

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

Lecture – 33
Autoencoder Variants II

Hello, welcome to the NPTEL online certification course on Deep Learning. So, we are discussing about the Autoencoder or the different types of autoencoder. So, in the previous lecture we have talked about what is known as sparse autoencoder and as we have said, that in case of autoencoder we want the autoencoder to learn a compressed domain representation of the input data or the autoencoder should learn salient features of the input data, it should not just memorize the input data.

So, in case of sparse autoencoder where we have assumed that the number of nodes in the hidden layer can be quite large or in some cases, it is it can even be larger than the number of nodes in the input layer or the output layer, but the compressed domain representation is enforced, by enforcing some constraint which we have said as sparsity constraint. And, what is sparsity constraint? In sparsity constraint we have seen that what we have restricted is the average activation of a node or a neuron in the hidden layer.

So, we have said that if $\hat{\sigma}_j$ is the average activation of the j th node in the hidden layer, then this average activation should be constraint or it should be equal to some sparsity values, sparsity parameter, which is equal to ρ , where ρ can be say 0.1, 0.0, 1.2 and so on a very small value depending upon the extent of sparsity that we want to impose.

So, for a given type of input data, if the sparsity constraint or the sparsity parameter is very low say, for example, 0.01, then the number of nodes, which will be active in the hidden layer for a given type of input is very small and most of the hidden layer nodes will be inactive. If, we increase this sparsity parameter, that is make the value of ρ instead of 0.01 say make it 0.2, then the number of nodes which will be active in the hidden layer, for the same input data will be larger than before.

So, if the sparsity parameter is 0.01, if the number of nodes which are active is the m , if I make the sparsity parameter to be 0.5, the number of nodes which become active is n ,

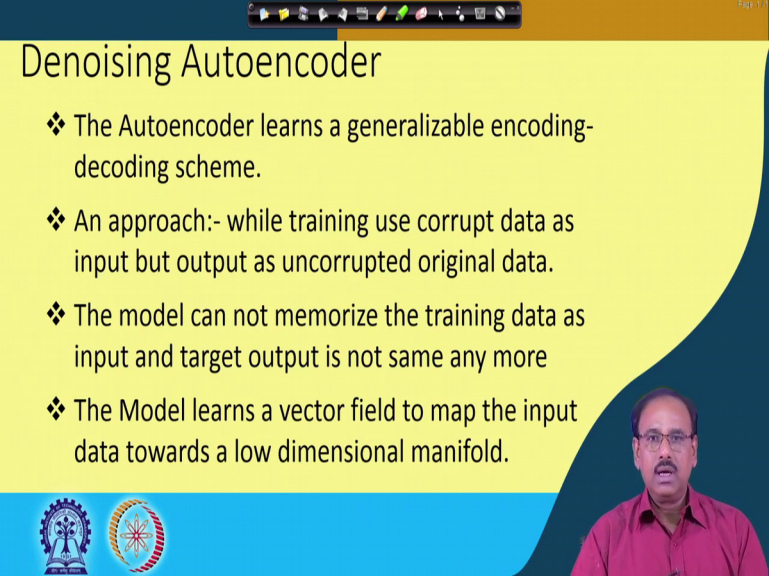
then it is quite obvious that n will be greater than m . So, depending upon the sparsity constraint or the value of the sparsity parameter that you specify you are controlling the number of nodes, which are to be which will be active in the hidden layer ok. So, in the process, the network learns a generalizable encoding decoding scheme, it just not simply memorize the input data for reproduction later by the decoder part.

(Refer Slide Time: 03:59)



So, today or in this lecture what we are going to discuss about denoising autoencoder and we will also discuss about contractive autoencoder and later on we will try to see some applications of the auto encoder.

(Refer Slide Time: 04:13)



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.

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

So, what is denoising auto encoder? So, as we said that an autoencoder learns a generalizable encoding decoding scheme, it does not simply memorize the input data. And, in the earlier case in earlier two types of auto encoders. In case of under complete autoencoder, that was done by restricting the number of nodes of the input layer. In case of sparse autoencoder, it was achieved that is generalizable learning or encoding decoding scheme was achieved by introducing a sparsity constraint, for the activation function of the hidden layer nodes.

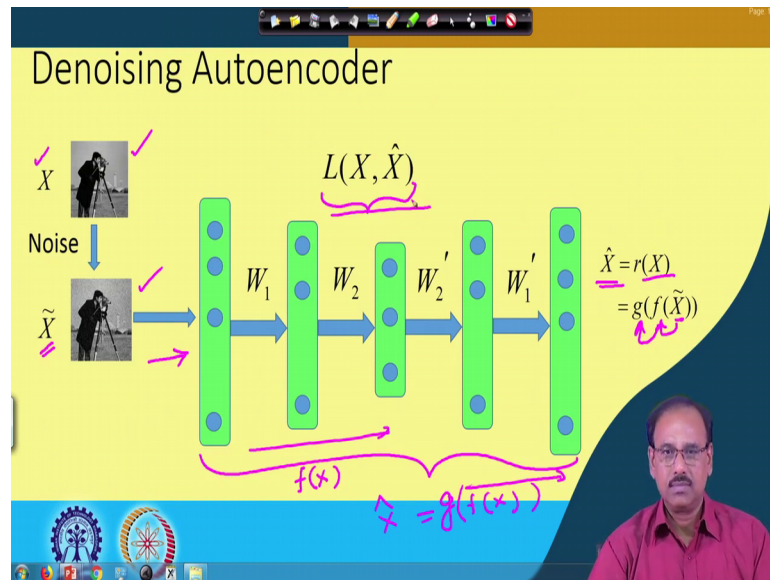
In case of denoising autoencoder, what you do is while training the neural network your the training data is corrupted or it is perturbed and the perturbed data is fed to the input of the autoencoder. Whereas, we expect that the output of the autoencoder should be the original unperturbed data.

So, now for training purpose, because you are feeding the input data which is a perturbed one or a noisy one whereas, I want the output data to be the original input or the original data training data. So, now, there is a difference between input and output. The input is perturbed data or noisy data output is original data.

So, because there is difference between input and output, this autoencoder or denoising autoencoder cannot simply memorize the input data, because while learning the input data and output data they are different. The input is the perturbed one, the output is the original one.

So, it cannot simply memorize the input data. Rather in this training while being trained, what the autoencoder learns is a vector field. So, it is a vector field that will map the input data towards a low dimensional manifold. So, we will come to what is a low dimensional manifold a bit later.

(Refer Slide Time: 06:29)



So, the training process is something like this. So, here what I have shown is say, this is my autoencoder, where this portion is the encoder part. And, if my input vector is a some X , then what this encoder does is it implements an encoding function, which is the $f X$. And, this part is decoder part the decoder part decodes this encoded data.

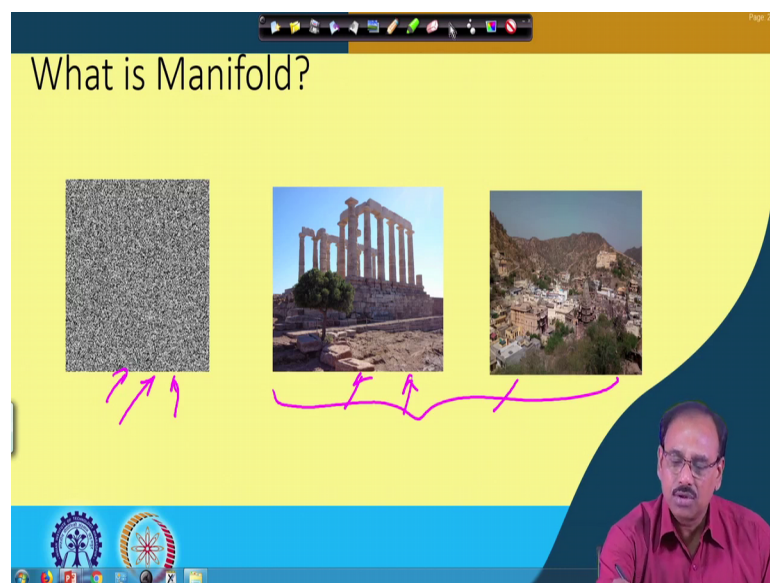
So, if I put that this function is g . So, decoder performs g of $f x$ and that is my reconstructed data x hat right. So, in case of this denoising autoencoder what we are doing is suppose my original input data or original training data is x , I am adding noise to it or perturbing this input data by a noise process or by perturbation process, to get a noisy data which is X tilde.

And, this X tilde is fed to the input of the autoencoder. And, what I want at the output is this X hat, where I want that this X hat should be this original X not X tilde; that means, during this back propagation learning the loss function, that I want to minimize is not X tilde, X hat, but the loss function that I want to minimize is what is $L X X$ hat. Where X is the original data and X hat is the reconstructed data, or reconstruct reconstructed X , which is X hat encoded, then decoded.

So, I want that \hat{X} should be equal to $g \circ f$ of \tilde{X} where \tilde{X} is the corrupt data or the perturbed data and I expect that this reconstructed data, should be same as my original data X . So, the loss function we are minimizing is $L(X, \hat{X})$, where this $L(X, \hat{X})$ can be a squared error loss, but that is what we want to minimize.

So, this is what this denoising autoencoder does. And, we have said that while doing. So, the denoising autoencoder learns a vector field, a vector field that maps an input data to a point on a low dimensional manifold.

(Refer Slide Time: 09:21)



So, let us see what is a manifold? Now, before coming to this figure just in simple term so in any particular residential area, whenever we try to give the address of a building. A building which is on a road say for example.

Student: (Refer Time: 09:42).

You name any road and we give a building number on that particular, road say a road may be Bidhan Sarani.

Student: Oh.

And, we are simply specifying say 51 Bidhan Sarani. So, the moment you say 51 Bidhan Sarani a person reaching Bidhan Sarani, that particular road can identify where that 51 Bidhan Sarani is. So, this address that you are giving is a 1 dimensional address right. On

the road you move in one direction come to the building which is number 51. So, 51 Bidhan Sarani uniquely identifies, your building on the road. So, this is an one dimensional address.

But, the building is not an 1 dimensional, in a 1 dimensional space. The building is actually in a 2 dimensional space. And to be more specific the building is in a 3 dimensional space, because the surface on which we are residing, this surface is part of a sphere or earth is nothing, but spherical approximately.

So, this surface is in 3D, but mostly we approximate that in 2D, because in a very smaller region this can be considered to be a 2D surface, but again so, every building or every place on this surface actually has a 2 dimensional address. I should specify the latitude longitude, but instead of doing that I am simply giving the building number on a road which is an 1 dimensional address.

So, that is, what is a manifold representation. A manifold is actually a very dense space region embedded in a high dimensional space, where in most of the input data, of the most of the sensible data resides. So, let us take another example with the set of these images that we have shown. Suppose I want to create an image of dimension m by n . So, as we told in our previous lectures that an image is nothing, but a vector into m into n dimensional space ok. So, there are large number of vectors in that m into n dimensional space.

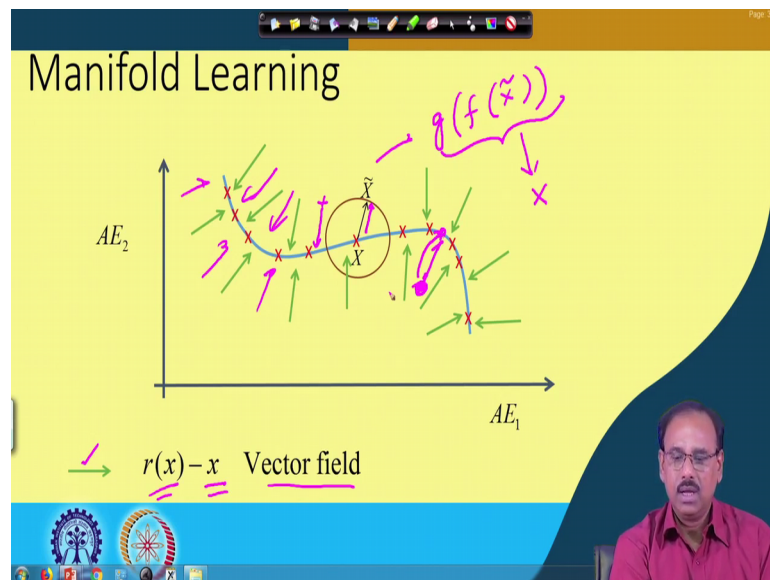
So, if I try to generate any image in that m into n dimensional space by choosing every pixel in that m by n image at random. You find that in majority of the cases or in most of the cases. The kind of image that will be generated is this kind of image, as we are creating every pixel in the image at random.

So, in the majority or most of the cases the kind of image that will be generated is this. And, the kind of image like this, or the kind of image like this, getting such a type of image by this random selection of pixel values is almost negligible. It may be or may not even be 0.00001 percent, which clearly says that I have an image space, which is m into n dimensional image space, where every possible image of size m into n , m by n is a point. If I choose a point at random, it is very likely that an image I will choose is this. It is very very unlikely that I will have I will get an image like this.

So, that clearly shows that most of the space or most of the volume in that m into n dimensional space represents images of this form, images of this form, not of this form, or a sensible image or a set of sensible images like this, occupies a very dense low dimensional space, which is embedded in the high dimensional space, which is m into n dimensional space.

And, this low dimensional space, which is very dense where in most of the input data or sensible images in this case they reside, this is what is manifold. ok. So, in case of autoencoder, when we are representing input data in a compressed domain representation. This compressed domain representation is nothing but a set of points on such manifolds right.

(Refer Slide Time: 14:57)



So, given this now, let us see that, what is this manifold learning? So, as we have said that for training this denoising autoencoder, I am feeding a data, I am using a training data X , it is quite likely that this training data X will belong to a manifold or it will be taken from a manifold. Then, I am disturbing this data or perturbing this data to make it data X tilde, which is the perturbed data.

So, in this case through a perturbation process or noisy process, so, this data X is perturbed to X tilde. And, using this I am training the autoencoder and what the autoencoder does is while training, my autoencoder output is as we have said before g of f of X tilde and I want that this g of X of X tilde should be my X not X tilde.

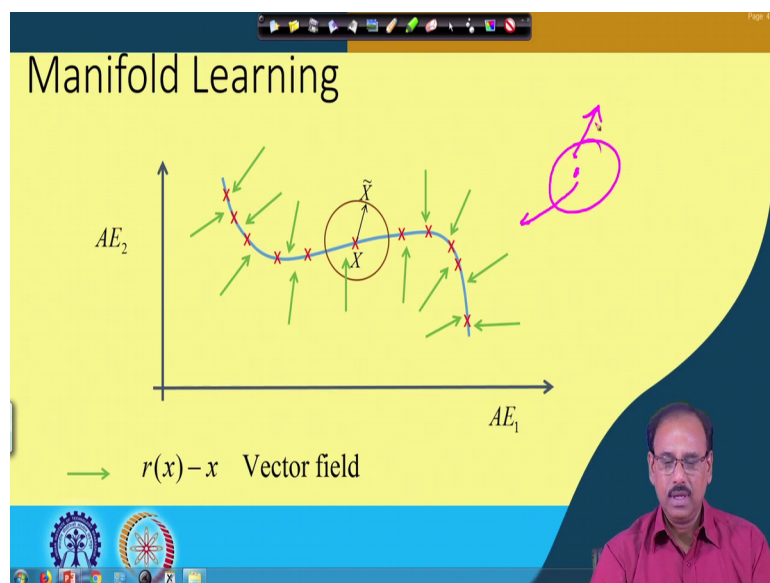
So, while training by this gradient descent procedure or back propagation learning effectively the autoencoder learns a vector field. The vector field is that given any point over here the autoencoder will try to move this data in a direction which is a nearest point on this manifold.

So, this reconstructed \tilde{x} minus my original x that is what is the vector field. So, over here this vector field or a vector field is represented by arrows which are in green color. So, I have vector fields like this over here. So, if I give any training vector somewhere over here, as the vector field in is in this direction and it is moving towards the manifold.

So, this data which is given over here will ultimately converge or ultimately be represented by a point on this particular manifold, which may be nearest to this input data that was given, this is what is known as manifold learning. So, the advantage of denoising autoencoder that we have just seen is that it prevents the autoencoder from simply memorizing the input data, because for training the training pair the input and output data is different.

But, in the process of training the autoencoder learns a vector field a vector field, which helps the autoencoder to move any point in a space to a point a nearest point in a nearest point on the manifold. Of course, this is possible in a space where the autoencoder has observed some inputs.

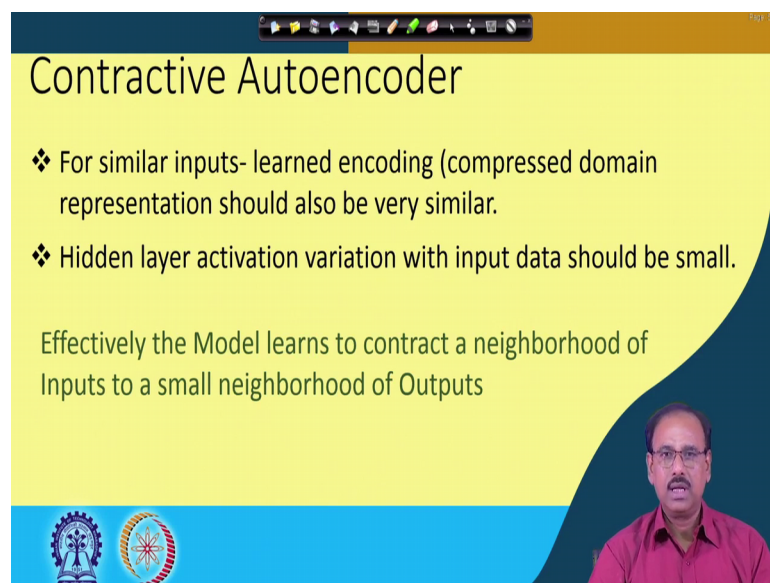
(Refer Slide Time: 18:15)



If, I have a point far away from this autoencoder and I did not have any data any training data over here. In that case my reconstruction error becomes quite large and it is quite possible that, this point will not be converged to a point on the manifold, but this point may go in the other direction as well.

So, in the region where the autoencoder has seen some training data, it learns a vector field. In the region of course, where the autoencoder have not seen a data, it does not have any idea of that particular area and it does not know what to do with that data.

(Refer Slide Time: 18:57)



Contractive Autoencoder

- ❖ For similar inputs- learned encoding (compressed domain representation should also be very similar.
- ❖ Hidden layer activation variation with input data should be small.

Effectively the Model learns to contract a neighborhood of Inputs to a small neighborhood of Outputs

So, that is what is manifold learning and a denoising autoencoder, learns a vector field which eventually moves the input data to a point on the manifold. The other kind of autoencoder that we are going to discuss is a contractive autoencoder.

The concept of contractive autoencoder is that, if the input data is very similar in that case encoded output of all those input data must also be very similar. So, that is the concept of an autoencoder. So, for similar inputs the large encoding or the compressed domain representation should also be very similar. So, this is what is the basic philosophy behind learning of a contractive autoencoder.

So, you know order to do this, what I have to impose is that the activation or the variation of the activations in the hidden layer activation of the nodes in the hidden layer, the variation of these activations with respect to input data should be small. That is if the

input data should be given to input data, if they are similar there may be slight variation, the activations in the in the hidden layer that would all should also be very very similar ok.

So, by doing this what such a model or the contractive autoencoder learns is it learns to contract a neighborhood of input to a small neighborhood of output right. So, as similar inputs are encoded in a similar way and when you decode that the decoder output will also be very very similar ok.

So, this ensures that similar inputs at the input or in a small neighborhood at the input will also be contracted to a very small neighborhood at the output that is what this contractive autoencoder does. And, as we are saying that the kind of regularization that we should impose in case of contractive autoencoder is that, the variation of the hidden layer activation should be small with respect to the input data.

(Refer Slide Time: 21:35)

The slide titled "Regularization" contains the following content:

- On the left, the Frobenius norm formula: $\|A\|_F = \sqrt{\sum_{j=1}^m \sum_{i=1}^{N_i} |a_{ij}|^2}$. A handwritten note "N x m" is written below it.
- In the center, the Jacobian matrix J is shown as a grid of partial derivatives:

$$J = \begin{bmatrix} \frac{\partial a_1^h(X)}{\partial x_1} & \frac{\partial a_1^h(X)}{\partial x_2} & \dots & \frac{\partial a_1^h(X)}{\partial x_m} \\ \frac{\partial a_2^h(X)}{\partial x_1} & \frac{\partial a_2^h(X)}{\partial x_2} & \dots & \frac{\partial a_2^h(X)}{\partial x_m} \\ \dots & \dots & \dots & \dots \\ \frac{\partial a_{N_h}^h(X)}{\partial x_1} & \frac{\partial a_{N_h}^h(X)}{\partial x_2} & \dots & \frac{\partial a_{N_h}^h(X)}{\partial x_m} \end{bmatrix}$$
- At the bottom, the loss function with a regularization term: $L(X, \hat{X}) + \lambda \sum_{i=1}^{N_h} \|\nabla_X a_i^h(X)\|^2$. The regularization term is circled in pink.

And that can be imposed by introducing a regularization again here the regularization is on activations not regularization on the weight values, which has done in case of multi-layer perceptron. So, here the regularization is on the activations. And, the regularization term can be something like minimization of the Frobenius norm of the Jacobian matrix of the activation functions in the detector.

So, what is Frobenius norm? Frobenius norm of a matrix A as shown over here, if my matrix is A , then Frobenius norm is $\sqrt{\sum_{i,j} a_{ij}^2}$ where a_{ij} is the i th j th element of the matrix A , take the sum of all these squared elements over all the elements. That is summation over all i and summation over all j and take the square root of this, that is what is Frobenius norm. And, the regularization term in case of contractive autoencoder can be squared Frobenius norm of the Jacobian matrix.

Now, what is the Jacobian matrix? As we have seen that when we have taken gradient of the loss function; the loss function over the scalar our weights were vectors right. So, when you take the gradient of the scalar function with respect to your vector the gradient becomes a vector. Over here for every hidden layer node, I have input which is a vector or a data vector. And, if I have multiple nodes in the hidden layer, then the activations taken together they are also vectors.

So, if I take the gradient of this vector with respect to input vector, it is a gradient of scalar with respect to a vector that becomes a vector. So, gradient of a vector with respect to vector should become a matrix. So, and that is what is Jacobian matrix. So, Jacobian matrix actually captures the variation of different components of the output vector with respect to the input vector.

So, the Jacobian matrix in this case is defined this way, suppose I have N_h number of nodes in the hidden layer. So, I will have N_h number of activation values, 1 activation value of every hidden layer node. And, by input data vector is the m dimensional there are m components of the input data vector. So, the Jacobian matrix simply becomes like this, a $N_h \times m$ which is the activation of the first node in the hidden layer for input vector X .

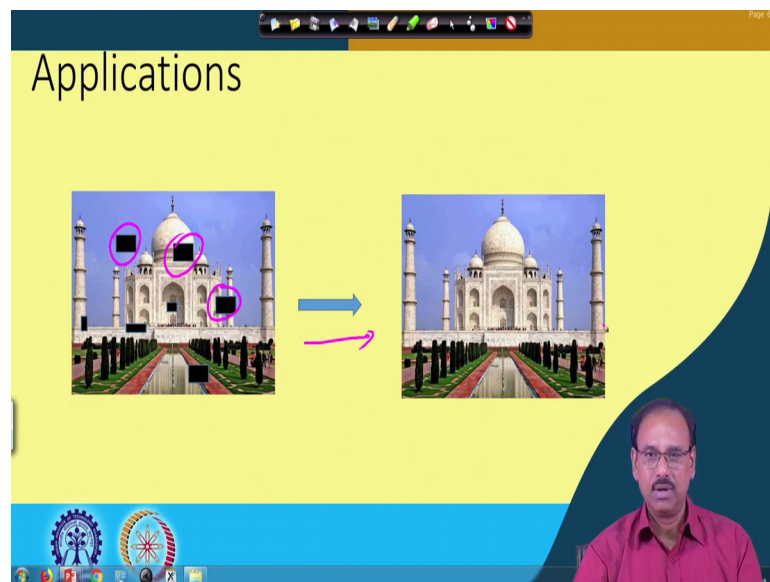
So, it is $\frac{\partial a_1}{\partial X_1}, \frac{\partial a_1}{\partial X_2}, \dots, \frac{\partial a_1}{\partial X_m}$ and so on. So, the first row actually gives you the gradient of the output of the first node in the hidden layer with respect to the input vector. Second row tells you the gradient of the output of the second node of the in the hidden layer with respect to the input vector X and so on.

Similarly, last row gives you the gradient of the output of the N_h th node in the hidden layer with respect to the input vector. And, all of them together gives you what is the Jacobian matrix. And, my regularization term now becomes a squared Frobenius norm of this Jacobian matrix.

So, you find that this del X a i h X all of them together are nothing, but components of this Jacobian matrix. So, this is, what is my regularization term. So, my back propagation learning algorithm now should try to minimize this along with the data loss that I have which is $L(X, \hat{X})$ right.

So, this regularization term ensures that variation of the activations in the hidden layer nodes with respect to the input will be very small minimum, once the network is learned property. And, that is what is our contractive auto encoder. And, as we said that this contractive autoencoder tries to contract a neighborhood of the input data to a small neighborhood of the output data.

(Refer Slide Time: 26:45)



So, given this let us now try to see, what are some possible applications of such auto encoder. One of the application is what is called in painting. The in painting says that, if I have an original image where the original images corrupt in the sense that some region I do not have any information in some region, or the information contained in some of the regions some smaller regions is totally corrupted. So, you find that because autoencoder learns a salient features of the image and from the salient features the decoder part can reconstruct the image.

So, it is possible that autoencoder will be able to remove such corrupted regions from the input image, which is the in painting as this shown here. So, here these black regions as shown these black regions are another missing regions and by in painting you can cover

those black regions. So, this is one of the applications where the autoencoder can be successfully applied.

The auto encoders can also be applied in abnormality detection. For example, I have a surveillance camera; the purpose is to detect any abnormal event. Say for example, I have a pedestrian road where only the pedestrians are supposed to move there should not be any car on the pedestrian road.

So, when I train the autoencoder I train the autoencoder with only normal sequences, where only the pedestrians are present. And, then when I impulse this autoencoder in the test sequence if any car comes on that pedestrian road, because while training the autoencoder has not seen the car. So, the car cannot be properly reconstructed by the autoencoder.

So, on the output side if I have if I can identify regions of the image frames, where the reconstruction error is very very large as we have said that car cannot be properly represented cannot be properly reconstructed, as car was not seen during training. Pedestrians were seen during training, so pedestrians will be properly constructed, but not the car.

So, in that particular region the reconstruction error where the car is present is very very high because it is not properly reconstructed. So, in the output video sequence whichever region I find that the reconstruction error is very large, I can identify that in those regions we have some abnormal activity taking place ok. So, this trained auto encoders can also be used for such abnormality or abnormal activity detection.

And, as it is obvious from the denoising autoencoder that auto encoder; obviously, can also be used for filtering purpose that is removal of the noise. So, there are many such applications where auto encoders can be applied. So, we will talk about other models of the deep learning other types of neural networks in our subsequent lectures. So, let us stop today.

Thank you.