Deep Learning
Prof. Prabir Kumar Biswas
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 31
Autoencoder Training

Hello, welcome to the NPTEL online certification course on Deep Learning. In previous few lectures we were discussing about the autoencoders. And, we have seen that autoencoder is an algorithm, which tries to learn a compressed domain representation of the input data or it tries to learn the structure present in the input data.

And, in order to do that what you do is given an input data, you pass it through a neural network which is; obviously, a multi-layer neural network, where you try to reconstruct, whatever input you are feeding the same signal the same data you try to reconstruct at the output, but while this information the data passes from the input to the output layer, it passes through one or more hidden layers.

So, in the basic configuration we have seen that an autoencoder consists of one hidden layer, where the number of nodes in the hidden layer is much less than the number of nodes in the input layer. That is a configuration which is known as under complete autoencoder. So, as it passes through a hidden layer, which is also known as bottleneck layer, the information which while it passes through the hidden layer having a lesser number of nodes than the input layer, the network learns a compressed domain representation.

What will happen if the hidden layer contains the same number of nodes as the input layer or the number of nodes, which is even larger than the input layer? Later on of course, we will see other configurations of the autoencoder where such thing is also possible, but in such cases the autoencoder will learn a compressed domain representation, by intuitive (Refer Time: 02:34) some constraint, which we will term as sparsity constant, we will come to that later.

But, so far what we have discussed is what is known as under complete autoencoder. So, there if we assume that your number of nodes in the hidden layer is same as the number of nodes in the input layer and; obviously, the same as number of nodes in the output

layer. And, or even more than that, in that case it is possible that autoencoder will simply try to memorize the input data. And, the function that it will learn is a simply identity function, whereby wherever is there at the input the same will be reproduced at the output.

But, by putting a hidden layer having limited number of nodes known as bottleneck layer as we have just said, the network is forced to learn a compressed domain representation or the network is forced to learn the salient features, which are present in the input data. And, that is the purpose of an autoencoder reconstruction of the input signal is not the purpose of autoencoder. But the purpose of autoencoder is that it learns a compressed domain representation, or it learns the salient features in the input data, and using this salient features the decoder side should be able to reconstruct your original data.

So, we are actually interested when it comes to the application of autoencoder will come to that a bit later, we are actually interested in the output of the hidden layer or the bottleneck layer, which is a compressed domain representation. So, we have also seen that as we are talking about the compressed domain representation, this is also nothing, but what is known as dimensionality reduction.

So, if your number of nodes in the hidden layer is much less than the number of nodes in the input layer, then in the compressed domain representation the dimensionality of the latent variable, which is mapped from the input data the dimensionality of the latent space data is much less than the dimensionality of the input data.

But, still the decoder should be able to reconstruct the input from that reduced data; so, this is what is dimensionality reduction. And, we have also discussed in our previous lecture, that when you talk about dimensionality reduction traditionally a very popular method for dimensionality reduction is what is known as principal component analysis. That is the input data is projected onto the eigenvectors into the Eigen space. And, if the input data is projected into the eigenvectors; obviously, if you have a set of data which is of dimension d, the number of eigenvectors will be d number of eigenvectors. But, for every eigenvector there will be a corresponding eigenvalue.

So, when you form the transformation matrix, you form the transformation matrix using the eigenvectors as rows in the transformation matrix. And, when you form this transformation matrix, as we have seen in your previous lecture that the first row in the

transformation matrix will be the eigenvector corresponding to the maximum eigenvalue. And, the last row in the transformation matrix will be the eigenvector corresponding to minimum eigenvalue. And in between all that rows are formed by eigenvectors arranged in descending order of the corresponding eigenvalues.

So, now for transformation purpose if we retain only few number of rows in the transformation matrix from the top. So, we retain only one row. In that case your d dimensional data is transformed into a one-dimensional data, which is nothing, but projection onto the eigenvector having my maximum eigenvalue.

If, we return only two eigenvectors, then the d dimensional input data will be transformed into two-dimensional principal components in Eigen space. And, we have also seen through reconstruction with examples of reconstruction that the power of such principal components. So, we have seen with examples that even with one principal component, it is possible to reconstruct most of the information present in the input data. And, there we have compared the principal component analysis PCA with autoencoders.

And, we have seen that in case of autoencoder, if we do not impose any non-linearity in the neural; in the neural network, then your principal components and the autoencoder outputs, they almost coincide. But, we have also discussed that autoencoders are much more powerful than principal components, because of the presence, because the neural; neurons in the neural network are capable of imposing or implementing non-linear functions.

So, as principal component analysis is a linear transformation from the input space to the eigen space, autoencoders can give you a non-linear transformation. So, as because autoencoders can give us non-linear transformation, we can even represent non-linear manifolds using the autoencoders, which is not possible using principal components. So, you have seen that if we do not impose the non-linearity in the neurons, or if the activation function of the neurons are linear, then autoencoder and principal component analysis they become almost identical.

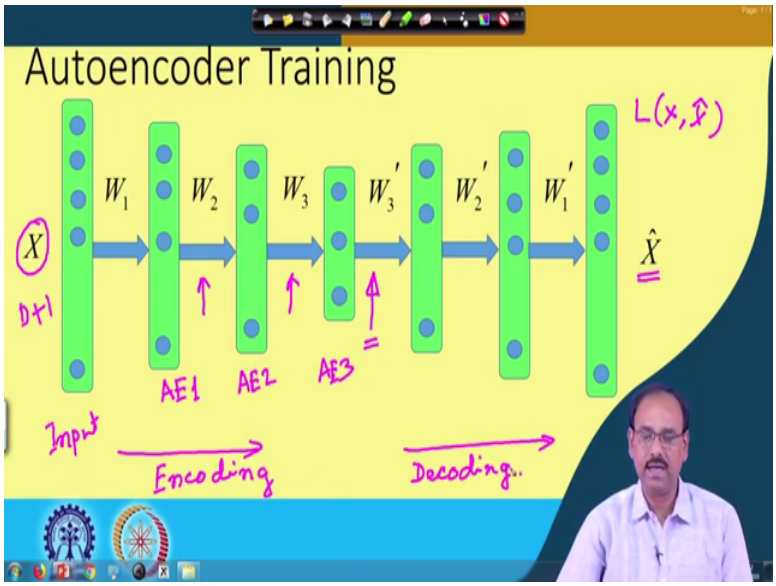
However, because of the presence of non-linearity autoencoder gives us much better representation in lower dimensional space than in case of principal components. So, that is what we have discussed in our previous lectures.

(Refer Slide Time: 09:12)

So, today what we are going to discuss about is how do you train a deep autoencoder. So, we are talking about deep autoencoder training, then subsequently we will talk about other versions of autoencoders like sparse autoencoder, denoising autoencoder, contractive autoencoder and so on. And, after few lectures we will also talk about convolution autoencoder, but I will come to that topic after we discuss about convolution and convolution neural network.

So, let us start with how do we learn a deep autoencoder. So, as we told before that in case of deep autoencoder an instead of having only one hidden layer, we have stacks of hidden layers placed one after another. So, that is what is nothing but, stacking of different autoencoder layers. And, the depth of the network depends upon, how many such autoencoder layers, you have in your auto encoded neural network.

(Refer Slide Time: 10:25)



So, a typical figure of a deep autoencoder is something like this. So, here you find that we have our, this input layer; so, this is the layer which is input layer. So, you have feeding your input data X to this input layer. So; obviously, the number of nodes in the input layer we have also told that before is same as the number of elements in vector X plus 1, why that additional 1, because we also wants to incorporate the bias term in the same layer.

So, if the dimensionality of the input vector X is D number of nodes in the input layer will be D plus 1 so, this is what we have discussed before. And, then you find that we have a number of autoencoders. So, I can call this one as say the first autoencoder; autoencoder 1, this is the autoencoder 2, this is autoencoder 3 and so on. And, here in this configuration I get the coded output or the reduced dimensional representation, which is also known as latent space representation at the output of autoencoder 3.

Of course, each of these autoencoders will give a reduced dimensional representation both the same input, but at different levels. So, output of autoencoder 1, what you get here is also a coded version of input vector X, output of autoencoder 2 is also a coded version of input vector X; here this is also a coded version of the input vector X, right. So, this is where we get maximal dimensionality reduction.

And, once I have this coded output, the coded output is decoded by this decoding layers, again I have a number of decoding layers to get my reconstruction or the reconstructed

signal exact. Same case of autoencoder what he said is we try to reduce the error between X and X hat or the loss function in this case is L X, X hat, which we have also said that the loss function that can be defined is sum of squared error. That is, what is the error in the construction of X hat, when you compare that with input X. And, training of the autoencoder or deep autoencoder we will try to reduce this error the sum of squared error to a minimum value.

So, you find that in case of autoencoder we have also said before, that I have an encoding part so, this is a portion which is encoding and this is a part of the autoencoder which is decoding. So, I have an encoding portion and I have a decoding portion and these two taken together forms an autoencoder. And, the depth or the number of layers that you have auto, that you have within this on autoencoder that tells you that what is the power of the autoencoder, ok.

Now, so, given this that as we said that for training such autoencoder I want to minimize the loss or minimize the error between the input vector X and the reconstructed vector X hat. So, what I want ideally is the reconstructed X hat should be identical with my input vector X. And, that has to be done using the back propagation learning algorithm that we have talked about earlier.

Now, you find that the number of weight vectors or number of weight matrices or the number of elements; weight elements, that you have to determine by training of the autoencoder is tremendous. So, I have weight matrix W 1, I have weight matrix W 2, I have weight matrix W 3, on the encoder side on the decoder side I have weight matrix W 3 hash, W 2 hash and W 1 hash. And, the number of weight matrices will go on increasing with the depth of the autoencoder that is the deeper the network is the number of such weight matrices will go on increasing.

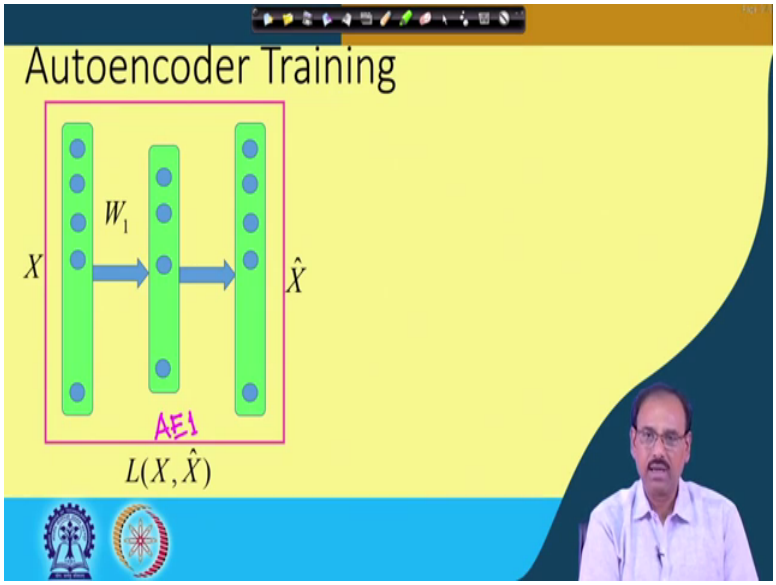
So, if I try to train this entire autoencoder end-to-end that is given the full autoencoder architecture, I have the set of training vectors given to the input, I have X hat as the output and by reducing the error between X and X hat, if I try to train this autoencoder. Then, I have to deal with so many weight matrices simultaneously.

And, that leads to a problem, one is obviously, the memory problem that I have to save so, many weight elements into the memory. The other problems are that if the weights that you have is close to the solution weights the convergence of the learning algorithm

is quite easy, but if the weight elements are too large then finding a global minimum becomes a difficulty. And, on the other hand if the weight elements because initially all of them are chosen at random so, if the weight elements becomes too low in that case the convergence of the learning algorithm becomes very slow.

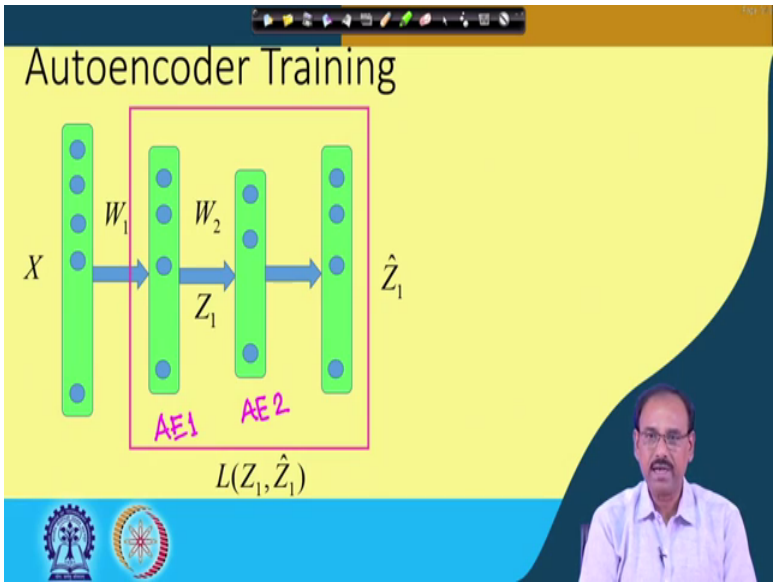
So, to avoid this problem the kind of approach for training or deep autoencoder can be that you go for a stage called pre-training, which will be finally, refined with end to end training mechanism, but before going for that to reduce your complexity. You go for pre training, and this pre training is done layer by layer; that means, while pre training you deal with lesser number of weight matrices. So, let us see that how we; how it is actually done?

(Refer Slide Time: 16:58)



So, we are talking about the layer by layer pre training mechanism. So, while doing this you I take only one layer of autoencoder at a time. So, as before I have this input layer, where the input vector X is applied and I take initially only one autoencoder say autoencoder layer 1. And, then I put a decoder, which decodes the coded output from autoencoder 1. So, because I have only one layer this decoder output should be same as reconstructed X so, I call it X hat. And while training this first autoencoder or while trying to determine what will be the value of W 1, you go for back propagation. And, this back propagation will try to minimize the error between X and X hash; X and X hat and while doing so, it learns the weight vectors W 1.

(Refer Slide Time: 18:12)

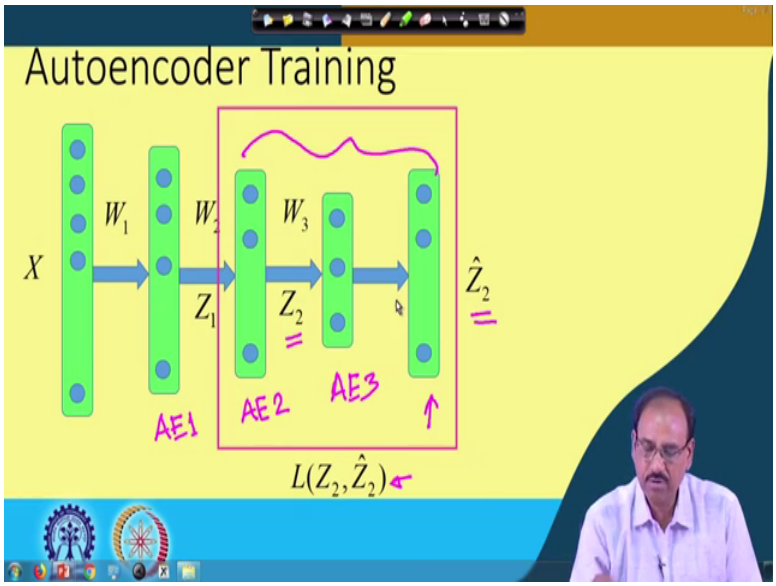


And, once this weight vector W 1 is learned then you discard the first level of decoder that we have used and discarding this now I put a second autoencoder. Now, before that you find that is a output of the autoencoder, which encodes X is say a latent space vectors Z 1. So, the next level of autoencoder, we will try to encode this Z 1. So, I put the second autoencoder and let me call this second autoencoder as AE 1, AE 2 sorry.

So, this AE 2 we will try to encode Z 1 and while doing so, it will try to learn the weight W 2 or weight matrix W 2. So, to train AE 2, now what I do is I put another decoder of course, these decoders I am putting, they are all intermediate finally these decoders will not be there.

So, I put another decoder and this decoder tries to decode the output of autoencoder 2; that means, it will try to reconstruct Z 1, which is input to autoencoder 2 and I call this Z 1 hat. And, for this learning, the back propagation learning it will consider only the autoencoder 1, autoencoder 2 and the decoder that we have used. And, while training this it will try to minimize the error between Z 1 and Z 1 hat. And, once that training is complete that learning is complete what we have learnt is the weight matrix W 2. And, once weight matrix W 2 is learnt, you remove this second decoder that we have placed.

(Refer Slide Time: 20:10)

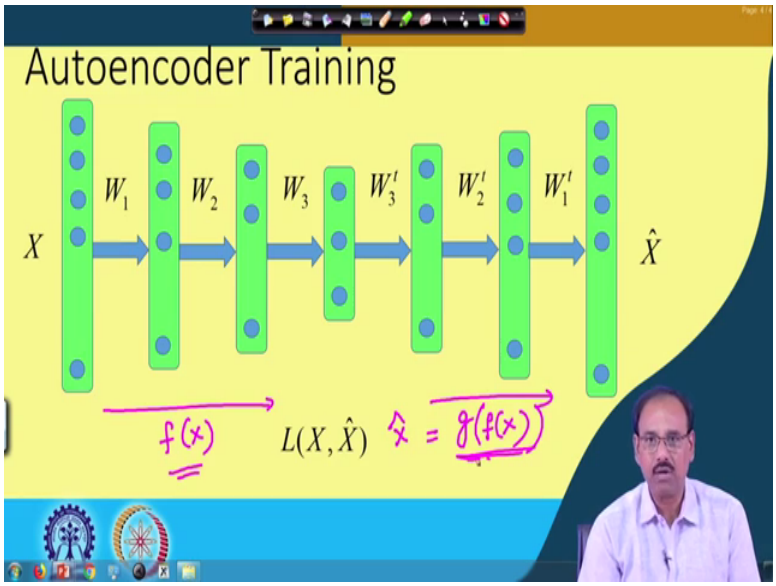


And, there the autoencoder 2 let me assume the output of the autoencoder 2 is Z 2. And, after got this I put the third autoencoder say AE 3, this is third autoencoder AE 3. So, now, we have to train this third autoencoder; that means, we have to find out that what is the weight matrix W 3.

So, again for training this I put another decoder over here and this decoder will decode the output of autoencoder 3. So, you find that the input to encoder 3 was Z 2 which was the output of autoencoder 2. So, this last decoder that we are talking about these decoder will try to decode this Z 2 would it by autoencoder 3 to give you Z 2 hat. And, for doing this the layers, which are involved is only these layers. And, it will try to minimize the error or the loss function between Z 2 and Z 2 hat or L Z 2, Z 2 hat.

And, once that is trained so you find that we have the pre trained values of W 1, W 2 and W 3, and if I assume that this is the last autoencoder layer that we had in our deep autoencoder then the next part comes is the decoder part. So, to state the decoder so, what I have to do is I have to again remove this autoencoder I have to have a decoder part. So, the decoder that you put is this by wrapping the encoder side. And, by taking the corresponding weights in the decoder side you take the transposes of the encoder weights.

(Refer Slide Time: 22:02)



So, this forms the total autoencoder or the encoder decoder pair, which forms the total autoencoder. So, here you find that W 1, W 2 and W 3, they were pre trained by using layer by layer mechanism; layer by layer learning mechanism. Now, I have this decoder also forming a full autoencoder and, now you go for fine tuning or finer training of the weight vectors, for that on the decoder side you initially assume that the weights are W 3 prime, W 2 prime and W 1 prime.

Now, you try to train this entire autoencoder chain using end to end training mechanism, that is you feed your training vector X to the input, output of the autoencoder becomes X hat which is the reconstructed value of X and by back propagation learning you try to minimize the loss function between X and X hat which is L X, X hat. And, here you find that because the encoder side was pre trained so, the values of the weight matrices W 1, W 2, W 3 are close to the actual values.

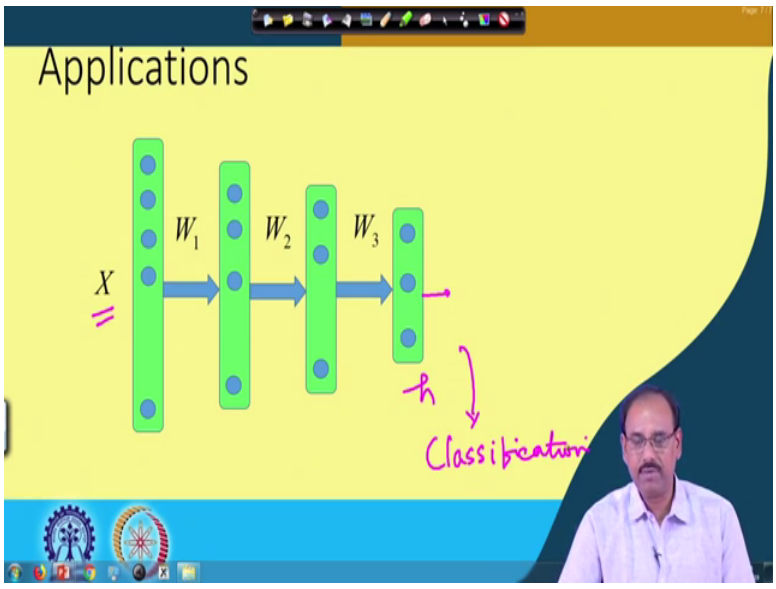
And, while you go for this end-to-end training now we have set of values, which are very close to the actual the convergence of this training algorithm is much better than, convergence of the algorithm if we try to train the intern network at a time. So, this is what is layer by layer pre training. So, what we have to do is after the pre training is complete we have to introduce the decoder part and then for fine tuning of the pre trained weights I had to go for one or more iterations for end to end training, giving the input as X and the output encoder output of the autoencoder X hat, try to reduce the error

between X and X hat, using the gradient descent or back propagation learning mechanism. So, that will ensure the convergence to be faster and likely to attain the global minimum of the loss function.

So, this is what is the pre training the layer by layer pre training of autoencoder. Now, suppose my autoencoder is trained. So, what the autoencoder is giving me, giving an input X. The encoder side actually gives me say a function f of x, that is my encoding function and say decoding let me call it a function g so, this gives me decoding of f of X. So, what is my X hat? X hat is nothing, but g of f x. So, the training mechanism try to minimize the error between f x and g of f x, and that is how you try to find that X and X hat will be identical when the autoencoders properly trained, ok.

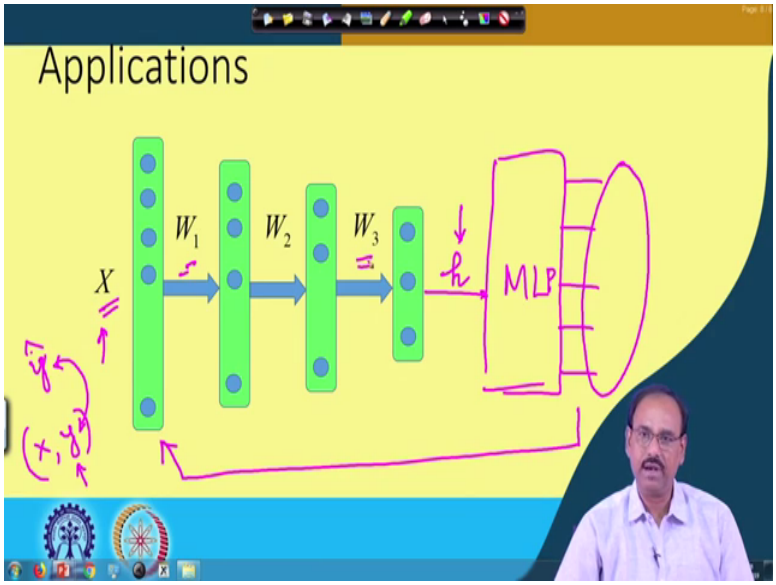
So, given this as we said before that we are not really interested in the reconstruction part, I have input X what do I do with X hat, that is not my m, but my m is actually the output of the autoencoder. That is the latent space representation of the reduced space representation that I get from X, and as we said that what I get at the output of the autoencoder say, let me call this as output vector h, which contains the salient features present in X, after discarding the redundancies or non-salient features which are present in X. So, h will contain the salient features and that is in the reduced dimension. So, using h how I can go for further applications.

(Refer Slide Time: 26:41)



So, one of the applications that can be there are various applications. So, one of the applications can be say classification. So, what I have is I have input vector X so, this input vector X, the output of the autoencoder gives me h, and here this h can be fed to a classifier for classification. It is not necessary that classifier; this classifier has to be a neural network, it can be any classifier, it can be a support vector machine, it can be a base classifier or whatever which we have discussed earlier. But, let us assume that we have a neural network we have talked about multi-layer perceptron before.

(Refer Slide Time: 27:28)



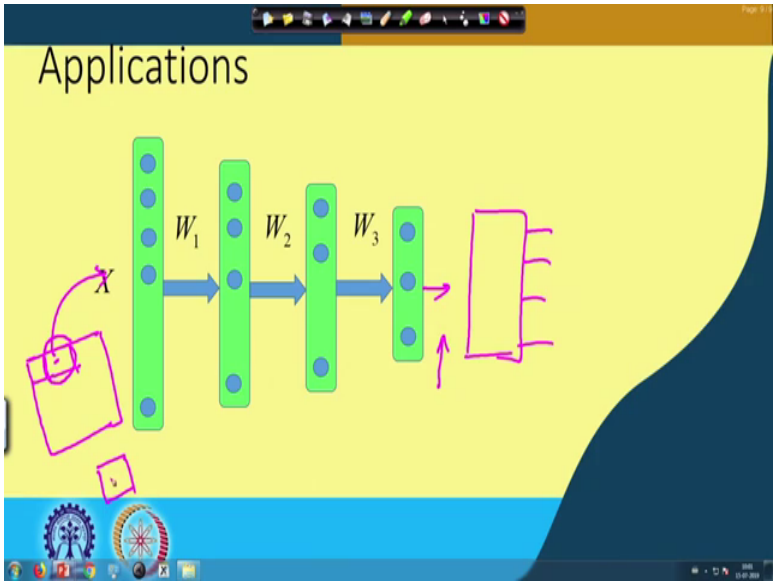
So, I can feed a multi-layer perceptron or MLP over here. So, the input to MLP is h and output of the MLP is the class identification of h or eventually the class identification of the input vector X. So, now, find that what the autoencoder does is it gives a coded output h of X, it does not give you what is the class belongingness of h or what is the class belongingness of X. So, if for the training vectors the class belongingness of X is also known that is the training vectors are given in the form of x, y, where x is the input vector and y is the class. Then, again I can go for an end-to-end training fine with finer refining of the weight matrices W 1, W 2, W 3.

And, now these training these back propagation training, we will consider the output of this MLP or we will consider the loss function with the output of the MLP, that also we have seen before, it can be cross entropy, it can be again sum of squared error. So, various such output error functions so, the loss function can be defined, which is defined

at the output of the MLP depending upon, whether the class say y hash, which is decided by this MLP taking this h that matches with y or not.

If it matches with y then there is no error or no training required, if it does not match then; obviously, we have to obtain this entire neural network with back propagation learning again. And, during that time also this W 1 to W 3, this encoder weight matrices can be defined further, this is one of the application. The other application can be even I can go for say pixel classification or segmentation operation of an input image.

(Refer Slide Time: 29:47)



So, for that what I can do is suppose I have an image an input, you take various patches of this input image convert that into a vector and feed that to the input. And, as we have seen before that output of the autoencoder will contain the salient features of each of these image blocks that is what the autoencoder is doing.

And, then at the output again I put a classifier, what this classifier will do it will put this input matrix, the input image or patch of the image into one of the classes. And, if that class and using that, again I can form an error function or a loss function and the autoencoder can again be trained using this loss function.

And, finally, once the autoencoder is trained again, now we have feed an input image again feed all the different blocks of the input image to this autoencoder, the autoencoder

will give a salient or the latent space representation of each of the blocks and the blocks can be classified by this trained M L P.

So, thereby I can classify each and every block in the image which is nothing, but a segmentation operation. I can also consider this to be a pixel by pixel classification or the semantic segmentation of the input image. And, for that what I can do is given and patch in the image, whatever representation I am getting at the output, I can consider that this is the salient features of the central pixel of this patch. So, this is my central pixel.

So, this classifier output basically classifies the central pixel. So, as I take different blocks of the image and pass it through the autoencoder and classifier chain, the classifier output will classify each and every pixel, which is the center of the block. So, it will classify each and every pixel within the image to different classes. So, the classification of pixels to different classes is nothing, but segmentation of the input image. So, the input image or the pixels of the input image are now classified into different class, and all the pixels belonging to the same class that forms a particular segment.

So, today what you have discussed is that how you can train an autoencoder we have talked about layer by layer training of the autoencoder, and once the autoencoder is trained what can be possible applications of such autoencoder. So, we will stop here today.

Thank you.