

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

Lecture - 30
Autoencoder Vs. PCA II

Hello welcome back to the NPTEL online certification course on Deep Learning. So, in our previous class we have started discussion on Autoencoder versus Principal Component Analysis.

(Refer Slide Time: 00:44)



So, what we had discussed in the previous class is how you get the principal components from the covariance matrix of the input data. So, let me just briefly tell you what we have discussed in the previous class. So, that our platform for discussion on comparison between autoencoder and principal components becomes complete.

(Refer Slide Time: 01:09)

What is PCA?

$$X = \{x^1, x^2, \dots, x^N\}$$
$$C_x = E \{(x - \mu_x)(x - \mu_x)^t\}$$
$$\mu_x = \frac{1}{N} \sum_{x \in X} x$$
$$\lambda_i \rightarrow i=1 \dots d$$
$$e_i \rightarrow A = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_d \end{bmatrix}$$
$$\lambda_1 > \lambda_2 > \dots > \lambda_d$$

So, what we presented in the previous class is that given a set of your input vectors X , where X we have considered to be a vector set of vectors X_1, X_2 up to say X_N , having N number of population where each of this vector X is of dimension d ; that means, there are d number of components in the feature vector. Then, first you find out the covariance matrix from this set of input vectors.

So, the covariance matrix is given by C_X is equal to expectation value of X minus μ_X into X minus μ_X transpose, where this μ_X is the mean of the input vector. So, mean μ_X is equal to sum of X over all X , where X is the set of these vectors. Once, you form this covariance matrix C_X , then for from this covariance matrix C_X is compute the eigenvectors and the eigenvalues. So, I have set of eigenvalues λ_i , where i varies from 1 to d as d is the dimensionality of the feature vectors. And for each λ_i , I have the corresponding eigenvector e_i .

And, once I had this e_i either set of eigenvectors, then we had formed our transformation matrix A , where A was formed as with the eigenvectors e 's as rows. So, the first row was e_1 , the second row was e_2 , and the last row was e_d . And, these eigenvectors were arranged in rows in such a way, that here λ_1 or the λ_1 is greater than λ_2 and so on it is greater than λ_t .

So, an eigenvector corresponding to the maximum eigenvalue is put as first element or the first row in my transformation matrix, and the eigenvector corresponding to the

minimum eigenvalue is put as the last row in the transformation matrix. So, all the eigenvectors are actually arranged in rows in order of the descending order of the corresponding eigenvalues.

(Refer Slide Time: 04:11)

What is PCA?

$$Y = A(X - \mu_x)$$

↓
KL → transformation.

$$A_{d \times d} \Rightarrow A_{p \times d} \quad p \ll d$$

$$Y = A(X - \mu_x)$$

↪ p

$$\hat{X} = A^{-1}Y + \mu_x$$

So, once I define this transformation matrix, then I have a transformation which is given by Y is equal to A into X minus μ_X . And, this transformation is what is known as $K L$ transformation. And, from the nature of this transformation, you find that X is an input vector and A is the transformation matrix, where every row in this transformation matrix of the eigenvectors. So, every component of Y is nothing, but projection of the vector X minus μ_X on to the i th eigenvector, which is the i th row in this transformation matrix.

So, i th component of my transform vector Y is nothing, but the projection of the vector X minus μ_X onto the i th eigenvalue. And, these components of the vector Y , the transform vector Y is are nothing, but my principal components. And, you will find that these principal components are nothing, but the transformation of the input vector X shifted by μ_X onto the eigenvector. So, this transformation basically gives you a mapping from the input space to an eigen space. And, as we said the eigenvectors been orthogonal, eigen space is also orthogonal.

So, effectively what is the transform and the way you obtain the data reduction is that in the transformation matrix A , where A is of dimension d by d right; it has d number of rows d number of columns. So, if I reduce the number of columns from the bottom, that

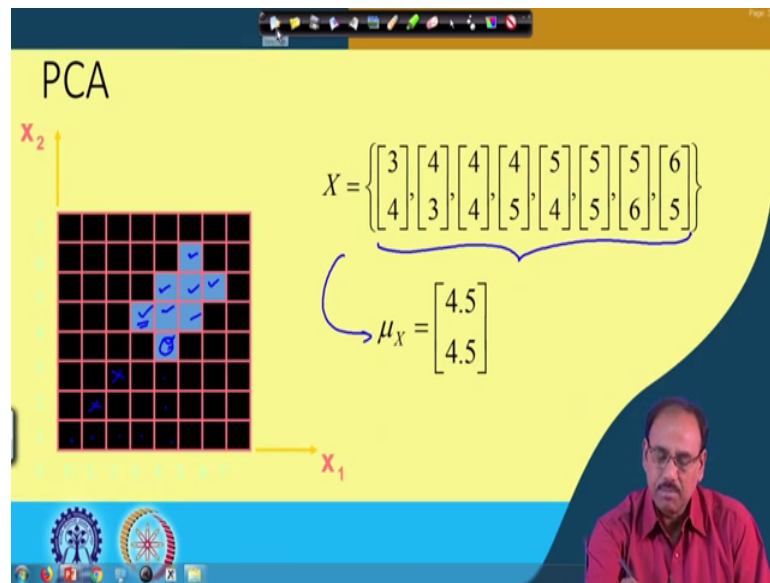
is I remove columns corresponding to minimum eigenvalues that ensures that in the reconstruction I will have minimum amount of error. So, that is the different analysis which is beyond the scope of this lecture. So, I assume that I truncate this transformation matrix A , by reducing some number of rows from the bottom.

So, what I do is I make a transformation matrix A with say p number of rows and of course, d number of columns I am not reducing the number of columns, where this p is much much less than d . So, using that when I go for this transformation Y is equal to A into X minus μX , this vector a that you get that will have p number of components. And, if I do not go for any truncation this vector a will have d number of components. So, as I reduce the number of rows p in my transformation matrix I go for reduction of the dimensionality ok.

But, this reduction is done in such a way by removing the eigenvectors in such a way, that when I try to reconstruct my original X from Y by an inverse transformation. So, from here; obviously, X will be A inverse Y plus μX . So, as in transformation matrix A , I am not retaining all the components or all the rows. So, obviously, the reconstructed X that I will have will not be actual X , but it will be an approximation of X which is \hat{X} ok.

So, this is what I get after reducing the dimensionality. So, as PCA gives you reduction in dimensionality and we have also seen that autoencoder gives you reduction in dimensionality. So, whether we can have some relation we can establish some relation between PCA and autoencoder. We will come to that a bit later, but before that let us try to see what this PCA is actually giving you.

(Refer Slide Time: 08:17)



So, to further illustrate the principal components I have taken this binary image. So, here you find that it is a binary image or all these elements, which are blue these are one; that means, I can say that I have pixels present in this location in these locations I do not have any pixel. So, I can form a vector with only the pixel locations where the pixels are present. So, accordingly my population of vectors will be given by this 3 4. So, this is 3 4 right, sorry this is 4 3 0 1 2 3 4 0 1 2 3. So, this is 3 4 3.

Similarly, this is the vector which is 3 4 and so on. So, all vector positions all the positions so; however, I have a pixel present, I take those positions as my vector population. And, from this set of vectors I compute the mean of the vectors which is μ_X and in this particular case you can compute that this mean vector will be 4.5 4.5. So, once I have this computation now as we said before that for principal components or for K I transformation I need to have the covariance matrix.

(Refer Slide Time: 09:44)

PCA

$$X = \left\{ \begin{bmatrix} 3 \\ 4 \end{bmatrix}, \begin{bmatrix} 4 \\ 3 \end{bmatrix}, \begin{bmatrix} 4 \\ 4 \end{bmatrix}, \begin{bmatrix} 4 \\ 5 \end{bmatrix}, \begin{bmatrix} 5 \\ 4 \end{bmatrix}, \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \begin{bmatrix} 5 \\ 6 \end{bmatrix}, \begin{bmatrix} 6 \\ 5 \end{bmatrix} \right\} \quad \mu_X = \begin{bmatrix} 4.5 \\ 4.5 \end{bmatrix}$$
$$(X_1 - \mu_X)(X_1 - \mu_X)' = \begin{bmatrix} -1.5 \\ -0.5 \end{bmatrix} \begin{bmatrix} -1.5 & -0.5 \end{bmatrix} = \begin{bmatrix} 2.25 & 0.75 \\ 0.75 & 0.25 \end{bmatrix}$$
$$(X_2 - \mu_X)(X_2 - \mu_X)' = \begin{bmatrix} 0.25 & 0.75 \\ 0.75 & 2.25 \end{bmatrix}$$

And, we defined covariance matrix as expectation value of X minus μ_X into X minus μ_X transpose. So, here I have a very small example. So, from here if I try to compute that covariance matrix so, the covariance matrix 4 I take this as my first vector X_1 . So, X_1 minus μ_X so, μ_X was 4.4.5 4.5, if you subtract 4.5 4.5 from 3 4 what you get is your X minus μ_X becomes minus 1.5 minus 0.5. So, X minus μ_X my into X minus μ_X transpose, if you do this multiplication, it simply becomes 2.25 0.75 0.75 and 0.25.

So, that is for the first vector in my vector population that I get. Similarly, when I compute X_2 minus μ_X into X_2 minus μ_X transpose in the same manner, that will give you 0.25 0.75 0.75 and 2.25, you can compute this and verify. So, this way you have compute X minus μ_X into X minus μ_X transpose for all the vectors that we have in set of vectors X . And, the expectation value is nothing, but average of these matrices that you are getting for each of the vectors I am getting X minus μ_X into X minus μ_X transpose. So, in this particular case I have 8 such vectors. So, I will have 8 such matrices and the average of all those matrices gives you the covariance matrix.

(Refer Slide Time: 11:36)

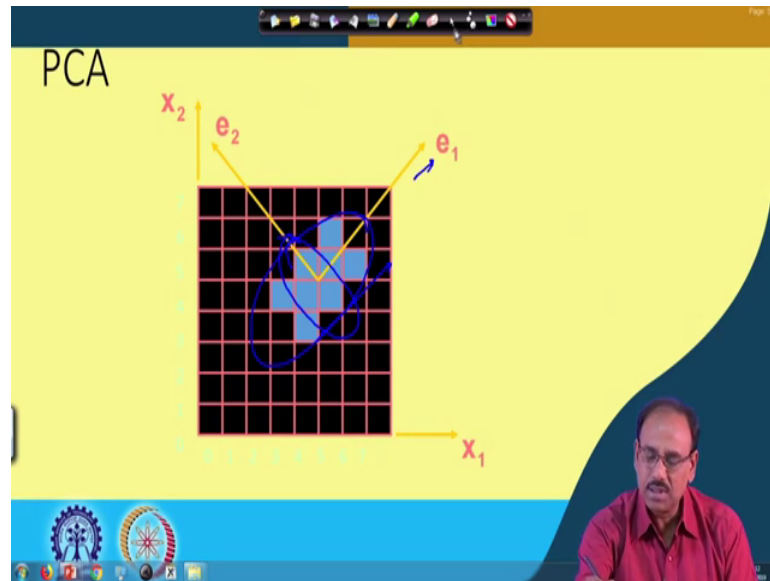
PCA

$$C_x = \begin{bmatrix} 0.75 & 0.375 \\ 0.375 & 0.75 \end{bmatrix}$$
$$\begin{vmatrix} 0.75 - \lambda & 0.375 \\ 0.375 & 0.75 - \lambda \end{vmatrix} = 0 \Rightarrow \lambda_1 = 1.125 \text{ \& } \lambda_2 = 0.375$$
$$\lambda_1 \Rightarrow e_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \lambda_2 \Rightarrow e_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

So, covariance matrix in this case is nothing, but if you compute that will come out to be the covariance matrix C_x equal to 0.75 0.375 0.375 and 0.75. And, from here you compute the eigenvalues as I said that it will be the determinant 0.75 minus lambda 0.375 0.375 and 0.75 minus lambda, you said this determinant equal to 0. So, you get a quadratic equation in lambdas you solve that and you get 2 values of lambda, lambda 1 comes out to be 1.125 and lambda 2 comes out to be 0.375. And, for these values of lambda you compute the corresponding eigenvectors as we said that the eigenvectors will be nothing, but $C_x e$ is equal to lambda e.

Where C_x is your covariance matrix e is the eigenvector and lambda is the eigenvalue. So, you solve those equations you get your eigenvectors. So, it comes out to be that in this particular case your e 1 for eigenvalue lambda 1 comes out to be one upon root 2 11. Similarly, e 2 for eigenvalue lambda 2 comes out to be one upon root 3 1 minus 1. Now, what comes next is the interesting one that is we want to see that what are this eigenvectors?

(Refer Slide Time: 13:08)



So, I simply superimpose this eigenvectors in the same image space. Here, so, here you find that if you look at this, you find that in the direction of e_1 . If, you look at this direction this is the direction in which your spade of data is maximum right and that is what we said that λ_1 , that is the eigenvalue indicates that what is the variation of data in that direction of the corresponding eigenvector.

Are similarly λ_2 this tells you that the variation of data in this direction is minimum right. So, I get λ_1 and λ_2 and here again you find that this sorry e_1 and e_2 and here again you find that e_1 and e_2 they are actually orthogonal, and they are centered at the centroid of the pixels. So, that also says that a K L transformation is a transformation, which gives you translation and rotation operations. Because, e_1 , e_2 are nothing, but rotated $X_1 X_2$.

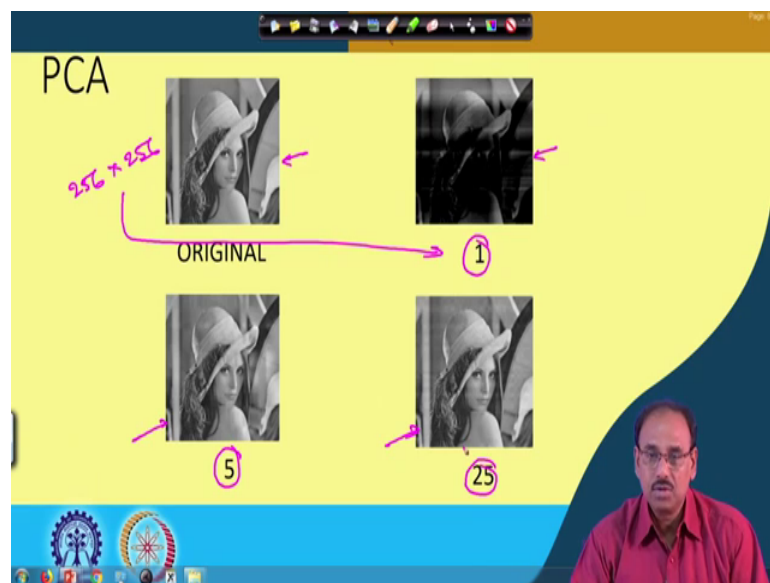
And, this coordinate system $e_1 e_2$ is translated to the mean of the vectors that we have. So, this also gives you the rotation and translation transformation right. Now, coming to the concept of your principal components, so, what are principal components over here? So, I have data point this. If, I take the projection of this data point see if I take the projection of this on to e_1 . So, this is the projection on e_1 and this is what is the principal component? Similarly, if I project this onto e_2 this is also the principal component.

So, this is the first principal component and this is the second principal component. If, I wanted to represent this by a single dimension or by scalars, I will not consider the projection onto e_2 rather I will consider only projection onto e_1 , because e_1 corresponds to the eigenvalue which is maximum so, this is my first principal component.

So, to think of this in other way the principal components or the K L transformation is actually a linear transformation. So, whatever doing is given your input vector input data you are linearly transforming it to your principal components right. So, principal components gives you a linear transformation or linear mapping from the input data space to the output data space. So, if I want to retain only one component you find that that component is nothing, but projections onto this eigenvector e_1 .

So, you have project a projecting onto a line. If, I want to retain two components; I want to convert the input data into two principal components, by this principal component analysis I will take projections onto e_1 , I will also take projections onto e_2 ; that means, every vector will now be mapped to a point in the plane $e_1 e_2$. So, this transformation is a linear transformation. So, given this now can we try to establish that what will be the relation between the principal component analysis and auto encoders.

(Refer Slide Time: 16:57)



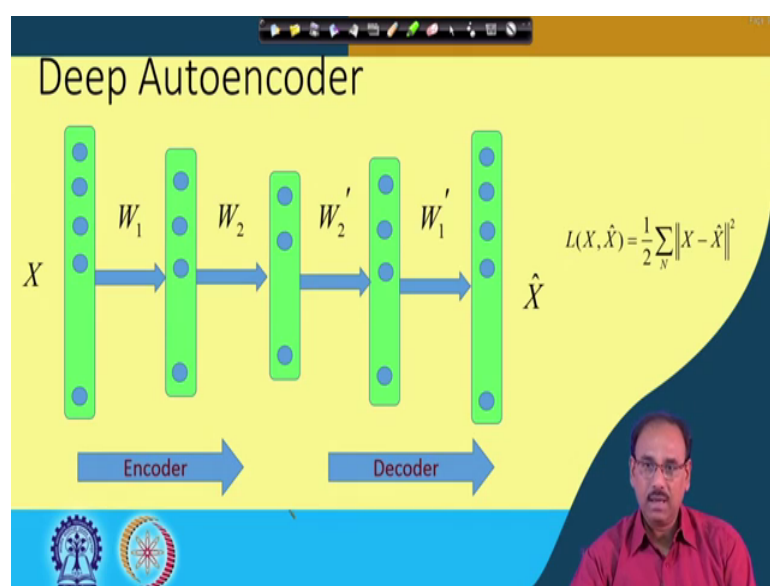
So, both before that just to illustrate what is the power of this principal component analysis of the principal components, you find that this is just an illustration that I have

this input image. So, this is the original input image. And, this is the image which has been reconstructed using only one principal component. And, you find that the reconstruction is amazing. So, that simply says that principal components retains the structure of the data, instead of one this original image is actually 256 by 256 number of pixels. So, you find that reduction is from 256 to by 256 to 1.

So, the amount of compression that has been achieved instead of 1 if I use 5 principal components then this is the reconstruction that you get. Instead of 5 if I use 25 principal components and this is the reconstruction that you get. So, here you can imagine what is the amount of compression that you are getting or what is the compression amount of compression that you are getting over here, 256 by 256 is to 256 is to 25. So, that shows you, what is the power of the principal components? So, this principal components using principal components I can go for dimensionality reduction.

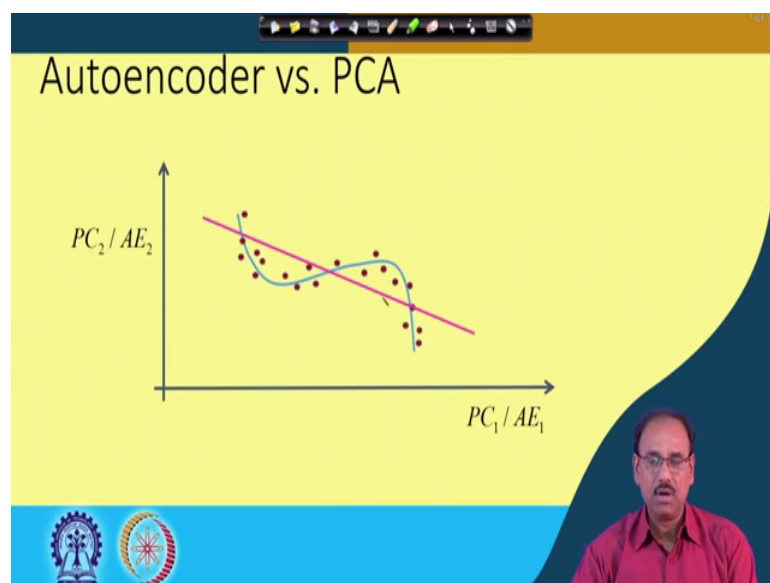
And, as we have seen that using auto encoders also we can go for a reduction. So, definitely I can establish some relation between these 2. So, what we have seen is in case of principal components this is a linear transformation, from N dimensional space or say d dimensional space, you are transforming it to say 2 dimensional space or 3 dimensional space depending upon the number of principal components that you want to use. And, this mapping is a linear mapping.

(Refer Slide Time: 18:02)



As against auto encoders being neural networks which can implement non-linear functions. So, the kind of mapping that we can use in case of auto encoders is a non-linear mapping; that means, in other words we can say that the principal components or the auto encoders can be thought of as generalization of principal components. So, whatever principal components analysis can do auto encoders can also do the same thing, but autoencoder can do something more, because here I can have non-linear mapping not simply linear mapping whereas, in case of principal components we have only linear mapping ok.

(Refer Slide Time: 19:51)

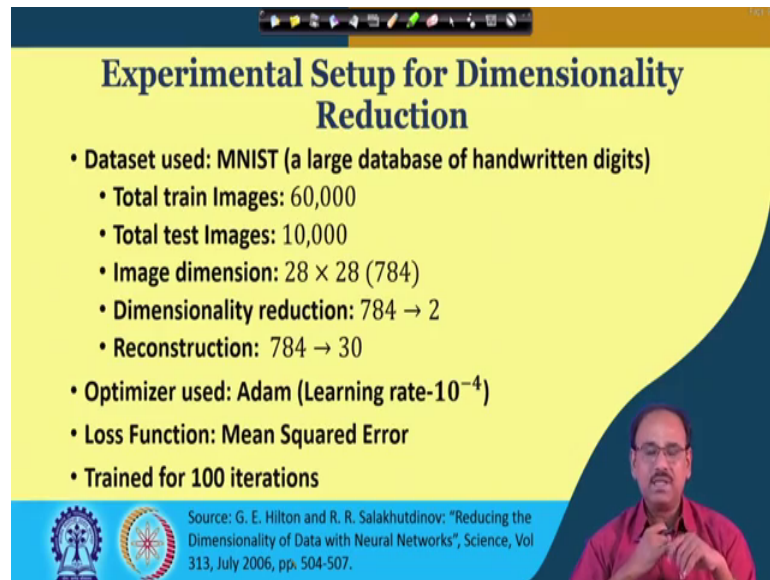


So, this is what I just said that given a set of data, which are just red dots in this figure and I want to convert this set of data using principal component analysis into principal components. So, if I use just two principal components; PC 1 and PC 2 I am transforming this set of data into on to a straight line as given by this pink line right. So, this is sorry I am transforming this set of data onto a plane as defined by PC 1 and PC 2. And, this being a linear transformation or linear mapping, what I can do is given this set of data this can be approximated on a straight line or the straight line is defined in space PC 1 PC 2 whereas, autoencoders are capable of going for imparting non-linear transformation.

So, using auto encoders I can even go to establish or to extract the non-linear structures which are present in the data. So, it is possible the same data or autoencoder will learn

the inner structure which is not just a linear, but it is non-linear as shown by the blue curve. So, all the set of data points with your red points can now be represented by points on this blue curve whereas, principal components will represent this set of data by points on the pink line.

(Refer Slide Time: 21:39)



Experimental Setup for Dimensionality Reduction

- Dataset used: MNIST (a large database of handwritten digits)
 - Total train Images: 60,000
 - Total test Images: 10,000
 - Image dimension: 28×28 (784)
 - Dimensionality reduction: $784 \rightarrow 2$
 - Reconstruction: $784 \rightarrow 30$
- Optimizer used: Adam (Learning rate- 10^{-4})
- Loss Function: Mean Squared Error
- Trained for 100 iterations

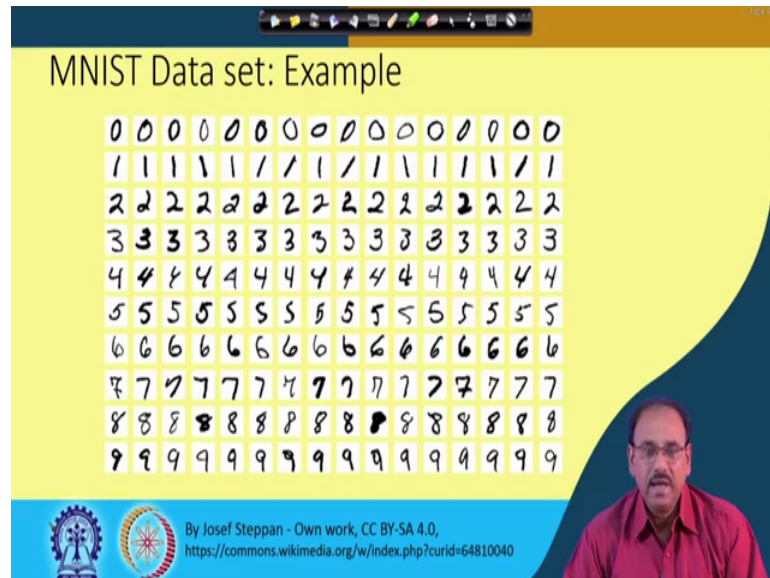
Source: G. E. Hilton and R. R. Salakhutdinov: "Reducing the Dimensionality of Data with Neural Networks", Science, Vol 313, July 2006, pp. 504-507.

So, we will present some experiment from this particular source which is given at the bottom. So, these experiments are done on MNIST data set and NIST data set which is a public domain data set. Which has total this data set is actually data set of handwritten digits from 0 to 9, it has total 60 000 training images and total 10 000 test images all handwritten digits, every image is of size of dimension 28 by 28; that means, it has got 784 number of pixels. So, when you think in terms of autoencoder at the input side will have 784 plus 1 to take care of the bias 785 number of neurons..

Dimensionality reduction from 784 to 2 so using autoencoder as well as PCA and then we will also talk about the reconstruction from the reduced dimension, where for deconstruction the dimension was reduced to 30 from 784. For dimensionality reduction demonstration it was reduced to 2, because the plane on which will be projecting this data or to dimensional planes. So, if it is reduced to 2 dimensional data then projection is I can visualize it, then the other things that optimizer, which is atom optimizer, we will come to that later we have not yet talked about the different types of

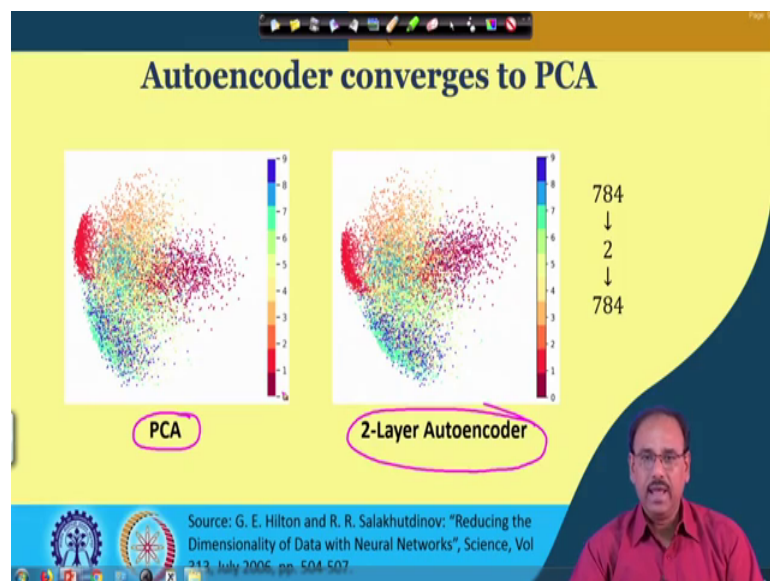
optimizers. The loss function that was considered is mean square error that we said and for training 100 iterations was used. So, this is what is the experiment setup.

(Refer Slide Time: 23:30)



Now, what is them this is an example of the MNIST data set I was as I said that these is data set of handwritten digits from 0 to 9 ok. And, again this is taken from the source as given at the bottom.

(Refer Slide Time: 23:47)



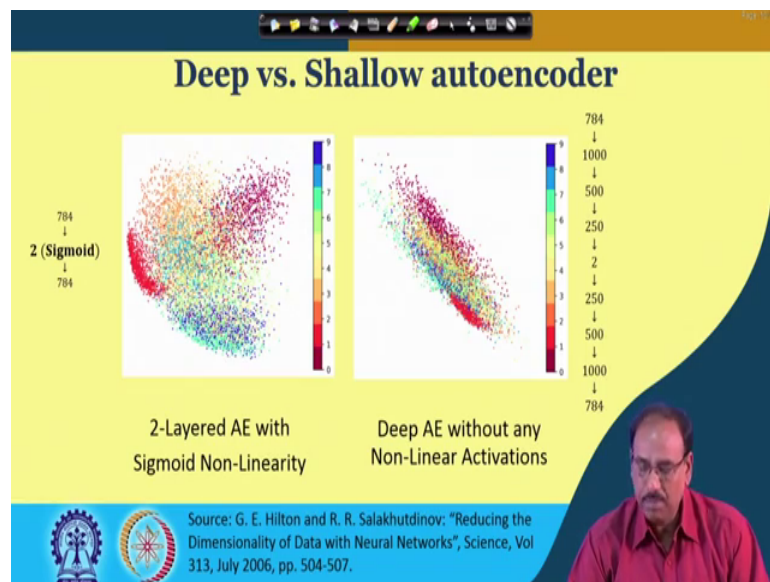
Now, for this data set, the data reduction was done using principal component analysis to 2 dimensional data, you remember our input was 784 right. So, is in principal component

analysis it was reduced to 2 dimensional vectors and also by using auto encoders it was reduced to 2 dimensional vectors, but in this particular case no non-linearity was used the activation function of the neurons was a linear function.

And, as is shown in these two data set, these two outputs you find that the output as given by PCA is almost identical to output as given by 2 layer it autoencoder, which has just one hidden layer right hidden layer and then output layer they are almost identical. So, and this is what we said that principal component analysis is our data dimension reduction technique auto encoder can also be thought of as dimensionality reduction technique.

So, somewhere there must be a relation between these 2 and this is what ok. So, you find that in some case PCA and auto encoders they give you identical results. Of course, will have difference of results as we said that, auto encoders are more general than principal component analysis, because auto encoders can impose can implement non-linear functions ok.

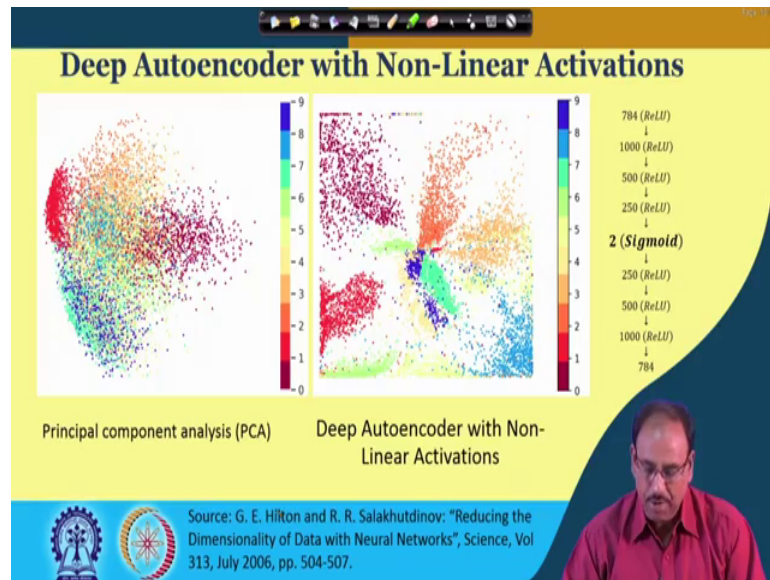
(Refer Slide Time: 25:30)



So, let us see what other results that we can obtain. So, this is a comparison between deep versus shallow autoencoder. So, on the left hand side what is shown as the output of 2 layer of encoder and on the right hand side what is shown as a deep autoencoder. And, here you find that a deep autoencoder having multiple number of layers as the architecture of the deep autoencoder is shown on the right.

So, deep autoencoder with multiple auto encoding layers or decoding layers can capture the structure of the data better, and that is where the beauty of the auto the or the beauty of the deep learning comes.

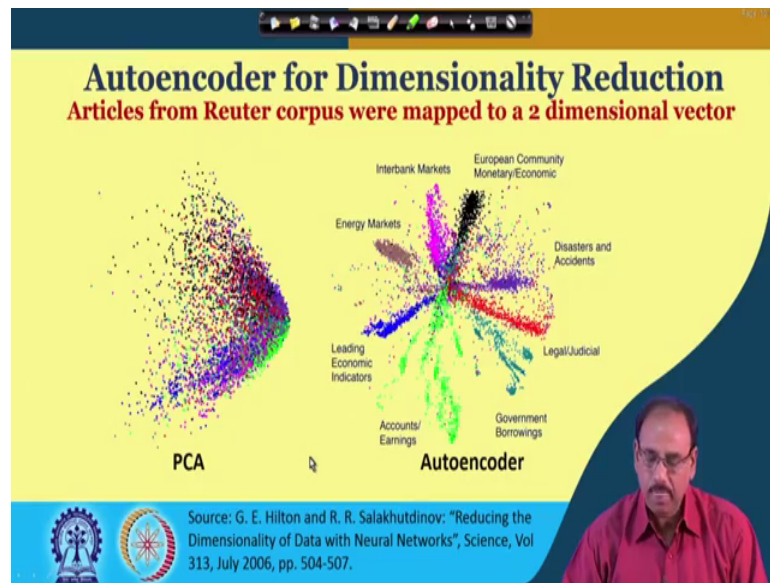
(Refer Slide Time: 26:19)



Coming to another example so, if we have, so in the previous case the output was from deep autoencoder without any non-linear activation function. Now, find that if we use non-linear activation function in a deep autoencoder, then as shown in this diagram on the right, the deep autoencoder with non-linear activation function can even capture the structure of the data better. So, here you find that the different digits this is 0 yeah this is actually 0 set of 0s, here it is set of 1s and so on.

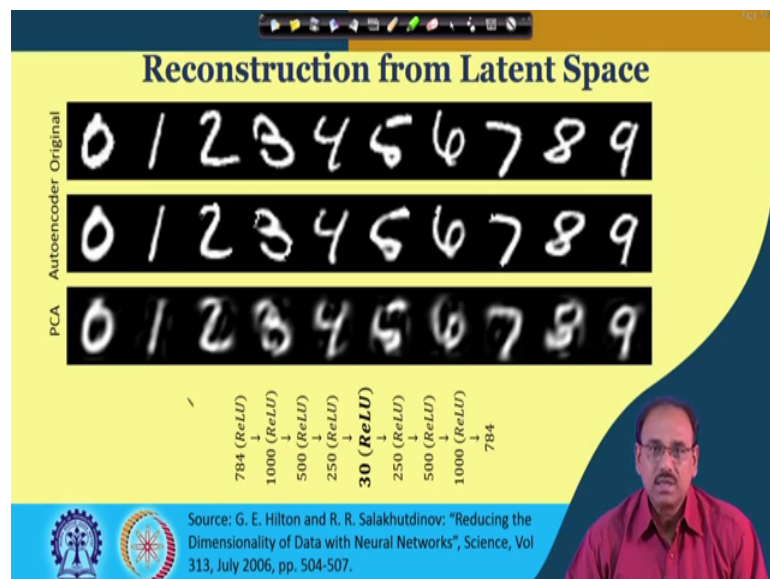
So, it can capture the inner structure of the data even better than shallow autoencoder ok. So, that is what is the beauty of auto encoders with non-linearity it can understand or it can learn, the inner structure of the data much better.

(Refer Slide Time: 27:24)



This is another example, where it was a data reduction from articles from Reuter corpus those kind images were reduced to 2 dimensional data. So, here again you find that autoencoder as compared to principal component that has captured, the inner structure much better than, what the principal components can do?

(Refer Slide Time: 27:49)

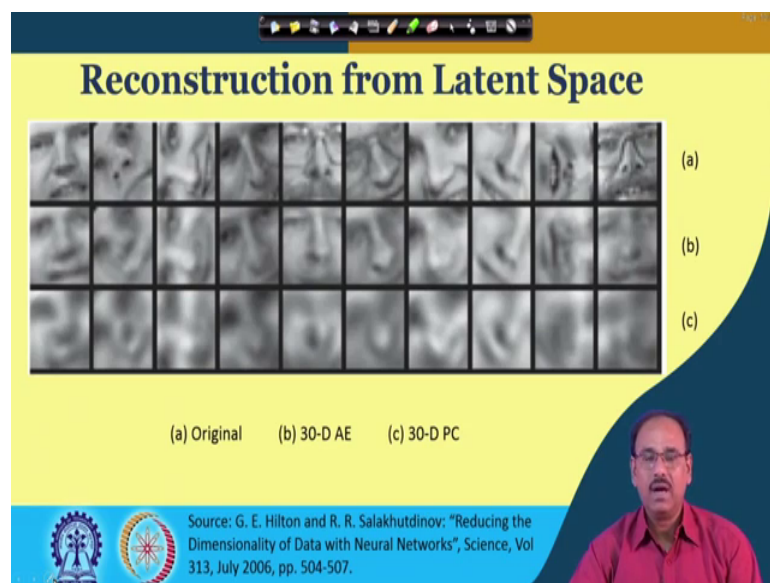


This is just another reconstruction example where we say that for the reconstruction that data reduction was done up to dimension of 30. So, from 784 it was reduced to dimension 30. So, here the top row gives a what was the original image, in the middle

row you have the reconstruction using autoencoder or the auto encoders architecture is given over here. And, the activation function was ReLU that is rectified linear unit and the bottom row is the reconstruction using principal component analysis, so, the principal components again having 30 principal components.

So, here again you find that though the radar data reduction was equal the data were reduced to the same extent both using autoencoder as well as principal components, but the reconstruction using the encoded data using autoencoder is much better than reconstruction from the principal components though the reduced dimensionality was same. So, that tells you what is the power of autoencoder over principal components. Earlier we have seen the power of principal components, now it shows that the auto encoders are even more powerful than principal components.

(Refer Slide Time: 29:16)



Similarly, this is another example of reconstruction, where we have a set of phase images. So, the top row is your original image, the second row is again output that is the reconstruction from 30 dimensional autoencoder outputs. And, the bottom row is reconstruction from 30 dimensional principal components. So, here again you find that the middle row, in the middle row, the data structure of the input data or the input images has been reconstructed much better than in the bottom row.

So, it also shows that auto encoders with non-linearity or deep on auto encoders within deep non-linearity are much more powerful than principal components ok. So, let me

stop this today's lecture here, we will next talk about the training algorithms of auto encoders. Of course, the training will be back propagation training as we said earlier and the errors or the loss function for back propagation training, that we will be using will be the sum of squared error loss. So, we will come back later.

Thank you.