

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

**Lecture - 32**  
**Autoencoder Variants I**

Hello, welcome to the NPTEL online certification course on Deep Learning. You remember in the previous class or for last few classes we are discussing about the Autoencoder.

(Refer Slide Time: 00:43)



And in the previous class what we have discussed is about autoencoder training. And there we have seen that for efficient training of autoencoders. What is better is to go for layer by layer pre training and once the encoder is trained layer by layer, which we are calling as pre training. Then, you introduce the decoder part and we can have final training passes, which are end to end training. And what we have considered so far is a type of auto encoder, which we said as under complete autoencoder.

And, in case of under complete autoencoder given an input vector, we want that the final output of the autoencoder will be a reconstructed version of the same input vector; that means, ideally the input and the output they should be identical. Now, given this requirement of an autoencoder, it is quite possible that the autoencoder in the process, we

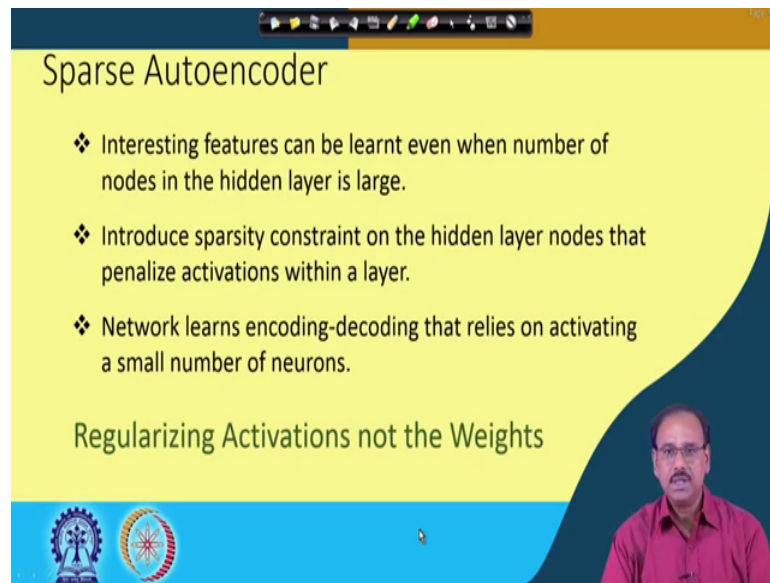
will learn an identity function or it will simply memorize, whatever is the input vector that is presented to the autoencoder.

But, that is not the aim what we want is that the autoencoder should learn or generalizable encoding decoding mechanism. So, for that autoencoder has to learn some structure or the salient features which are present in the input data. And, it should be possible that those salient features are learnt in a compressed domain. Or in other words, the encoder of the output encoder should learn a compressed domain representation of the input data, where this compressed domain representation contains the salient features of the input data.

And, using these salient features of the compressed domain representation the decoder part of the autoencoder should be able to reconstruct your original data, which is fed to the input of the encoder. So, in case of under complete autoencoder that we have seen so far this is achieved by introducing a bottleneck layer, which is the hidden layer and in the hidden layer, the number of nodes or the neurons we have is much less than the number of nodes in the input layer or the number of nodes in the output layer. So, in effect what we have forced is that when the information passes through this autoencoder from input layer to the output layer. In between it passes through a bottleneck layer, where the number of nodes is much less.

So, as a result you are restricting the autoencoder from learning us or simply memorizing the input data that you are feeding, but the autoencoder learns a compressed domain representation of the input data. So, that is just one version of the autoencoder, which we have discussed, which is under complete autoencoder. So, there are other types of auto encoders that we are going to discuss now. So, what we will be discussing is the sparse autoencoder, the denoising autoencoder, the contractive autoencoder and so on. And of course, as we said that a convolution auto encoder, we will deal sometimes later.

(Refer Slide Time: 04:30)



**Sparse Autoencoder**

- ❖ Interesting features can be learnt even when number of nodes in the hidden layer is large.
- ❖ Introduce sparsity constraint on the hidden layer nodes that penalize activations within a layer.
- ❖ Network learns encoding-decoding that relies on activating a small number of neurons.

**Regularizing Activations not the Weights**

The slide features a yellow background with a dark blue curved shape on the right side. At the bottom left, there are two circular logos. At the bottom right, there is a small video inset showing a man with glasses and a red shirt.

So, first let us see that what is sparse autoencoder? So, as we have just said that, in case of under complete autoencoder. The autoencoder learns a compressed domain representation or it learns the salient interesting salient features of the input data, by passing the information through a bottleneck layer. However, for learning the interesting features, it is not necessary that I have to pass the information through a layer, a hidden layer or a bottleneck layer, where the number of nodes is much less than the number of nodes at the input layer.

So, such interesting features can be learned even when the number of nodes in the hidden layer is large. It may be same as the number of nodes in the input layer or even it might even be larger than the number of nodes in the input layer. And even in such cases it is possible to learn the interesting features in a compressed domain. And the way it is done is you introduce a sparsity constraint.

So, a sparsity constraint is introduced on the hidden layer nodes and what this sparsity constraint does is it penalizes the activation function of the hidden layer nodes. Or in other words you are you are trying to restrict the activation of the hidden layer nodes in the autoencoder. And while doing so, the autoencoder learns a encoding decoding mechanism, which relies on activating a small number of neurons.

So, even though we have a large number of nodes or large number of neurons in the hidden layer, but all of them are not activated. You are activating a small number of

neurons out of those large number of neurons. So, in effect individual neurons in the hidden layer, effectively learn a particular type of feature of the input data. And different neurons are activated for different types of features. And obviously, not all the nodes are activated. So, that is what is done in case of auto in case of sparse autoencoder. And as we are restricting the activation of the hidden layer nodes, so, that is done by using by introducing a regularization term.

So, you remember that when we talked about the back propagation algorithm, which minimizes a cost function or a loss function. The loss function consisted of two terms; one was the data loss term and the other component was regularization loss term. So, in this case or in the previous case when we talked about training of the multi-layer perceptron, the regularization term was on the weights or the weight values. So, there you remember we had mod of w square where w was the weight matrices or the components in the weight matrix.

So, the regularization in that case was on the weights, but in case of sparse autoencoder as we are trying to restrict or limit the activations of the hidden layer nodes. So, the regularization that we are doing is on the activations, it is not on the weights as we have done in case of MLP or Multi-Layer Perceptron. So, in this case the regularization will be on the activations of the hidden layer node.

(Refer Slide Time: 08:24)

**Sparsity Constraint**

$a_j^h \rightarrow$  Activation of  $j^{\text{th}}$  Neuron in hidden layer h

$a_j^h \rightarrow 1 \Rightarrow$  Neuron is active

Average activation  $\rightarrow \hat{\rho}_j = \frac{1}{m} \sum_{i=1}^m a_j^h(x_i)$

Constraint  $\rightarrow \hat{\rho}_j = \rho$

$\rho \rightarrow$  sparsity parameter (typically a small value)

The slide includes a graph of a sigmoid function with a vertical line at a specific input value, and a small video inset of a man in a red shirt in the bottom right corner.

So, let us see that how this regularization is done? So, suppose a  $a_j^h$  as you see over here is the activation of the  $j$ th node in the hidden layer  $h$ . And we assume that the type of non-linearity or the non-linear activation functions of the nodes which are being used is a sigmoidal activation, a sigmoidal non non-linear function. So, as it is sigmoidal non-linear function; that means, if this node  $a_j^h$  is active the output of the neuron will be close to 1. And, if the node is not activated for certain input, then the output of the neuron will be close to 0. And, that is what is your sigmoidal activation function. So, if you remember a sigmoidal activation function which was something like this.

So, the sigmoidal activation function was of this form, so as the value of  $x$  increases or the input increases the output of the sigmoidal activation function goes closer to 1, asymptotically reaches the value equal to 1 and as the input reduces, the output of the sigmoidal function sigmoidal activation function asymptotically goes towards 0.

So, that is what we are using over here we are assuming that the non-linear activation function of the nodes are sigmoidal activation. So, if the node is active the output will be close to 1, if the node is not active the output will be close to 0. And, given this I can compute the average activation of the  $j$ th node. So, let me put this average activation of the  $j$ th node as it is a  $\hat{\rho}_j$  or  $\rho_j$ .

(Refer Slide Time: 10:47)

The slide is titled "Sparsity Constraint" and contains the following text:

- $a_j^h \rightarrow$  Activation of  $j^{\text{th}}$  Neuron in hidden layer  $h$
- $a_j^h \rightarrow 1 \Rightarrow$  Neuron is active
- Average activation  $\rightarrow \hat{\rho}_j = \frac{1}{m} \sum_{i=1}^m a_j^h(x_i)$
- Constraint  $\rightarrow \hat{\rho}_j = \rho$
- $\rho \rightarrow$  sparsity parameter (typically a small value)

Handwritten notes in pink ink next to the constraint equation show the values 0.2, 0.02, and 0.5. A video inset in the bottom right corner shows a man in a red shirt speaking.

So, the average activation of the  $j$ th neuron in the hidden layer is  $\hat{\rho}_j$  where  $\rho_j$  is nothing, but sum of  $a_j^h x_i$  is the  $i$ th input vector right. So, the activation of

this  $j$ th node in the hidden layer  $h$  for an input vector  $x_i$  and if there are  $m$  number of such input vectors, then the average over all these input vectors of this activation function is nothing, but  $\frac{1}{m} \sum_{i=1}^m a_{jh} x_i$ , where  $i$  varies from 1 to  $m$ .

So, that is the average activation of node  $j$  or neuron  $j$  in the hidden layer  $h$  over all the input vectors that we fit. And, given this we put a constraint that suppose I want, that average activation of a hidden layer node should be equal to  $\rho$ , where this  $\rho$  should be equal to  $\hat{\rho}_j$  the average activation of  $\hat{\rho}_j$  should be equal to  $\rho$ , where this  $\rho$  is sparsity parameter.

And typically if we impose this we assume that this sparsity parameter is very very small say something like 0.2 or even 0.02, maybe 0.5 depending upon the degree of sparsity that we want to impose on this network. So, once I impose this sparsity parameter and I want that the average activation of a hidden layer node should be equal to this sparsity parameter, which is a constraint.

(Refer Slide Time: 13:00)

**Sparsity Constraint**

$$\text{Regularizer: } \sum_{j=1}^{N_h} \left[ \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j} \right] \Rightarrow \sum_{j=1}^{N_h} KL(\rho \parallel \hat{\rho}_j)$$

$$J_{\text{sparse}}(W) = L(X, \hat{X}) + \lambda \sum_j KL(\rho \parallel \hat{\rho}_j)$$

Data Loss                      Regularization Loss.

Now, using this now I can introduce a regularization term as we said that this regularization is on the activation this regularization is not on the weight values. So, the regularization term can be as KL divergence between the two distributions one defined by  $\hat{\rho}_j$  and the other defined by  $\rho$ . So, the regularization term now becomes this that is  $\rho \log \rho$  upon  $\hat{\rho}_j$  plus  $(1-\rho) \log (1-\rho)$  upon  $(1-\hat{\rho}_j)$

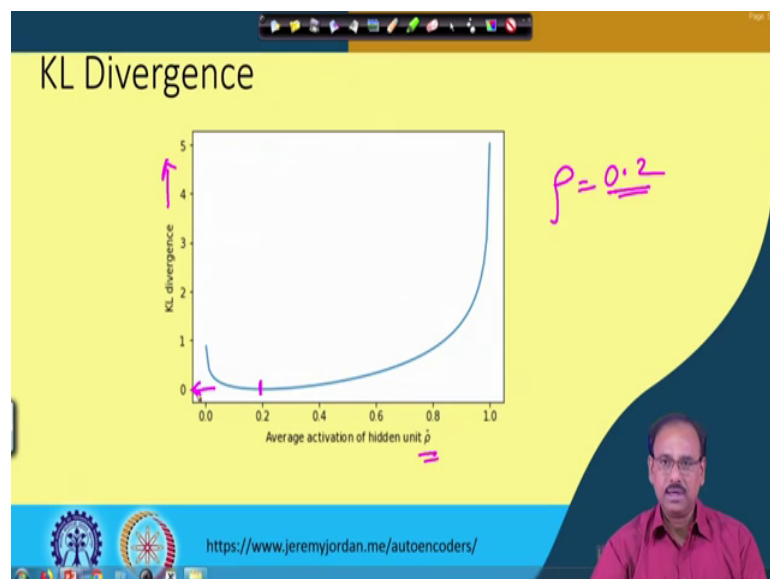
take the sum over this over all the nodes in the hidden layer. So, I assume that the number of nodes in the hidden layer is equal to  $h$ .

So, this is the regularization term and this is what is the KL divergence between two distributions Bernoulli distributions given by  $\rho$  and  $\hat{\rho}_j$ . So, that is the KL divergence and in fact, this KL divergence gives you the distance between two different distributions. So, given this now I can have a complete loss function for this parts at autoencoder, which is defined as  $J_{\text{sparse}} + \lambda W$ ;  $W$  is the weight values or weight matrix components, which is now  $L \times X \times \hat{X}$ , where this  $L \times X \times \hat{X}$  is the data loss component. So, this is what is the data loss component, where  $X$  is your input vector and  $\hat{X}$  is the output vector or the reconstructed  $X$ , which is the output of the autoencoder.

And in addition to that I also introduce this KL divergence between the two distributions; one given by  $\rho$  the other given by  $\hat{\rho}_j$ . And, this hyper parameter  $\lambda$  this represents that, what is the relative weight, that I am putting in between the data loss component and the regularization loss component. So, this is what is regularization loss?

So, this is my overall loss function. And, now you can apply the back propagation learning algorithm for minimization of the overall loss function. So, once I introduce this regularization loss, which is given by this the KL divergence between distributions Bernoulli distributions given by  $\rho$  and  $\hat{\rho}_j$ , let us see what is its implication on the back propagation learning or the gradient descent operation.

(Refer Slide Time: 16:16)



So, you remember that and this is what tells you that how the KL divergence between the two distributions that varies. So, here you find that if I put rho, which is the constraint if I put rho is equal to 0.2, the KL divergence. So, this side is the KL divergence between the distributions given by rho and rho j hat. So, here you find that if rho is equal to 0.2, then when your rho j hat is also 0.2, then the divergence is minimum and this is equal to 0. And as the values between rho and rho j hat differs the divergence shoots off quite fast it increases quite fast.

So, when rho and rho j hat they are equal then the divergence is minimum and it is equal to 0 ok. So, this is the implication of the KL divergence, which we said that it measures the distance between two different distributions. So, as I was saying that what will be the implication of introducing the scale divergence as a regularization loss in our overall loss function.

(Refer Slide Time: 17:36)

Sparsity Constraint

$$\delta_i^k = O_i^k (1 - O_i^k) \sum_{j=1}^{M_{k+1}} \delta_j^{k+1} W_{ij}^{k+1}$$

$$\delta_i^k = O_i^k (1 - O_i^k) \left[ \sum_{j=1}^{M_{k+1}} \delta_j^{k+1} W_{ij}^{k+1} \right] + \lambda \left[ -\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i} \right]$$

So, if you remember you try to recollect that, when you talked about the back propagation learning of multi-layer perceptron's, we had a term given as delta i k, where delta i k consisted of the back propagated error from the subsequent layers ok, from the output side towards in outside. So, this delta i k was the back propagated error, which is basically a combination of errors from all those nodes or all those paths to which this i th node in the k th layer was feeding the input.



So, all the errors so if the  $i$ th node in the  $k$ th layer is this. So, this is  $i$ th node in the  $k$ th layer and it is feeding input to all different layers in the subsequent all different nodes in the subsequent layer, then the error which is back propagated from all these paths, they are accumulated together. So, that is what is given by this summation operation, they are accumulated and then it is multiplied by the local gradient, which is  $O_{ik}$  into  $O_{i1}$  minus  $O_{ik}$ , where  $O_{ik}$  is the output of the  $i$ th node in this  $k$ th layer and for weight updation this has to be multiplied by the input to this  $k$ th layer node, which is actually output of a previously unknown.

So, this is a term which we had used in back propagation learning of multi-layer perceptron. And when we introduce this KL divergence between the 2 distributions, this term gets modified by this  $ok$ . So, you find that this first part that summation, which is the back propagated error term from the subsequent layers that remains as it is.

And, additional term in this gradient descent procedure that comes into picture is  $\lambda$  times minus  $\rho$  by  $\rho_j$  hat plus  $1$  minus  $2$  upon  $1$  minus  $\rho_j$  hat  $ok$ . Whereas we said that  $\rho$  is the average activation,  $\rho$  is the constraint, and  $\rho_j$  hat is the average activation of a node in the hidden layer.

So, this is the modification or this is an additional term, which has to be introduced in the back propagation learning. All other steps in the back propagation learning will remain as it is. So, by introducing this the KL divergence, what we are doing is as the average activation of the hidden layer nodes is being restricted. So, it is quite likely that most of the nodes in the hidden layer will be inactive and few nodes in the hidden layer will be active, for a given input data. If the input data is varied for some other input data, it is quite possible that some other set of hidden layer nodes will be active while some other set of the hidden layer nodes will be inactive.

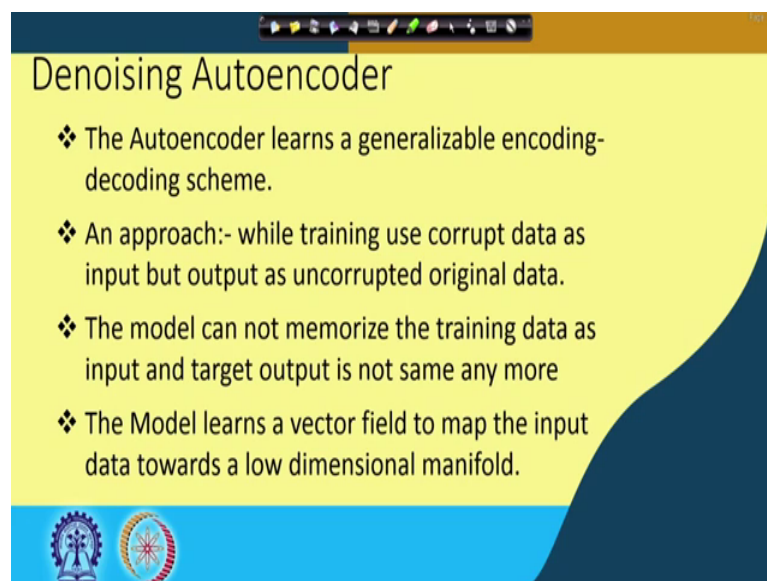
So, as a result the individual nodes in the hidden layer, they are activated based on a particular type of interesting or salient features in the input data or we can say that different nodes in the different in the hidden layer they are activated for different types of features  $ok$ . Or in other words different nodes, learn, different types of features in the input data. So, given an input data, if a kind of input features the  $k$  or is present in the input data, then say  $k$ th node in the hidden layer will be active. Whereas, if the input

data contains a kind of feature say  $p$ , then  $p$  th node in the input in the hidden layer will be active.

So, different nodes in the hidden layer they are activated for different types of features or in other words the individual nodes in the hidden layer, they learn different types of features given in the input data and that is what is your sparse autoencoder.



So, in case of sparse autoencoder as we have just seen that even though the number of nodes in the hidden layer is large or maybe even larger than the number of nodes in the input layer, but still it learns a compressed domain representation not simply remembers the input data, but it learns a compressed domain impression, representation of the input data and that compressed domain representation is learnt by introducing this sparsity constraint. So, that is what is our sparse autoencoder. Now, we talk about another type of autoencoder, which is known as denoising autoencoder.

(Refer Slide Time: 23:18)



### Denoising Autoencoder

- ❖ The Autoencoder learns a generalizable encoding-decoding scheme.
- ❖ An approach:- while training use corrupt data as input but output as uncorrupted original data.
- ❖ The model can not memorize the training data as input and target output is not same any more
- ❖ The Model learns a vector field to map the input data towards a low dimensional manifold.



So, in case of denoising autoencoder, the concept is for training when have feed an input data, instead of feeding an input data, you feed a corrupted version or slightly perturbed version of the input data. So, why we want this, because as we said earlier, that our aim is that the autoencoder should be sensitive enough to the input data, so, that it is able to reproduce the input data.

At the same time the autoencoder should be insensitive enough to the input data that it simply does not remember the input data, but it should be able to generate a compressed domain representation of the input data. Or in other words the autoencoder will learn and remember the salient features of the input data; it will not remember the input data as a whole.

That means the encoding decoding scheme, which is learnt by the autoencoder should be generalizable ok. It is not that whatever input is fed it will simply remember or memorize that input data. So, for doing that in earlier two versions in over complete autoencoder and sparse autoencoder, in case of over complete autoencoder we had put a constraint on the number of nodes in the hidden layer. So, that the autoencoder is forced to learn a compressed domain representation.

In case of sparse autoencoder what we had done is we had put a sparsity constraint to ensure that only a few number of nodes in the hidden layer, they are active for a given data a type of input data not all of them ok. So; that means, individual nodes in the hidden layer they learn or they are active for particular type of features in the input data ok. But, in case of denoising autoencoder what you are doing is we are training the autoencoder with a corrupt version or a perturbed version of the original data. Whereas, we expect that the output from the autoencoder should be the original uncorrupted data.

So, now because your input and output they are different right. Input is a corrupted data, output is uncorrupted original data. So, because the input and output they are different and using this pair the autoencoder is trained the autoencoder cannot memorize that input data anymore, because the input data and the target they are different they are not same. And in effect what the autoencoder does is the autoencoder model learns a vector field. A vector field, that maps the input data towards a low dimensional manifold.

So, we will come to what is manifold, we will see what is manifold and we will see that how the autoencoder learns this vector field, so, that the input data will be pushed towards a point on a low dimensional manifold. So, we will come back to this denoising autoencoder and other versions of the autoencoder in our subsequent lectures.

Thank you.