# Deep Generative Models

CS772A: Probabilistic Machine Learning

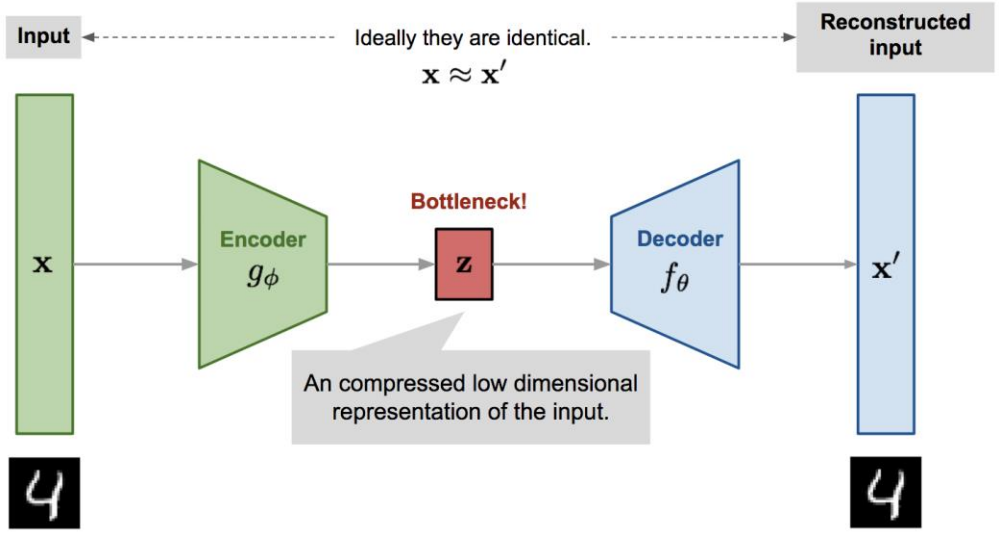
Piyush Rai

# Plan today

- Variational Auto-encoder (VAE)
- Generative Adversarial Network (GAN)
- Denoising Diffusion Model

# Variational Auto-encoder (VAE)

- VAE* is a probabilistic extension of autoencoders (AE). An AE is shown below




$$L_{\text{AE}}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}^{(i)} - f_\theta(g_\phi(\mathbf{x}^{(i)})))^2$$

- The basic difference is that VAE assumes a prior $\boldsymbol{p(z)}$ on the latent code $\boldsymbol{z}$
  - This enables it to not just compress the data but also generate synthetic data
  - How: Sample $\boldsymbol{z}$ from a prior and pass it through the decoder
- Thus VAE can learn good latent representation + generate novel synthetic data
- The name has "Variational" in it since it is learned using VI principles

*Autoencoding Variational Bayes (Kingma and Welling, 2013)

Pic source: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

# Variational Autoencoder (VAE)

- VAE has three main components
  - A prior $p_\theta(z)$ over latent codes
  - A probabilistic decoder/generator $p_\theta(x|z)$, modeled by a deep neural net
  - A posterior or probabilistic encoder $p_\theta(z|x)$ approx. by an "inference network" $q_\phi(z|x)$

Here $\theta$ collectively denotes all the parameters of the prior and likelihood

Using the idea of "Amortized Inference"

- VAE is learned by maximizing the ELBO

Here $\phi$ collectively denotes all the parameters that define the inference network

ELBO for a single data point

$$\mathcal{L}(\theta, \phi|\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right]$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - \mathbb{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})\right)$$

Maximized to find the optimal $\theta$ and $\phi$

$q_\phi$ should be such that data $x$ is reconstructed well from $z$ (high log-lik)

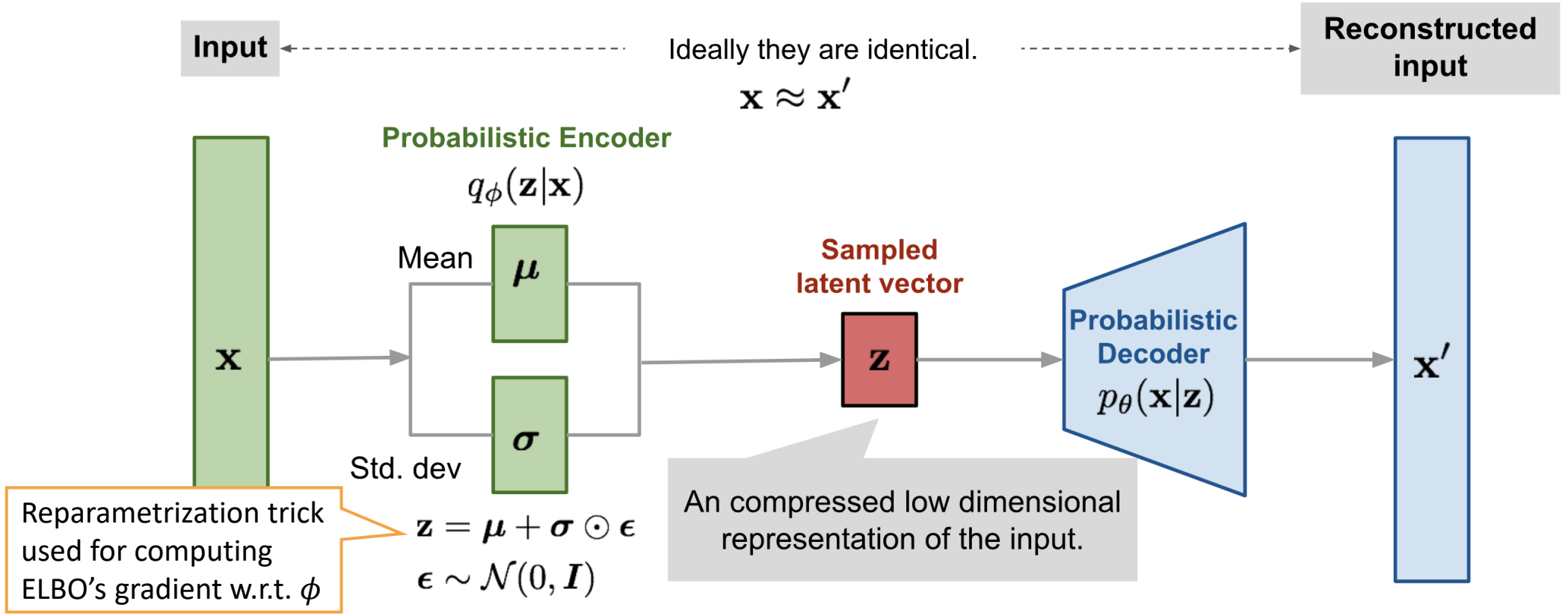
$q_\phi$ should also be simple (close to the prior)

- The Reparametrization Trick is commonly used to optimize the ELBO
- Posterior is inferred only over $z$, and usually only point estimate on $\theta$

# Variational Autoencoder: The Complete Pipeline

- Both probabilistic encoder and decoder learned jointly by maximizing the ELBO

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right]$$
$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] - \mathbb{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z})\right)$$



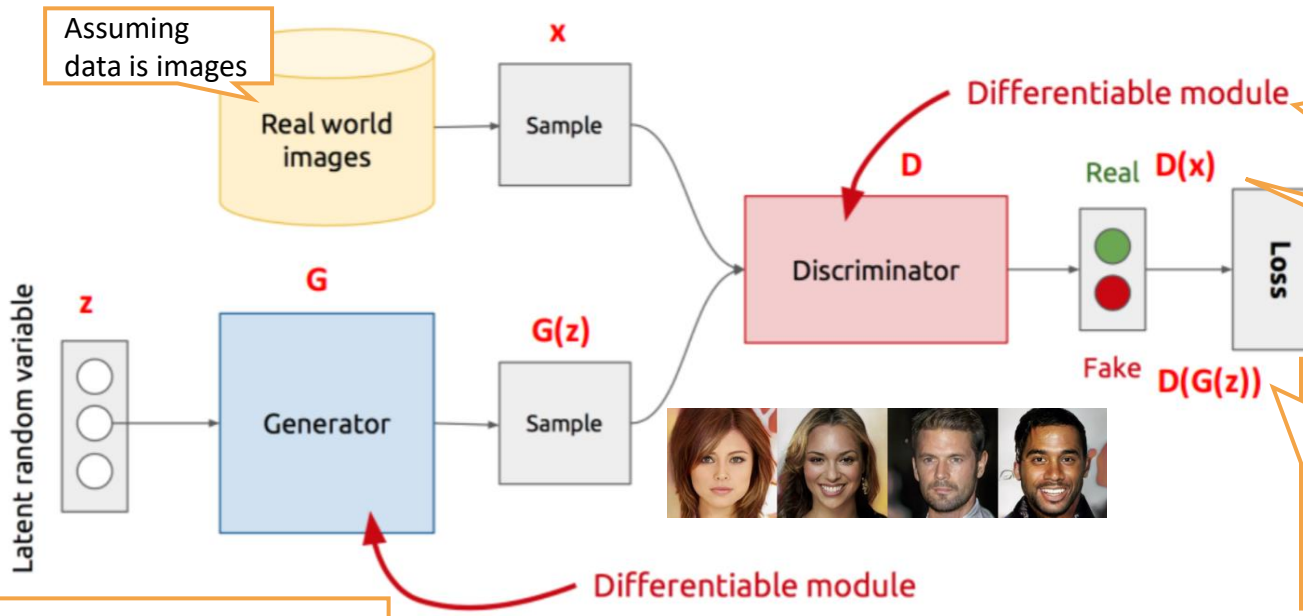

Pic source: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

# Generative Adversarial Network (GAN)

- GAN is an implicit generative latent variable model
- Can generate from it but can't compute $p(x)$ - the model doesn't define it explicitly
- GAN is trained using an adversarial way (Goodfellow et al, 2013)

Unlike VAE, no explicit parametric likelihood model $p(x|z)$

Thus can't train using methods that require likelihood (MLE, VI, etc)




The discriminator can be a binary classifier or any method that can compare b/w two distributions (real and fake here)

Discriminator network is trained to make $D(x)$ close to 1

Discriminator network is trained to make $D(G(z))$ close to 0 and generator network is trained to make it to be close to 1 to fool the discriminator into believing that $G(z)$ is a real sample

Min-max optimization

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))]$$

# Generative Adversarial Network (GAN)

- The GAN training criterion was

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}))]$$

- With $\boldsymbol{G}$ fixed, the optimal $\boldsymbol{D}$ (exercise)

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

Distribution of real data

Distribution of synthetic data

- Given the optimal $\boldsymbol{D}$, The optimal generator $\boldsymbol{G}$ is found by minimizing

$$V(D_G^*, G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}\left[\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(x)}{p_{data}(x) + p_g(x)}\right]$$

$$= \mathrm{KL}\left[p_{data}(x) \middle\| \frac{p_{data}(x) + p_g(x)}{2}\right] + \mathrm{KL}\left[p_g(x) \middle\| \frac{p_{data}(x) + p_g(x)}{2}\right] - \log 4$$

Jensen-Shannon divergence between $p_{data}$ and $p_g$. Minimized when $p_g = p_{data}$

Thus GAN can learn the true data distribution if the generator and discriminator have enough modeling power

# GAN Optimization

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z))]$$

- The GAN training procedure can be summarized as

$\theta_g$ and $\theta_d$ denote the params of the deep neural nets defining the generator and discriminator, respectively

1 Initialize $\boldsymbol{\theta}_g$, $\boldsymbol{\theta}_d$;
2 **for** *each training iteration* **do**
3  **for** $K$ *steps* **do**
4  Sample minibatch of $M$ noise vectors $\mathbf{z}_m \sim q_z(\mathbf{z})$;
5  Sample minibatch of $M$ examples $\mathbf{x}_m \sim p_D$;
6  Update the discriminator by performing stochastic gradient *ascent* using this gradient:
 $\nabla_{\boldsymbol{\theta}_d} \frac{1}{M} \sum_{m=1}^{M} [\log D(\mathbf{x}_m) + \log(1 - D(G(\mathbf{z}_m)))]$. ;
7  Sample minibatch of $M$ noise vectors $\mathbf{z}_m \sim q_z(\mathbf{z})$;
8  Update the generator by performing stochastic gradient *descent* using this gradient:
 $\nabla_{\boldsymbol{\theta}_g} \frac{1}{M} \sum_{m=1}^{M} \log(1 - D(G(\mathbf{z}_m)))$. ;
9 Return $\boldsymbol{\theta}_g$, $\boldsymbol{\theta}_d$

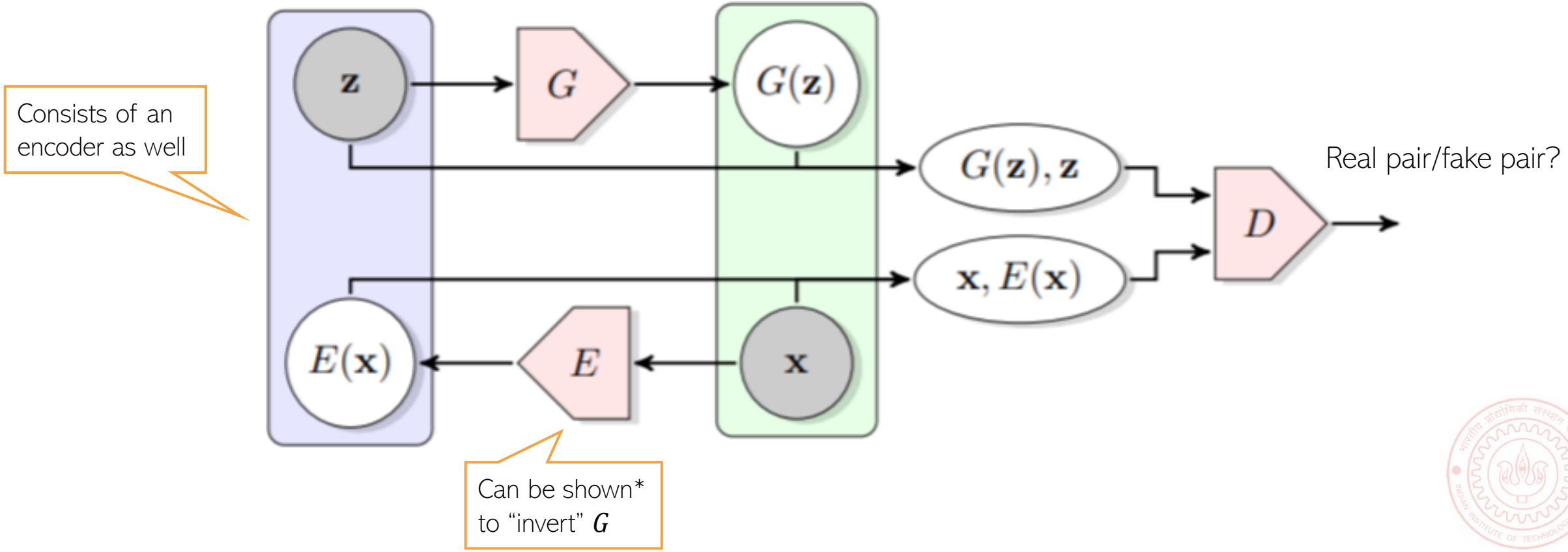
In practice, for stable training, we run $K > 1$ steps of optimizing w.r.t. $D$ and 1 step of optimizing w.r.t. $G$

In practice, in this step, instead of minimizing $\log(1 - D(G(z))$, we maximize $\log\left(D(G(z))\right)$

Reason: Generator is bad initially so discriminator will always predict correctly initially and $\log(1 - D(G(z))$ will saturate

# GANs that also learn latent representations

- The standard GAN can only generate data. Can't learn the latent $\boldsymbol{z}$ from $\boldsymbol{x}$
- Bidirectional GAN* (BiGAN) is a GAN variant that allows this

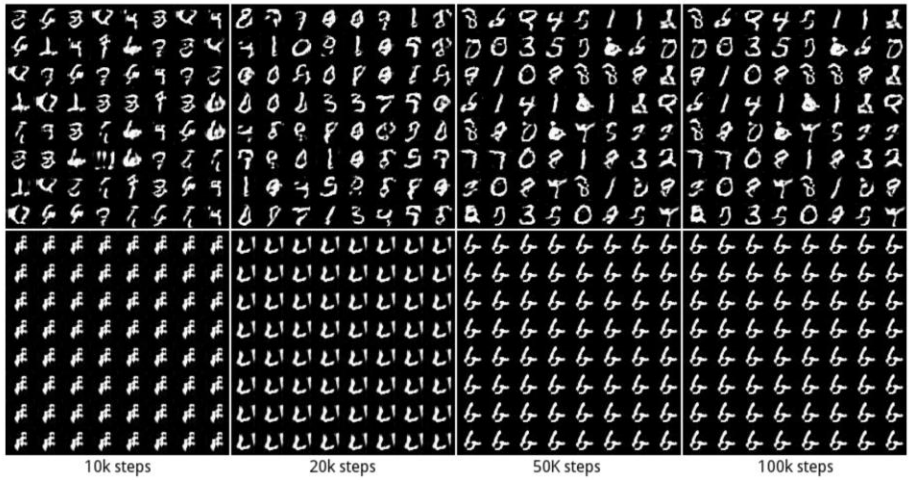



*Adversarial Feature Learning (Donahue et a Dumoulin l, 2017)

#Adversarially Learned Inference (Dumoulin et al, 2017)

# GAN: Some Issues/Comments

- GAN training can be hard and the basic GAN suffers from several issues
- Instability of training procedure
- Mode Collapse problem: Lack of diversity in generated samples
  - Generator may find some data that can easily fool the discriminator
  - It will stuck at that mode of the data distribution and keep generating data like that




GAN 1: No mode collapse (all 10 modes captured in generation)

GAN 2: Mode collapse (stuck on one of the modes)

- Some work on addressing these issues (e.g., Wasserstein GAN, Least Squares GAN, etc)

# Evaluating Generation Quality

High IS and low FID is desirable

Both IS and FID measure how realistic the generated data is

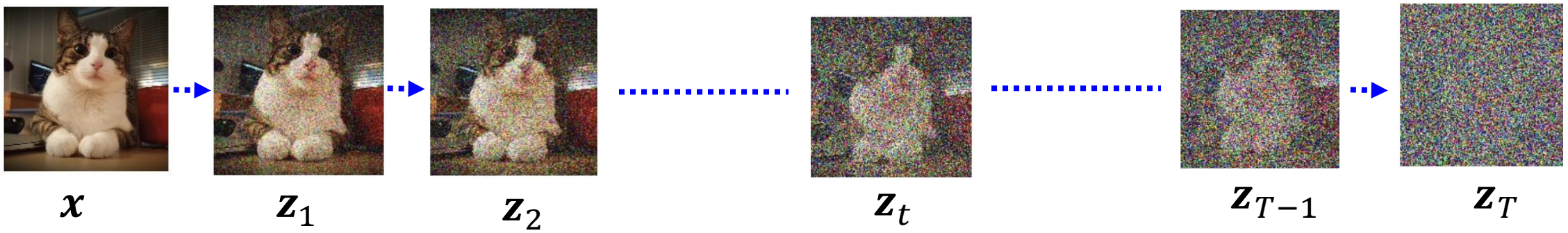
- Two measures that are commonly used
  - Inception score (IS): Evaluates the distribution of generated data
  - Frechet inception distance (FID): Compared the distribution of real data and generated data
- Inception Score defined as $\exp(\mathbb{E}_{x\sim p_g}[\mathrm{KL}(p(y|x)||p(y))])$ will be high if
  - Very few high-probability classes in each sample $x$: Low entropy for $p(y|x)$
  - We have diverse classes across samples: Marginal $p(y)$ is close to uniform (high entropy)
- FID uses extracted features (using a deep neural net) of real and generated data
  - Usually from the layers closer to the output layer
- These features are used to estimate two Gaussian distributions

Using real data $\mathcal{N}(\mu_R, \Sigma_R)$ $\mathcal{N}(\mu_G, \Sigma_G)$ Using generated data

- FID is then defined as $\mathrm{FID} = |\mu_G - \mu_R|^2 + \mathrm{trace}(\Sigma_G + \Sigma_R - (\Sigma_G\Sigma_R)^{1/2})$

# Denoising Diffusion Models

- Consider gradually corrupting an image ($\boldsymbol{z_0 = x}$) till it becomes pure noise ($\boldsymbol{z_T}$)




- Each step $\boldsymbol{z_{t-1} \rightarrow z_t}$ is a pre-defined Gaussian perturbation (forward process)

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t|\sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I})$$

$$\mathbf{z}_t = \sqrt{1-\beta_t}\mathbf{z}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon} \quad (\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}))$$

$\beta_t \in (0,1)$ and $\beta_1 < \beta_2 < \cdots \beta_{T-1} < \beta_T$

Usually pre-defined but can also be learned

**implies**

Imp: Thus we can also compute $\boldsymbol{z_t}$ from $\boldsymbol{x}$ directly in a single step

$$q(\mathbf{z}_t|\mathbf{x}) = \mathcal{N}(\mathbf{z}_t|\sqrt{\alpha_t}\mathbf{x}, (1-\alpha_t)\mathbf{I})$$

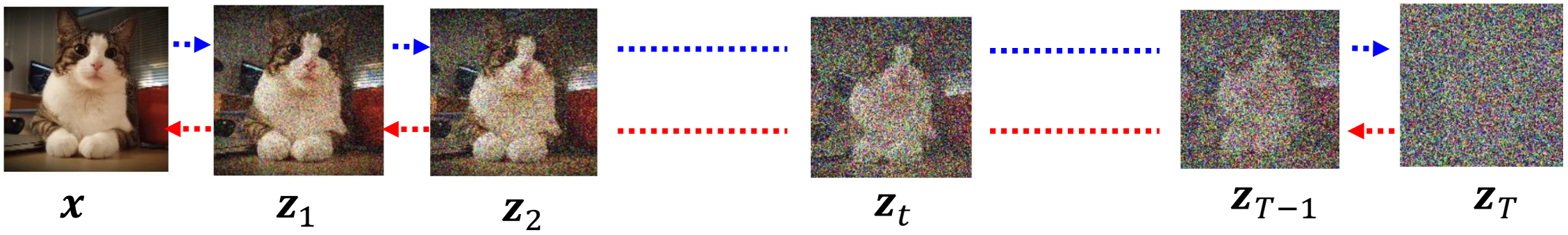
where $\alpha_t = \prod_{\tau=1}^{t}(1-\beta_\tau)$

$$\mathbf{z}_t = \sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}$$

$$q(\mathbf{z}_T|\mathbf{x}) = \mathcal{N}(\mathbf{z}_T|\mathbf{0}, \mathbf{I}) \quad \text{as } T \rightarrow \infty$$

# Generating Data by Reversing Diffusion

- Reversing the diffusion (red arrows) would enable generating data from pure noise




- To reverse the diffusion, we need the distribution of $z_{t-1}$ given $z_t$, i.e., $q(z_{t-1}|z_t)$

The denoising distribution

Intractable because $q(z_t)$ and $q(z_{t-1})$ are difficult to compute

$$q(z_{t-1}|z_t) = \frac{q(z_{t-1})\, q(z_t|z_{t-1})}{q(z_t)}$$

$$q(z_t) = \int q(z_t|x)p(x)dx$$

Since the true data distribution $p(x)$ is not known, we can't compute this integral

# Towards a Tractable Reverse Diffusion

- Although $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$ isn't tractable, the following distribution is tractable

$$q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) = \frac{q(\boldsymbol{z}_{t-1}|\boldsymbol{x})\, q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, \boldsymbol{x})}{q(\boldsymbol{z}_t|\boldsymbol{x})}$$

$$= \frac{q(\boldsymbol{z}_{t-1}|\boldsymbol{x})\, q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}{q(\boldsymbol{z}_t|\boldsymbol{x})}$$

- Reason: $q(\boldsymbol{z}_{t-1}|\boldsymbol{x})$ and $q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})$ are Gaussians, so $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})$ is Gaussian

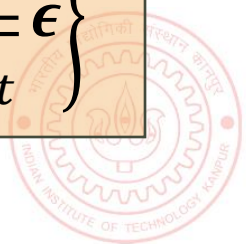
$$q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}_{t-1}|m(\boldsymbol{x}, \boldsymbol{z}_t), \sigma_t^2 \mathbf{I})$$

$$m(\boldsymbol{x}, \boldsymbol{z}_t) = \frac{(1-\alpha_{t-1})\sqrt{1-\beta_t}\boldsymbol{z}_t + \sqrt{\alpha_{t-1}}\beta_t \boldsymbol{x}}{1-\alpha_t}$$

$$\sigma_t^2 = \frac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t}$$

Using $\boldsymbol{x} = \frac{1}{\sqrt{\alpha_t}}\boldsymbol{z}_t - \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}}\boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

$$= \frac{1}{\sqrt{1-\beta_t}}\left\{\boldsymbol{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{\epsilon}\right\}$$



# Towards a Tractable Reverse Diffusion

- We saw that the reverse diffusion distribution is the Gaussian

$$q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}_{t-1}|m(\boldsymbol{x}, \boldsymbol{z}_t), \sigma_t^2 \mathbf{I})$$

where $$m(\boldsymbol{x}, \boldsymbol{z}_t) = \frac{1}{\sqrt{1-\beta_t}}\left\{\boldsymbol{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{\epsilon}\right\}$$

Issue: At generation time, we don't have $\boldsymbol{x}$ (the goal is to generate $\boldsymbol{x}$ which is only available for training data) so we can't use $m(\boldsymbol{x}, \boldsymbol{z}_t)$ at generation time since it depends on $\boldsymbol{x}$

- Let's approximate $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})$ by another Gaussian that doesn't depend on $\boldsymbol{x}$

$$p(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{w}) = \mathcal{N}(\boldsymbol{z}_{t-1}|\mu(\boldsymbol{z}_t, \boldsymbol{w}, t), \Sigma(\boldsymbol{z}_t, \boldsymbol{w}, t))$$

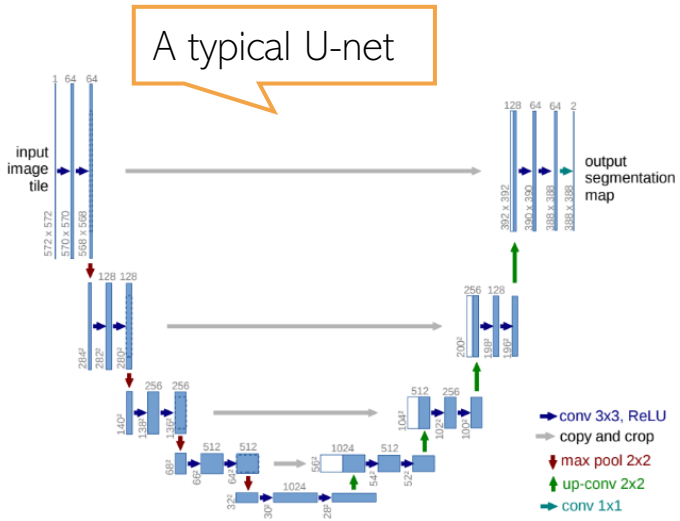
- Usually, $\Sigma(\boldsymbol{z}_t, \boldsymbol{w}, t)$ is chosen to be spherical. A popular choice: $\Sigma(\boldsymbol{z}_t, \boldsymbol{w}, t) = \beta_t \mathbf{I}$
- The mean $\mu(\boldsymbol{z}_t, \boldsymbol{w}, t)$ is defined to mimic the form of $m(\boldsymbol{x}, \boldsymbol{z}_t)$

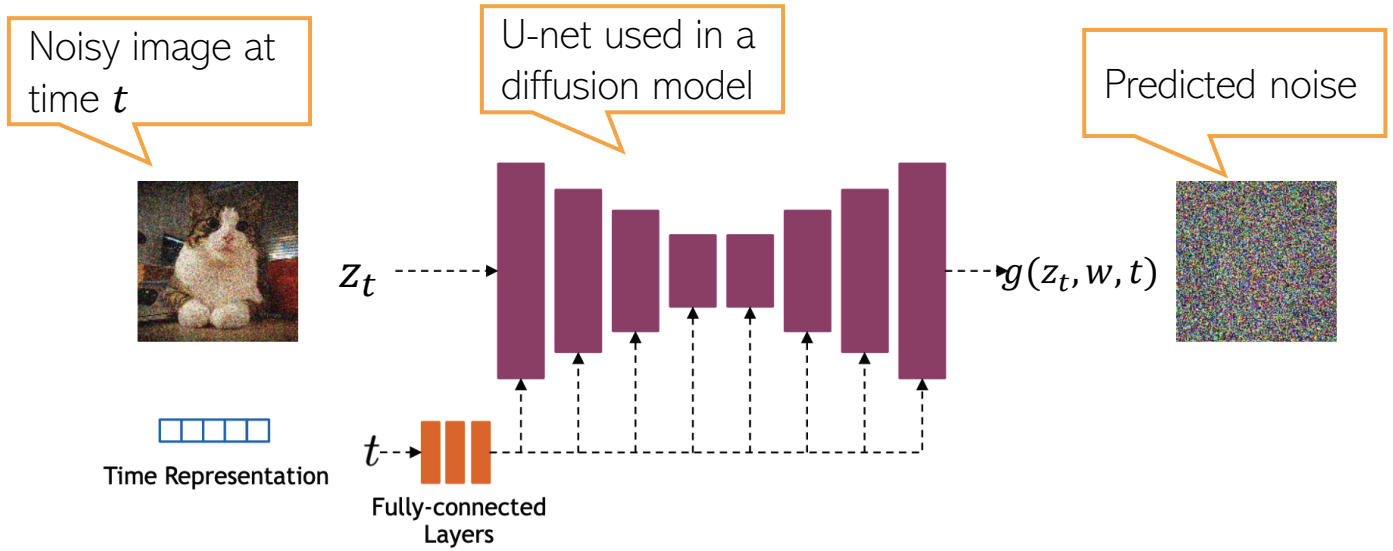
$$\mu(\boldsymbol{z}_t, \boldsymbol{w}, t) = \frac{1}{\sqrt{1-\beta_t}}\left\{\boldsymbol{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} g(\boldsymbol{z}_t, \boldsymbol{w}, t)\right\}$$

Our goal will be to model $g()$ as a neural network which predicts the noise $\boldsymbol{\epsilon}$ which was added to $\boldsymbol{x}$ to get its noisy version $\boldsymbol{z}_t$

# Noise Predictor Network

- A "U-net" model (a neural net) is commonly used as the noise predictor network




- An embedding (positional embedding) of the time-step $t$ is fed into the residual blocks of the U-net architecture

Figure source: Arash Vahdat

# Denoising Diffusion Model: The Training Algo

- The overall training algo is as follows

**Input:** Training data $\mathcal{D} = \{\mathbf{x}_n\}$
Noise schedule $\{\beta_1, \ldots, \beta_T\}$

**Output:** Network parameters $\mathbf{w}$

---

**for** $t \in \{1, \ldots, T\}$ **do**
  $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1-\beta_\tau)$ // Calculate alphas from betas
**end for**

**repeat**
  $\mathbf{x} \sim \mathcal{D}$ // Sample a data point
  $t \sim \{1, \ldots, T\}$ // Sample a point along the Markov chain
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ // Sample a noise vector
  $\mathbf{z}_t \leftarrow \sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}$ // Evaluate noisy latent variable
  $\mathcal{L}(\mathbf{w}) \leftarrow \|\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t) - \boldsymbol{\epsilon}\|^2$ // Compute loss term
  Take optimizer step
**until** converged

**return** $\mathbf{w}$

Pseudo-code from Bishop & Bishop (2023)

# Denoising Diffusion Model: Generation

- Using the training model, we can now generate data as follows

**Input:** Trained denoising network $\mathbf{g}(\mathbf{z}, \mathbf{w}, t)$

Noise schedule $\{\beta_1, \ldots, \beta_T\}$

**Output:** Sample vector $\mathbf{x}$ in data space

---

$\mathbf{z}_T \sim \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ `// Sample from final latent space`

**for** $t \in T, \ldots, 2$ **do**

$\alpha_t \leftarrow \prod_{\tau=1}^{t}(1-\beta_\tau)$ `// Calculate alpha`

`// Evaluate network output`

$\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) \leftarrow \frac{1}{\sqrt{1-\beta_t}}\left\{\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)\right\}$

$\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ `// Sample a noise vector`

$\mathbf{z}_{t-1} \leftarrow \boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) + \sqrt{\beta_t}\boldsymbol{\epsilon}$ `// Add scaled noise`

**end for**

$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\left\{\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}}\mathbf{g}(\mathbf{z}_1, \mathbf{w}, t)\right\}$ `// Final denoising step`

**return** $\mathbf{x}$

Generation can be slow because it requires several steps

Reducing the number of steps is an active area of research

One such approach is DDIM (denoising diffusion implicit model) which relaxes the Markov assumption in the noise process

Pseudo-code from Bishop & Bishop (2023)