, now what I would like to do is just look into the performance of my auto encoder. So, say that this is where the kind of a noisy input, which was given down.

(Refer Slide Time: 09:58)



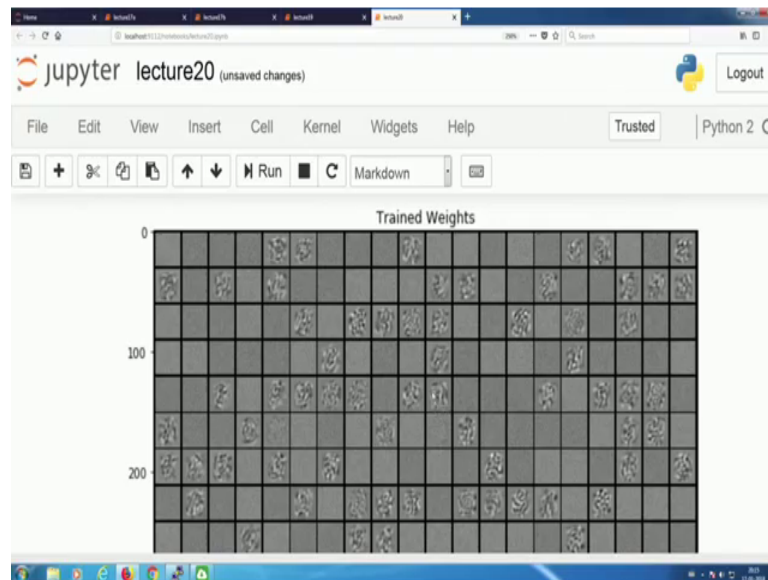
This is what it gets reconstructed over here. So, sorry and yeah yeah. So, I have this output being created for this kind of an input, which is given done and you can see it pretty much, a lot of these noise is gone down. So, you can, one example is now, you know how to create a new electric in order to remove noise from your images provided and then you can change provided. You have like really unnoisy versions of your ideal inputs available to you and now, let us look into the visualizations for this one. So, it is the same way that we had the weights copied down.

(Refer Slide Time: 10:38)



Before the training and then after the training.

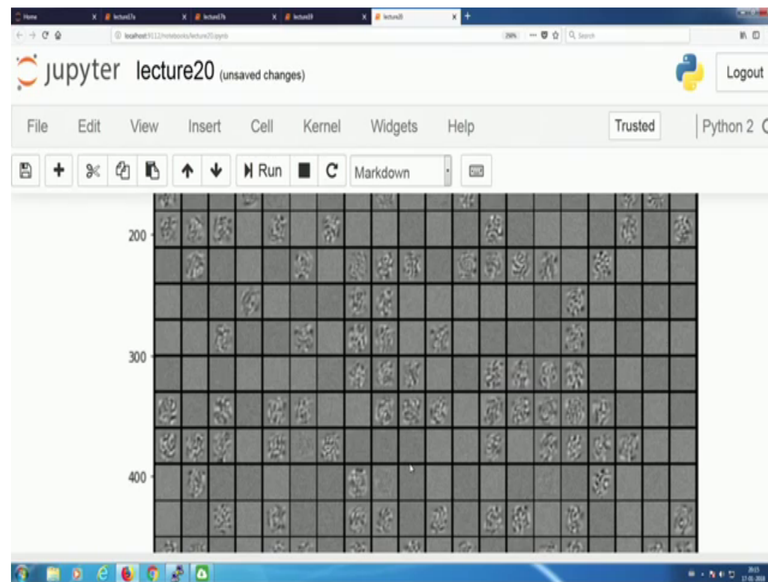
(Refer Slide Time: 10:42)



So, this were sort of your initial weights, which connect down 784 neurons onto your 200 onto your 400 neurons, the same logic that there are 784 or 28 cross 28 elements, which make up this weight matrix and you have 20 cross 20, such weight matrices or 400. And now, here is what you start looking into this train weights.

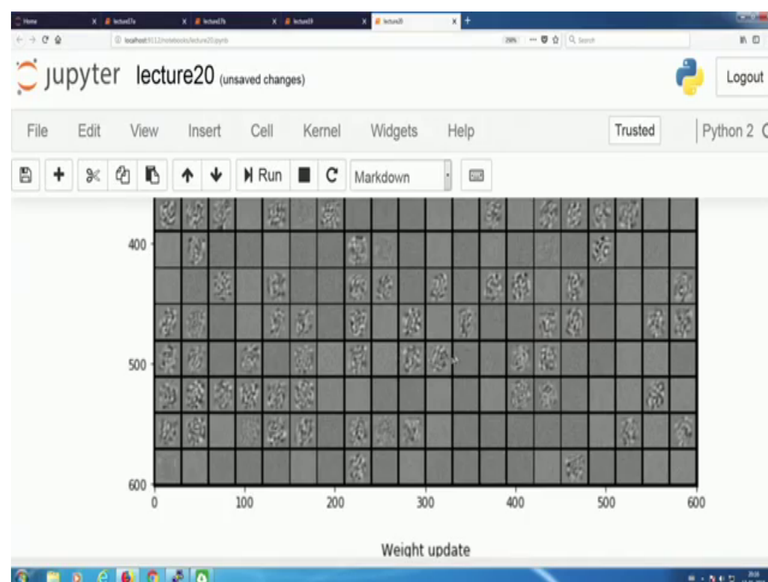
Now, clearly look, I mean these weights, which come down at the center, these kind of some patterns as if wave let us some jagged kind of wave let patterns, they are much more meaningful.

(Refer Slide Time: 11:16)



There are distinct features, which you see over here on, these kernels as come to what you were seeing down with a simple one, in the earlier case with just using sparsity or even without using sparsity.

(Refer Slide Time: 11:32)



So, there you could see that there were changes, when we were looking into the change matrix over here, which was this, the weight update which comes down, but when you are looking at trying to find out, if there is something significant to be understood from the weight matrix, you do not see though a lot of times, it these weight matrices might

not make any sense to be physically interpreted, but this is you need to keep something in mind that operating with these weight matrices is, what tries to boost up the features and give you all the features. So, it is not necessary that these kernels would have some physical significance meaning.

So, it is the same way as trying to look into a sobel kernel. I mean that might not necessarily give you any meaning coming out of it, but the moment you start convolving a sobel kernel with an image and then you see, you will either accentuate your gradients along the horizontal or the vertical direction, based on what kind of a kernel you are using. So, here also it is the same thing, based on this if you are doing a correlation operation and then your non-linear.

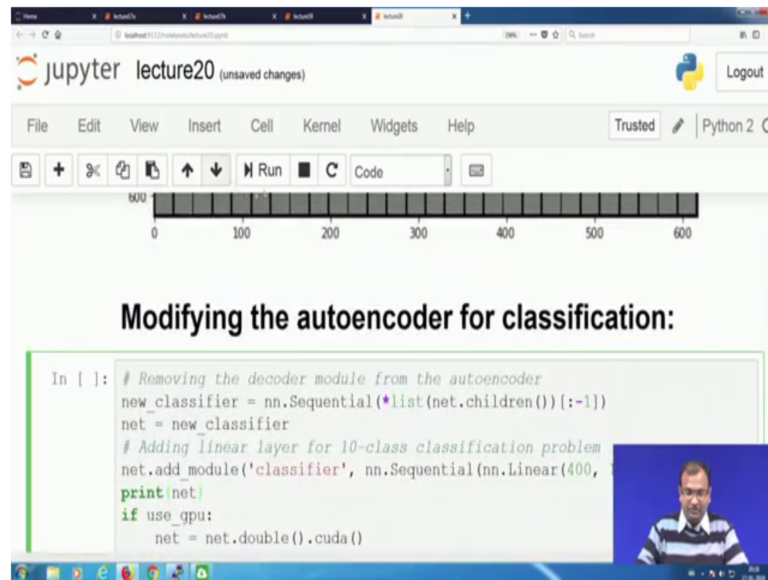
So, the whole thing which happens in the first layer, you would start to get down some meaningful outputs from there and then this brings us to the point that it was easier to learn down more significant and prominent bits, with noise incorporated. Now, also remember one thing that this is still a sparsity incorporated auto encoder. So, we also need to look down into the decoder side of it and really try to find out, if I am able to get down, the sparse matrix coming down.

So, this is what my initial bits are there, one which connects down my 400 neurons to 784 neurons. Now, after my training you see a majority of these actually turn up as 0 bits and whatever you see over here, is non zero bits. These also have a significant amount of 0s over there. So, there are just few dotted points, which are these connections of high negative or high positive value and rest everything is your value and this is exactly what you were trying to look at any point of time. So, you can save this image, zoom into it and really look into it or just play around with the code, you can extract each simple patch over there and look into it and a greater detail.

Now, this is what I was inferring about. So, once you have your sparsity incorporated over there and if you end up learning a much better over complete representation, in your earlier layer connecting down 784 neurons to 400 neurons in the subsequent next layer, where you have an L1 sparsity in position given down. You would be having a matrix, which has a majority of values which are 0s and that is what comes down over here as you learn down.

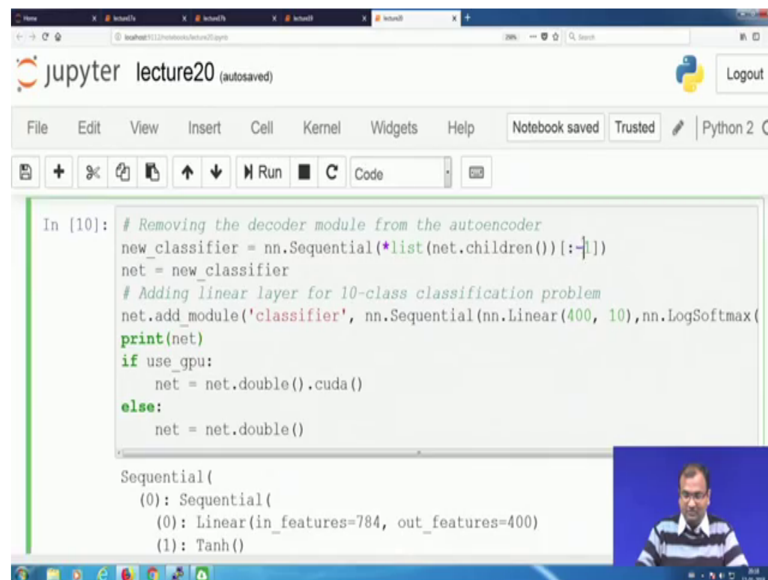
So, if you look into your weight updates, they will also be open down a similar kind of a form, because you had to bring down those very high values on to 0 values also. So, the updates are of the similar range. Now, comes the next part of it which is just to modifying your auto encoder, in order to do a classification.

(Refer Slide Time: 14:03)



Now, here what we do is, we chunk off the last layer.

(Refer Slide Time: 14:09)

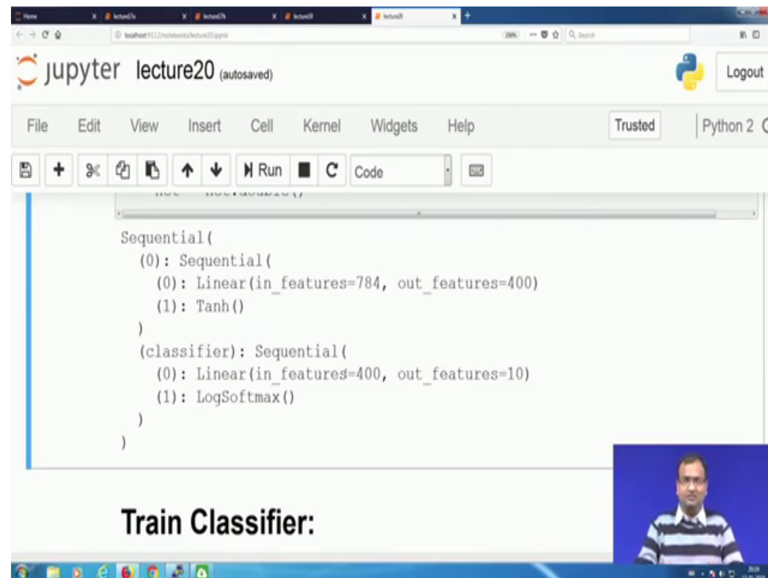


Just delete this last line, over there the last pointer, which was 400 neurons to 784 neurons and then connect down 400 neurons to 10 neurons and then I have a log soft

max for my classification output over there, if I have a gpu, it gets converted to a gpu and then this is my architecture.

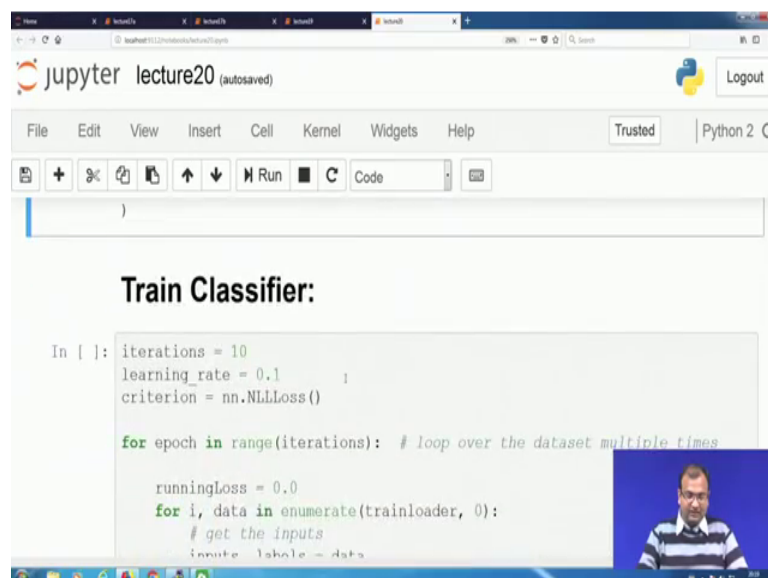
So, what I essentially did was no further changes.

(Refer Slide Time: 14:30)



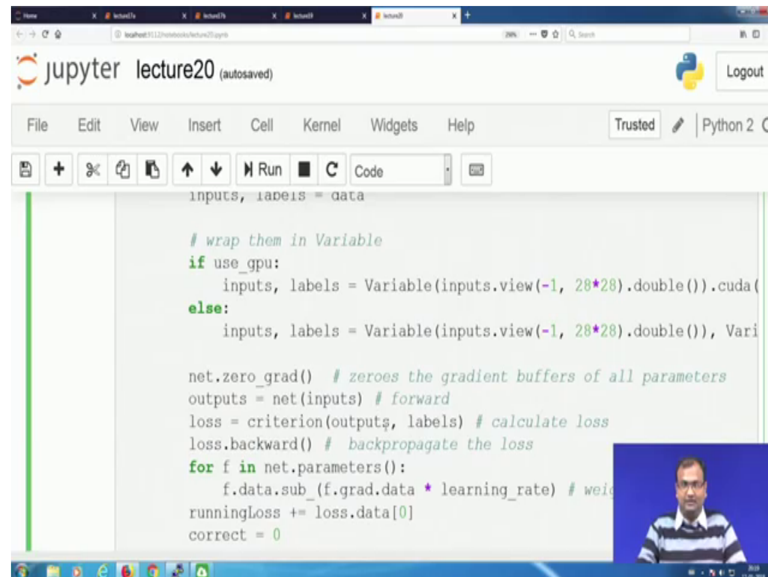
Anywhere over there, but keep one thing in mind that, while I am trading down a classifier

(Refer Slide Time: 14:33)



I no more would need to make use of the noisy data in any way right. So, and then the noise is of no use and. In fact, what I choose to do over here, is I even do away.

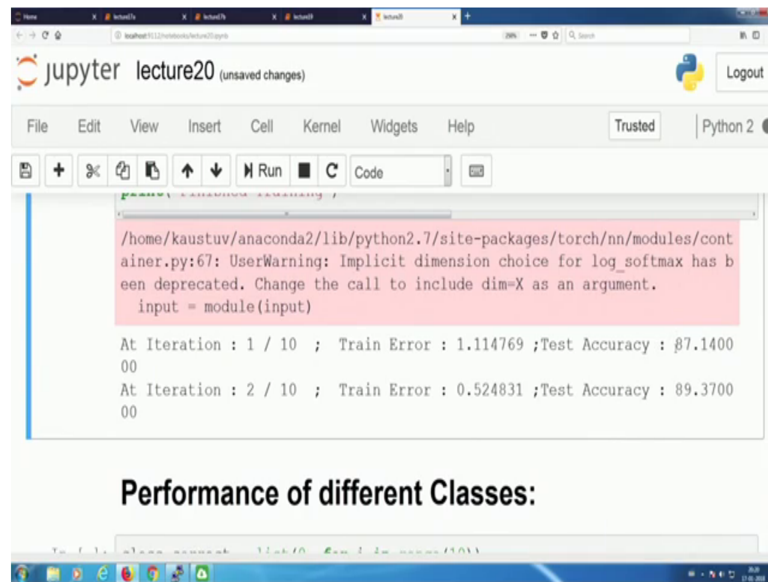
(Refer Slide Time: 14:40)



With my optimizer of an Adam, the Adam optimizer used and I can use a simple gradient descent, the vanilla gradient descent, in order to solve it out, because by now, I have my weights which I learned on, which are really good in shape and this is a newer network, whose forward pass is defined in a very different way. You do not even have the l on sparsity, coming down in this particular kind of a network as well you had all everything incorporated over there.

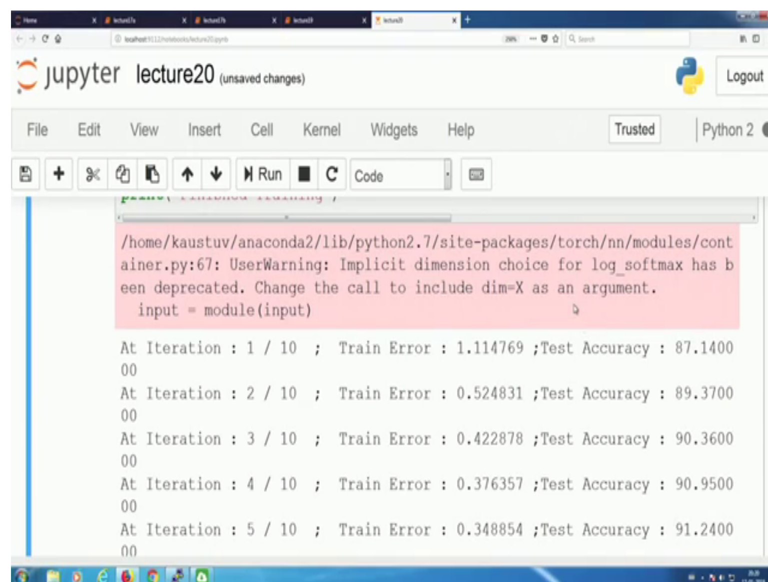
So, now let us run the classifier. So, I see this is what comes down as my final network of the classifier. Now, if I run, train start training of the classifier which I needed to do actually. So, let us wait for sometime yeah. So, you start somewhere at a training accuracy.

(Refer Slide Time: 15:35)



Now remember, in the earlier case we trained it down for 10 epochs and over 10 epochs you went down to an accuracy of something.

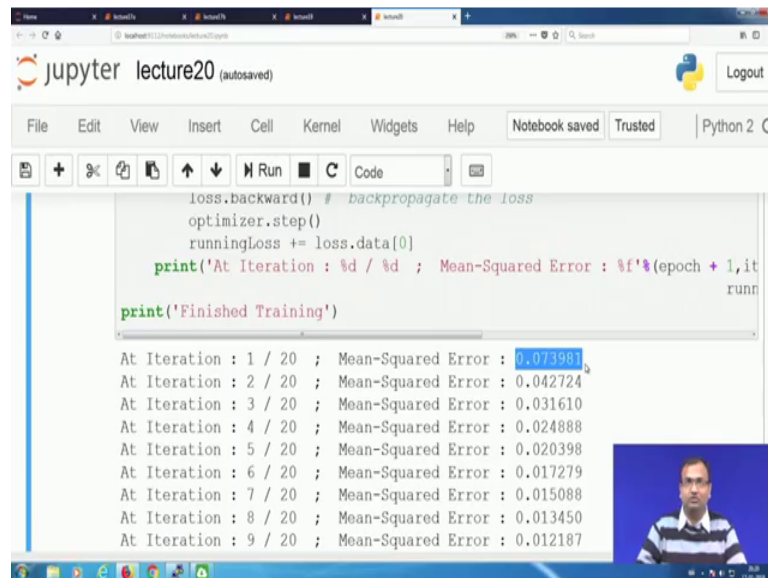
(Refer Slide Time: 15:42)



91.7 percent or something over 20 epochs, you just barely cross down 92 percent whereas, here when we start with this one, you start with the training accuracy of 87 percent and this is also quite concurrent that, while we were just training down the auto encoder for denoising purposes.

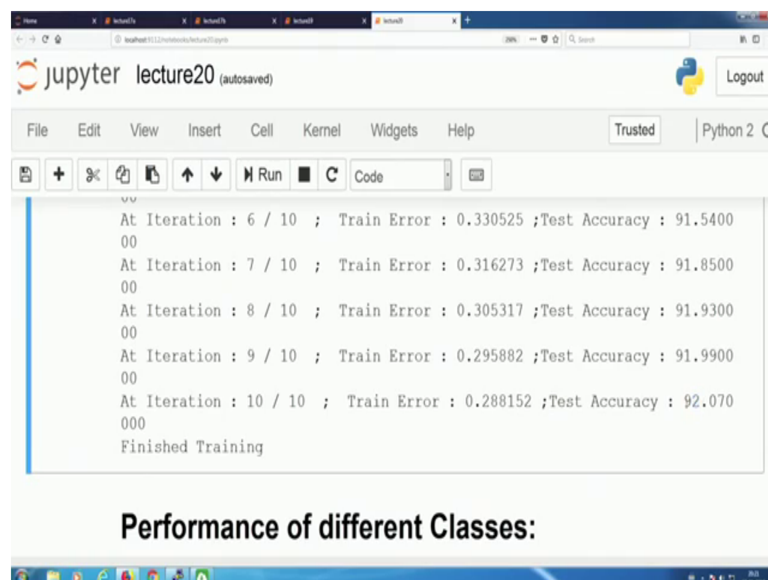
So, let us get down over here.

(Refer Slide Time: 16:07)



So, your starting error, MSE error was also quite low as well as your MSE error at the end was really low. So, as network which has been able to really reconstruct itself, whatever data is given down in a really good way that is a network which has learned to actually learn down, features in a real good way and that is what we see, when we come down to this classification. So, the starting point of my classification is what starts with an accuracy of 87.1 percent and then we go down and then quite within the 10th epoch, we come down to an accuracy of 92 percent.

(Refer Slide Time: 16:35)



Now, one thing you might ask, is my saturation accuracy does not change in any way and that is like really something to do around with the whole aspect of statistical machine learning, because your saturation accuracy is basically, the upper bound of how good you will be able to classify and you cannot just break that. It is a function of the data, which you have, it is a function of the amount of training samples you have, it is a function of the kind of a architecture or the way you are actually trying to model down a discriminator, it is a function which is also dependent on what is the kind of a loss function you are using, it is a cumulative effect of all of them which come down and what we essentially have played down is just one single aspect.

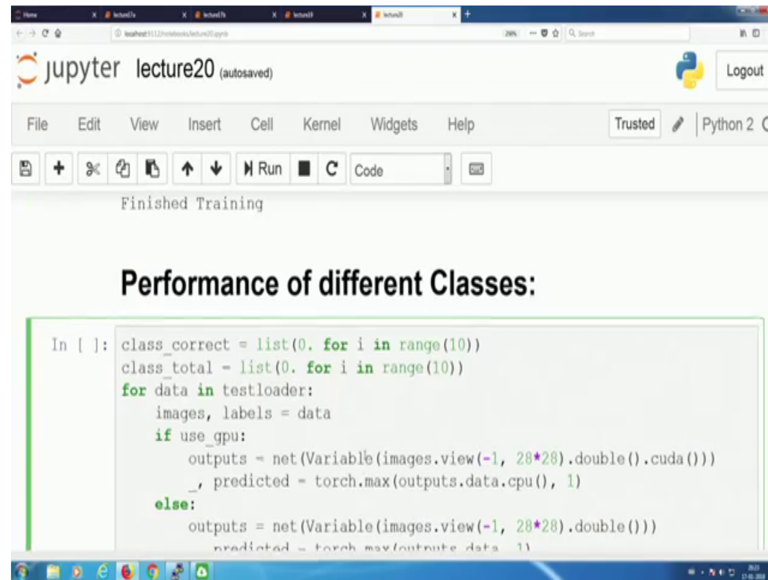
So, I have not increased my training data although, I was putting down noise, but that is just small jitters around the training data. There has been a significant increase in my training data as such I have not changed my architecture. I have not changed the way I am evaluating a network or helping it learn, which is my cost functions.

The only thing which has been changed are slight modifications to my cost function or the data, which comes down. Now, that given. Having said that the only point, where it has a significant impact is the starting point and then that it has the lead in the whole race to finishing it off the final point of the race is still the same, you can keep this running for a longer duration of time change out your learning rates, you can still see.

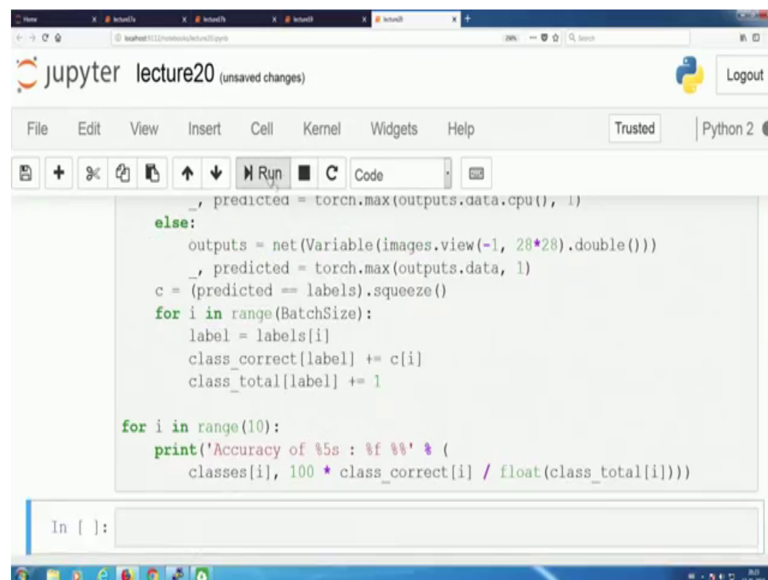
But we have our experiences. So, there is something which I will come down in a later lecture, which is with actually trying to look into your learning rules and there we will be introducing something called as a weight decay factor. So, your learning rates over, then we will start on decaying, then they will change after a fixed number of epochs and that will actually help you come down very close to the saddle. So, you remember your saddling aspect and in the cost function plane. So, it comes down to the global minima, but then since the gradient is really high, on account of this learning rate eta. So, it keeps on oscillating around over there.

Now, if I keep on changing this oscillation factor, I keep on reducing my learning rate. So, it would gradually oscillate and come down to the global minimum point and that is what we will be doing, but that comes down slightly in a later on lecture. So, let us look Into what happened with the performance in a per class basis.

(Refer Slide Time: 18:52)

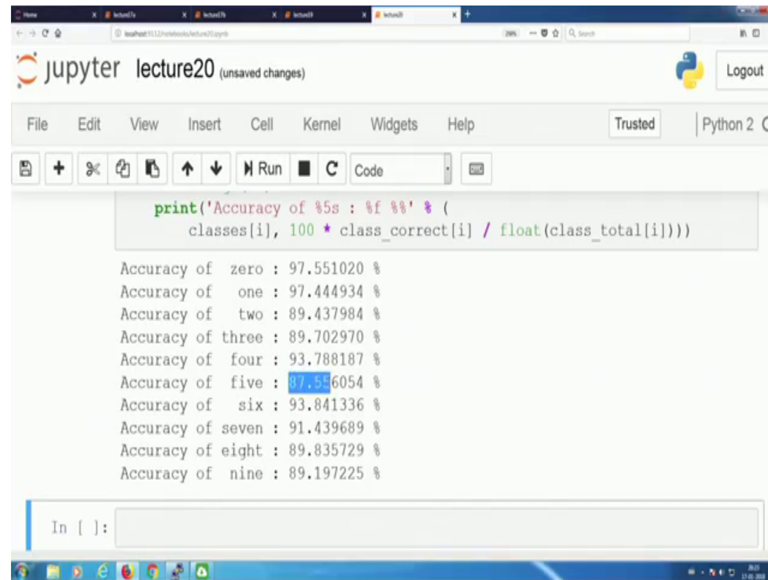


(Refer Slide Time: 18:54)



So, the last number 5.

(Refer Slide Time: 18:58)



Yes, nothing increased significantly, but we do see that class number 0 and 1, they come down to a point of 97 percent, which was really high. So, in the earlier case, we did not see that high result going down to 97 percent. We had something close to around in case of class 0. It was about 97 percent, but then in class 1, it was still restricted to about 90 to 193 percent. So, these are changes which come down by introducing that extra noisy factor, which helped it learn down.

So, that brings us to a point that noise at certain aspects is also good. It is not necessary, that you need to always remove out noise and in fact, for a lot of these applications, as was in the paper and then what we had studied in the earlier lectures and today's example you do see that adding noise actually create. So, much robust system and a much better way of learning down these weights and representations. So, that is all what we have for lecture 20 and this whole family of water encoders, which we had done. So, that would bring us to a concrete conclusion of this first four weeks and the first month of the class. In the next month onwards, we are going to start with the next version, which is called as convolutional neural networks and then that is the next generation of neural networks, which come down, which do not necessarily have a connection, which is called as a fully connected, but we are going to play around with how neurons are connected to the inputs in a very different way.

So, till then stay tuned and wait and watch for the exciting ones in the coming weeks as well.