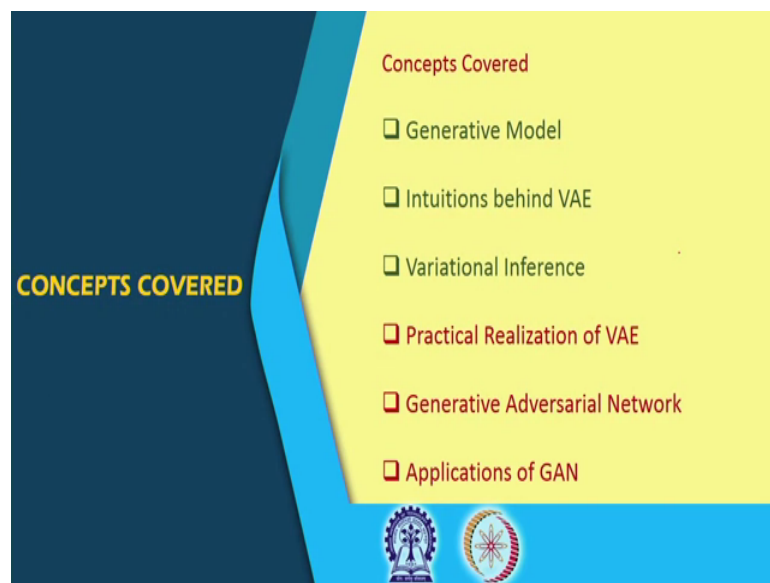


**Deep Learning**  
**Prof. Prabir Kumar Biswas**  
**Department of Electronics and Electrical Communication Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 60**  
**Generative Adversarial Network**

Hello, welcome back to the NPTEL online certification course on Deep Learning. So, for last few classes we are discussing about the Generative Network, where what we have discussed is that if you feed in latent vector or latent variable to the generative network, then the generative network will give you an output data or reconstructed image or the reconstructed object. And the kind of generative network that we are discussing so far is what is known as adversarial auto encoder.

(Refer Slide Time: 00:58)



So, we have started our discussion on adversarial auto encoder, we have discussed about what is variational inference, today what we are going to discuss about how do you practically realize the variational auto encoder. Then we will briefly discuss about another generative model which is known as generative adversarial network and we shall conclude our lecture or conclude this course with the applications of generative adversarial network.

(Refer Slide Time: 01:36)

**Variational Autoencoder : Variational Inference**

- ❑ Our initial objective: minimize  $KL(Q(z|x) || P(z|x))$
- ❑ Which is same as maximizing  $\sum_z Q(z|x) \log \frac{P(x,z)}{Q(z|x)}$

*Variational Lower Bound*

➤ So, aim now is: *maximize*

$$L = \sum_z Q(z|x) \log \frac{P(x,z)}{Q(z|x)} = \sum_z Q(z|x) \log \frac{P(x|z)P(z)}{Q(z|x)}$$

So, this is what we have discussed in the previous lectures that our initial objective was to minimize the KL divergence  $Q(z|x) || P(z|x)$  and we have seen earlier that this is same as maximizing the  $\sum_z Q(z|x) \log \frac{P(x,z)}{Q(z|x)}$ . What  $P(x,z)$  is the joint distribution upon  $Q(z|x)$  and we have seen that this is what is known as a variational lower bound and the concept of variational lower bound that came because we have seen that  $P(x)$  computation of  $P(x)$  was intractable.

So, what we try to find out and we have seen that this particular expression the variational lower bound gives a lower bound of  $P(x)$ . So, in order to maximize of  $x$  it is equivalent to maximization of the variational lower bound because when we are when we know that this variational lower bound is always less than  $P(x)$ . So, if I can maximize this variational lower bound  $P(x)$  is also going to be maximized. So, this is an indirect way of increasing the probability distribution  $P(x)$ .

So, our aim was now to maximize this variational lower bound and this variational lower bound can be rewritten as  $\sum_z Q(z|x) \log \frac{P(x,z)}{Q(z|x)}$  into  $\sum_z Q(z|x) \log \frac{P(x|z)P(z)}{Q(z|x)}$ .

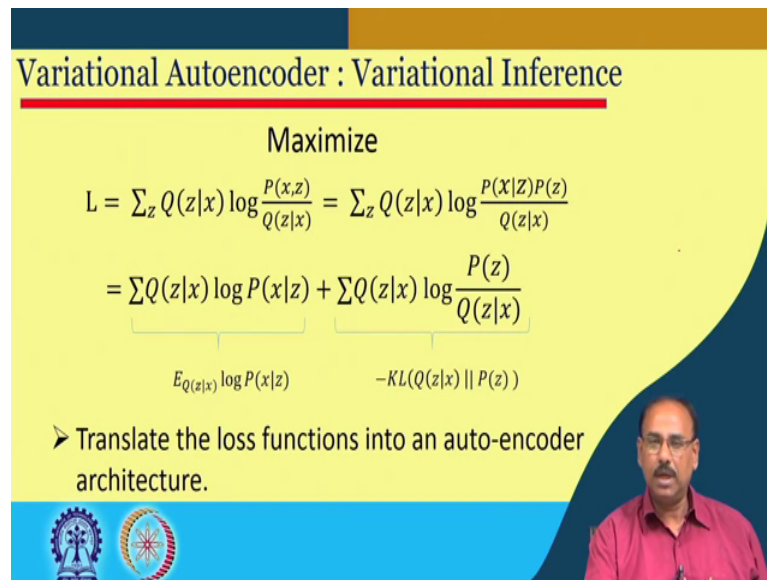
(Refer Slide Time: 03:14)

Variational Autoencoder : Variational Inference

Maximize

$$L = \sum_z Q(z|x) \log \frac{P(x,z)}{Q(z|x)} = \sum_z Q(z|x) \log \frac{P(x|z)P(z)}{Q(z|x)}$$
$$= \underbrace{\sum_z Q(z|x) \log P(x|z)}_{E_{Q(z|x)} \log P(x|z)} + \underbrace{\sum_z Q(z|x) \log \frac{P(z)}{Q(z|x)}}_{-KL(Q(z|x) || P(z))}$$

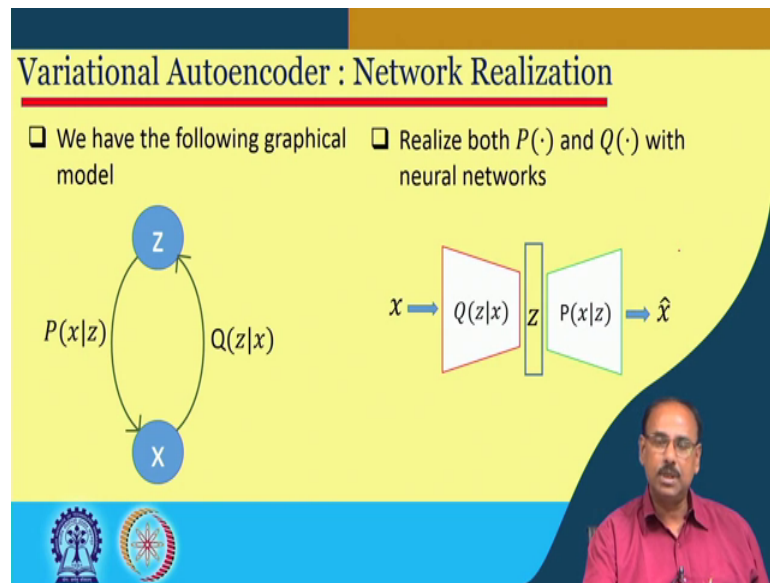
➤ Translate the loss functions into an auto-encoder architecture.



So, this expression can be rewritten as sum of  $Q$  of  $z$  given  $x$  log of  $P$  of  $x$  given  $z$  plus sum of  $Q$  of  $z$  given  $x$  log of  $P$   $z$  upon  $Q$  of  $z$  given  $x$ . So, here you find that the second term that is sum of  $Q$   $z$  given  $x$  log of  $P$   $z$  upon  $Q$   $z$  given  $x$  this is again another KL divergence, KL divergence between  $P$  of  $Q$   $z$  KL divergence between  $Q$  of  $z$  given  $x$  and  $P$  of  $z$ . So, this can be simply written as the first term is now expectation value of log of  $P$   $x$  given  $z$  and you find that this log of  $P$   $x$  given  $z$  is nothing, but log likelihood. And when you try to maximize this basically the first term maximization of this means you are going for maximum likelihood estimation.

And the second term which is the minor scale divergence of  $Q$   $z$  given  $x$   $P$   $z$  maximization of this term effectively means that you want to minimize the KL divergence between  $Q$  of  $z$  given  $x$  and  $P$  of  $z$   $x$  and  $P$  of  $z$ . So, then we have seen that how we can translate this loss function or maximization of the loss function into an auto encoder architecture.

(Refer Slide Time: 04:46)



So, for that again we have gone back to our graphical model that given your latent variable or latent vector  $z$  and the data which is to be reconstructed  $x$ . So, there is a probability involved probability density distribution involved that  $P$  of  $x$  given  $z$  that is given  $z$ , I want to maximize what is the likelihood  $P$  of  $x$  given  $z$ .



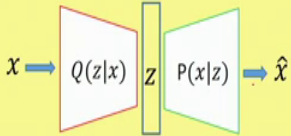
And similarly to get your latent variables or latent vectors from the training data I have another probability distribution involved what is which is  $Q$  of  $z$  given  $x$ . So, in terms of network realization you realize both  $P$  and  $Q$  with the neural networks and the network model that we would get is something like this. That  $Q$  of  $z$  given  $x$  is your encoder network,  $P$  of  $x$  given  $z$  is the decoder or the generator network and  $P$  of  $z$  given  $Q$  of  $z$  given  $x$  actually gives you the latent variable which is  $z$ .

So, here comes the difference between the traditional auto encoder and the variational auto encoder. In case of traditional auto encoder the latent vector  $z$  was deterministic, but with this variational auto encoder because we have a term  $P$  of  $z$  that is the probability distribution of the latent variable  $z$ . So, instead of getting a deterministic latent variable as in case of auto encoder it over here what we expect is the encoder part will give us the probability distribution or the parameters of the probability distribution of  $P$  of  $z$ .

(Refer Slide Time: 06:34)

### Variational Autoencoder : Network Realization

- ❑ The  $z$  codes we get here should match with the distribution of  $P(z)$  and we can decide what prior distribution to choose for  $P(z)$ .
- ❑ Usual practice is to select a Normal distribution  $N(0, I)$  for the prior.





So, it is therefore, expected that  $z$  codes does it latent variable that we get that should match with the distribution of  $P$  of  $z$  and we assume a normal distribution or normal a prior of normal distribution for  $P$  of  $z$  for this normal distribution is 0 mean and unit variance. So, that will be a prior for  $P$  of  $z$ .

(Refer Slide Time: 06:59)

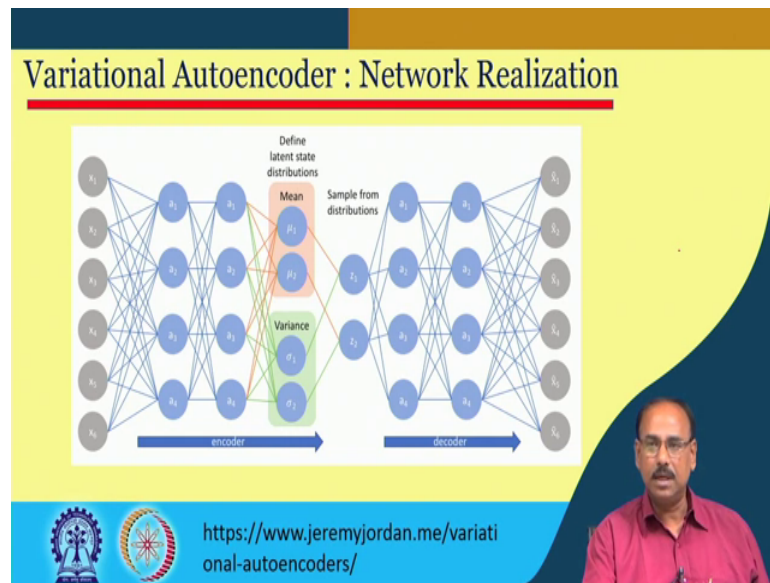
### Variational Autoencoder : Network Realization

- ❑ Instead of generating a fixed code for an input, Encoder now gives parameters of the distribution of the latent code.
- ❑ For a given input  $x$ , we need to generate mean vector  $\mu(x)$  and diagonal covariance matrix,  $\Sigma(x)$ .
- ❑ We need to SAMPLE a code from that latent distribution and pass forward to the Decoder.



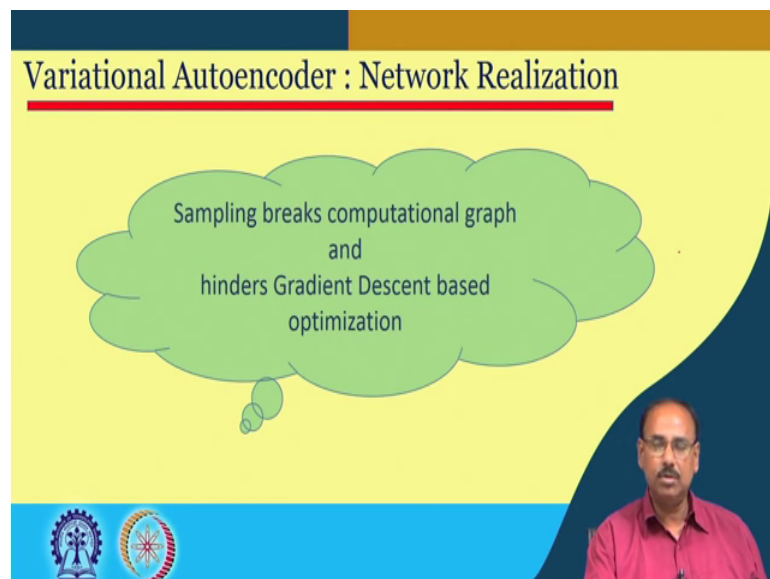
So, as you have seen that here instead of getting a fixed latent variable  $z$  what you are getting is a distribution or the parameters of the distribution.

(Refer Slide Time: 07:09)



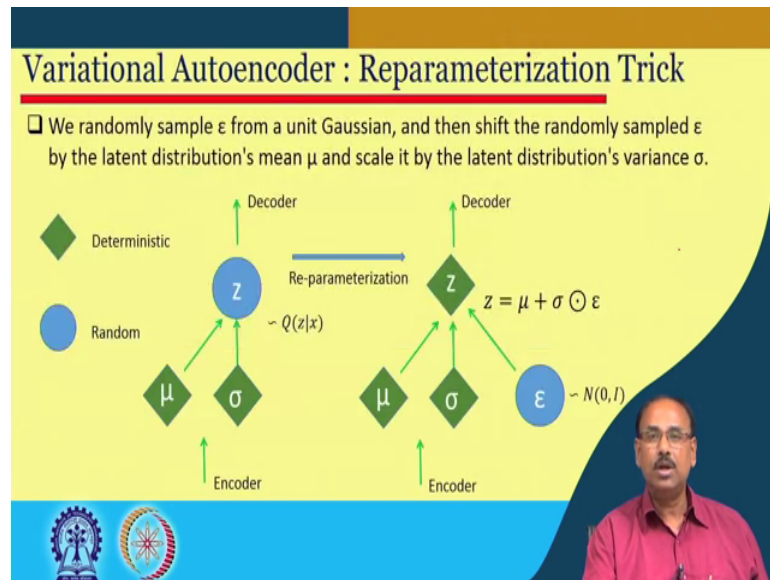
And this is the network realization that we have the encoder network on the left and the decoder network on the right, in between what you have is as the encoder is giving you the distribution of  $z$ . I can sample a set from that distribution and feed it to the decoder or the general network and the decoder of the generator will give me the generated or the reconstructed data that is  $x$ .

(Refer Slide Time: 07:33)



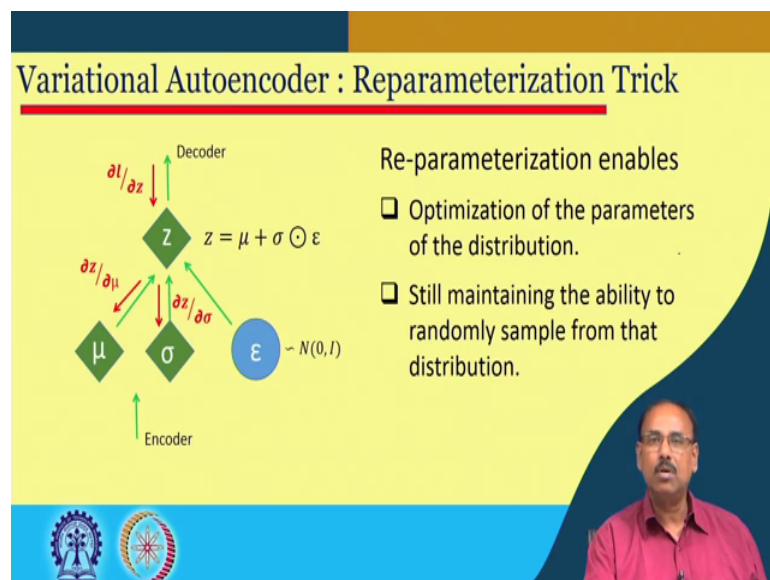
And then we have discussed about the problem in always in this sampling which makes it difficult for the back propagation learning because gradient cannot flow through the sampling process.

(Refer Slide Time: 07:47)



So, in order to avoid this problem we have made use of the Re-parameterization trick.

(Refer Slide Time: 07:52)



So, this is what we have discussed in details in our previous lecture.



(Refer Slide Time: 07:56)

Variational Autoencoder : Coding the Cost Functions

$$E_{Q(z|x)} \log P(x|z) - KL(Q(z|x) || P(z))$$

Maximize Minimize

The slide features a yellow background with a blue and orange header. The equation is centered, with two arrows pointing downwards to green and blue ovals labeled 'Maximize' and 'Minimize' respectively. A small inset video of a man in a pink shirt is visible in the bottom right corner.

So, now the problem is that we are trying to maximize the expectation value of log P of x given z minus KL divergence between Q of z given x and P z. So, maximization of this expression means we are trying to maximize expectation value of log of P x given z and we want to minimize the KL divergence Q z given x P z. So, we have both this maximization and minimization involved.

(Refer Slide Time: 08:33)

Variational Autoencoder : Coding the Cost Functions

- ❑ Maximizing  $E_{Q(z|x)} \log P(x|z)$  is a maximum likelihood estimation. It is observed all the time in discriminative supervised model, for example Logistic Regression, SVM, or Linear Regression.
- ❑ In the other words, given an input z and an output x, we want to maximize the conditional distribution  $P(x|z)$  under some model parameters.
- ❑ So we could implement it by using any classifier with input z and output x, then optimize the objective function by using for example log loss or regression loss.

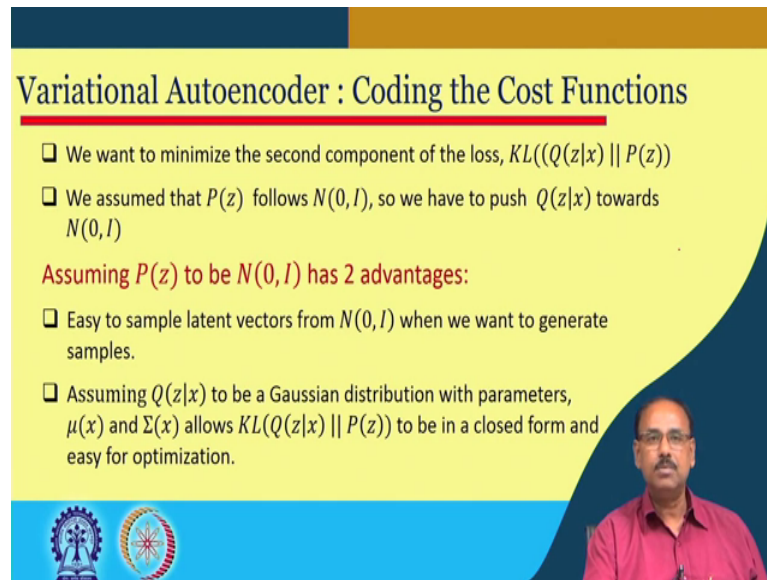
The slide features a yellow background with a blue and orange header. It contains three bullet points explaining the maximization part of the cost function. A small inset video of a man in a pink shirt is visible in the bottom right corner.

And for this maximization you find that the maximization of expectation value of P of x given z is nothing, but a maximum likelihood estimation which we have encountered a



number of times in our discriminative network. So, this can be solved using any of the classifier of the input is  $z$  and the output is  $x$  then you optimize the objective function by using different techniques for example, log loss or regression loss and all that.

(Refer Slide Time: 09:04)



**Variational Autoencoder : Coding the Cost Functions**

- ❑ We want to minimize the second component of the loss,  $KL(Q(z|x) || P(z))$
- ❑ We assumed that  $P(z)$  follows  $N(0, I)$ , so we have to push  $Q(z|x)$  towards  $N(0, I)$

**Assuming  $P(z)$  to be  $N(0, I)$  has 2 advantages:**

- ❑ Easy to sample latent vectors from  $N(0, I)$  when we want to generate samples.
- ❑ Assuming  $Q(z|x)$  to be a Gaussian distribution with parameters,  $\mu(x)$  and  $\Sigma(x)$  allows  $KL(Q(z|x) || P(z))$  to be in a closed form and easy for optimization.

The slide features a yellow background with a blue and orange header. At the bottom, there are logos of institutions and a video feed of a man in a red shirt.

Whereas for minimization of the other part that is the KL divergence as we said that we assume a prior distribution of  $P$  of  $z$  which is normal distribution with 0 mean and unit variance for all the attributes of the vector  $z$ . So, by when we try to minimize the scale divergence that effectively means that we are pushing the parameters of  $Q$   $z$  given  $x$  towards that normal distribution of 0 mean and unit variance.


So, by using this prior this normal prior it is advantageous in 2 ways. Firstly, it becomes very easy to sample it latent vectors from this normal distribution. And secondly, assuming that  $Q$  of  $z$  given  $x$  to be Gaussian distribution with parameters mean and the covariance matrix sigma it allows this KL distribution to be in a closed form and that becomes easy for optimization.

(Refer Slide Time: 10:10)

## Variational Autoencoder : Coding the Cost Functions

$$KL(N(\mu(x), \Sigma(x)) || N(0, I)) = 0.5 * [tr(\Sigma(x)) + \mu(x)^T \mu(x) - k - \log \det(\Sigma(x))]$$

- k is dimension of the latent code
- $tr(\Sigma(x))$  is trace of a covariance matrix
- $\Sigma(x)$  is the diagonal covariance matrix. So, its determinant can be computed as product of its diagonal entries.
- In practice  $\Sigma(x)$  can be predicted only as vector containing the diagonal entries



So this closed form representation is something like this that the scale distribution can now be represented as 0.5 into trace of the covariance matrix sigma plus the vector mu transpose mu minus k minus log of determinant of the covariance matrix, where this k is actually the dimensionality of the latent code or the latent vector z.


(Refer Slide Time: 10:41)

## Variational Autoencoder : Coding the Cost Functions

$$KL(N(\mu(x), \Sigma(x)) || N(0, I)) = 0.5 * [tr(\Sigma(x)) + \mu(x)^T \mu(x) - k - \log \det(\Sigma(x))]$$

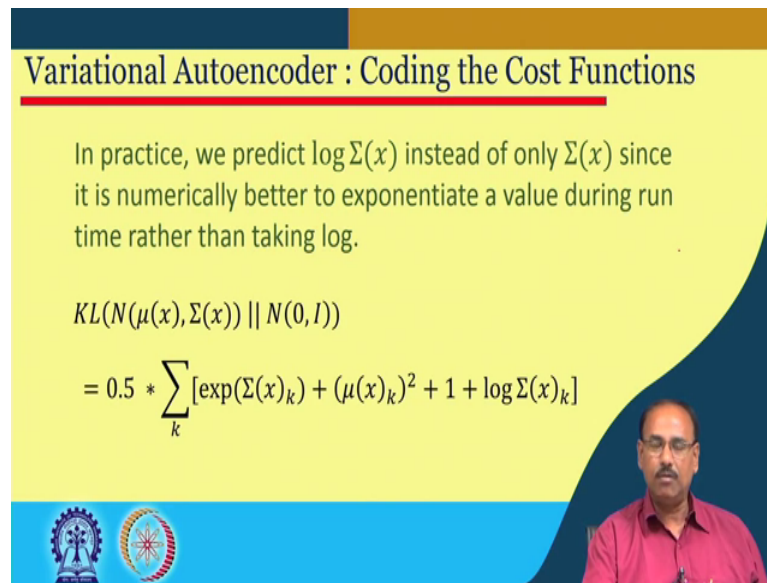
$$= 0.5 * \left[ \sum_k \Sigma(x)_k + \sum_k (\mu(x)_k)^2 + \sum_k 1 - \log \left[ \prod_k \Sigma(x)_k \right] \right]$$

$$= 0.5 * \left[ \sum_k \Sigma(x)_k + \sum_k (\mu(x)_k)^2 + \sum_k 1 - \sum_k \log \Sigma(x)_k \right]$$

$$= 0.5 * \sum_k [\Sigma(x)_k + (\mu(x)_k)^2 + 1 + \log \Sigma(x)_k]$$


And this can be further simplified as all of this I will go come to the final expression.

(Refer Slide Time: 10:47)



Variational Autoencoder : Coding the Cost Functions

In practice, we predict  $\log \Sigma(x)$  instead of only  $\Sigma(x)$  since it is numerically better to exponentiate a value during run time rather than taking log.

$$KL(N(\mu(x), \Sigma(x)) || N(0, I))$$
$$= 0.5 * \sum_k [\exp(\Sigma(x)_k) + (\mu(x)_k)^2 + 1 + \log \Sigma(x)_k]$$

The slide features a yellow background with a blue and orange header. At the bottom left, there are two circular logos. A speaker overlay of a man in a pink shirt is visible in the bottom right corner.

So, this finally, comes down to 0.5 into sum over k the sigma x k plus mu x k square plus 1 plus log of sigma x k. So, this is the final closed form expression for that loss function that we get or the KL divergence form that we get and this is the advantage of assuming normal distribution prior for Q of z given x.

But in practice, we predict log of sigma x instead of sigma x since it is numerically better to exponentiate a value during runtime rather than taking a log operation. So, your final expression that it becomes the KL divergence of Q of x depends z and the normal distribution 0 1 that is equal to 0.5 into. So, instead of sigma x k now it becomes exponential of sigma x k plus mu x k square plus 1 plus log of sum of x k, you find that this log of sum of x k has come from the product term which is basically determinant of the covariance matrix sigma, right. So, this is log not sum of x k, but log of the different components of the covariance matrix in x k, right.

(Refer Slide Time: 12:22)

Variational Autoencoder :After training

Visualizing Reconstructions:

6 3 2 1 6 / 2 0 / 5 Input

6 3 2 1 6 / 2 0 / 5 Reconstructions

Using as Generative Model

- Sample a random vector from  $N(0, I)$
- Feed forward the vector through the pre-trained Decoder


6 5 9 3 9 0 4 6 / 7 Generated Samples

So, now, let us see that what are different kind of output that we can have using this variational auto encoder. So, this gives the reconstructed output from the variational auto encoder for given inputs you have the different reconstructions and this second set of results is showing that if I sample a vector from a normal distribution 0 mean unit variance, then the generator network generates or reconstructs the signal as given in the last one in the last row.


So, you find that in many cases this generator the generated samples or the generated data that closely resemble the amnesty database the characters which are available in the amnesty database.

(Refer Slide Time: 13:15)

### Variational Autoencoder : Generative Model



VAE generating novel faces after trained on CelebA dataset




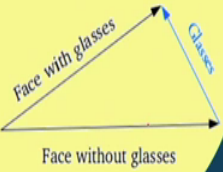
This is again another set of output again reconstructed from this generative model the variational auto encoder which is trained on CelebA database.

(Refer Slide Time: 13:26)

### Variational Autoencoder : Vector Arithmetic

How do you interpolate between two samples ?

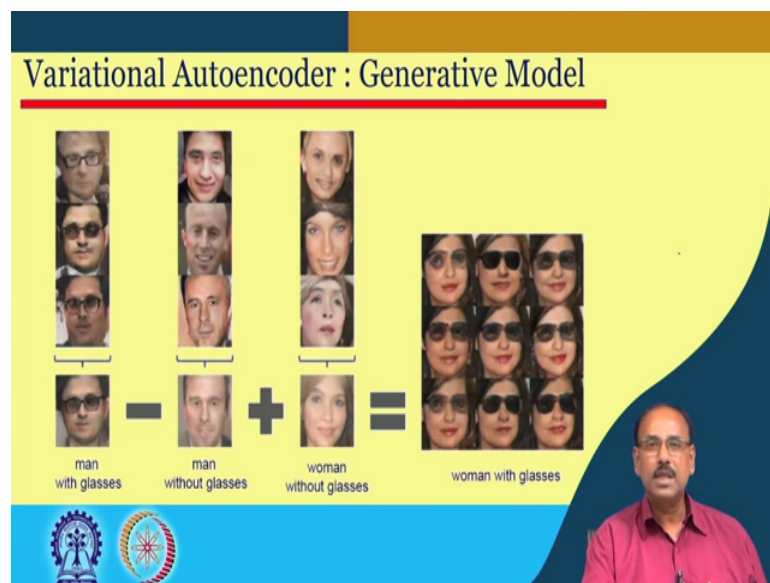
- Take a face image with glasses and find the latent code ( $C_1$ )
- Take another face without glasses and find latent code ( $C_2$ )
- $C_3 = C_1 - C_2$  gives code for glasses
- Take a new face without glasses and find latent code ( $C_4$ )
- $C_3 + C_4$  will overlay glasses on this new image
- Such transitions are possible only if the latent space is continuous instead of clusters



With variational auto encoder we can have another trick following the vector arithmetic approach, that is if we take a set of face images with classes and the corresponding latent code we say is  $C_1$ , then take another set of face images which are without glasses the corresponding latent code is  $C_2$  then  $C_3$  the vector  $C_3$  which is  $C_1$  minus  $C_2$  that gives you the code for the glasses.

Now, if we have a face image without glass we can impose this glass onto that face image. So, you take a new face image without glass and its corresponding latent code is C 4 then you generate a latent code which is C 3 plus C 4. So, you find that C 3 was the latent code for the glass and C 4 is the latent code of a face without glass. So, when we are making C 3 plus C 4 effectively I am generating a latent code for a face with a glass. So, this transition is possible because the latent space is continuous instead of clusters of points that you get in case of traditional auto encoder.

(Refer Slide Time: 14:40)





So a set of results on this so, it says that we had a set of images with glass and we also have a set of images of man without glass you subtract these two you get the latent code of the glass then you have set up images of women which are without glasses. Then, you add the latent code of the glass with latent code of women and then you feed this latent code the latent vector to the generator network and the generator network outputs faces of women with glasses.

So, this is a nice trick where we can play with the latent codes in the latent space and I can generate the output data in various combinations. So, that was your our variational auto encoder now briefly I am going to discuss about what is generative adversarial network. So, generative adversarial network is another form of generative model for reconstruction of the data from the latent code.

(Refer Slide Time: 15:49)

## Implicit Generative Models

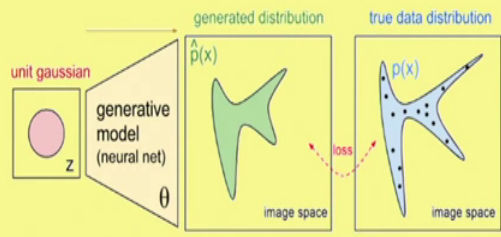
- ❑ Implicitly defines a probability distribution.
- ❑ Sample code vector,  $z$ , from a simple and fixed distribution (e.g. spherical Gaussian or Uniform).
- ❑ A generator network is trained as a differentiable network to map  $z$  to a data point  $x$ .





So, unlike in case of variational auto encoder where the distribution the probability distribution of the latent code was quite explicit, in case of generative model this implicitly defines the probability distribution. So, a sample code vector  $z$  from a sample your sample code vector  $z$  from a simple and fixed distribution which may be say normal or uniform distribution feed it to the generator network, which is trained as a differentiable network and this generator network then map  $x$  to your data maps this latent variable  $z$  to the data point  $x$ .

(Refer Slide Time: 16:31)

## Implicit Generative Models



<https://openai.com/blog/generative-models/>

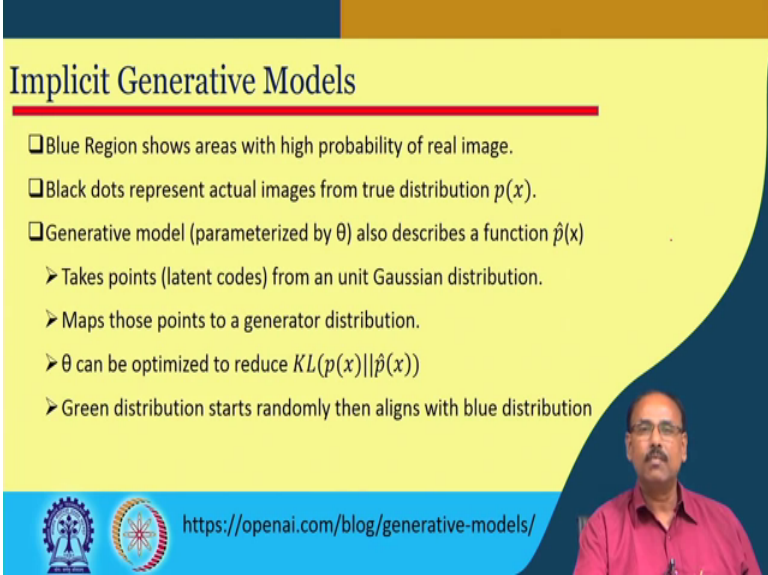




So, effectively what you have is you have the data from a true data distribution and the generator model what it does is, it takes a sample from some distribution may be say unit Gaussian distribution and this then it generates another distribution of data which is the  $\hat{p}(x)$ . Then while cleaning this genetic model what we want to do is we want to minimize the KL divergence between  $\hat{p}(x)$  and  $p(x)$ ,  $p(x)$  was the true distribution of the data and  $\hat{p}(x)$  is the distribution of the generated data.

So, we want to minimize this scale divergence and while minimization of that you try to update the parameters of the generative model. So, that effectively or eventually your  $\hat{p}(x)$  becomes similar to  $p(x)$ .

(Refer Slide Time: 17:28)



**Implicit Generative Models**

- ❑ Blue Region shows areas with high probability of real image.
- ❑ Black dots represent actual images from true distribution  $p(x)$ .
- ❑ Generative model (parameterized by  $\theta$ ) also describes a function  $\hat{p}(x)$ 
  - Takes points (latent codes) from an unit Gaussian distribution.
  - Maps those points to a generator distribution.
  - $\theta$  can be optimized to reduce  $KL(p(x)||\hat{p}(x))$
  - Green distribution starts randomly then aligns with blue distribution

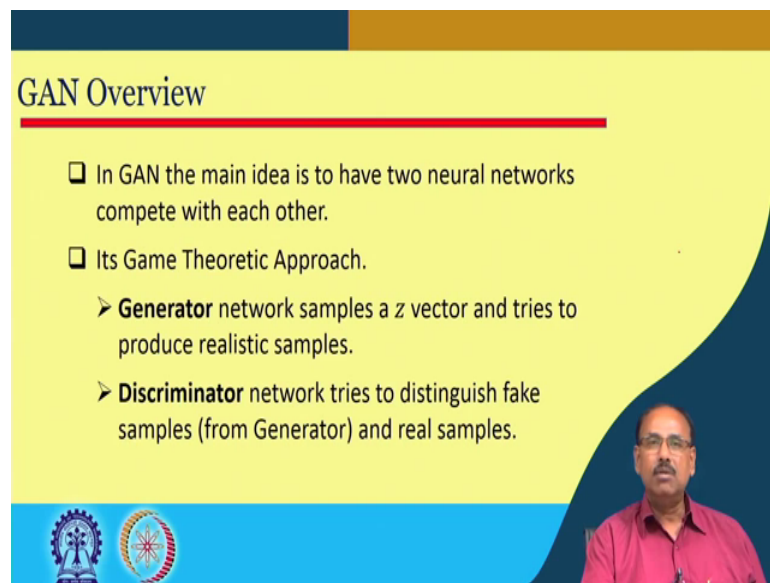
<https://openai.com/blog/generative-models/>

So, as we have shown that the Blue Region in the previous diagram that was your probability distribution of the real image, where as the black dots that was actual images from the true distribution, actual images samples from the true distribution  $p(x)$ . Generative model for which the parameters are theta, this gives you distribution which is  $\hat{p}(x)$ .

Then this was generated by taking points from sampling points from and unique from say a Gaussian distribution and then theta is optimized by minimizing KL divergence between  $p(x)$  and  $\hat{p}(x)$ . And eventually the green distribution initially that starts randomly and then gradually it aligns for the blue distribution as was shown over here.

So, the green distribution is the distribution of the generated data and the blue distribution is the distribution of the real data. So, initially green distribution starts at random and while the training goes on this green distribution eventually becomes similar to the blue distribution.

(Refer Slide Time: 18:43)



The slide is titled "GAN Overview" and features a yellow background with a blue and orange header. It contains a bulleted list of key concepts. In the bottom right corner, there is a small video inset of a man in a red shirt. The bottom left corner has two circular logos.

### GAN Overview

- ❑ In GAN the main idea is to have two neural networks compete with each other.
- ❑ Its Game Theoretic Approach.
  - **Generator** network samples a  $z$  vector and tries to produce realistic samples.
  - **Discriminator** network tries to distinguish fake samples (from Generator) and real samples.

So, this is how the generative model works. So, in generative adversarial network which is popularly known as GAN, the main idea is to have two neural networks and they will compete with each other, in so effectively it is a GAN theoretic approach. One of the neural network is a generator that samples  $z$  vector from the latent space and tries to produce a realistic sample and the discriminator network which is a competitor of the generator network or adversary to the generator network that is why it is adversarial network.

So, the discriminant network it tries to distinguish between the fake samples which are actually coming from the generator and the real samples which are fed to the discriminator to decimate between the real samples and the generated samples now called as fake samples which are generated by the generator. So, while this contest goes on eventually both the generator and the discriminator both of them learns the distribution and this is implicit, it is not explicit unlike in case of variational auto encoder.

(Refer Slide Time: 19:52)

### GAN Overview

- ❑ Assume  $D(x)$  represents probability of belonging to real class for a given sample,  $x$
- ❑ Discriminator will try to increase  $D(x)$  for real samples and decrease  $D(x)$  for fake/generated samples
- ❑ Generator will try to increase  $D(x)$  for generated samples

The diagram illustrates the GAN architecture. A generator  $G$  takes noise  $z$  as input and produces a fake image. A discriminator  $D$  takes both real and fake images as input and outputs probabilities for real and fake classes. The diagram also shows training paths for the Discriminator and Generator.

Legend:  
— Discriminator training  
— Generator training

So, this is what is our network in fact, let us assume that  $D(x)$ , it represents the probability of belonging to real class for a given sample  $x$ . So, the discriminator will try to decrease  $D(x)$  for real samples we will try to increase  $D(x)$  for real examples, because it wants to discriminate between the real samples and the fake samples or the generated samples.

So,  $x$  being input to the discriminant network if this  $x$  comes from a real sample the discriminator will try to increase  $D(x)$  that is it is probability to distribution; whereas, if this  $x$  is a fake sample which comes from the generator network then the discriminator will try to reduce  $D(x)$ . In turn generator will try to increase  $D(x)$  for its own generated samples and that is how they are locked in a GAN.

(Refer Slide Time: 20:51)

### GAN Overview

**Training the Discriminator**

$$\max_D V(D, G) = \underbrace{E_{x \sim p_{data}(x)} \log D(x)}_{\text{Maximize probability for real}} + \underbrace{E_{z \sim p_z(z)} [\log \{1 - D(G(z))\}]}_{\text{Minimize probability for generated}}$$

The slide also features a small video inset of a man in a red shirt in the bottom right corner and logos of institutions in the bottom left corner.

So, by doing this now there are two training components; one of the training component is training of the discriminator and the other training component is training of the generator so, for training we need the cost function. So, for training of the discriminator as we said that decimator wants to maximize the probability for the real data and it minimizes the probability for the generated data or effectively it maximizes the probability or log likelihood of 1 minus D of G z, where G z is the data generated by the generator network from the latent variable which is z.

So, while training the discriminator it is an maximization operation where it is maximization over the log likelihood, log of D x plus the value of estimation value of log of 1 minus D of G z. So, maximization of this will train the discriminator.

(Refer Slide Time: 22:02)

### GAN Overview

**Training the Generator**

$$\min_G V(D, G) = E_{z \sim p_z(z)} [\log\{1 - D(G(z))\}]$$
$$\equiv \max_G E_{z \sim p_z(z)} [\log D(G(z))]$$

Maximize probability for generated

The slide features logos of institutions at the bottom left and a portrait of a man in a red shirt at the bottom right.

On the other hand the second component which is training of the generator; obviously, the generator wants to maximize the probability of its own data which is generated by the generator itself. So, it tries to maximize log of  $D(G(z))$  or expectation value of log of  $D(G(z))$ . So, you find that these two are in opposite direction; the discriminator tries to minimize  $D(G(z))$  probability of the probability distribution of the generated data.

On the other hand the generator tries to maximize log of  $D(G(z))$  so, they are actually working in the opposite direction. In effect what is done is, the generator is trying to fool the discriminator so that discriminator cannot discriminate between which is the real image and which is the fake image. And while doing so, both of them learn the distributions.

(Refer Slide Time: 22:57)


### GAN Training : Alternate updates of D and G

```
for number of training iterations do
  for k steps do
    • Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
    • Sample minibatch of  $m$  examples  $\{x^{(1)}, \dots, x^{(m)}\}$  from data generating distribution  $p_{data}(x)$ .
    • Update the discriminator by ascending its stochastic gradient:
      
$$\nabla_{\theta_D} \frac{1}{m} \sum_{i=1}^m [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$$

    end for
    • Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
    • Update the generator by descending its stochastic gradient:
      
$$\nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z^{(i)})))$$

    end for
  end for
```

Goodfellow et al. "Generative Adversarial Networks",  
NeurIPS, 2014



So, this is just what is shown as the algorithm how the generator and the discriminator networks are trained. So, given a set of noise samples or the latent variables  $z$  the generator network generates a number of data samples, using these data samples that discriminator tries to maximize the same cost function which is  $\log$  of  $D$   $x$  plus  $\log$  of  $1$  minus  $D$   $G$  of  $z$  and once the discriminator is learnt it stabilizes then you come to the training of the generator wherein you try to maximize  $D$  of  $G$   $z$ , ok.

So, these two training operations; these two training operations occur one after another effectively making both of the generator and that discriminator quite strong.

(Refer Slide Time: 23:59)

**GAN Applications: High Resolution Image Synthesis**

Karras, Tero, Timo Aila, Samuli Laine, and Jaakko Lehtinen.  
"Progressive growing of gans for improved quality, stability, and variation." ICLR, 2018.

So, this GAN has been applied in many application domains there was a paper in ICLR in 2018 where this generative adversarial network was used for synthesis of high resolution images. So, this particular slide, shows 2 images which are synthesized and you find that though the images are synthesized, but they are really looking like real images, right. So, that shows what is the power of generative adversarial network.

(Refer Slide Time: 24:35)

**GAN Applications: Image to Image Translation**

Labels to Street Scene    Labels to Facade    BW to Color

Aerial to Map    Day to Night    Edges to Photo

Isola, Phillip, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros.  
"Image-to-image translation with conditional adversarial networks." CVPR, 2017

It has also been used for image to image translation. So, this is a semi peer paper in 2017 where you find that given the semantic segmentation of a scene using the generative



adversarial network it is possible to generate or to possible to synthesize real looking images. Similarly from the input levels you can also generate real look images given black and white image or grayscale images you can generate color images. So, all these are possible using this generative adversarial network.

(Refer Slide Time: 25:12)




This shows another one that if we have a sequence of input frames which are basically semantic segmentation of different frames, then it is possible to generate a real looking video in different style. Again for that we have to train the generated generator network and the adversarial network in a proper way.

(Refer Slide Time: 25:36)

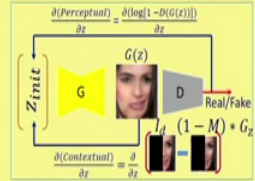
### GAN Applications: Image Inpainting

**Input:** Masked/damaged image,  $I_d$ , with binary mask,  $M$   
**Intermediate Output:** Image,  $I_G$ , after iterative optimization for  $z$   
**Final Output:** Inpainted image,  $\hat{I}_d = M * I_d + (1 - M) * I_G$

Stage 1: Pre-training a GAN



Stage 2: Iterative search for  $z$



Yeh, R. A., Chen, C., Yan Lim, T., Schwing, A. G., Hasegawa-Johnson, M., & Do, M. N. (2017). Semantic image inpainting with deep generative models. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

This is a piece of work which is for image inpainting, image inpainting says is a problem where if you have an input image where there are large areas where the information is not available or the image is damaged. So, we can try to fill up those damaged regions using the information of the overall image.

So, this generative adversarial network has become very effective in that area also. So, for that what you do is, you use a generated network which is pre-trained over the domain of images for which we want this inpainting operation to be active. And then use this generator network to generate a real-looking image and then our aim is that when we compute the cost function.

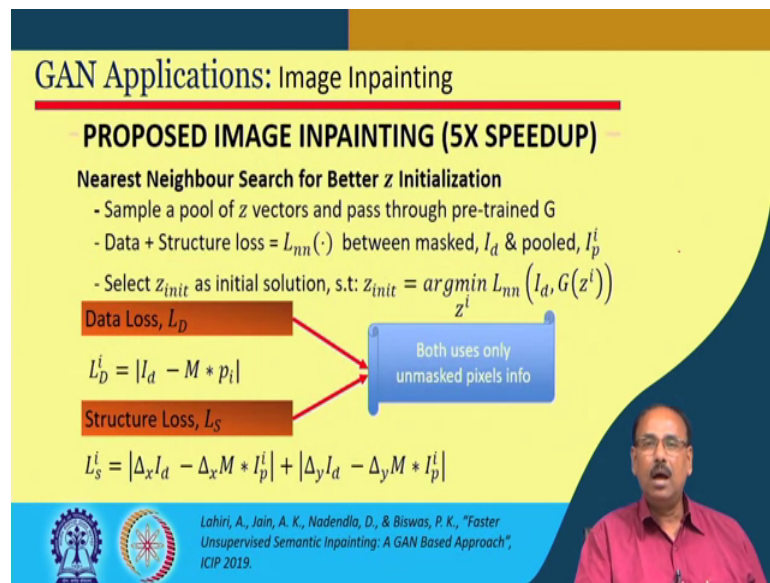
The cost function we will have two components; one is the perceptual component where the discriminator tries to discriminate between this generated image or the fake image and the real image; that means, how realistic this generated image so that gives you our perceptual loss.

And another loss component which is contextual loss, the contextual loss says that there may be some portions in the image for which the information was not available or it was damaged. But, the image which has been generated or reconstructed by the generator in that if I take that portion of this generated image which is matching with the undamaged part of your input damaged image. This undamaged part of the input

damaged image and the corresponding portion in the generated image they should match and that is what gives you the contextual loss.

And what you do is you try to minimize both of these losses the perceptual loss and the contextual loss and while trying to minimize this you go on updating the input vector or the latent vector, and given a latent vector you are getting the generated output.

(Refer Slide Time: 27:44)



**GAN Applications: Image Inpainting**

**PROPOSED IMAGE INPAINTING (5X SPEEDUP)**

**Nearest Neighbour Search for Better z Initialization**

- Sample a pool of z vectors and pass through pre-trained G
- Data + Structure loss =  $L_{nn}(\cdot)$  between masked,  $I_d$  & pooled,  $I_p^i$
- Select  $z_{init}$  as initial solution, s.t:  $z_{init} = \underset{z^i}{\operatorname{argmin}} L_{nn}(I_d, G(z^i))$

**Data Loss,  $L_D$**

$$L_D^i = |I_d - M * p_i|$$

**Structure Loss,  $L_S$**

$$L_S^i = |\Delta_x I_d - \Delta_x M * I_p^i| + |\Delta_y I_d - \Delta_y M * I_p^i|$$

Both uses only unmasked pixels info

Lahiri, A., Jain, A. K., Nadendla, D., & Biswas, P. K., "Faster Unsupervised Semantic Inpainting: A GAN Based Approach", ICIP 2019.

We had done some piece of work on this, where we have done some improvement of the previous work. Where what we have done is, we have added we have used two type of loss. One is the data loss which is similar to the loss function which has been used by this previous authors A Et al where they have published this work presented this work in CVPR 2017. This is the recent work that we have presented in ICIP 2019 in Taipei where in addition to data loss we have also introduced another loss component which is the structure loss.

And in order to improve the computational complexity what we have done is, we have taken random samples attain samples from your latent space and for each of these ten samples. We have generated ten images using the generator network and out of these images you have taken we have taken that one which is closest to the damaged one the damaged input image and the corresponding latent vector z was our initial vector and we updated on this.

So, using both of this we have found that computationally our method has become much more efficient and at the same time that reconstruction, because we have introduced a second loss which the structure loss that is in the gradient space. So, introduction of the structure loss also has improved the quality of our in painted images.

(Refer Slide Time: 29:22)



So, this shows the output the you find that the images in the leftmost column these are the damaged images which are fed to the generative adversarial network. The three images in the middle column, these are the outputs in generated as reported by a at all in their CVPR 2017 paper. And the rightmost column is the output that we have obtained and this has been as we have said that this particular work has been presented at international conference on image processing 2019 at Taipei.

(Refer Slide Time: 30:04)

**GAN Applications: Video Inpainting**

**PROPOSED VIDEO INPAINTING (80X SPEEDUP)**

- **Reuse Noise Priors**
  - Exploit temporal redundancy
  - Temporal neighbours should have close  $z$  representations

Diagram illustrating the reuse of noise priors: Frame  $t-1$  is processed by  $G, D, z_t^{init}$  to produce  $z_{t-1}^{final}$ . This  $z_{t-1}^{final}$  is then used as the initial vector  $z_t^{init}$  for Frame  $t$ , which is processed by  $G, D, z_t^{init}$  to produce  $z_t^{final}$ .

- **Group Consistency Loss**
  - Penalize if a local temporal neighbourhood of  $W$  frames differ
  - Helps in reducing sudden flickering effects across frames
  - Loss =  $|z_k - z_j| \forall i \in [1, 2, \dots, W]; \forall j \in [1, 2, \dots, W]$

Diagram illustrating the group consistency loss: A group of frames  $z_0^{final}, z_1^{final}, \dots, z_W^{final}$  is shown. Arrows indicate the loss calculation between adjacent frames  $z_{i-1}^{final}$  and  $z_i^{final}$ .

Lahiri, A., Jain, A. K., Nadendla, D., & Biswas, P. K., "Faster Unsupervised Semantic Inpainting: A GAN Based Approach", ICIP 2019.

The same concept we have extended for video and painting. So, in case of video inpainting because as you know that video is nothing, but a sequence of frames played at a particular frame rate. So, what we have done is we have grouped those frames into something called group of frames.

And we have tried to do the inpainting for every group of frame. So, for that we had the first group of the frame, first frame of every group was inpainted using the same concept as we have applied in case of images and that particular  $z$  vector that you get that is applied to for inpainting of all the frames in the same group.

And if I take frame  $z$   $t$  minus 1 an initial vector  $z$   $t$  minus 1 for a frame  $t$  minus 1; finally, whatever the final vector we get at convergence that becomes the initial vector for the next frame. And using this also we have observed that our computations is much more efficient than if we use frame by frame as proposed by A Et al. Of course, A Et al did not propose it for the video; they are proposed it proposed their technique for the images. So, what we have done is we have used to that technique for different frames in the video.

(Refer Slide Time: 31:37)



And this particular set of video sequence tells you what are the computations that we have obtained. So, on the left most one of the video samples these are the damaged video sequence, in the middle as we said that what is obtained using the approach of A Et al and the right 2 columns, the right 2 video sequences are what we obtained using our technique. And you find that apparently the output that has been given by our approach using this GAN based approach for unsupervised semantic in painting this appears to give better result then what you get using A Et al.

So, with this we come to the end of today's lecture and in fact, to the end of this particular course on deep learning. I hope you have enjoyed the course and you have been able to learn the different methods of deep learning their applications their optimization techniques. Of course, we have started our course with the conventional machine learning techniques where the feature extraction is manual and then you use the machine learning techniques for feature classification or the data classification there.

And then we have moved on to deep learning or even the feature extraction part has been made part of your machine learning approach and we have talked about different types of networks that deep neural networks for deep learning applications for machine learning applications. And this variational auto encoder or the generative adversarial network which are used for generation of the data or reconstruction of the data or synthesis of the

data, this is these approaches are actually the recent trends in a deep learning domain. So, I hope you have enjoyed this course and the course was really useful for you.

Thank you.