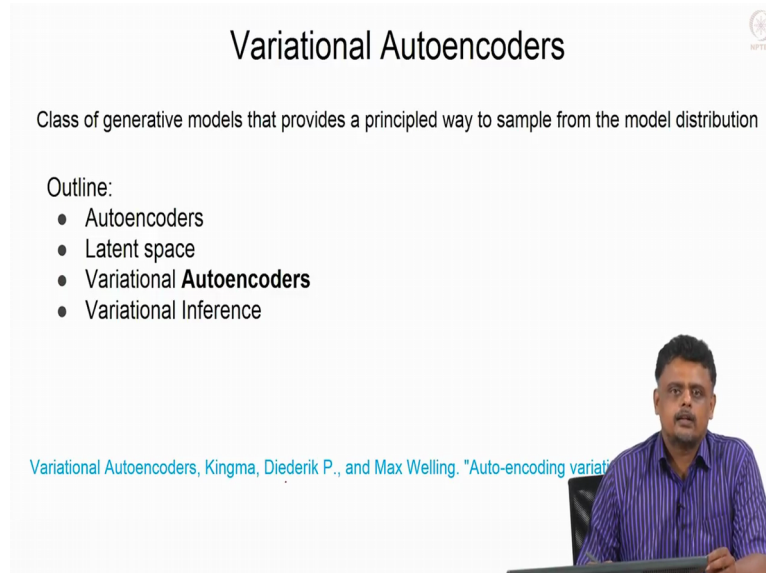


Machine Learning for Engineering and Science Applications
Professor Dr Ganapathy Krishnumurthi
Department of Engineering Design
Indian Institute of Technology, Madras
Variation Auto-Encoders (VAE)

(Refer Slide Time: 0:19)



Variational Autoencoders

Class of generative models that provides a principled way to sample from the model distribution

Outline:

- Autoencoders
- Latent space
- Variational **Autoencoders**
- Variational Inference

Variational Autoencoders, Kingma, Diederik P., and Max Welling. "Auto-encoding variational

The slide features a small inset video of a man in a blue striped shirt sitting at a desk, speaking. The IIT Madras logo is visible in the top right corner of the slide.

Hello and welcome back in this video we will look at variational auto encoders which are class of generative models that provide a principled way to sample from the data distribution or the model distribution. This is a brief outline of our talk in this video, so we will start off with some introduction to auto encoders and what is meant by latent vector or latent space and then we will move on to variational auto encoders and finally we will conclude with variational interference which is the probabilistic interpretation of VAEs. So in this video we will primarily focus on how variational auto encoders are used in deep learning context.

(Refer Slide Time: 0:53)

Autoencoders

The diagram illustrates the autoencoder architecture. It consists of three main components: an input data block labeled 'x', an encoder block, a features block labeled 'z', and a decoder block. The input data 'x' is processed by the encoder to produce the features 'z'. These features 'z' are then processed by the decoder to produce the reconstructed input data '-x'. The diagram is titled 'Autoencoders' and includes a small logo in the top right corner.

- Autoencoders are trained to reconstruct input data and in the process learn reduced dimensional representation
- Compressed features are meant to capture factors of variation in training data.
- Can we generate new images from an autoencoder? Typically no constraints on the learnt representation

So we will look at auto encoders, so they are class of neural network which are trained to reconstruct the input. So if you are thinking of images as input, typically the most standard involves seems to have a lot of applications in the computer vision or image or using images. So let us say input data is some image let us say data set 28 cross 28 and the idea behind this neural network is to reconstruct the output, so the output will be another image which will be 28 cross 28 and optimization in this case proceeds by making sure that your reconstruction and inputs are consistent with each other, reconstruction loss is minimised.

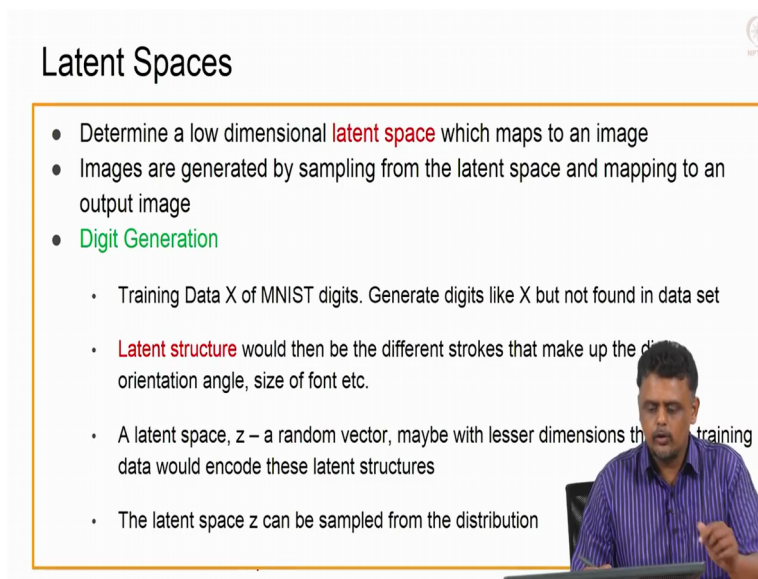
Typical structure for auto encoder basically the input, you can think of it as a fully connected neural network or an MLP then drag size the 28 by 28 image 4784 pixels and you (1:50) layers with decreasing number of hidden neurons, so we come to a layer where which you call the bottleneck layer and from there we have again successively increasing number of hidden neurons in every layer till we get to the output which is the same size as the input layer.

Now this layer from which the output is reconstructed is called the, we can call it the representation or the latent space okay, so we will denote it by Z ok. So the idea behind using auto encoders is to obtain a reduced dimensions representation of input in the sense that it retains most of the significant variations in your input enough to reconstruct, so that is the whole idea behind training the auto encoders. However, there is no structure to the, it is very hard to enforce some structure to the Z that we estimate. So typically you would have some

past constraints and things like that so that you get some meaningful representation, otherwise it is difficult to impose some structures in Z .

So what we do is to take this a little bit further in fact, the names are similar of the names are similar and the connection is very superficial because we can we can see that actually see later, this come from probabilistic arguments which are much more structured and leads to a more meaningful Z . So this Z as I mentioned earlier is referred to as the latent space or a latent vector which is what we are trying to infer.

(Refer Slide Time: 3:24)



Latent Spaces

- Determine a low dimensional **latent space** which maps to an image
- Images are generated by sampling from the latent space and mapping to an output image
- **Digit Generation**
 - Training Data X of MNIST digits. Generate digits like X but not found in data set
 - **Latent structure** would then be the different strokes that make up the digit, orientation angle, size of font etc.
 - A latent space, z – a random vector, maybe with lesser dimensions than training data would encode these latent structures
 - The latent space z can be sampled from the distribution

So what does this give us in terms of advantages, okay. So 1st that is the latent space or the latent vector that we estimate is basically reduced dimensionality, so in the case of images if you have very large images maybe it is possible to get a latent space representation which is only maybe 2 or 3 parameters ok, or maybe hundreds of parameters instead of millions of parameters when you think of naturally images. And what we do with this latent representation, the idea is the representation is actually can be actually used to create new images or generate new data with sampling randomly from it. So what we ideally want is a probabilistic or a probability density functions for $F Z$ so that we can sample from it and from there we can generate images which are close to the training data that we use ok.

So for instance, let us look at the digit generation right, so we have training data X of MNIST digits, it is a 28 cross 28 images in $(())(4:31)$ of them. And our purpose is to generate digits like X but not really the same ones found in dataset alright, so which is like saying we want to maximise this P of X that is typically we want in a generative model, we want a probability

model in our input data. So what would be a latent structure or latent space represent, so in this case these are the things that we are unable to observe so we only observe once the person has finished reading so we do not know what are the different strokes that he would have used or she would have used.

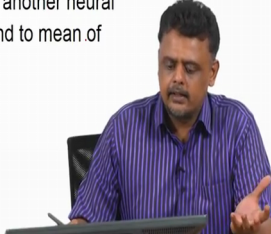
And there are also other things like orientation, how big are fonts, how thick are the strokes, so on and so forth, these are some of the latent structures or the latent space involved in the output that we see, which is basically the digits, the observation that we make is the digit itself or the image of the digit itself but what went on to creating it we do not know ok. So what we are trying to do is to model that latent data, so basically we model that as in the form of a latent space from which we sampled this vector Z ok, so Z is a random vector usually with lesser dimensions than that of the input data. And what we also want is a distribution for Z okay, so we can have a prior distribution for Z ok from which we can sample Z and we can also define a posterior distribution and this is what most people are interested in that is given this data set, what would be the most likely Z values that we can have okay.

So once we have this distribution then we can sample this Z from the distribution and map it to Sample X , so that is the idea behind variational auto encoders is to infer this distribution P of Z given X so that we can use that to sample from one of the distributions X ok.

(Refer Slide Time: 6:34)

Variational Autoencoders

- Training Data (input image) is mapped to a latent space using a neural net
 - Latent space posterior distribution and **prior distribution** are modelled as Gaussian
 - Output of net is two parameters – mean and covariance – Parameters of Posterior distribution
- A random sample from the latent space distribution is assumed to generate the input data
- The latent space vector is mapped to input image using another neural network – reconstructed output is assumed to correspond to mean of Gaussian - leads to reconstruction loss



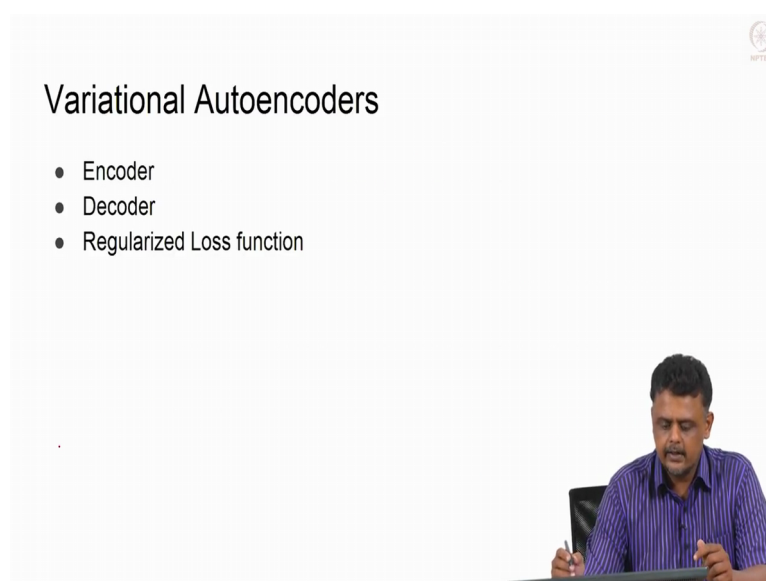
Basically, as I mentioned earlier the idea is to map training data in this case input images to a latent space using a neural network ok. And the latent space is basically the Z , which is posterior distribution P of Z , X so that is the P of Z , X Z given X sorry. And the prior

distribution, we have some assumptions about prior distribution and we usually model them as Gaussian distribution ok. The output of the network which provides P of Z given X basically has, it does not exactly gives you samples from P of Z given X , instead it gives you since it is a matrix, it does a Gaussian, it gives you two parameters basically the mean and covariance value of the distribution okay.

Now we draw a random sample from the latent space, so once we know the mean so we would know μ and we also know the covariance matrix, so using given this information, we can sample Z from this latent space and use that to generate data which is similar to the trained data X ok. So how is that done? So again not very surprising, we will use another neural network which will Sample from μ , Z that are depicted take that as input and map it to an output which is very similar to your input data or input images. So if you take MNIST digits, you would map from your sample of latent vector to a digit which looks like it came from the training data distribution ok.

Of course you can also interpret the output of this network, the network that takes P of Z given X and gives you a sample from the training data similar to the training data distribution. And we would like to if we interpret our output again as you know as the Sample from Gaussian distribution then it leads to a reconstruction loss, we will see how that is done in the next few slides.

(Refer Slide Time: 8:35)



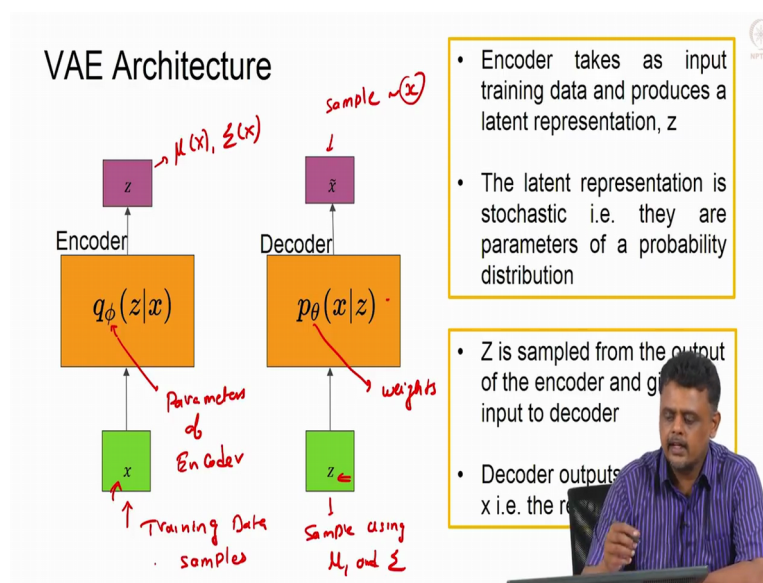
Variational Autoencoders

- Encoder
- Decoder
- Regularized Loss function

So variational encoders consist of these 3 components; the encoder, decoder and the regularised loss function okay, so this is from the deep learning point of view. So as you saw

earlier, the encoder takes x (8:46) as input and provides you the parameters of the latent space distribution, and decoder takes a Sample from the latent space distribution and provides you with the output which is similar to data in the training data ok, and these 2 are basically parameterise, the encoder and decoder both are deep neural network right so we can use them ok, can be convolutional neural networks or just fully connected once, we can use either of them so typically for images you make sense to use convolutional neural network ok. And we have a regularised loss function, we will see what that is in order to optimise the parameters of this neural network.

(Refer Slide Time: 9:33)

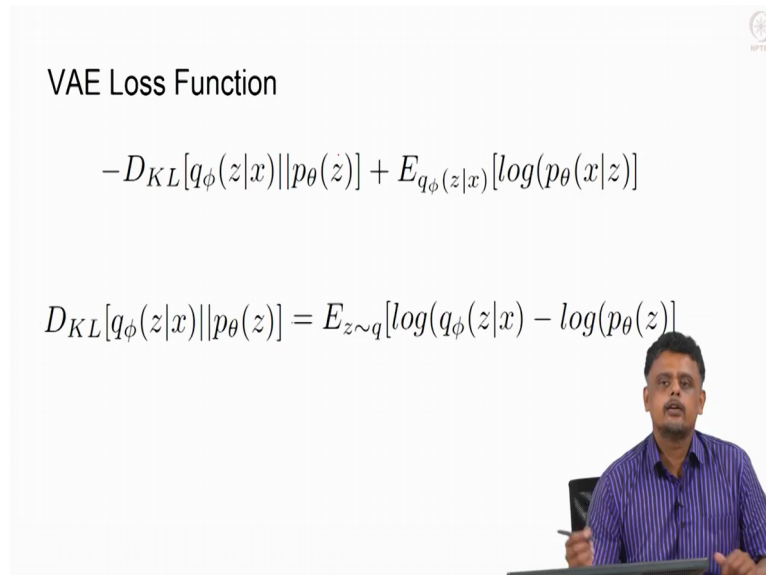


So the VAE architecture, so this is your training data as input so training data samples are given as input. Q of Z given X is actually represented by a neural network, so this is the Φ or the parameters of the or the weight parameters or weights of the encoder, right. And it gives you output Z in this case again slight (10:11) it does not exactly, you do not exactly get Z but you would get the parameters that depends upon X ok, so for every X that you give as input, you have a corresponding μ and Σ coming for it and you have this decoder which is again the neural network with the data are the again weights or the (10:35) of parameters, weights of that network which takes its samples, so this is Sample using μ and Σ right.

You sample using μ and Σ and you get a Z , this is given as input to this neural network and it provides an output \tilde{x} which is a sample, which is similar to \tilde{x} which is sample which is similar to x ok. So in this case we have given a particular x as input from which we have sampled Z right and so we would expect the output \tilde{x} to be the same as x , \tilde{x}

will be same as X that we gave as input over here ok. So this is the VAE architecture and in order to train this we have a loss function which consist of 2 parts, we will look at what they are and that ensures that this neural network is consistent in terms of a particular Z giving rise to a particular sample from the training data ok.

(Refer Slide Time: 11:53)



VAE Loss Function

$$-D_{KL}[q_{\phi}(z|x)||p_{\theta}(z)] + E_{q_{\phi}(z|x)}[\log(p_{\theta}(x|z))]$$
$$D_{KL}[q_{\phi}(z|x)||p_{\theta}(z)] = E_{z \sim q}[\log(q_{\phi}(z|x)) - \log(p_{\theta}(z))]$$

So the VAE loss function consist of 2 terms ok, so this is your data reconstruction loss is typical in most networks, and this is your regulariser. So if you look at the regulariser, the regularised is nothing but the KLD version, so the KLD versions between the distribution that is the output by the encoder network and your prior model for distribution of Z. So we have actually in this case in the case of VAEs, both of them are modelled as Gaussian distribution okay. So the KLD version if you recall defines you how similar two distributions are, if they are exactly the same then you get 0 ok so that is the optimum value that we get for that distribution.

(Refer Slide Time: 13:07)


VAE loss function

$$D_{KL}[q_{\phi}(z|x)||p_{\theta}(z)]$$

$q_{\phi}(z|x) \rightarrow \mathcal{N}(\mu, \Sigma) \rightarrow \text{Gaussian}$
 \downarrow
 constrained to be diagonal
 \leftarrow
 $\mathcal{N}(0, \mathbb{I})$

$$E_{q_{\phi}(z|x)}[\log(p_{\theta}(x|z))] \rightarrow \text{Recon loss}$$

\leftarrow decoder
 $\mu \leftarrow \theta$
 Binary cross entropy



So the sum of these 2 is what you optimise, so let us take a closer look at each one of them. So as I said earlier, so Q_{ϕ} of Z , X is the output of the encoder. So which gives you, in this case we have modelled them as Gaussian so you will get a μ in Σ corresponding to a Gaussian right. P_{θ} of Z is another Gaussian right, 0 mean and unit standard deviation okay, again if you look at the covariance metrics, it is constrained to be diagonal okay, and so is this one so identity metric. So our prior for Z is basically a standard normal distribution and we constrain the output to the parameters of the this Posterior distribution Q of Z X which is output prior networks whose parameters are ϕ are again integrated as the mean and covariance metrics of the Gaussian distribution, of course with covariance metrics they constrain to be diagonal ok.

Now this is your reconstruction loss, look at it so this is the output of the decoder right, the output of the decoder is actually an image and the output in this case if you are looking at image, it is looking to generate images. Now if we assume that this distribution that the if we interpret the output also as as the mean of the Gaussian distribution then you can again once again model the output as as the parameters of the Gaussian distribution and the mean is given by the output of the network characterised by data ok, so if you take the log of that then you will get, you will end up with the usual least square loss function. On the other hand, if you are looking at let say binary images wherein the pixel values are 0, 1, then you can also use binary cross entropy.

Binary cross entropy between your input image where the pixel values are either 1 or 0 and output of the decoder with values ranging from 0 to 1 which is done by using a Sigma at the output, then you can interpret the each of the output pixel values as probabilities drawn from the Gaussian and which we can use either as a if you take a log of that then you again you get the usual least square loss function. On the other hand if you treat the pixel values as binary variables themselves and you can still use the cross entropy ok that which comes only binary distribution ok. So each of them each of these are possible, so we have a combination of two loss functions, one is the reconstruction loss and the other one corresponds to the constraint on the output of the encoder making sure that the parameters correspond to that of a standard normal distribution, so that is the regularisation that we impose ok, so this is from a deep learning point of view.

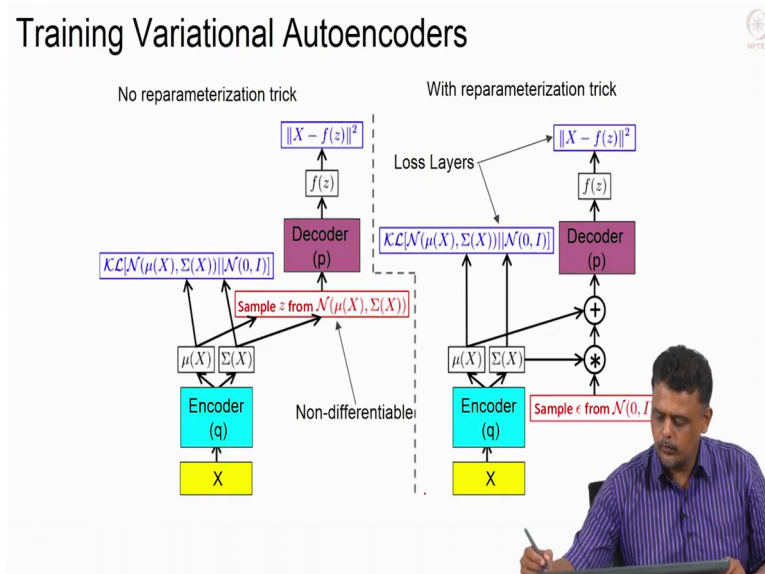
Once again this is essential because we can we can say well since you are cosigning it to be drawn from a standard normal distribution, why do we even need the encoder network right? So you can always say I will just draw from Z and then I will try to reconstruct some arbitrary X , so the idea behind doing this is that we want to make sure not all Z will give rise to the X that we have in our training data. So we specifically want to work with Z , the vectors that we are now using as hidden representations or latent space representation to correspond to the data samples in our training data. So we have an encoder network that takes particular X from our training data distribution and maps it to Z and we try to reconstruct the same X using the decoder from the Z that we have obtained okay.

Now the loss function to be optimised, the loss person to be optimised has to be done on a image by image basis, so in the sense there are 2 ways of looking at this; one is just take a look at this loss function. So once you pass through the, once the encoder has given you a particular μ and σ , then you can sample a lot of Z values from that using μ and σ , pass it through the decoder function to reconstruct and of course the expectations values will be over the all the Z values that you generated for that particular X . Now as I mentioned earlier, this will become slightly harder to do because you have to sample lots of Z because not all Z that you sample will correspond to the X that you have at hand.

So to make this thing simpler, so we will have the encoder function so that expectation value can be done over for the loss function can be done over all the samples in your training data, so where it works is if you have the training data, let us say many bats or the entire training dataset, you will pass it through the encoder to obtain μ and σ , we will sample Z from

each of the output corresponding to every X of the encoder and pass it through the decoder to get the reconstruction loss so that we can do (19:24) on that loss function okay. Still one problem left because the sampling function if I say is not a differentiable function right, you cannot differentiate the sampling process so there is a worker called the reparameterisation trick, we will see what that is.

(Refer Slide Time: 19:42)



So just to recall that briefly, this is the image on the left is what we look at, so we have an X , this is the bunch of X that is the samples from the training data goes through the encoder, gives you parameters of a Gaussian distribution from which you sample a Z right here, then you pass it through the decoder which will give you a reconstruction of your input ok. So then you have some of 2 loss functions, this is what we saw earlier as the data reconstruction loss and this is to constrain this is the regulariser which makes sure that the parameters that you are generating from the encoder remain close to the normal distribution.

Now doing it this way so we want to back prop through the entire network all the way, but that is difficult to do so through the sampling process because the sampling itself is a non-differentiable function right. So to evaluate this reparameterisation trick what it does is, it uses a standard trick right so if you have if you have a sample from a standard normal distribution, you can convert it into a sample from a Gaussian distribution with a particular mean and standard deviation that is what is exactly the here. So everything else remains the same, the usual encoder X goes to the encoder to give you μ and Σ but you Sample Epsilon, for every X you will Sample Epsilon from standard normal distribution right, and then you will do the

trick of multiplying by Sigma and adding Mu to it right and that is given as input to the decoder.

Now back prop can go through the entire network in order to update all the ways okay, since this particular since this process of assembling Sigma from (μ, σ) (21:34) do not depend on the parameters of your either the decoder or the encoder ok. So once you have trained to the entire dataset, what does it we do with it? How do we let us say how do we generate samples? It is actually very simple, all you do is you Sample from the standard normal distribution. Since we have constrained the encoder or regularise the encoder to that way, you can sample some Z from the standard normal distribution. You can get rid of the encoders, you can discard this, now you have got Z you know how to Sample Z from, and you just have to put Z through the decoder ok and it will provide you Sample.

Now how this, we have already looked at auto encoders so the similarities the following, both of them have an encoder and a decoder we saw that, so we take that the classical auto encoder takes as input your training data samples and successive hidden layers there is a bottleneck layer which is treated as ideal representation and that hidden representation is then mapped back to your input, so which is exactly what happens here. We have this Q which maps X to Z that is the bottleneck layer in your auto encoder and then you have the decoder which takes the input in your representation and provides you with the output ok, so this structure is called as auto encoder, otherwise the principles are different.

The advantage behind doing it this way that is making the output of the encoder (μ, σ) (23:27) we are not treating it as some deterministic value there but rather treating it as the parameters of the distribution from which that is to be drawn. Now this provides some interesting results because it provides meaningful or a structured Z representation in the sense that if you sample close to the Z values that are estimated for trained data X , you will get similar X right. And if you systematically sample Z over the range of Z values that are being generated using this network, then you can see that we have mapped to some attributes in the images that we are using ok.


It will not be exact but it gives you a very smooth transient, as you change Z gradually, the images will also transient smoothly okay. The structure of the images can be captured in the distribution of Z rather than providing some arbitrary hidden space representation or latent space representation, you have a very structured Z that comes out of this process and that is

made possible by treating the output of the encoder as (\cdot) (24:37) rather than as a deterministic hidden layer in auto encoders okay.

(Refer Slide Time: 24:54)

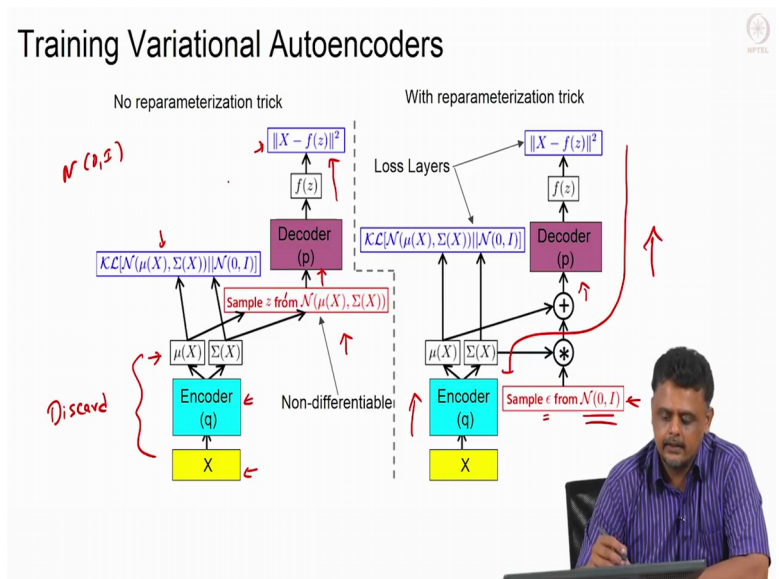
VAE Loss Function

$$-\underbrace{D_{KL}[q_\phi(z|x)||p_\theta(z)]}_{\text{Regularizer}} + \underbrace{E_{q_\phi(z|x)}[\log(p_\theta(x|z))]}_{\text{Data Reconstruction Loss}}$$

$$D_{KL}[q_\phi(z|x)||p_\theta(z)] = E_{z \sim q}[\log(q_\phi(z|x)) - \log(p_\theta(z))]$$


Another point is that the loss function that we derived from the point of view of deep neural networks if we go back to what we saw earlier, we have sum of these 2 terms right so this we wrote down as data reconstruction loss plus a regulariser, so that is from a deep learning viewpoint however, it is actually possible to derive this loss function from probabilistic arguments. So starting from trying to maximise your the likelihood of the data, it is possible to derive this particular loss function to be optimised, so this is a much more principled way of creating a generative model.

(Refer Slide Time: 25:42)



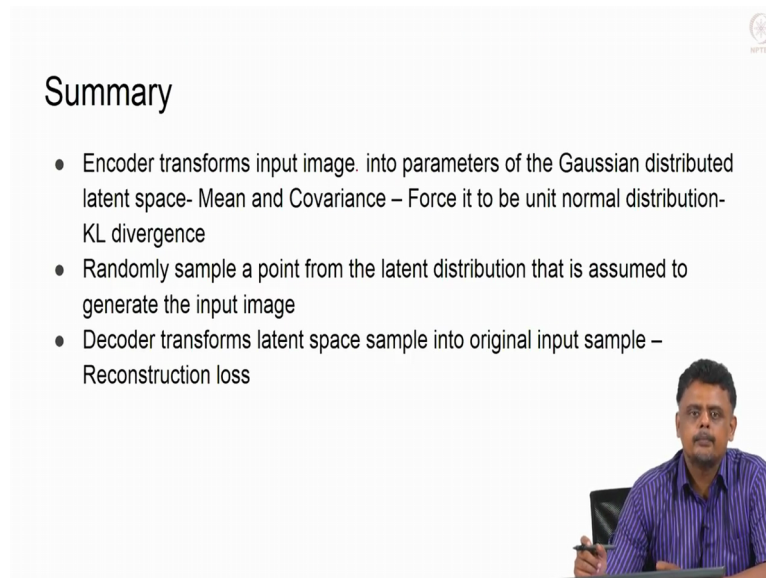
The innovation here is that the output of the generative model that is this P of Z given X, those are the parameters of the distribution, that has been replaced by neural network. So the

parameters of P of Z given X this PDF has been replaced with neural network which basically regresses your μ and σ of your P of Z , X . So μ and σ corresponding to be of Z , X are regressed by your neural networks. Similarly the process of mapping from Z which your Sample from P of Z provides to data, the data on which we actually want to get, the generative data is also accomplished using a neural network.

This provides you get advantage because neural networks can pretty much learn very complicated functions from higher dimensional space because the inputs to for instance if you take the input to the encoder, let us say it is a picture, it is an image, it is a 100 cross 100 image that is like dimension of 10,000 right. So your 10,000 input dimensions can be easily handled by a neural network, it provides a structure to do that, it is a non-linear function mapping. Similarly from mapping from Z to X is again this function that is denoted here that is also mainly possible by the neural network, so the F here represents the neural network, the P and Q are the probability distributions ok.

So we have parameterised the probability distribution as Gaussian and we have estimated the parameters of the Gaussian using neural networks. So this is the strong point of this variational auto encoder in addition to having a principled way of obtaining the loss function, which means that the Z that you generate from your training data have are meaningful, this is not possible in let us using a classical auto encoder because the only thing you can do is possibly make its parts to have a good representation, but in this case varying Z smoothly as many people have shown which will refer to later, varying Z smoothly can give you variation of your input ok, so that is the advantage of using variational auto encoder.

(Refer Slide Time: 28:04)



The slide is titled "Summary" and contains the following text:

- Encoder transforms input image. into parameters of the Gaussian distributed latent space- Mean and Covariance – Force it to be unit normal distribution- KL divergence
- Randomly sample a point from the latent distribution that is assumed to generate the input image
- Decoder transforms latent space sample into original input sample – Reconstruction loss

In the bottom right corner of the slide, there is a small inset image of a man with glasses and a blue striped shirt, sitting at a desk and looking towards the camera.

So to summarise, the encoder is a neural network here then coder is a neural network that transforms the input image into the parameters of the Gaussian distribution ok. And this Gaussian distribution corresponds to the latent space, so we have a mean and covariance corresponding to the latent representation and we regularise it by making it close to the unit normal distribution that is how we regularise the network. Now we randomly sample from the latent distribution and we assume that dataset generates the input image that the input to the encoder or say input image is the input to the encoder.

Then we spend that Z through the decoder which is again another neural network, it is a function mapping which transforms that Z into the input image ok. And we optimise, in order to optimise we do the reconstruction loss that is make the output of the decoder as close as possible to the input image to the encoder. So during this process over your given input data trains your decoder appropriately, so once the training is done you can discard the encoder if you want and just sample from unit standard unit normal distribution, pass it through the decoder and obtaining samples of data that you want to generate.