

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

Lecture - 29
Autoencoder Vs. PCA I

Hello, welcome to the NPTEL online certification course on Deep Learning. So, since our previous class we have started discussion on Auto encoders. So, what we discussed yesterday or in our previous class is what is an Autoencoder and a particular variant of auto encoder that we have introduced is what is known as under complete autoencoder.

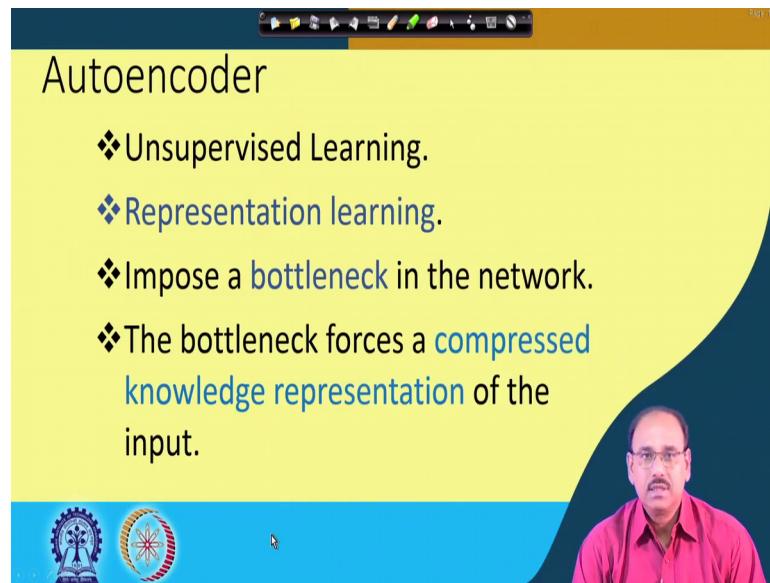
(Refer Slide Time: 00:45)



Today in this lecture we will discuss about the auto encoder versus principal component analysis that is whether principal components and the auto encoder outputs they are related, if that related how they are related what is the similarity and what is the dissimilarity between these two.

And then we saw in subsequent lectures we will discuss about other Autoencoder topics, like training Autoencoders, Sparse Autoencoder, Denoising Autoencoder, Contractive Autoencoder, Convolution Autoencoder and all that.

(Refer Slide Time: 01:37)



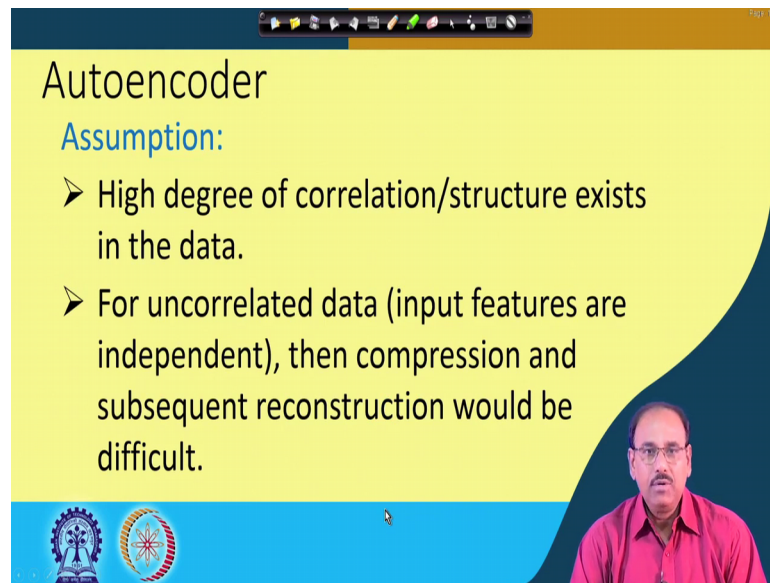
Autoencoder

- ❖ Unsupervised Learning.
- ❖ Representation learning.
- ❖ Impose a **bottleneck** in the network.
- ❖ The bottleneck forces a **compressed knowledge representation** of the input.

So, before we start today's topic on Autoencoder versus PCA, let us just briefly recapitulate what we have discussed in our previous class. So, we have said that auto encoder is an unsupervised learning technique. So, a learning technique which forces the feed forward or deep neural networks to learn what is known as the representation learning. That is given an input vector or an input signal the network or auto encoder learns compressed domain representation or learns a structure which is present in the input data.

And the way in the neural network of the auto encoder learns this representation data representation of data structure is by imposing a bottleneck layer in the network. And this bottleneck layer actually forces a compressed knowledge representation of the input and that is what the auto encoder learns and this compressed domain knowledge representation is subsequently used for other applications.

(Refer Slide Time: 02:57)



Autoencoder

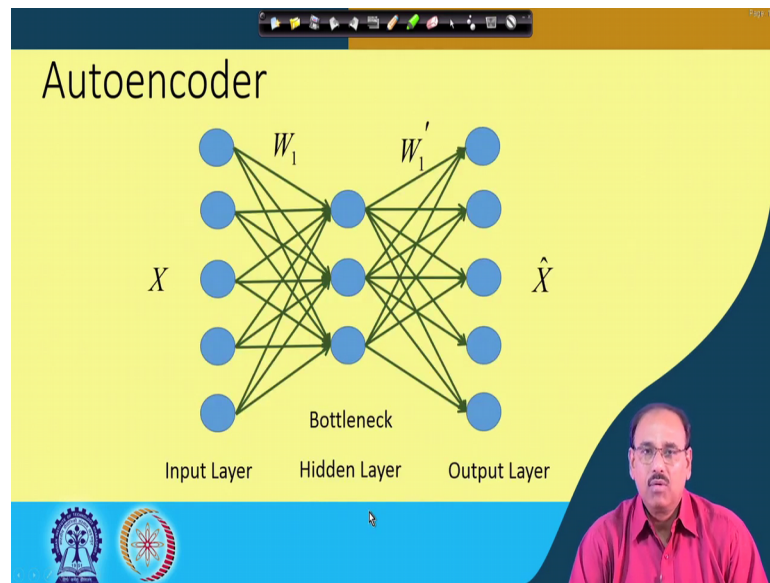
Assumption:

- 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, while doing this we assume something, the assumption is the degree of correlation or the structure that exists in the input data is quite high. And in fact if the input data or the input feature vectors are uncorrelated or they are statistically independent then compression and subsequent reconstruction would be difficult of course, we will be able to compress. But in the compression will be highly lossy compression if there is no redundancy because whatever information is present in the input data unless there is redundancy of there is correlation then going for any sort of compression leads to a loss of data.

And once in that compressed domain representation the data or the information is lost whichever way I try to reconstruct my signal, the original signal from that lossy compressed representation my output will always be lossy. That means, the decompressed data or reconstructed data cannot be identical to the input. So, the basic assumption in use of auto encoders, when you go for encoding in compressed domain or representation in compressed domain the basic assumption is the data is highly correlated.

(Refer Slide Time: 04:19)



So, based on this we have seen a basic auto encoder architecture which is something like this that you have an input layer you have an output layer. So, input layer actually accepts the input data.

So, if the dimensionality of the input data is n at the input layer I will have n number of nodes, in addition there will be one more node to take care of the bias. And in fact we have seen earlier that addition of this bias in the input vector allows us to go for an unified fixed representation, that is the bias term can be taken can be considered as an additional term in the weight vector.

So, number of nodes in the input layer will be n plus 1, if the input data vector has dimensionality n . Similarly in the output layer which reconstructs the input as \hat{X} , so our input is X and the output is \hat{X} . So, the output layer will consist of n number of nodes, because we want that the input X should be reconstructed at the output.

And in case of a basic model of an auto encoder we have a hidden layer in between input layer and output layer and what we have said in case of under complete autoencoder that the number of nodes in the hidden layer is much less is less than the number of nodes in the input layer or the number of nodes in the output layer. So, this is what he is known as a bottleneck layer. So, in bottleneck layer as the number of nodes is less than the input layer nodes. So, what this network does is the network passes the input information through a restricted layer whether the number of nodes is must much less and then

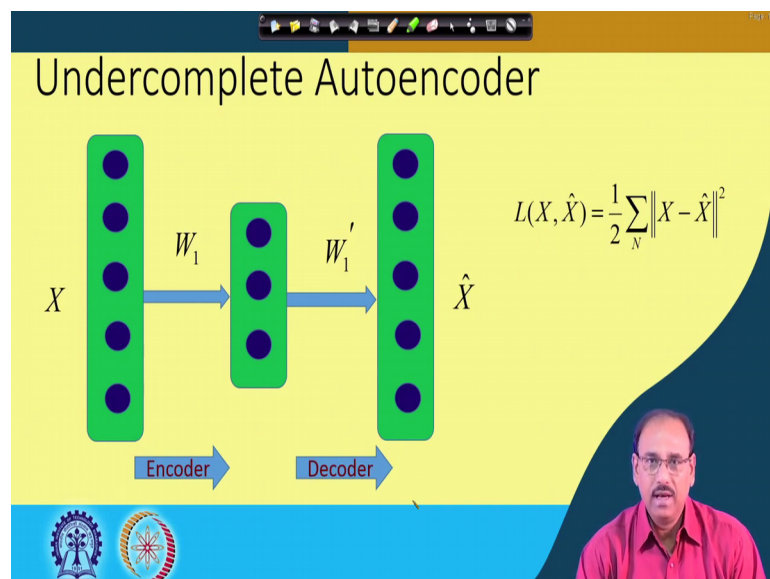
subsequently as this information passes through this restricted layer, then the decoder side that is the output layer tries to reconstruct the original input from this restricted output.

So, as the information passes through this restricted layer, the network tries to learn a compressed domain representation of the data. So, you imagine what will happen if I do not have this bottleneck layer, that is if the number of nodes in the hidden layer is same as the number of nodes in the input layer that is the size of the data or even more than the number of nodes in the input layer.

In that case it might be possible that the network will simply learn an identity function, that is given an input it goes to an intermediate representation and then knows how to reconstruct the same output. And in the process if I have large number of nodes in the hidden layer the network eventually may not learn the compressed domain representation or the structure present in the data which is not our m . So, that is the reason that in the hidden layer or in the bottom neck layer you put some restriction on the number of nodes that you can have.

Later on we will see that when we talk about the sparse auto encoder that it is not even necessary to have the restriction on the number of nodes. But we can add some other regularization term where the red node activations will be restricted. So, instead of trying to restrict the number of nodes you try to restrict the node activations.

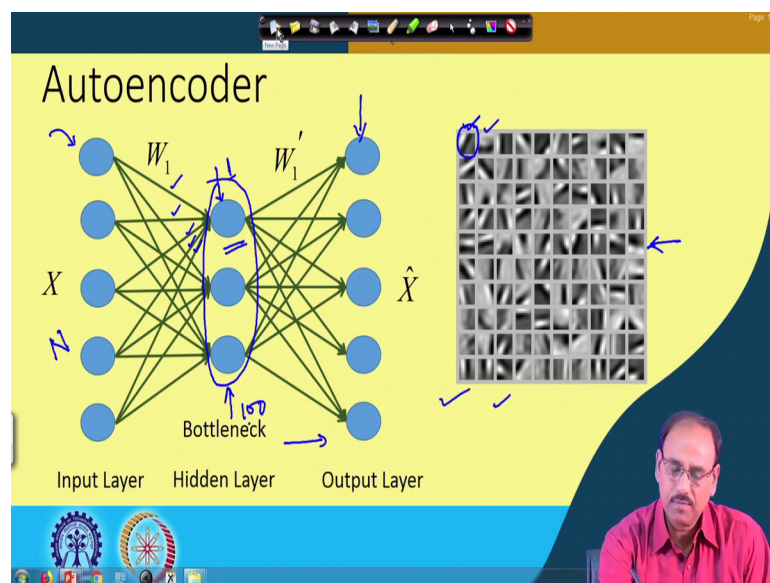
(Refer Slide Time: 08:07)



So, this is the basic structure of an auto encoder and as we said that all subsequent representations, we will use this form of representation diagram to represent an autoencoder. So, I have an encoder part which is from input layer to the hidden layer and I have a decoder part which is from the decoder to the output layer and this whole thing taken together the encoder decoder together is known as Auto encoder. And as it is an under complete auto encoder that we are trying to depict the number of nodes in the hidden layer, which is the bottleneck layer is less than the number of nodes in the input layer or the number of nodes in the output layer.

And while training this auto encoder the loss function that you try to minimize is the squared error loss between the input and output. So, that is $\|X - \hat{X}\|_2^2$ norm of that $X - \hat{X}$ square take the summation over all the data points all the training samples that you are feeding for training this network. So, this is the basic structure of an auto encoder which has got one input layer one output layer with a hidden layer or a bottleneck layer in between.

(Refer Slide Time: 09:17)



Now, when you go for; so what does this auto encoder try to learn? So, here what has been shown is that if I feed an input to an autoencoder which is an image. So, in our case x is an image and the output that is \hat{x} which is reconstructed is also an image right. And in ideal case if the auto encoder is properly trained then \hat{X} will be same as X . Now, once this auto encoder is properly trained, what does this encoding layer or the

bottleneck layer actually learn? So, this is an example which has been obtained from training such an autoencoder with large number of input images.

So, you find that on the right hand side the example image set that we have. So, this particular image this particular sub image is actually the image or the structure which is learnt by the first auto encoder. Now, here you find that this output is not exactly from this particular network that we are showing here, here you find that there are 100 such sub images or hundred structures. That means, the auto encoder which has been trained with for this kind of a example has 100 number of nodes in the middle layer or in the bottleneck layer and every node learns some structure.

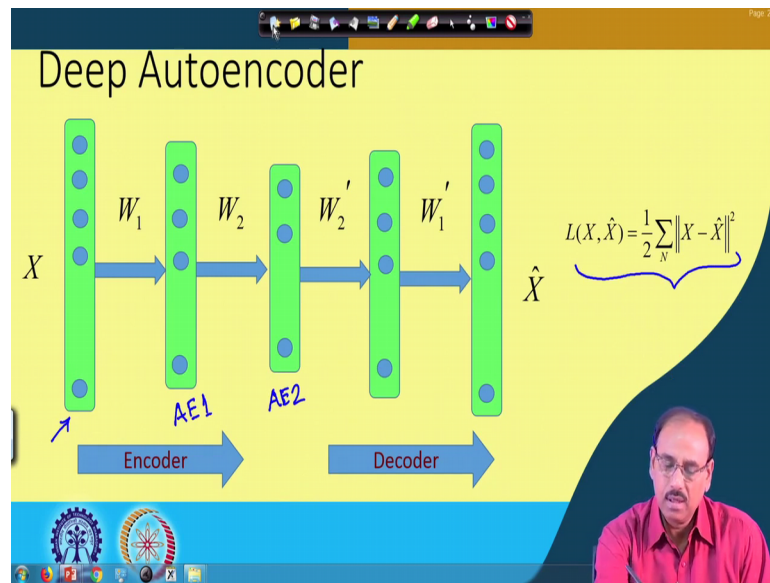
So, this first node it learns this structure similarly second node may learn this structure and so on and how this image has been formed. It is nothing but this weights from the input which are connected to the first node, you remember the way we have got this vector is by concatenating the columns of the input image.

So, when you form these structures when I reconstruct the structures which are learnt by these hidden layer nodes these are these weight vectors which are folded back in the form of an image ok. So, you find that if there are n number of nodes in the input. So, I have an image consisting of n number of pixels, here also I have n number of vectors of course, the n plus 1 considering the bias term. Now, when you form this you remove that bias term, so among the remaining vectors remaining components of the weight vector I fold it back in the form of an image.

So, this is such an image so these are the structures as shown in this set of images which are learnt by this encoding layer and as you see over here these sub images appear to be edges oriented in various directions. So, edges are nothing but the detailed information's which are present in the image. So, this simple example shows that this input layers or nodes in the input layer actually learn the structures which are present in the image. It does not simply pass the input image to the output layer and then what this decoder side does is the decoder side use makes use of these structures which is which are present in the image and from these structures it tries to reconstruct the original image.

So, when this network is properly learned this is the form of structure which will which is actually learnt by this encoding layer right. So, this is how an encoder learns the structure which is present in the data.

(Refer Slide Time: 13:07)

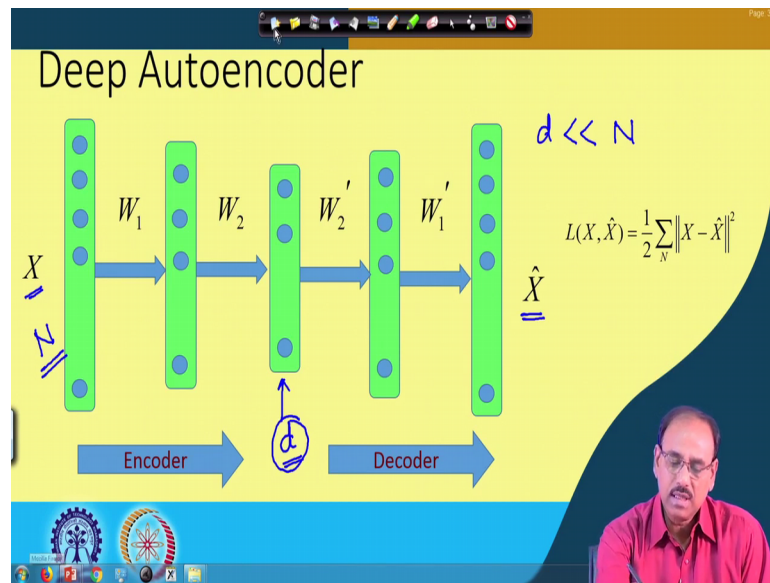


In case of Deep autoencoders, so earlier what we have shown is a basic structure of an auto encoder where I have an input layer, I have an output layer and I have one hidden layer which is a bottleneck layer. In a deep autoencoder I can have a number of such auto encoding layers which has stacked one after another. So, in this diagram what has been shown is this is your input layer this is the input layer, this is the auto encoder layer one so I put it as AE1 this is AE2 or this is actually the coding layer in this particular diagram.

I can have AE1 AE2 AE3 AE4 and so on I can go on stacking such auto encoder layers ok. Then accordingly on the decoder side also I will have stacking of a number of such decoding layers ok. So, in case of deep autoencoder the number of layers number of encoding layers and the number of decoding layers that you make part of the auto encoder that decides, what is the depth of the auto encoder that we are going to design or the auto encoder that you are going to use.

However for training the auto encoder we still use the squared error loss between the input and the output for training the auto encoder. So, given this here you find that what this auto encoder does as we have already said that given an input data of dimension say N .

(Refer Slide Time: 14:53)



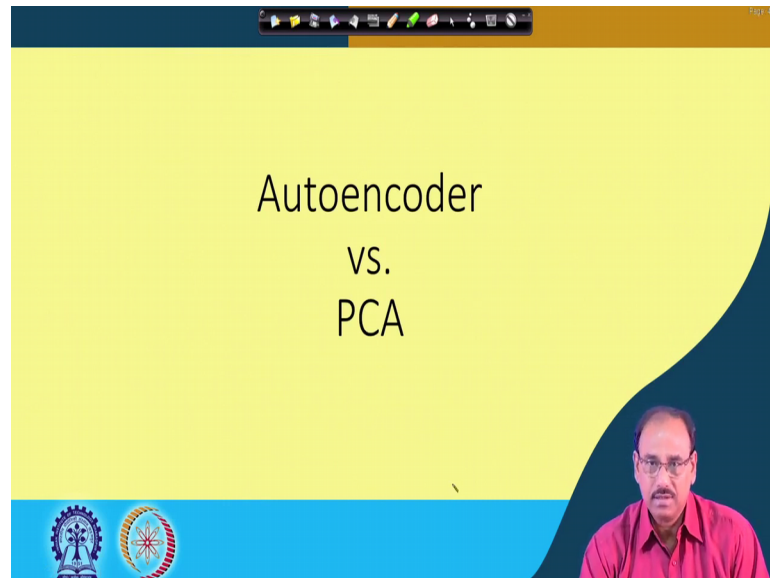
So, X having dimension N if in the encoding layer or in the bottleneck layer I have the number of nodes which is equal to d , where d is much less than N . So, here this auto encoder learns a compressed representation of this N dimensional data to a d dimensional representation, which is the latent also called as latent space representation. And it is expected that if the auto encoder is properly trained then from this latent space representation that decoder will be able to decode this latent space representation of the data to give you the reconstructed data \hat{X} , which is almost a replica of your input data X .

So, in other sense we can say that while coding the auto encoder actually gives you a transformation that transforms the data from a higher dimensional space to a lower dimensional space. And while doing so it ensures that your reconstruction error end to end reconstruction error when the data will be reconstructed that is minimized.

So, this lower dimensional representation indicates that the loss it tells that the loss that you incur while compressing the data or while trying to extract from the structure from the data the loss incurred will be minimum. So, in other case you can consider the function of the auto encoder is to go for dimensionality reduction of the input data. And if I consider the function of the auto encoder as a dimensionality reduction function, then we have to see that what is the other dimensionality reduction function that we have and

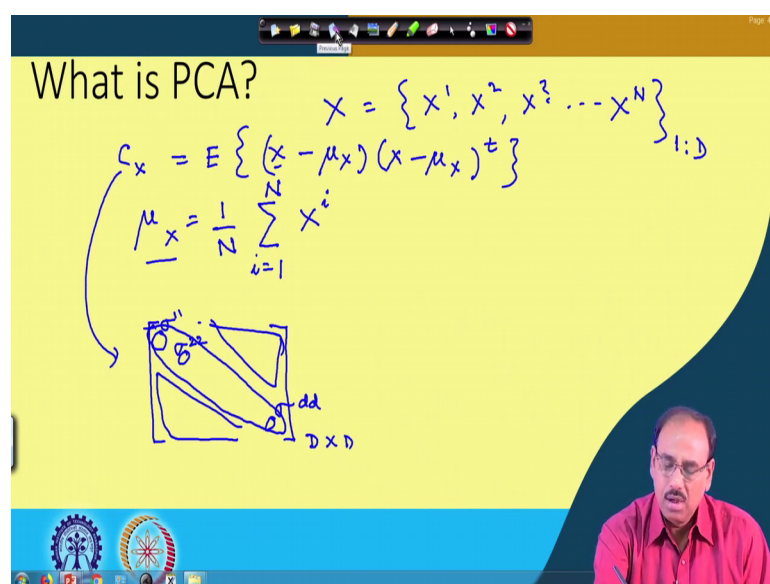
it is known that traditionally the dimensionality reduction is done by an algorithm known as principal component analysis.

(Refer Slide Time: 16:57)



So, naturally then the question comes that how does principal component analysis compared with auto encoders? So, in order to do that before going to that comparison for the benefit of those who does not know what is principal component analysis. Let me briefly say what is principal component analysis?

(Refer Slide Time: 17:29)



So, in case of principal component analysis our input is let us assume that input is a set of vectors X . So, I put this as set of vectors $X_1 X_2 X_3$ up to say X, X_N . So, assuming that we have N number of input vectors and each of the input vector maybe of dimension say let me put as capital D . So, capital D is the dimension of the input vectors. That means, each of $x_1 x_2$ up to x_n each of them has t number of components.

Now, once I have such a collection of vectors X you define the covariance matrix as C_x which is defined as expectation value of X minus μ_X into X minus μ_X transpose. What is μ_X ? μ_X is nothing but mean of the input vectors. So, I have N number of input vectors. So, this will be $\frac{1}{N}$ sum of X_i , where i varies from 1 to N . So, this is my μ_X and X is each of these individual vectors.

So, I define the covariance matrix of the set of input vectors as the expectation value of X minus μ_X into X minus μ_X transpose and now if you analyze this covariance matrix. So, what will be the size of this covariance matrix as the vector is N dimensional. So, this covariance matrix will be D by D matrix as D is the dimension of the feature vectors.

So, this covariance matrix will be a D by D matrix and in this covariance matrix the diagonal elements will give you the variance of the individual components of the vectors. That means, if I take the first vector, first component of X_1 , first component of X_2 , first component of X_3 and so on and I compute the variance of all those first components that variance will be my σ_{11} which is the first component in this diagonal vector. Similarly σ_{22} will be the variance of the second component σ_{dd} will be the variance of the d th component of the last component. And all the off diagonal elements in this matrix will give you the covariance of different components.

So, σ_{12} is the covariance between the first component and the second component, σ_{45} is the covariance between the fourth component and fifth component and so on, so this is what is your covariance matrix. So, once I have this covariance matrix then from the covariance matrix I can compute the eigenvalues and the eigenvectors. So, suppose how do you compute the eigenvalues and the eigenvectors given a covariance matrix C_x ?

(Refer Slide Time: 21:21)

What is PCA?

$$C_x = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \dots & \sigma_{1d} \\ \sigma_{21} & \sigma_{22} & \dots & \sigma_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{d1} & \sigma_{d2} & \dots & \sigma_{dd} \end{bmatrix}$$
$$C_x e_i = \lambda_i e_i$$
$$\begin{vmatrix} \sigma_{11} - \lambda & \sigma_{12} & \dots & \sigma_{1d} \\ \sigma_{21} & \sigma_{22} - \lambda & \dots & \dots \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{d1} & \sigma_{d2} & \dots & \sigma_{dd} - \lambda \end{vmatrix} = 0$$

$\lambda_i : i=1 \dots d$
 e_i

Having say sigma 1 1 sigma one two up to sigma one d sigma 2 1 sigma 2 2 up to sigma 2 d and so on. This is sigma d 1 sigma d 2 up to sigma d d. So, this is say my covariance matrix and I want to compute the eigenvectors e_i and the eigenvalues λ_i .

So, the way you have compute this eigenvalue is from each of these diagonal elements you subtract λ and then make determinant. So, the determinant will be $\sigma_{11} - \lambda$ sigma 1 2 up to sigma 1 d sigma 2 1 sigma 2 2 minus λ sigma 2 3 goes on sigma d 1 sigma d 2 sigma d d minus λ make a determinant and equate this to 0.

So, once you put this you will find that this determinant will give you a polynomial of degree d and it will be a polynomial in λ . So, once I solve this I will get d components or t values of λ . So, I will get λ_i , where i varies from 1 to d and then for each of this λ_i I can compute the corresponding eigenvector. So, the way you compute eigenvector is if for λ_i the corresponding eigenvector is say e_i .

Then the equation that has to be satisfied is $C x e_i$ have to be equal to λ_i times e_i . So, I know what is C x I know what is λ_i you solve this equation I get the i th eigenvector which is e_i . So, this is how given a set of vectors I can compute the covariance matrix, from the covariance matrix I can compute the eigenvectors or eigenvalues and for every eigenvalue I can compute the Eigen corresponding eigenvector.

And you see that if this covariance matrix is a real and symmetric which usually is then the eigenvectors are orthogonal and what this lambda tells you or the eigenvalue tells you it simply tells you that what is the scatter or what is the variation of that data in the direction of the corresponding eigenvector. So, if lambda 1 is very high that in k that indicates that the variation of data in the direction of the corresponding eigenvector which is e 1.

So, lambda one very high indicates that the variation of data in the direction of e 1 is very high right. So, given this once I have this eigenvectors then I can define a transformation. So, how do you define this transformation?

(Refer Slide Time: 24:51)

The slide contains the following handwritten content:

What is PCA?

$$A = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_d \end{bmatrix}$$

$e_1 \rightarrow \lambda_1$
 $e_2 \rightarrow \lambda_2$
 \vdots
 $e_d \rightarrow \lambda_d$

$\lambda_1 > \lambda_2$
 $\lambda_i > \lambda_j$
 $\lambda_i > \lambda_j$

$$A = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}_{2 \times d}$$

$$y^k = A(x^k - \mu_x) \Rightarrow KL$$

$\Rightarrow 2 \times 1$
 e_i
 y_1^k
 y_2^k

For defining a transformation you make you form a transformation matrix A, this transformation matrix A is formed using the eigenvectors as the rows in the transformation matrix. So, the first row in this transformation matrix is e 1 the second row is e 2 the last row is e d, you remember that we had d number of eigenvectors as our input vector is of dimension d.

And how I get this transformation matrix or how I arrange such eigenvectors into rows of this transformation matrix is in this transformation matrix e 1 corresponding to that my eigenvector is lambda 1 and for vector e 2 my corresponding eigenvalues the eigenvalue is lambda 1 and corresponding to this I have my eigenvalue which is lambda 2.

So, I arrange these eigenvectors as rows in this transformation matrix in descending order of the corresponding eigenvalues. So, here e_1 is the first row e_2 is the second row that indicates that I have λ_1 greater than λ_2 right. So, for two for a pair of eigenvalues say λ_i λ_j where both i and j varies from 1 to d because I will have λ_i that is from λ_1 to λ_d .

So, for this pair of eigenvalues λ_i and λ_j , if λ_i is greater than λ_j that indicates that e_i the eigenvector e_i will occupy a higher row than e_j in this transformation matrix. So, but this e_i I have the corresponding eigenvalue λ_i for e_j I have the corresponding value λ_j . So, as λ_i is greater than λ_j in this transformation matrix e_i will occupy a higher position than e_j , so that is how this transformation matrix is formed.

So, once I have this transformation matrix, then I can define a transformation, my input vectors are x . I can have a transformation which is given by A times say X_k the k th vector minus μ_j μX which is the mean of the vectors. So, this defines the transformation. So, this gives me a transform vector which is Y_k . So, for k th vector this transformation gives me a transform vector Y_k .

So, if you look at this transformation, what this transformation is doing if I take the first component of X_k . So, difference of first component of X_k and first component of μ_x right. This is transformed or the vector X_k minus μ_k is being projected onto vector e_1 , because this is nothing but the dot product of e_1 with X_k minus μX that gives that gives me the first component of Y_k .

Similarly, the dot product of X_k minus μ_x with e_2 which is the second row in my transformation matrix gives me the second component of Y_k right. So, this transformation that you get this is what is popularly known as KL transformation and I can use this KL transformation for data reduction in the sense, that if I want to reduce the dimension from d to 2. What I will do is in this transformation matrix A that I form this a instead of considering all the eigenvectors I will only consider e_1 and e_2 the eigenvectors e_1 and e_2 and the transformation will be same as this A times X_k minus μ_x , where A is now this is actually 2 by d matrix I have 2 rows and d number of columns right.

So, this is a $2 \times d$ matrix. So, when you go for this transformation this Y_k you find that it will be a 2×1 vector. That means, it is a two dimensional vector. So, just by truncation of this transformation matrix I can transform the data from n dimension to two dimension or d dimension to two dimension. So, that is what gives me a reduction in the dimensionality of the input data and this is what is popularly known as KL transformation and the components of this transform vector Y_k that you get that is Y_{k1} the first component and Y_{k2} that is the second component.

After this transformation these are what are known as principal components and the eigenvectors are the principal directions. So, effectively what you are doing is you are transforming your input data into a space which is known as Eigen space and the eigenvectors bring orthogonal the Eigen space is also orthogonal.

And the projections in the Eigen space in every Eigen direction are the principal components of the input data and by arranging the transformation matrix A in this form. That is arranging the rows as Eigenvectors in descending order of corresponding eigenvalues ensures that the error that you encounter by truncating some of the rows from the lower side, ensures that the error that you encounter will be minimum. So, let me stop here today I will take up this illustrations with principal component in the next class.

Thank you.