

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

**Lecture – 28**  
**Autoencoder**

Hello, welcome to the NPTEL online certification course on Deep Learning. Till our previous class we have talked about the back propagation learning.

(Refer Slide Time: 00:40)



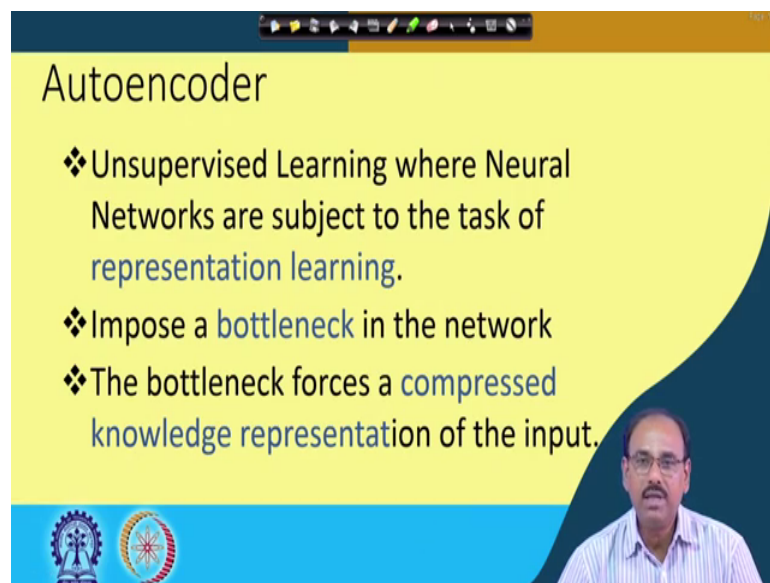
And, we have seen that how the back propagation learning is actually implemented or takes place in a feed forward neural network at the network level. And, also how the gradient is back propagated within a particular node, or different layers, or different circuits within a particular node. And, as we said in the previous class now onwards I will assume that you know back propagation learning and where it whenever the learning is discussed I will simply referred that back propagation learning algorithm is used, I will not go into details of the learning algorithm until and unless some details is essential.

So, in today's discussion we are going to start to discuss on Autoencoders. So, today and subsequent few lectures we will talk about under complete autoencoder, we will try to find out what is that relationship between an autoencoder and principle component analysis or PCA. Maybe, I will discuss something about PCA for those of you who are

not our of this. We will talk about the other variants of the autoencoder, namely sparse autoencoder, denoising autoencoder, contractive autoencoder and so on.

And, then we will also talk about convolution autoencoder, but not as a continuation of this series on autoencoders, but we will come back to convolution autoencoder, after we discussed about convolution and convolution neural network.

(Refer Slide Time: 02:33)



The slide is titled "Autoencoder" and features three bullet points: "❖ Unsupervised Learning where Neural Networks are subject to the task of representation learning.", "❖ Impose a bottleneck in the network", and "❖ The bottleneck forces a compressed knowledge representation of the input." The slide has a yellow background with a dark blue curved shape on the right side. At the bottom left, there are two logos: one of a gear and one of a circular diagram. At the bottom right, there is a small video inset showing a man with glasses and a mustache, wearing a light blue shirt, speaking.

So, today what we are going to talk about the autoencoder and under complete autoencoder. Now, what is this autoencoder? As the name suggests that autoencoder is nothing, but an algorithm, that codes itself or that encodes itself.

So, you can say that autoencoder is an unsupervised learning algorithm, where the neural networks are subject to the task of representation learning. And what is this representation? The representation is nothing, but how you code or how you encode the input data that is fed to the network. And, learning this representation learning this code is what is known as a representational learning.

And, we say autoencoders are unsupervised learning, because when you train an autoencoder for coding an input or for encoding an input. We do not use data's which are leveled data's, unlike in case of classification problems that we have discussed earlier. So, if you remember what we discussed in case of classification that for training the

network or for training of your classification algorithm, machine learning algorithm, we need a lot of training data.

And, what that set of training data tells you is that, it tells you that what is the class belongingness of a particular training data. And, it is only from that information of class belongingness I can compute the error, because if my machine says that our training data or if my machine in first that the training data belongs to some category say a 5 whereas, the ground truth says that that particular training data belongs to category 1. So, there is a mismatch my machine says it is category 5 whereas, the ground truth is category 1.

So, there is an error and your learning algorithm as we said using back propagation tries to minimize this error; that means, as the machine has interpreted to E 5 what modifications or what updations in the weight vectors we have to do. So, that the machine really interprets this data to belong to category 1.

So, those are the supervised learning algorithms, because the data that you used for training or learning or the labeled data, but in case of autoencoder the data that we use are not labeled data. However, we still have back propagation algorithm, because we want that whichever way the machine represents or the autoencoder represents the input data from that representation it is possible, it should be possible that we should be able to reconstruct the input data.

That means I have to find out that after encoding whether the encoded data can be reconstructed. So, if for encoding I call it a forward encoding algorithm. So, some mapping function  $f$  that gives me the encoded data, then another mapping function  $g$  should be able to convert or transform that encoded data to my original input.

And for learning what you do is you compare the original input and this reconstructed input and try to minimize the error between these two. So, this autoencoder as we said that it is an unsupervised learning and the task of the neural network in this case is to go for representation learning, or try to encode, or learn how to encode, or how to code the input data. And, in order to do this what you do is you introduce bottleneck in the network.

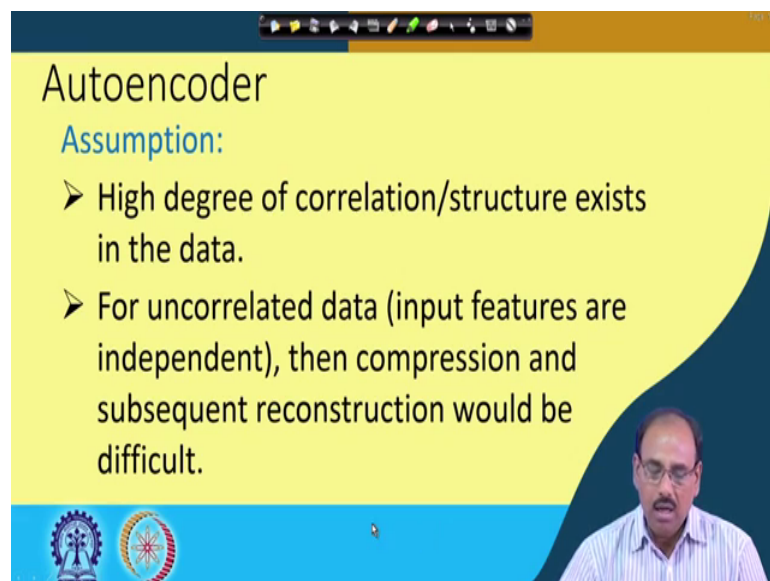
Because, as we said that our learning algorithm will be that I have an input, I have some coding in between then I have a reconstructed output. And, I want that the reconstructed

output should be similar to the input or they should be identical if possible. So, there is a possibility that the network may eventually learn an identity mapping ok.

So, if the network learns an identity mapping, it does not learn, it does not really learn the representation. So, in order to enforce or in order to force that the network learns the representation or network learns what is the inner structure of the data, It is necessary that in the network you impose a bottleneck layer, we will come to ah bit later details of all this is done. And this bottleneck actually forces a compressed knowledge representation of the input.

That means, if my input vector input data is of dimension say  $d$  in this compressed knowledge representation, it will be mapped to a vector of dimension say  $M$  where  $M$  is much less than  $d$ . So, how it is done we will come to this a bit later.

(Refer Slide Time: 07:51)



Autoencoder

Assumption:

- High degree of correlation/structure exists in the data.
- For uncorrelated data (input features are independent), then compression and subsequent reconstruction would be difficult.

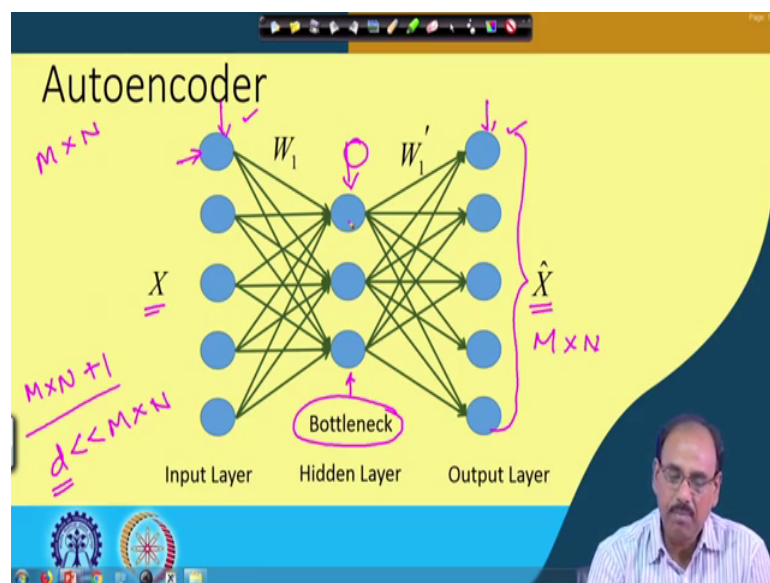
So, for this we have certain assumptions the assumption is there is a high degree of correlation or structure that exists in the data. If, the components of the input data are not correlated; that means, if the features are independent of one another, then this compressed domain representation and subsequent reconstruction of the original input will be difficult in fact, it may not be possible at all.

So, when the neural network goes for representational learning or tries for representation of the input data in compressed domain, what it tries to do is it removes the correlation or

the redundancy present in the data. And, what is what it preserves is only the uncorrelated part. And, from this uncorrelated part then it should be subsequently possible to reconstruct the original input data.

So, that is what an autoencoder is and as we said that the name autoencoder in indicates that it encodes the data or it codes the data on it is own. And, this is an unsupervised learning, because for training an autoencoder we do not use any label data. What we want is whatever is fed to the input the autoencoder outputs the same thing.

(Refer Slide Time: 09:20)



So, for this I need 2 different functions; one is the encoding part, one is the decoding part. So, the encoding part will encode the input data to and compressed domain representation knowledge representation. And, the decoder part will decode the data from that compressed representation from the encoded output to your original input. So, if my input was  $X$  it will reconstruct  $\hat{X}$  I want that  $X$  and  $\hat{X}$  should be similar or the error between  $X$  and  $\hat{X}$  should be minimum.

And, that is what is given by the decoder part. So, I should have an encoder half, I should also have a decoder half. And, this is the structure the base structure of an autoencoder. So, you find that in this autoencoder, I have an input layer this is the input layer and I have a hidden layer and I have an output layer.

So, this hidden layer is actually the bottleneck layer. So, in the bottleneck layer what you are doing is you are compressing the data, you are going for compressed domain representation. So, you find that the number of nodes in the hidden layer or the number of nodes in the bottleneck layer is much less than the number of nodes in the input layer. And, also you find that the number of nodes in the input layer is same as the number of nodes in the output layer.

Because, finally, at the output I want that whatever was the compressed domain representation in the hidden layer from this contest domain representation, it should be possible to reconstruct my original input. So, original input was  $X$  I should be able to reconstruct this  $X$  which is  $\hat{X}$ .

So; obviously, the dimensionality of  $X$  and the dimensionality of  $\hat{X}$  should be same. So, what does it mean? Say for example, I use this network for compressed domain representation of an input image. And, suppose image is of size  $M$  by  $N$ . So, there are  $M$  into  $N$  number of pixels.

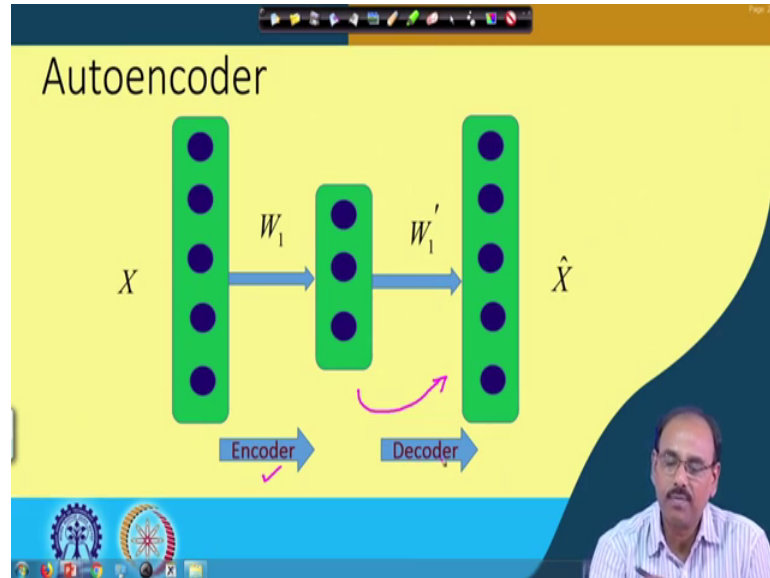
So, as we said before that by column concatenation this  $M$  by  $N$  matrix can be represented by an one dimensional vector is having  $M$  into  $N$  number of elements, that we have to do by column concative concatenation. So, if each of the in each  $N$  pixel is fed to one of the nodes on the input layer.

So, the number of nodes in the input layer has to be  $M$  into  $N$ , because every node input in the input layer gets 1 pixel. In addition we have to have one more node to represent the bias. So, the number of nodes in the input layer has to be  $M$  into  $N$  plus 1. Whereas, on the reconstruction side I do not need any bias, I just want my  $X$  back. So, that is what I need an  $\hat{X}$ . So, the number of nodes in the output layer has to be  $M$  into  $N$  as against  $M$  into  $N$  plus 1 on the input side.

Now, suppose I want that this entire image should be represented by a  $d$  dimensional vector or a vector having  $d$  number of components, where  $d$  is much less than  $M$  into  $N$ . So, in that case the number of nodes in the hidden layer has to be equal to  $d$ , which is my bottleneck layer, but the reconstruction purpose I need one more additional node to take care of the bias. So, the number of nodes in the hidden layer for the decoding side or for the reconstruction side will be  $M$  into  $N$  plus 1. So, this is the base architecture of an

autoencoder. Now, we find that so I have an input layer, I have an output layer; I have an hidden layer, or a bottleneck layer.

(Refer Slide Time: 14:01)



Now, this same architecture, now onwards for simplicity I will represent like this, that input layer will be an array of nodes, hidden layer will be another array of nodes, where array of array size in the hidden layer will usually be less than the array size in the input layer.

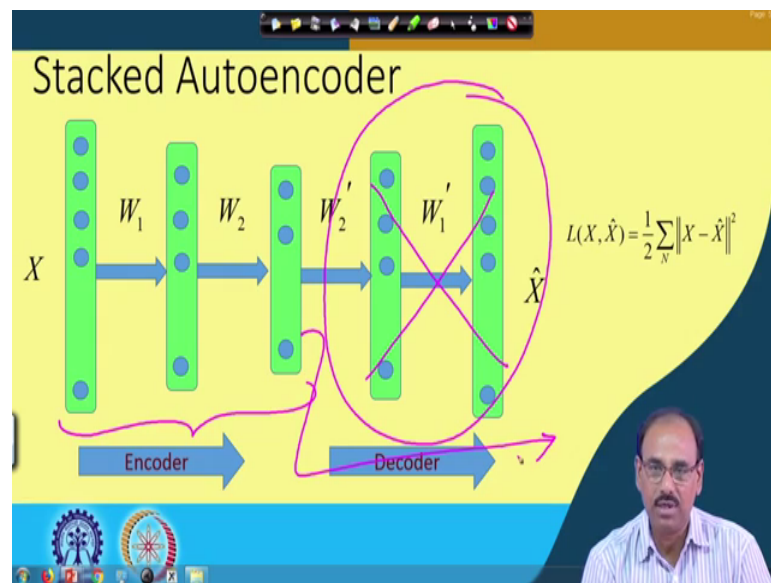
Similarly, on the output side also I will have an array of nodes. And, I will have a set of weights say  $W_1$  and  $W_1'$ . So, this  $W_1$  connects the input layer to the hidden layer and  $W_1'$  connects from the hidden layer to the output layer right. It is also possible, that instead of having just one hidden layer I can have multiple hidden layers.

Now, before that this side when you are going for input to the hidden layer, the hidden layer actually gives you the encoded information in a lower dimension in general. Later on when we talk about sparsity or sparse autoencoder, we will see that it is not necessary that I will have to go for dimensionality reduction. I can have our dimension explosion as well or may be of the same dimension, but there your compressed representation is done by some other mechanism.

However, for the timing let us assume that we are going for compressed domain representation, knowledge representation. So, from input to the reduced dimensional

representation that is a part which is the encoding part. So, this is the encoder side; so this is what is your encoder. And, similarly from compressed domain representation to the reconstruction of the original signal this is what is the decoder part. So, I have an encoder I also have a decoder.

(Refer Slide Time: 16:09)



It is also possible that I may not have only one hidden layer or only one bottleneck layer. I can have multiple number of hidden layers. So, I can have a situation something like this. So, on the encoder side I have a number of hidden layers on the decoder side also I will have a number of hidden layers. I will come back to this one a bit later.



(Refer Slide Time: 16:19)

Expectation

- ☑ Sensitive enough to input for accurate reconstruction
- ☐ Insensitive enough that it does not memorize or overfit the training data

↓

Loss Function  $\Rightarrow L(X, \hat{X}) + \text{Regularizer}$

Now, in this case as we said that when I have an autoencoder, what I have is I have input data and output of the autoencoder is also and data. And, I want that the output should be a faithful reconstruction of the input. And, while doing so the information passes through the bottleneck layer, where in the bottle neck layer I have a compressed domain representation or coded version of the input data.

Now, I my expectation from such an autoencoder is twofold. Firstly, I want that the autoencoder should be sensitive enough to the input for accurate reconstruction, because what we said is my reconstructed vector  $\hat{X}$  should be as close as possible to input  $X$ .

So; that means, my autoencoder should be accurately should be able to accurately reconstruct my input signal. So, that is what is it is sensitive enough to input for accurate reconstruction. Now, if accurate reconstruction is my  $M$  not the representation then an identity function is sufficient. So, it might be possible that autoencoder simply learns the identity function.

So, if it simply learns the identity mapping it will always reconstruct your output faithfully as the input, but that is not our  $M$ ; our  $M$  is actually what is happening at the bottle a bottleneck layer, that is how the input data is represented in the compressed domain, that is what it is my interest. And this encoded data will be useful for some later applications, because I have the original if my reconstruction is the is the only  $M$  why do

I need it, I have the original, why do I have to go through the bottleneck and then reconstruction.

So, the other expectation from an autoencoder is it should be insensitive enough that it does not memorize or over fit the training data; that means, it does not learn the identity function. So, I have 2 conflicting requirements or 2 conflicting expectations from an autoencoder, that it should be sensitive and at the same time it should not be sensitive enough. So, how do you impose or how do you try to satisfy both this requirements both this conflicting requirements simultaneously?

So, that is actually done by designing your loss function, which takes care of both of them. And, this loss function as we said that the loss function will be used for back propagation learning algorithm, when you train the autoencoder.

So, the loss function in this case will be given by this that this loss function will have 2 components; one is  $L(X, \hat{X})$  which is the error between the input  $X$  and very constructed  $\hat{X}$ . So, if I want to minimize this; so minimization of this the error takes care of this, that the autoencoder is sensitive for faithful reconstruction of the input. Whereas, the second the comment, which conflicts the previous the comment is through a regularizer term.

So, in the loss function you have an error function between  $X$  and  $\hat{X}$  and you also have a regularizer term. So, the regularizer term we will try to make it insensitive to the input and it will force the autoencoder to learn the low dimensional representation, where it learns the salient features of the input.

So, that using the salient features that decoder will be able to reconstruct the data. So, it does not simply learn the identity function. So, both these requirements conflicting requirements are made are satisfied by defining a loss function of this form. So, this is what I said that the way variant of the autoencoder, which is known as under complete autoencoder.

Takes care of the regularization is by introducing or by introducing restriction on the number of nodes in the hidden layer or the bottleneck layer. So, as we said usually number of nodes in the bottleneck layer or the hidden layer is much less than the number of nodes in the input layer, or similarly the number of nodes in the output layer.

So, in such cases the network is made insensitive to the input by restricted number of nodes in the hidden layer. And, when I have an architecture of this form this is what is known as an under complete autoencoder architecture. And, for training such an autoencoder you simply minimize the loss function, which is given by  $L(X, \hat{X})$ , which is half of  $\|X - \hat{X}\|_2^2$  and some of this over all the training vectors.

And, you find that here as we said that this is an unsupervised learning because I do not need to have a knowledge of the class belongingness of the input vector  $X$ , what I only need is that the  $X$  and  $\hat{X}$  should be similar that is  $\hat{X}$  should be faithful reconstruction of my input  $X$ . So, this is what is my under complete autoencoder.

Now, as we said before that it is not a necessary that the autoencoder have to have only one hidden layer or the bottleneck layer, I can have and stacked autoencoder where a number of hidden layers are stacked one after another. So, this diagram shows that I have stacked autoencoder where; obviously,  $X$  is fed to the input layer. I have the first hidden layer in the autoencoder on the encoding side, the connection weights between the input layer and the first hidden layer autoencoder, layer is given by the weight vectors or weight matrix  $W_1$ .

Then, I have the second hidden layer and, in this case the second hidden layer happens to be the bottle neck layer. And, the connection weights between the first auto encoding layer to the second hidden layer the second auto encoding layer is the set of connection weights  $W$ .

So, this completes my encoding part, then on the decoder side from the encoded data it goes to the first hidden layer on the decoding side, and the connection weights from this to this is given by  $W_2$ . And, similarly from this hidden layer, to the output layer, the connection weight is given by  $W_1$ .

And, what I simply want is that the  $\hat{X}$  should be a faithful reconstruction of  $X$  that is the input data. And, for that the loss function that we have to minimize is the  $L_2$  norm between  $X$  and  $\hat{X}$ . And, this has to be summed over all the training data and you minimize this  $L_2$  norm using back propagation learning algorithm. And, while this is minimized you find that this weights  $W_1$ ,  $W_2$ , similarly  $W_2$  and  $W_1$

they will be modified, they will be updated until and unless the L 2 norm or the error between  $X$  and  $\hat{X}$  is 0 or it is within an acceptable limit.

At that point we say that my autoencoder is properly trained. Now, what you do after the autoencoder is properly trained? I said earlier that reconstruction is not my  $M$ , I want to reconstruction I want to reconstruct the input data from the encoded data just to ensure that my encoding is proper, that is whatever my is my encoded data from the encoded data, I should be able to reconstruct my encoding input data. Which ensures that in the process of encoding I have not to lost any information, but this decoding part of the reconstruction part is not the one that I am interested in. What I am interested in is simply this encoding part my interest is up to this.

So, for subsequent applications if I want to apply this even for classification purpose what I can do is after this autoencoder is trained; that means, my encoding part is proper, I can simply forget about this part of the network. You simply cut it off, your feed your input, I have the encoded output and feed this encoded output to your other application modules, this is what is my aim. My aim is not the decoding. Decoding is only a helped to ensure that my encoder is working properly.

So, this is what the autoencoder will do and for this again we said that what we want is that autoencoder will be trained using the back propagation learning and for during this back propagation learning will make use of the errors of the gradient of the error that you get at the output layer, between the input and the reconstructed output, and based on that you modify all the connection weights. So, that your encoder is properly trained. So, I will stop here today we will continue with our discussions on autoencoders in subsequent lectures.

Thank you.