EECS 230 Deep Learning

Lecture 16: Generative Models

Some slides from Frank Noe, Simon Prince and Pascal Poupart

Outline

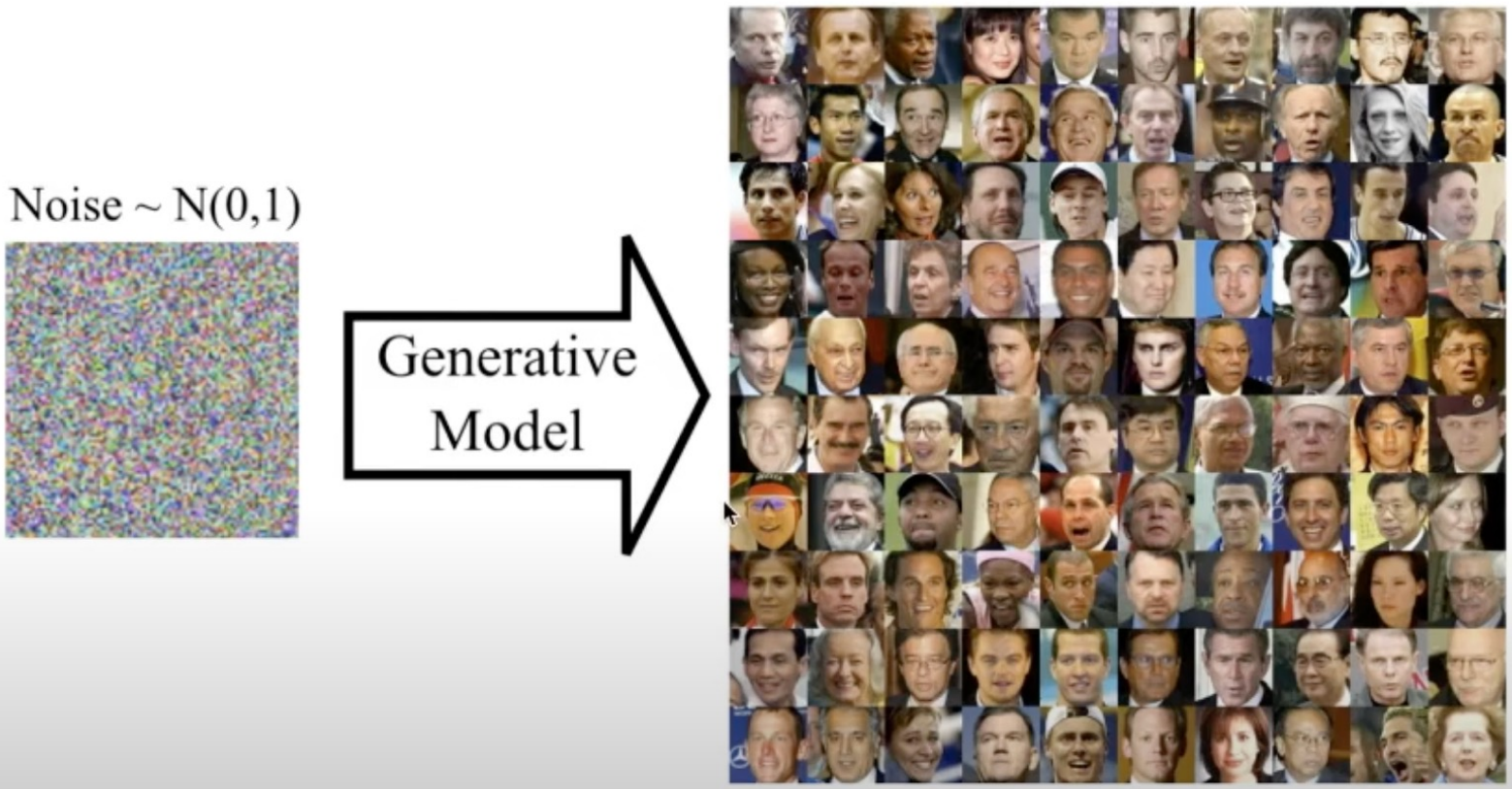
- ❑Generative model
- ❑Variational autoencoder
  - ❑Autoencoder
  - ❑Variational autoencoder
- ❑Generative Adversarial Network
- ❑Diffusion Model

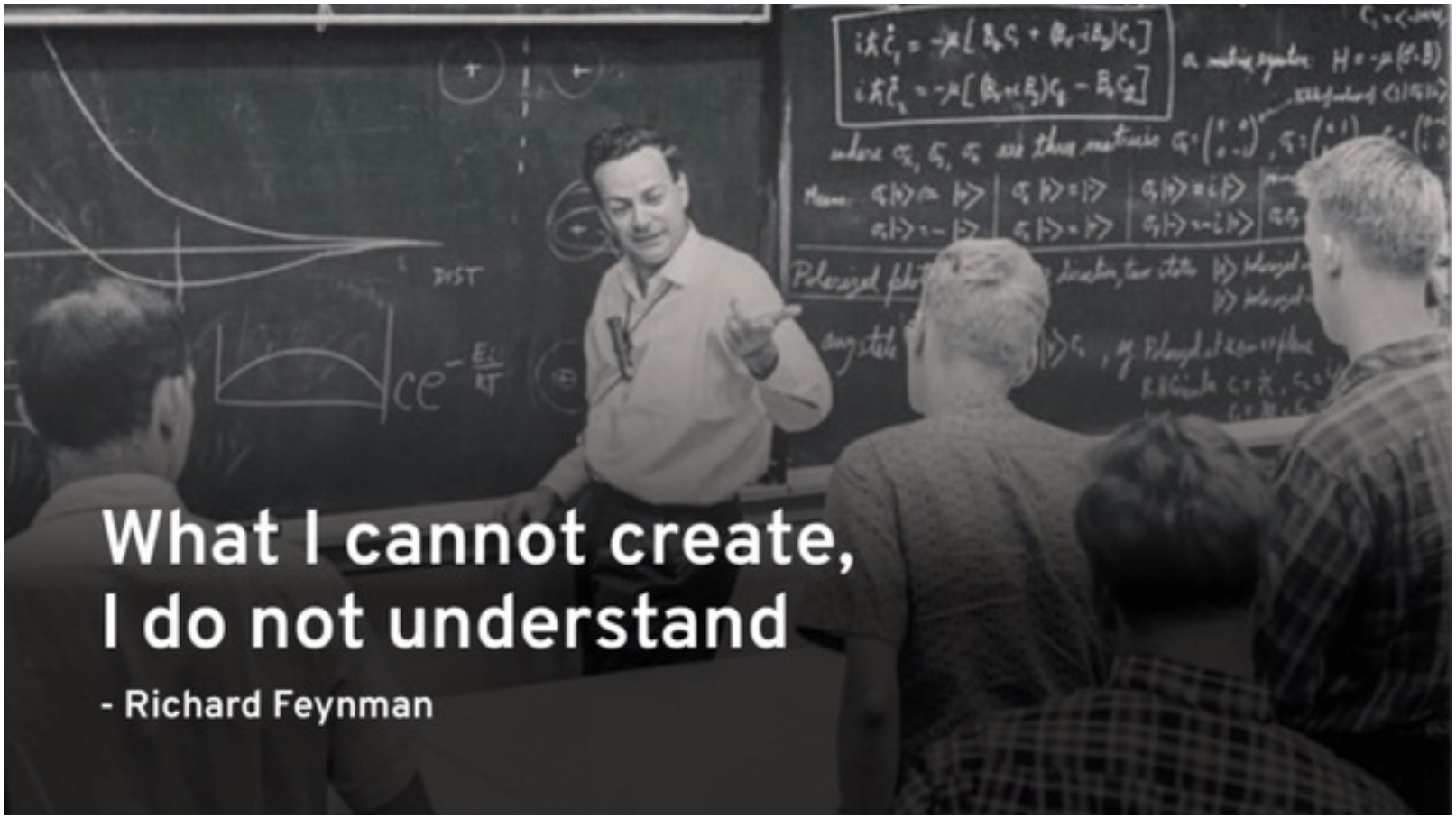
Generative model

So far…

- Discriminative model $P(y|x)$
  - Given input data x, predict y
  - E.g., classification, regression
- Generative model
  - Model data distribution P(x)
  - Sample from P(x) to generate new data

Generative model





UCMERCED

Types of generative neural networks

- ❑Boltzmann machines
- ❑Sigmoid belief networks
- ❑**Variational autoencoders (inference net + generator net)**
- ❑**Generative adversarial networks (generator net + discrimator net)**
- ❑Normalizing flows
- ❑**Diffusion Models**
- ❑…

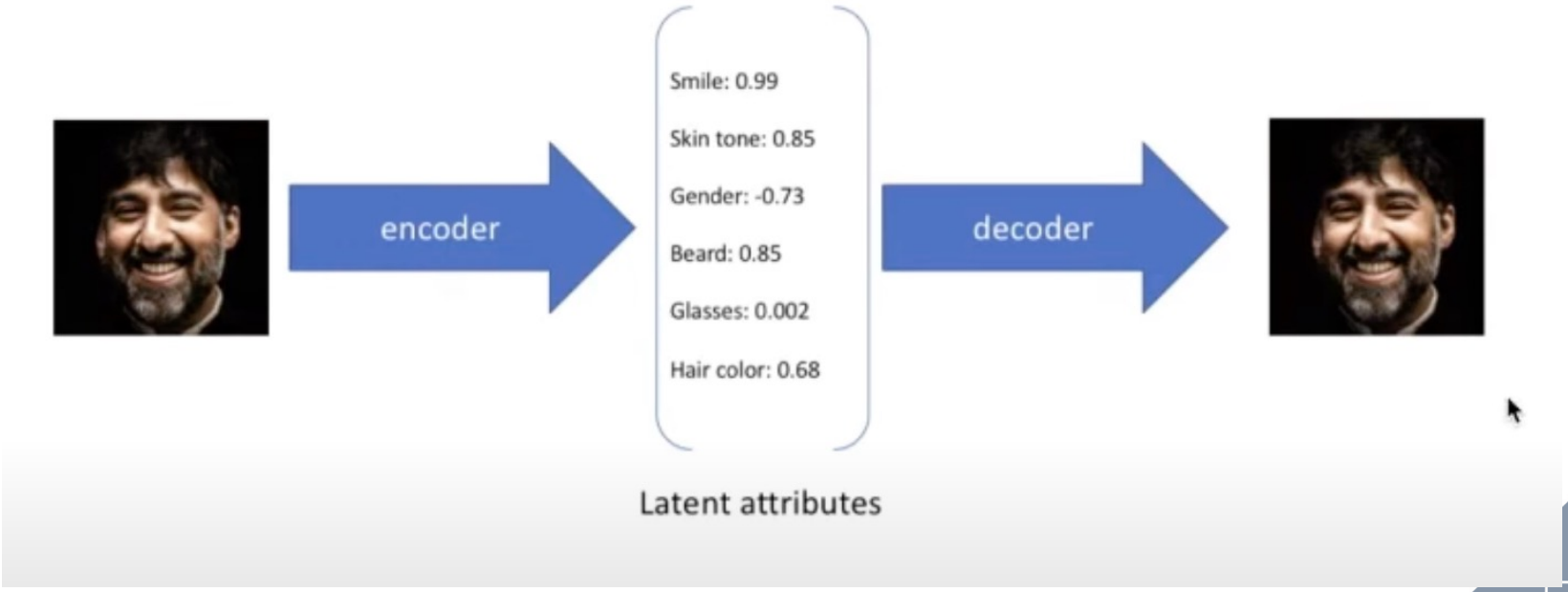
Variational Autoencoder

Autoencoder

- Special type of feed forward network for
  - Compression
  - Denoising
  - Sparse representation
  - Data generation

Autoencoder

- Encoder: f( )
- Decoder: g( )
- Autoencoder: g(f(x)) = x



Latent variable encodes “essential” information about input data

Linear autoencoder

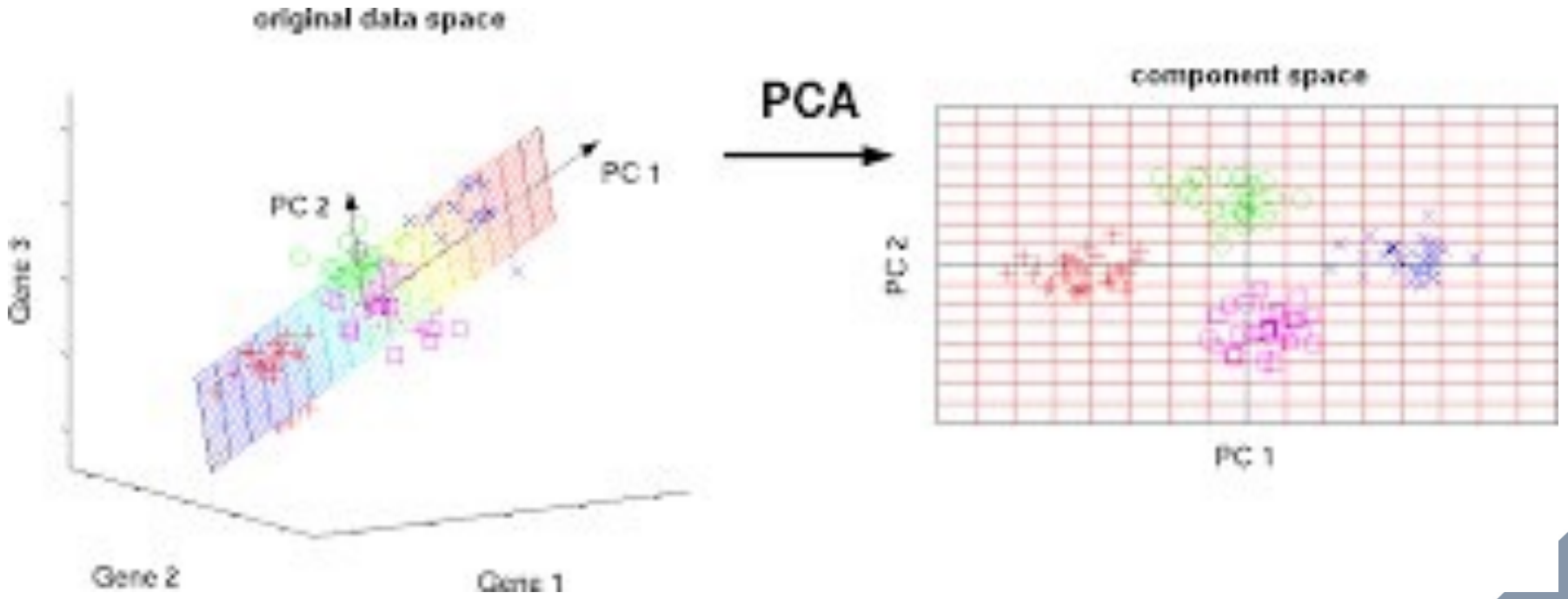
- Objective: find weights W_f and W_g that minimize reconstruction error

$$\min_{\boldsymbol{W}} \frac{1}{2} \sum_{n} \left|\left| \boldsymbol{W}_g \boldsymbol{W}_f \boldsymbol{x}_n - \boldsymbol{x}_n \right|\right|_2^2$$

- When using Euclidean norm (i.e., squared loss), solution is the same as principal component analysis (PCA)

Recap: principle component analysis

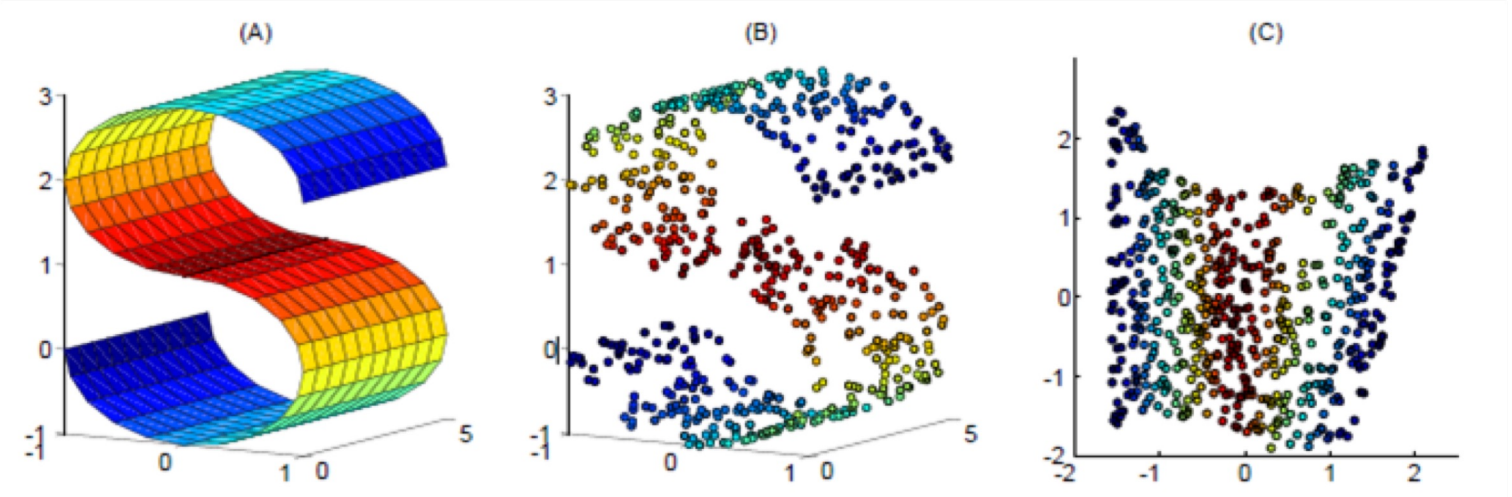
- Components with maximum variance



Non-linear autoencoder

- f( ) and g( ) are both non-linear functions

$$\min_{W} \frac{1}{2} \sum_{n} \left|\left| g\left(f\left(\boldsymbol{x}_n; \boldsymbol{W}_f\right); \boldsymbol{W}_g\right) - \boldsymbol{x}_n \right|\right|_2^2$$



UCMERCED

Sparse representation

- When more hidden nodes than inputs, use regularization to constrain autoencoder
- Example: force hidden nodes to be sparse

$$\min_{\boldsymbol{W}} \frac{1}{2} \sum_{n} \left|\left| g\left(f\left(\boldsymbol{x}_n; \boldsymbol{W}_f\right); \boldsymbol{W}_g\right) - \boldsymbol{x}_n \right|\right|_2^2 + \mathrm{c}\, \underbrace{nnz\left(f\left(\boldsymbol{x}_n; \boldsymbol{W}_f\right)\right)}_{\text{Sparse hidden nodes}}$$

where $nnz\left(f\left(\boldsymbol{x}_n; \boldsymbol{W}_f\right)\right)$ is the number of non-zero entries in the vector produced by f.

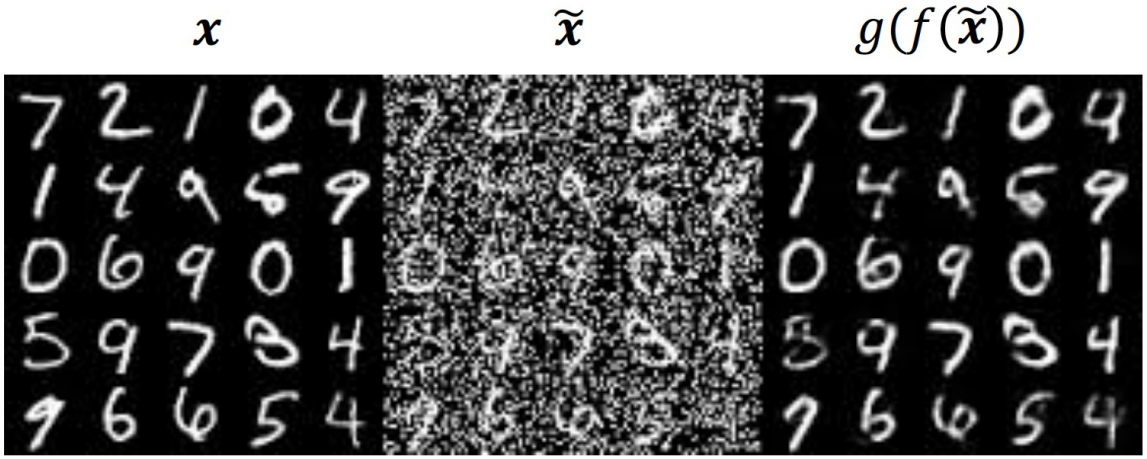
- Aporoximate objective: L_1 regularization

$$\min_{\boldsymbol{W}} \frac{1}{2} \sum_{n} \left|\left| g\left(f\left(\boldsymbol{x}_n; \boldsymbol{W}_f\right); \boldsymbol{W}_g\right) - \boldsymbol{x}_n \right|\right|_2^2 + \mathrm{c}\, \left|\left| f\left(\boldsymbol{x}_n; \boldsymbol{W}_f\right) \right|\right|_1$$

Denoising autoencoder

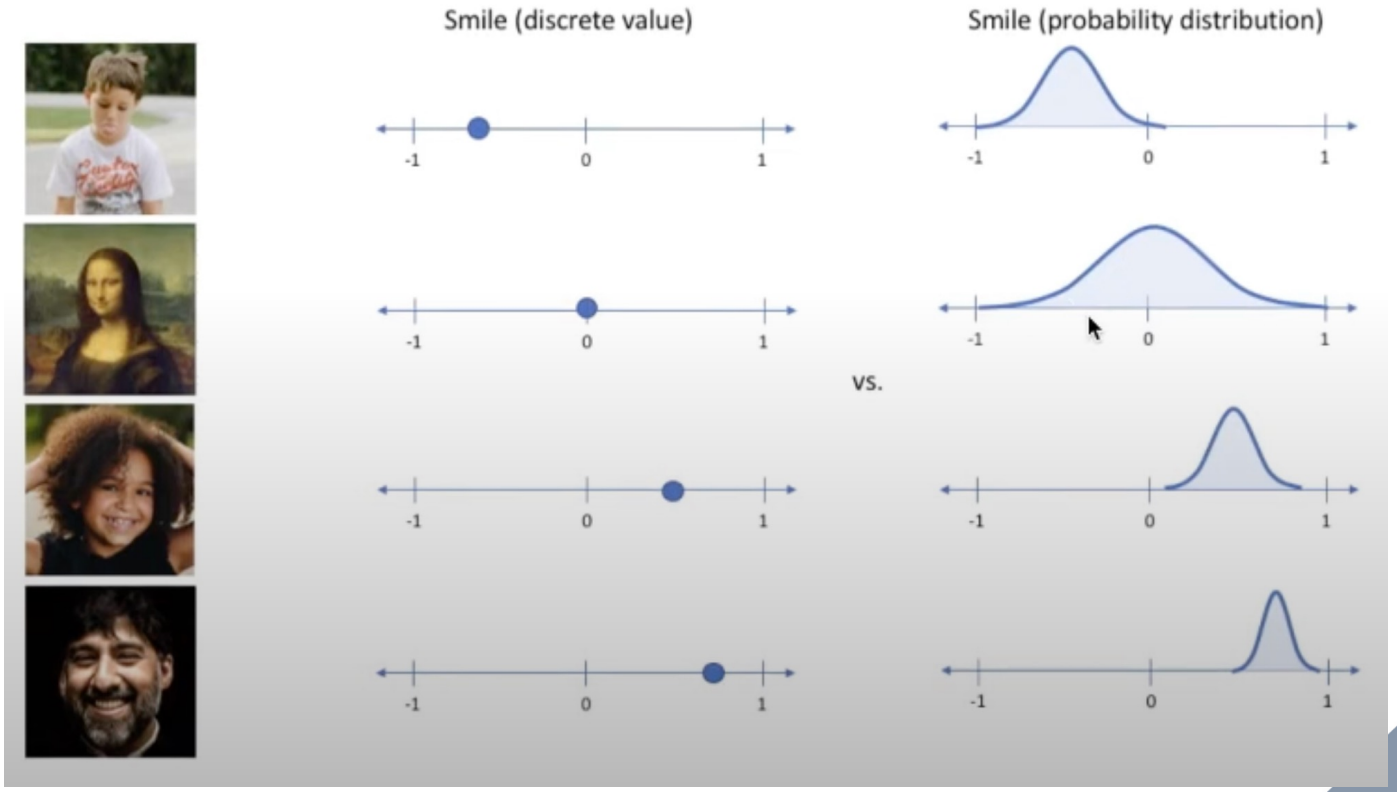
- Consider noisy version $\tilde{x}$ of the input x

$$\min_{W} \frac{1}{2} \sum_{n} \left\| g\left(f\left(\tilde{\boldsymbol{x}}_n; \boldsymbol{W}_f\right); \boldsymbol{W}_g\right) - \boldsymbol{x}_n \right\|_2^2 + c \left\| f\left(\tilde{\boldsymbol{x}}_n; \boldsymbol{W}_f\right) \right\|_1$$

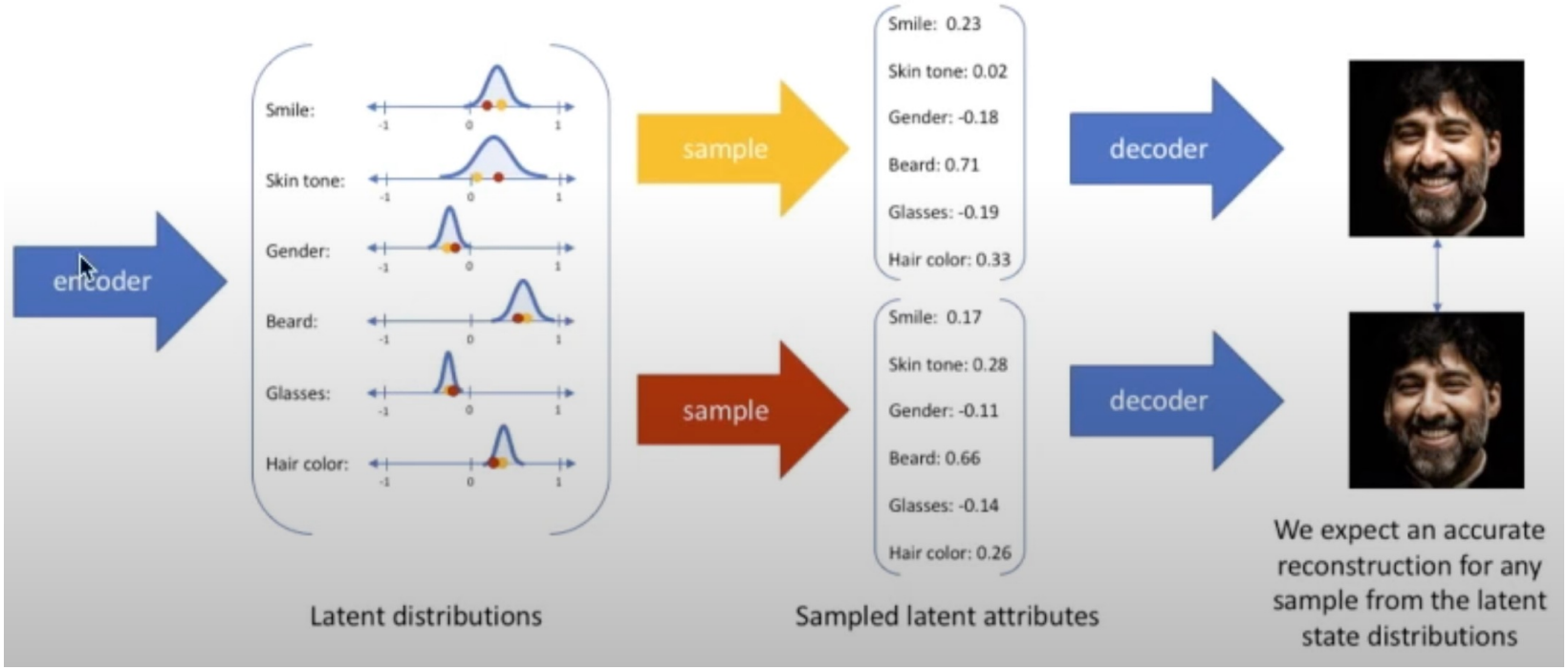


Variational/Probabilistic autoencoder

- Instead of a single value for each attribute, represent each attribute as a range of values
- VAE: Describe latent attribute in probabilistic terms



Variational/Probabilistic autoencoder



Variational/Probabilistic autoencoder

- Let f() and g() represent conditional distributions
  - f: $Pr(h|x; w_f)$
  - g: $Pr(x|h; w_g)$
- The decoder g() can be treated as a generative model
  - First sample h from Pr(h)
  - Then sample x from $Pr(x|h; w_g)$

Variational autoencoder

- Idea: train encoder $\Pr(h|x; w_f)$ to approach a simple and fixed distribution, e.g., $N(h; 0, I)$

$$\max_{W} \sum_{n} \log \Pr(\boldsymbol{x_n}; \boldsymbol{W_f}, \boldsymbol{W_g}) - \underbrace{c\, KL(\Pr(\boldsymbol{h}|\boldsymbol{x}_n; \boldsymbol{W}_f) \,||N(\boldsymbol{h}; \boldsymbol{0}, \boldsymbol{I}))}$$

Kullback-Leibler divergence
Distance measure for distributions

Variational Autoencoder Likelihood

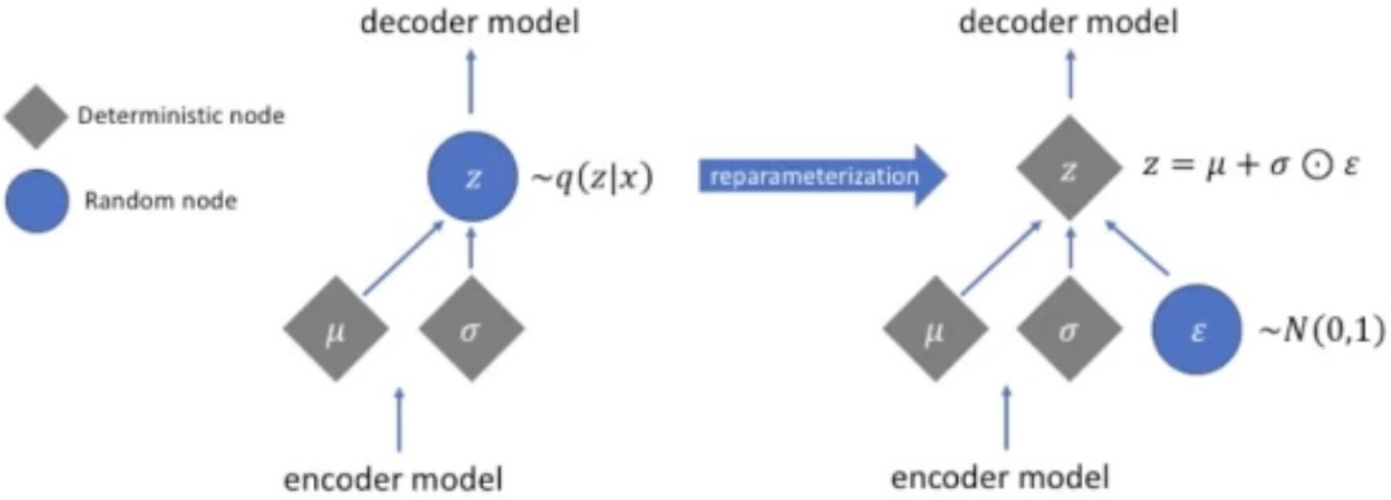
- How to compute $\Pr(x_n; W_f, W_g)$?

$$\Pr(\boldsymbol{x}_n; \boldsymbol{W}_f, \boldsymbol{W}_g) = \int_{\boldsymbol{h}} \Pr(\boldsymbol{x}_n | \boldsymbol{h}; \boldsymbol{W}_g) \Pr(\boldsymbol{h} | \boldsymbol{x}_n; \boldsymbol{W}_f)\, d\boldsymbol{h}$$

- Obtain mean and variance of Pr(h) by neural network

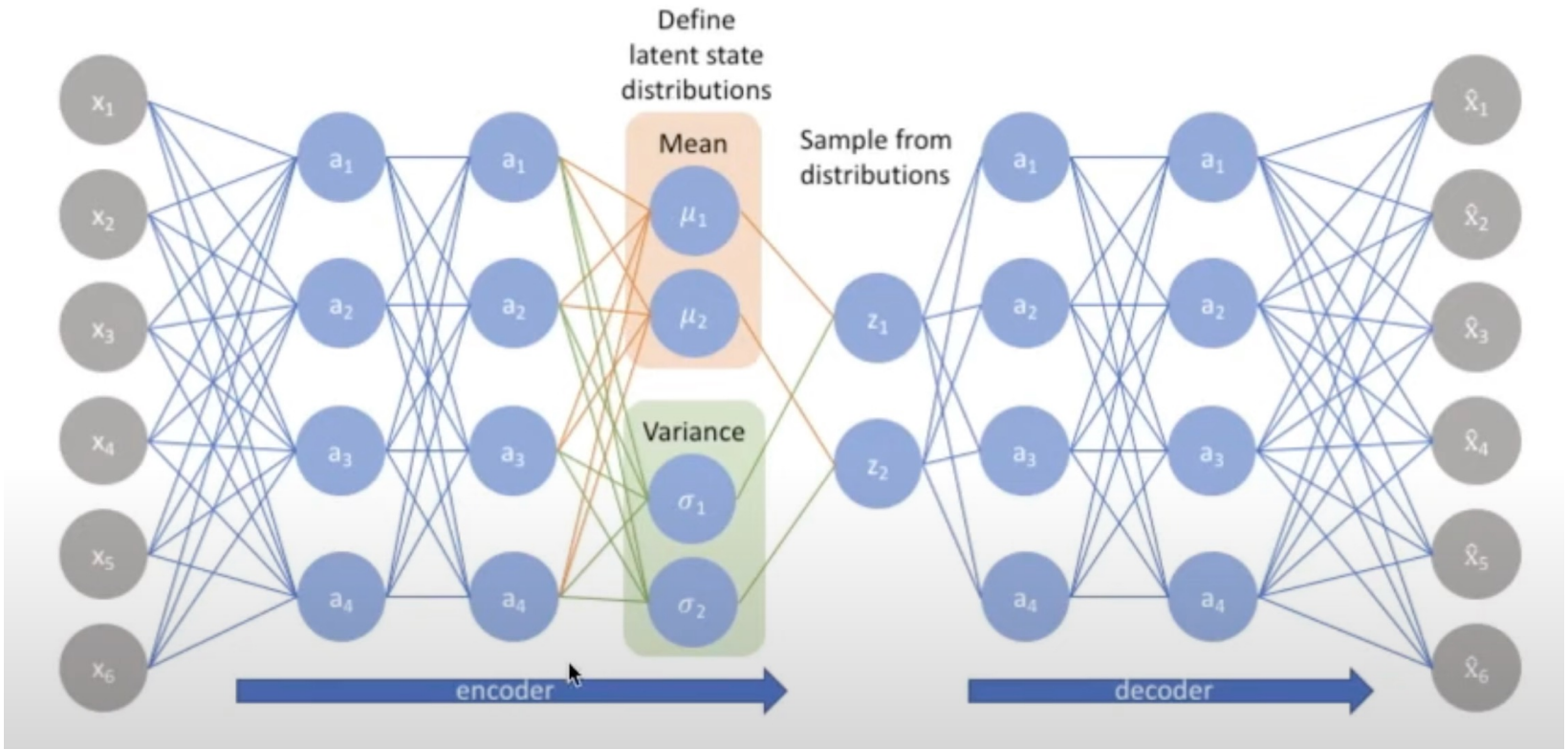
$$\Pr(\boldsymbol{h} | \boldsymbol{x}_n; \boldsymbol{W}_f) = N(\boldsymbol{h}; \mu_n(\boldsymbol{x}_n; \boldsymbol{W}_f), \sigma_n(\boldsymbol{x}_n; \boldsymbol{W}_f)\boldsymbol{I})$$

Reparameterization trick



UCMERCED

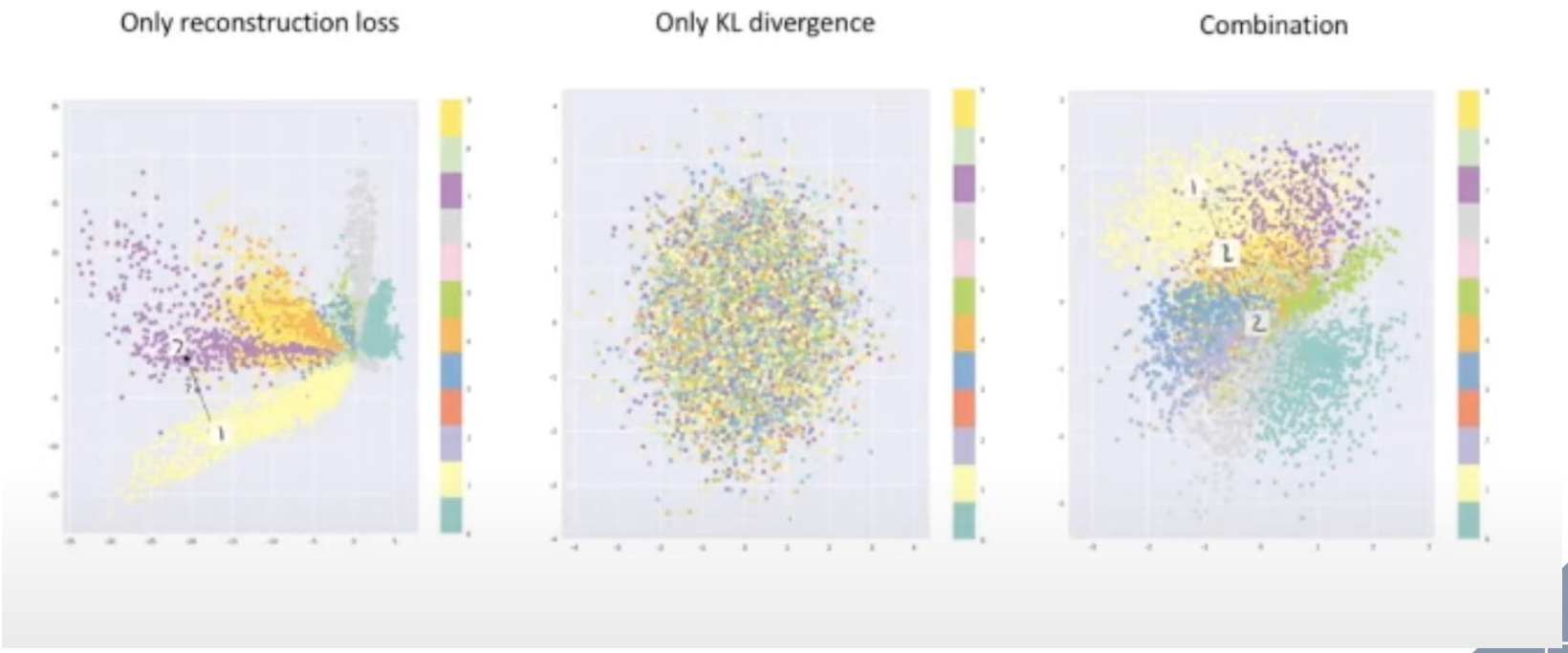
VAE implementation



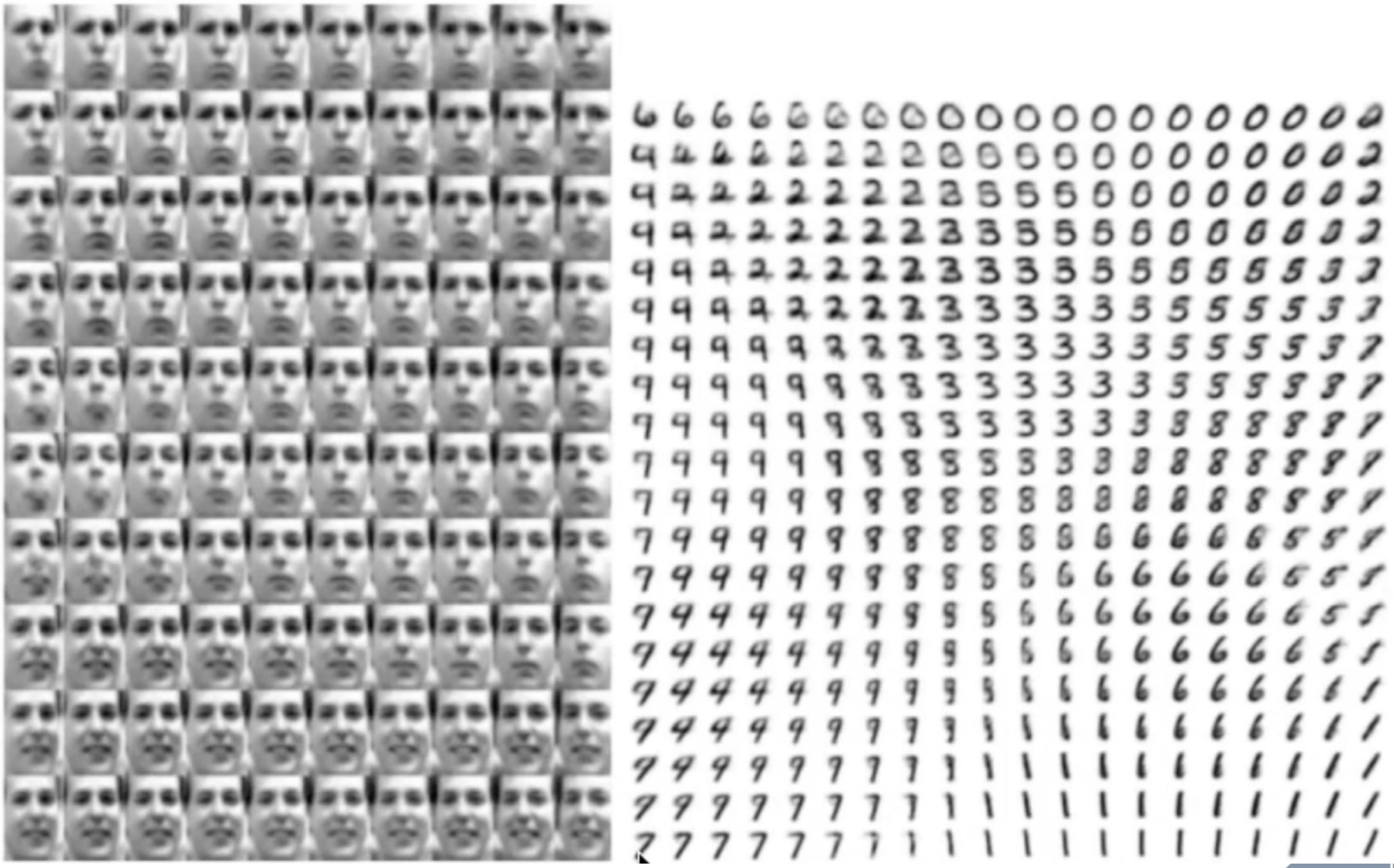
MNIST VAE

$$\max_{\boldsymbol{W}} \sum_{n} \log \Pr\left(\boldsymbol{x_n}; \boldsymbol{W_f}, \boldsymbol{W_g}\right) - \underbrace{c\, KL(\Pr\left(\boldsymbol{h}\middle|\boldsymbol{x}_n; \boldsymbol{W}_f\right) ||N(\boldsymbol{h}; \boldsymbol{0}, \boldsymbol{I}))}$$

Kullback-Leibler divergence

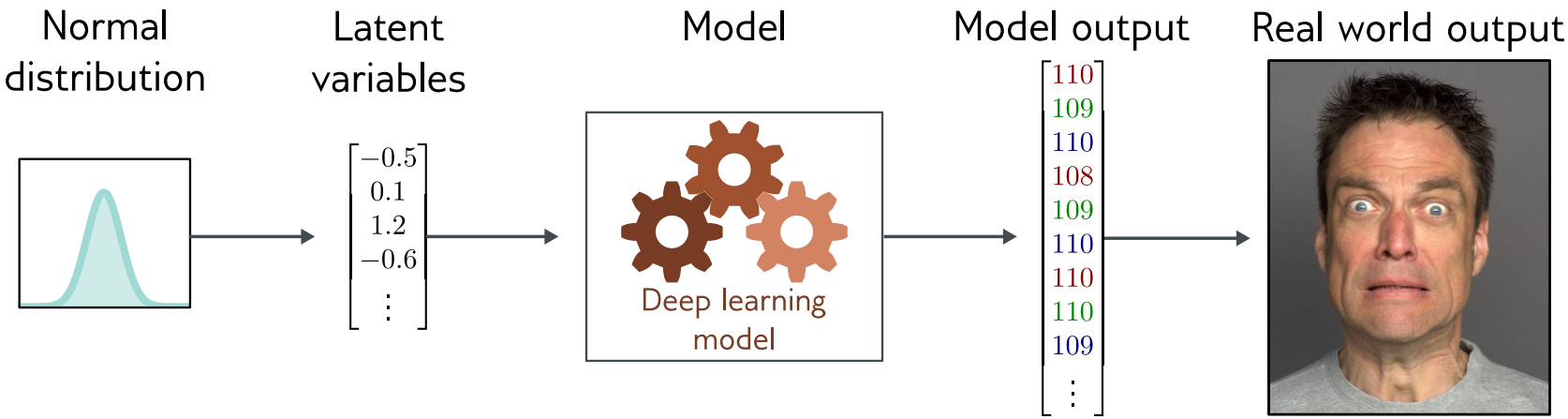


Examples from VAE



Generative Adversarial Network

Latent variable models



Latent variable models map a random "latent" variable to create a new data sample

In GANs, this is called the generator:

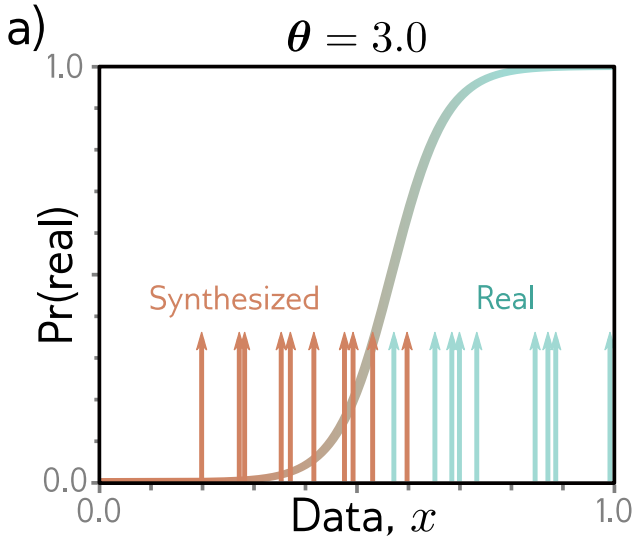
$$\mathbf{x}_* = \mathbf{g}[\mathbf{z}_i, \boldsymbol{\theta}]$$

Generative adversarial models

- GAN loss function
- DCGAN results and problems
- Tricks for improving performance
- Conditional GANs
- Image translation models
- StyleGAN

GAN example

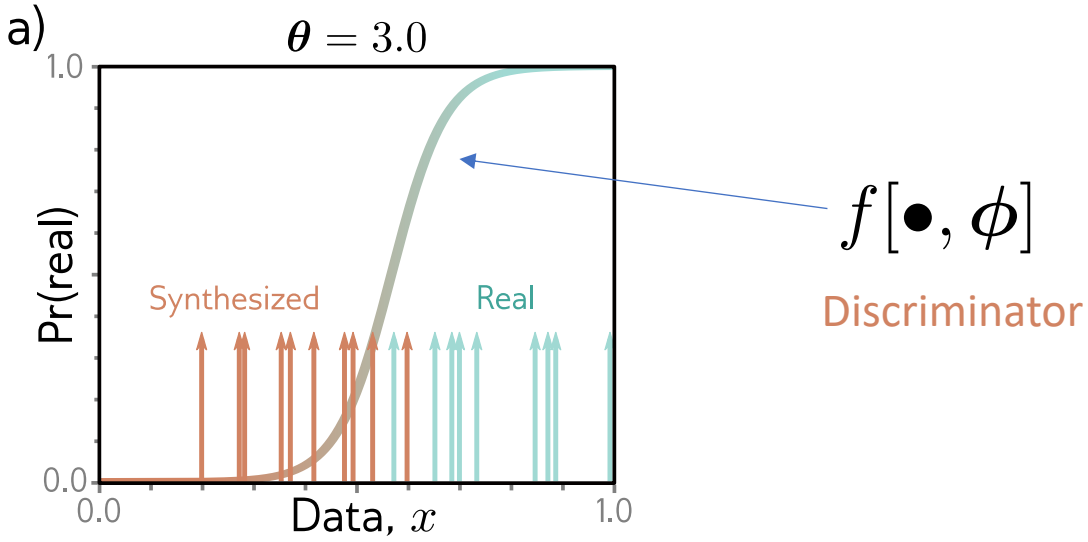
$$x_j^* = \mathrm{g}[z_j, \theta] = z_j + \theta$$



Slide credit: Simon Prince

GAN example

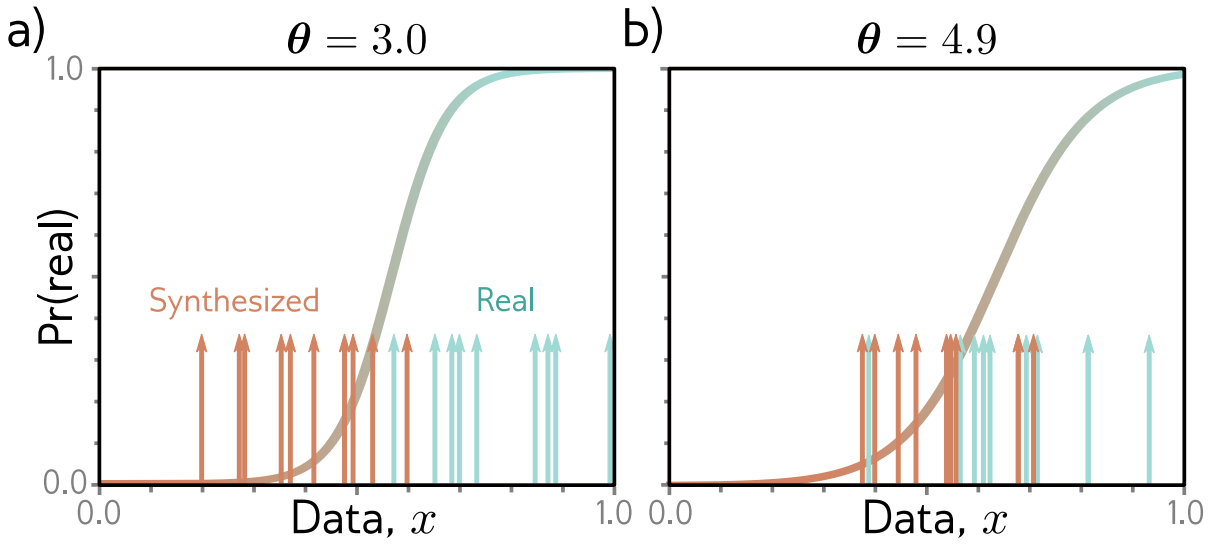
$$x_j^* = \mathrm{g}[z_j, \theta] = z_j + \theta$$



Slide credit: Simon Prince

GAN example

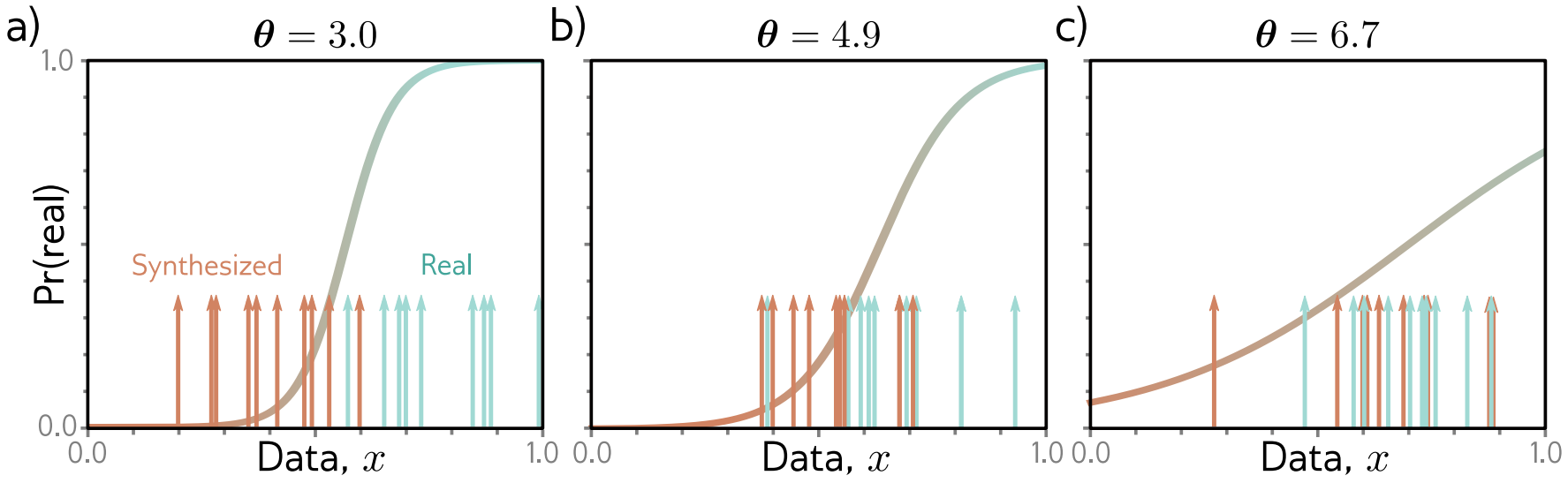
$$x_j^* = \mathrm{g}[z_j, \theta] = z_j + \theta$$



Slide credit: Simon Prince

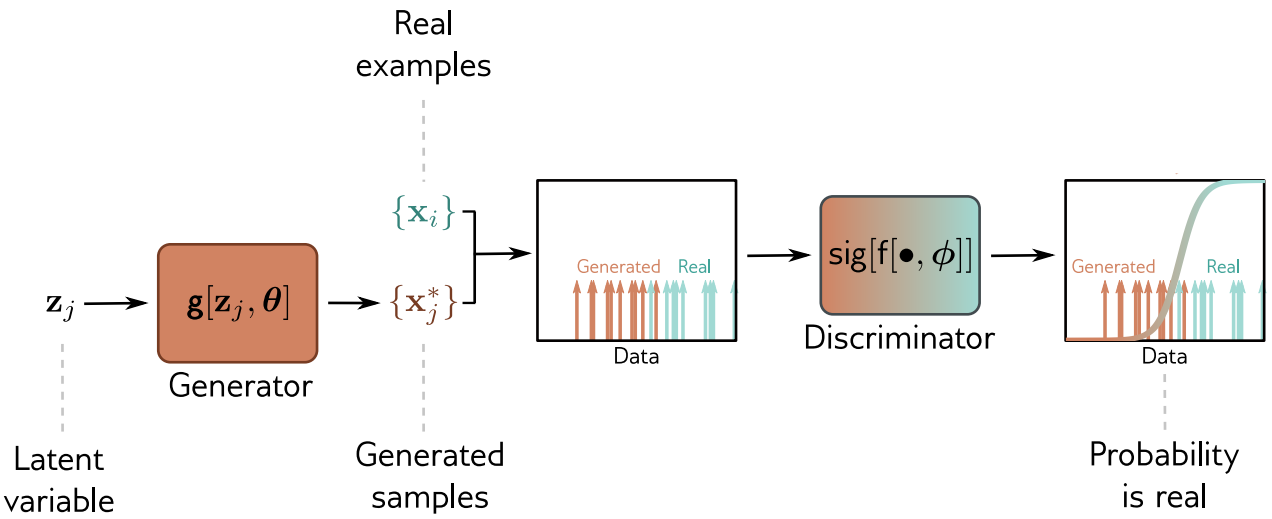
GAN example

$$x_j^* = \mathrm{g}[z_j, \theta] = z_j + \theta$$



Slide credit: Simon Prince

GAN



Slide credit: Simon Prince

GAN cost function

Discriminator uses standard cross entropy loss:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[\sum_i -(1-y_i) \log\Big[1 - \operatorname{sig}[\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\Big] - y_i \log\Big[\operatorname{sig}[\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\Big] \right]$$

Discriminator: generated samples, y = 0, real examples, y = 1:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[\sum_j -\log\Big[1 - \operatorname{sig}[\mathrm{f}[\mathbf{x}^*_j, \boldsymbol{\phi}]]\Big] - \sum_i \log\Big[\operatorname{sig}[\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\Big] \right]$$

Generator loss: make generated samples more likely under discriminator (i.e. make discriminator loss larger)

$$\hat{\boldsymbol{\phi}}, \hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[\underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left[\sum_j -\log\Big[1 - \operatorname{sig}[\mathrm{f}[\mathbf{g}[\mathbf{z}_j, \boldsymbol{\theta}], \boldsymbol{\phi}]]\Big] - \sum_i \log\Big[\operatorname{sig}[\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\Big] \right] \right]$$

Slide credit: Simon Prince

GAN Cost function

$$\hat{\boldsymbol{\phi}}, \hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[\underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left[\sum_j -\log\Big[1-\operatorname{sig}[\mathrm{f}[\mathbf{g}[\mathbf{z}_j, \boldsymbol{\theta}], \boldsymbol{\phi}]]\Big] - \sum_i \log\Big[\operatorname{sig}[\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\Big] \right] \right]$$

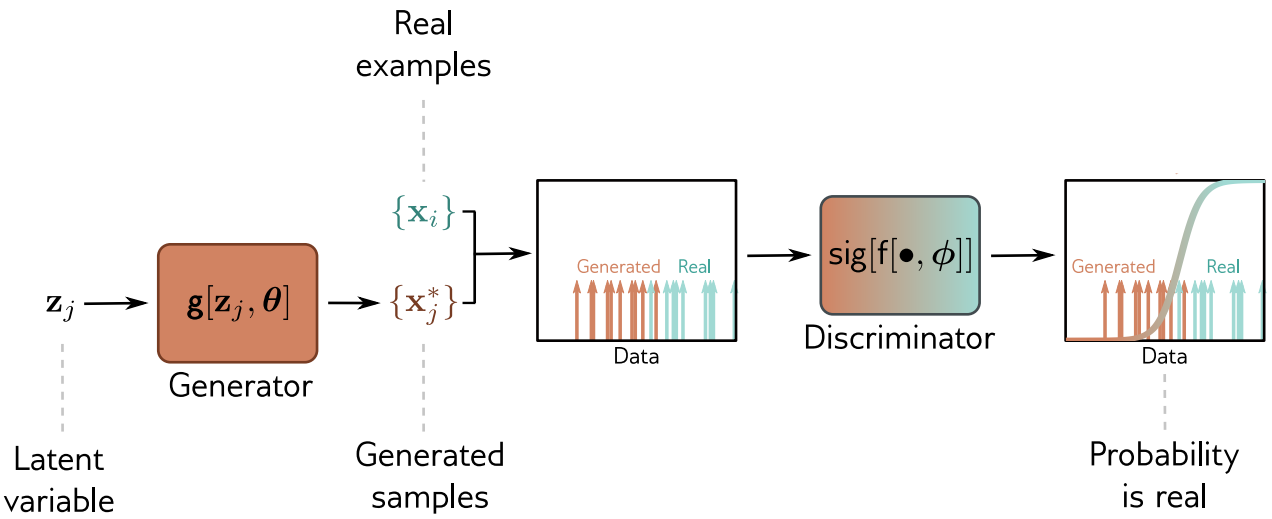
Can divide into two parts:

$$L[\boldsymbol{\phi}] = \sum_j -\log\Big[1 - \operatorname{sig}[\mathrm{f}[\mathbf{g}[\mathbf{z}_j, \boldsymbol{\theta}], \boldsymbol{\phi}]]\Big] - \sum_i \log\Big[\operatorname{sig}[\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\Big]$$

$$L[\boldsymbol{\theta}] = \sum_j \log\Big[1 - \operatorname{sig}[\mathrm{f}[\mathbf{g}[\mathbf{z}_j, \boldsymbol{\theta}], \boldsymbol{\phi}]]\Big]$$

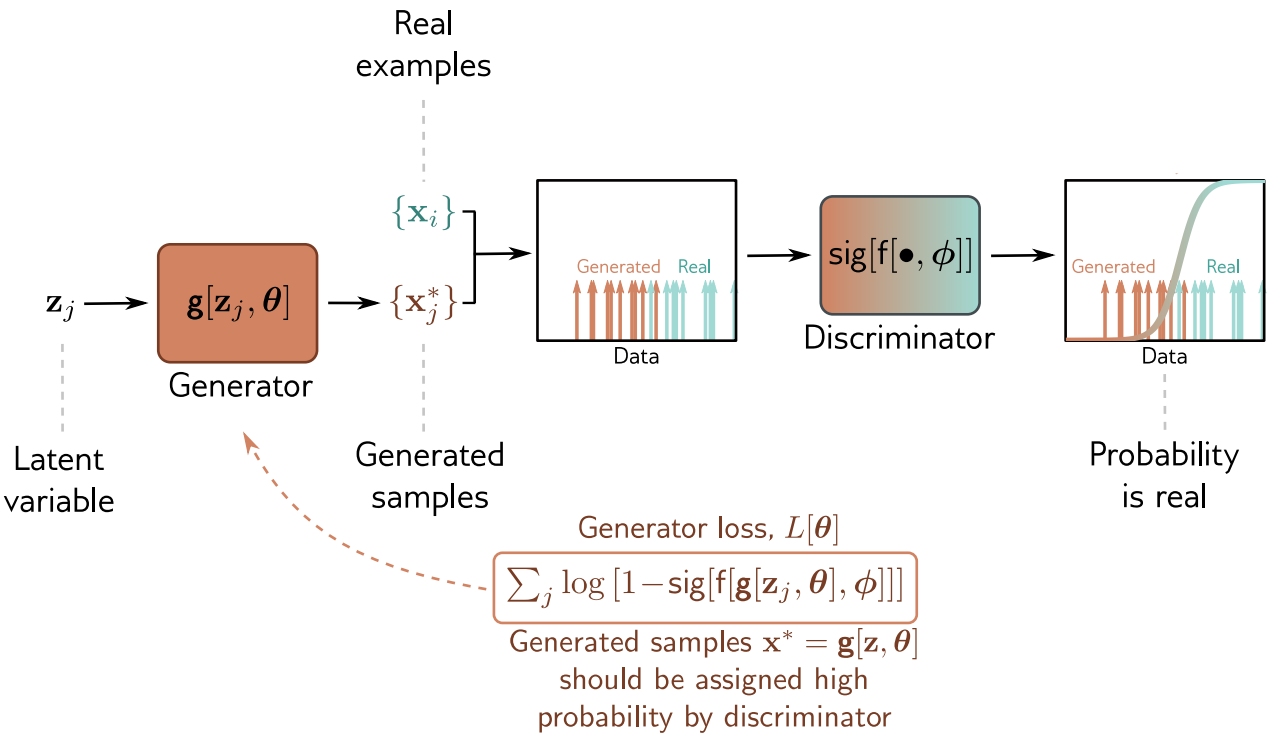
Slide credit: Simon Prince

GAN loss function



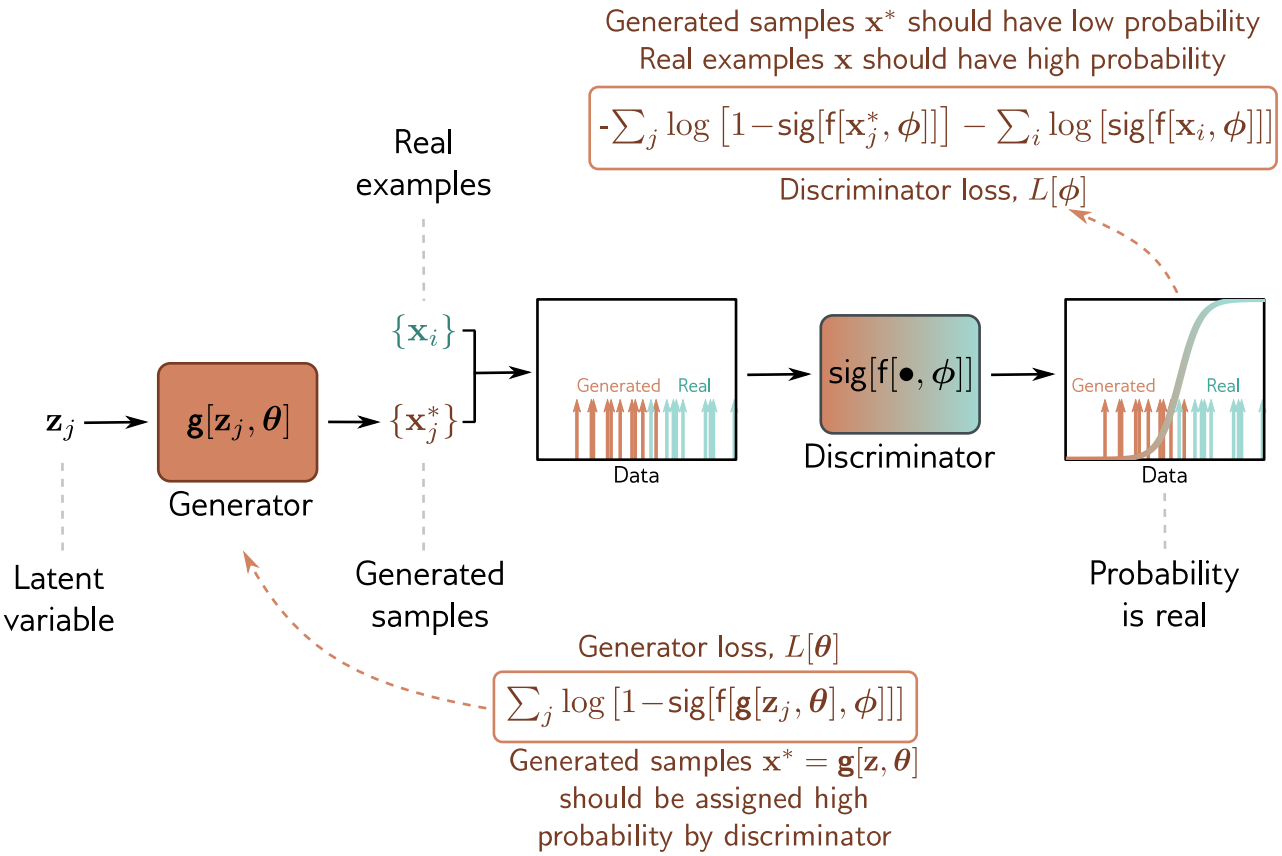
Slide credit: Simon Prince

GAN loss function



Slide credit: Simon Prince

GAN loss function



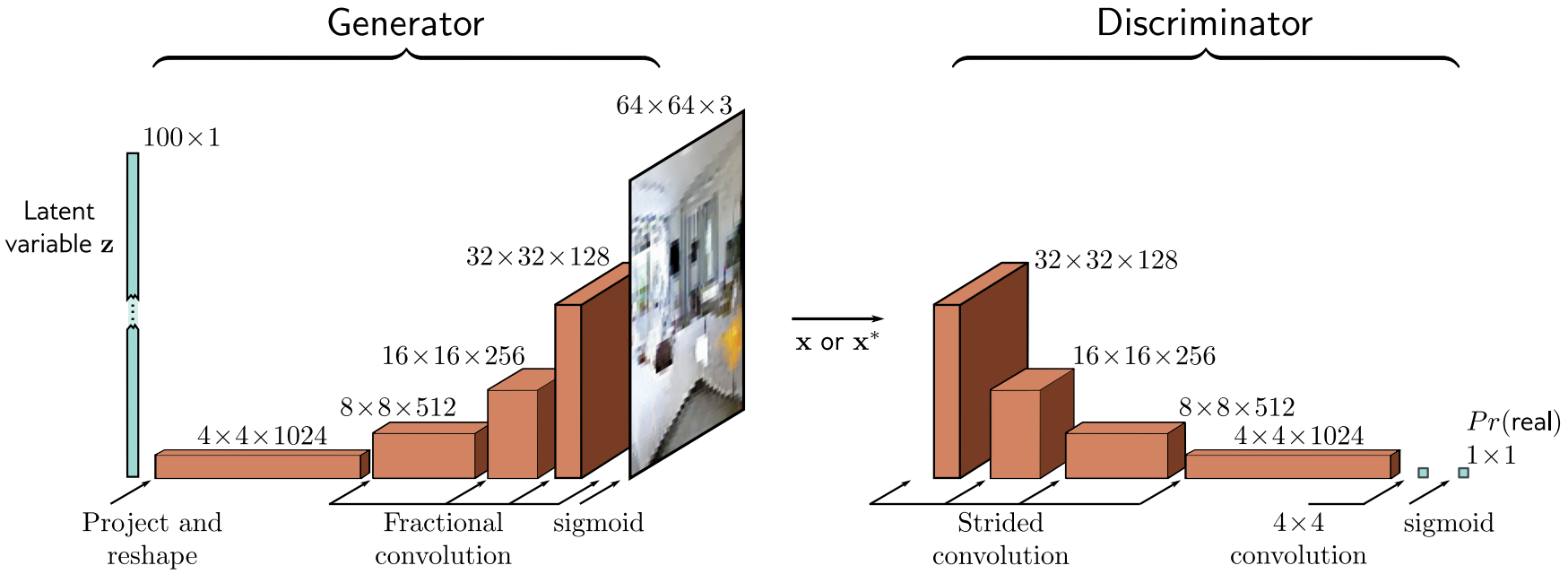
Slide credit: Simon Prince

Generative adversarial models

- GAN loss function
- DCGAN results and problems
- Tricks for improving performance
- Conditional GANs
- Image translation models

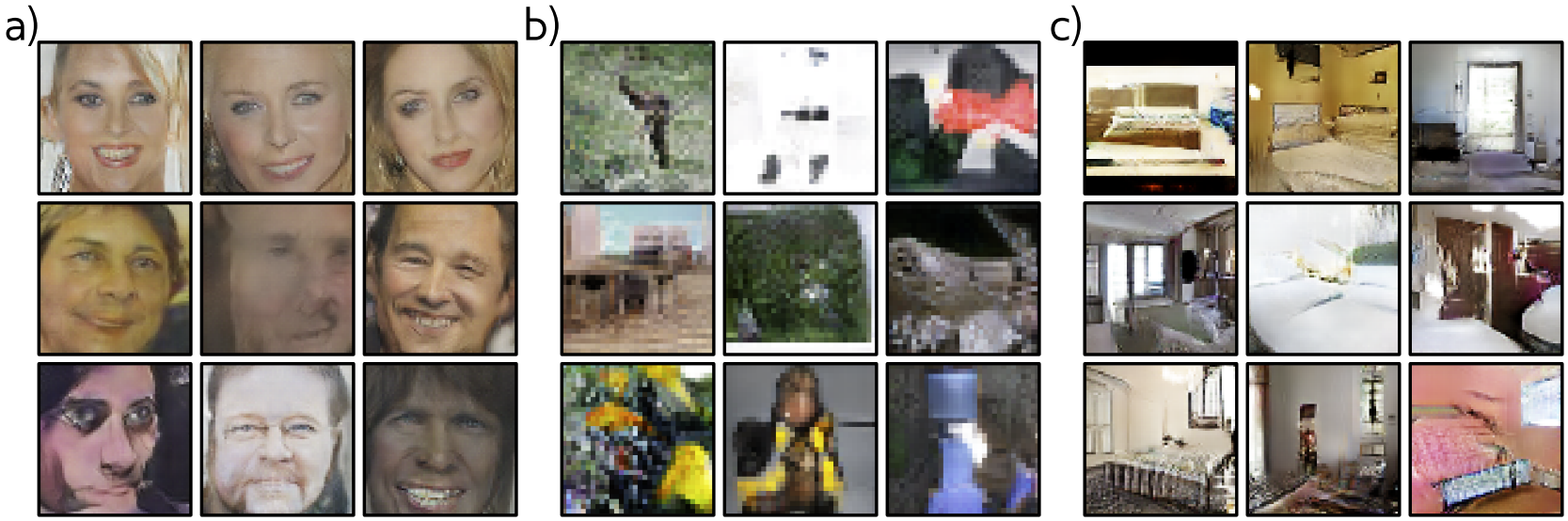
Slide credit: Simon Prince

Deep Convolutional (DC) GAN



Slide credit: Simon Prince

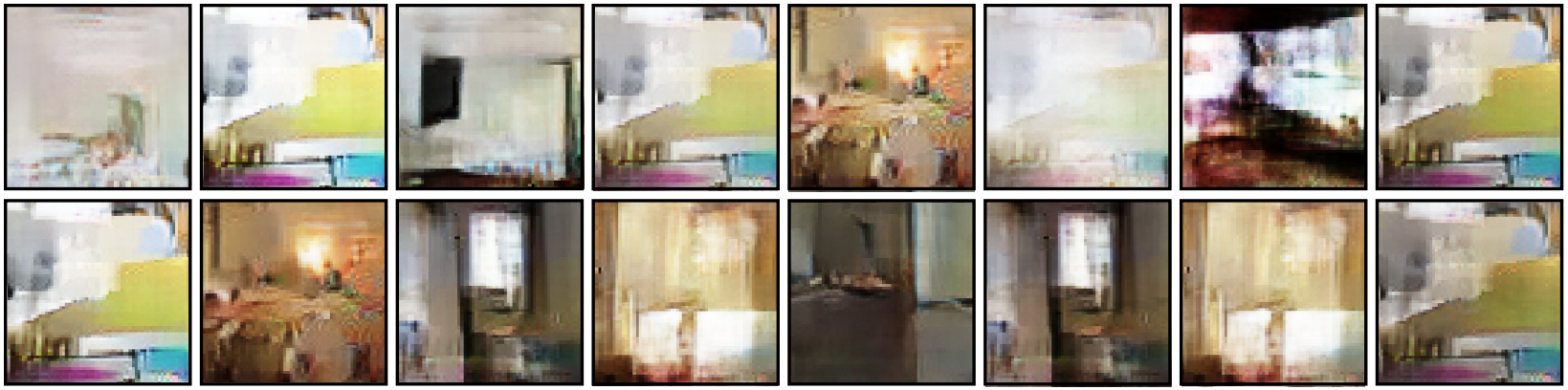
DC GAN Results



Slide credit: Simon Prince

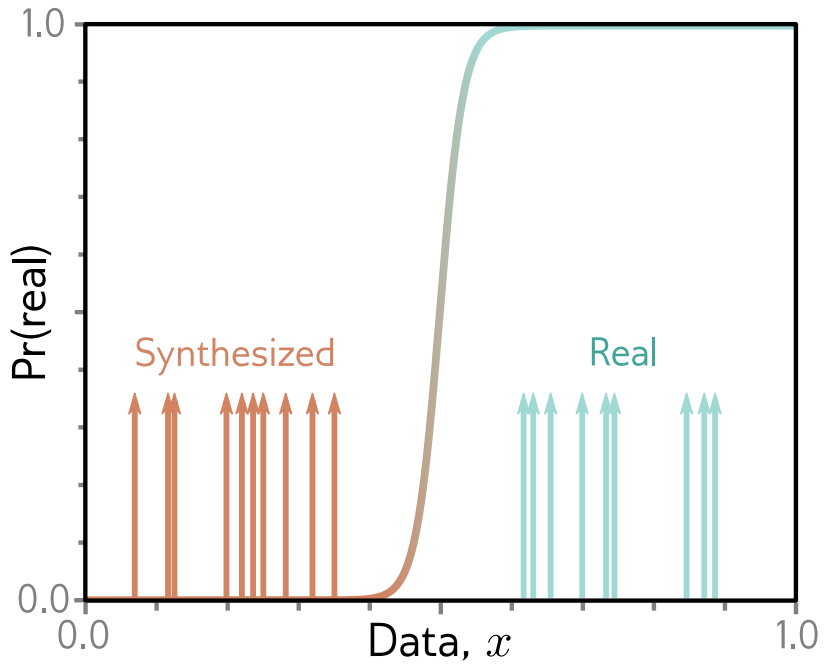
Mode collapse

Training GANs is difficult. Often fails.



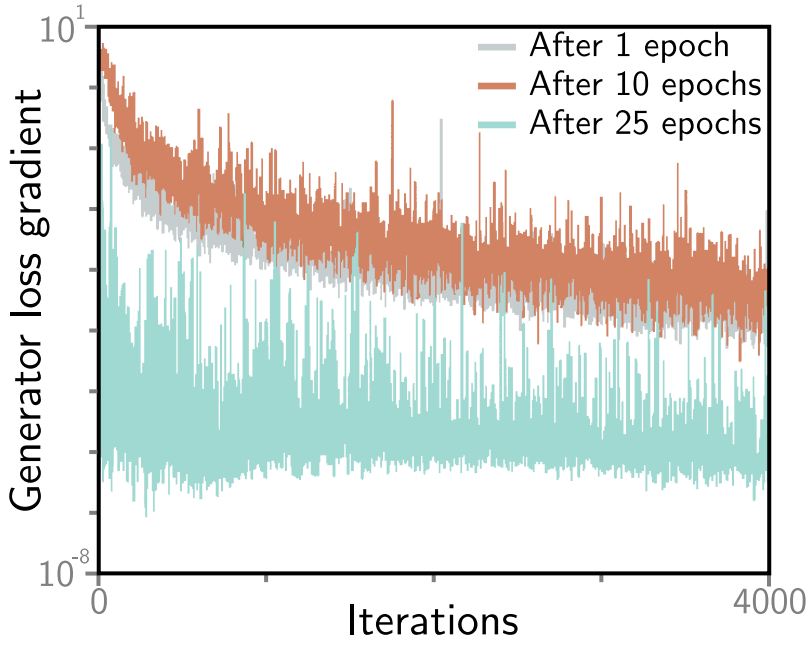
Slide credit: Simon Prince

GAN training instability



Slide credit: Simon Prince

GAN training instability



Fix generator and continue training discriminator, then generator gradients disappear.

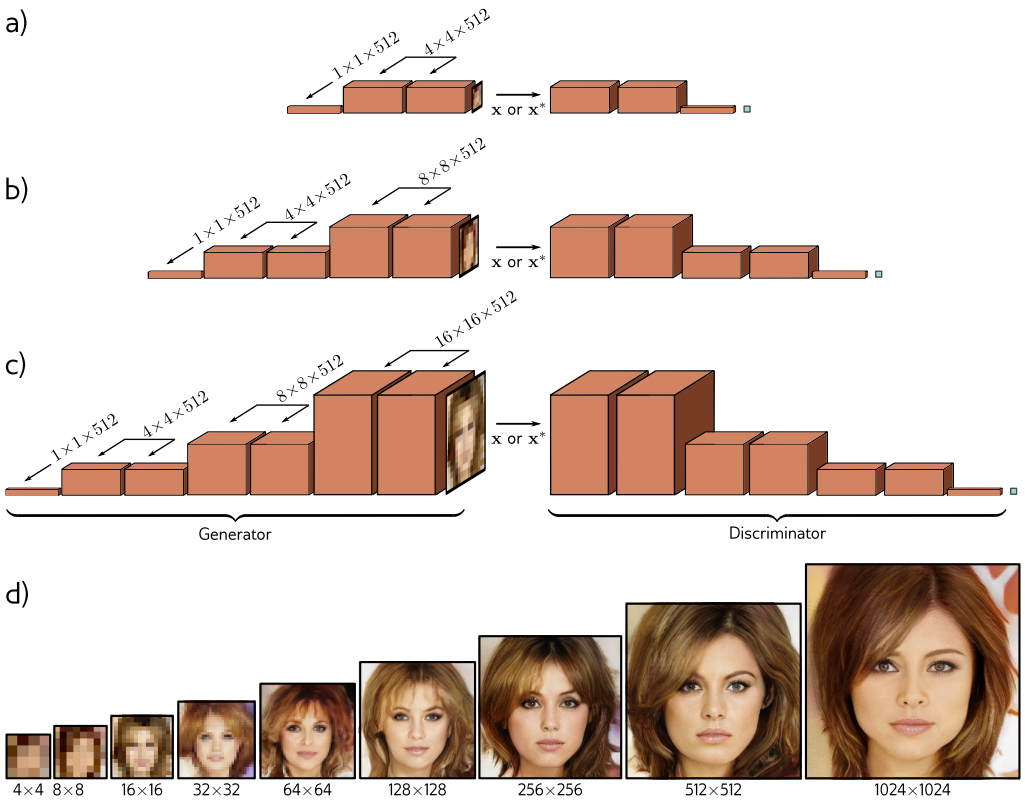
Slide credit: Simon Prince

Generative adversarial models

- GAN loss function
- DCGAN results and problems
- Tricks for improving performance
- Conditional GANs
- Image translation models

Slide credit: Simon Prince

Trick 1: Progressive growing



Slide credit: Simon Prince

Trick 2: Truncation

Only choose random values of latent variables that are less than a threshold τ.

Slide credit: Simon Prince

Example results

Slide credit: Simon Prince

Interpolation

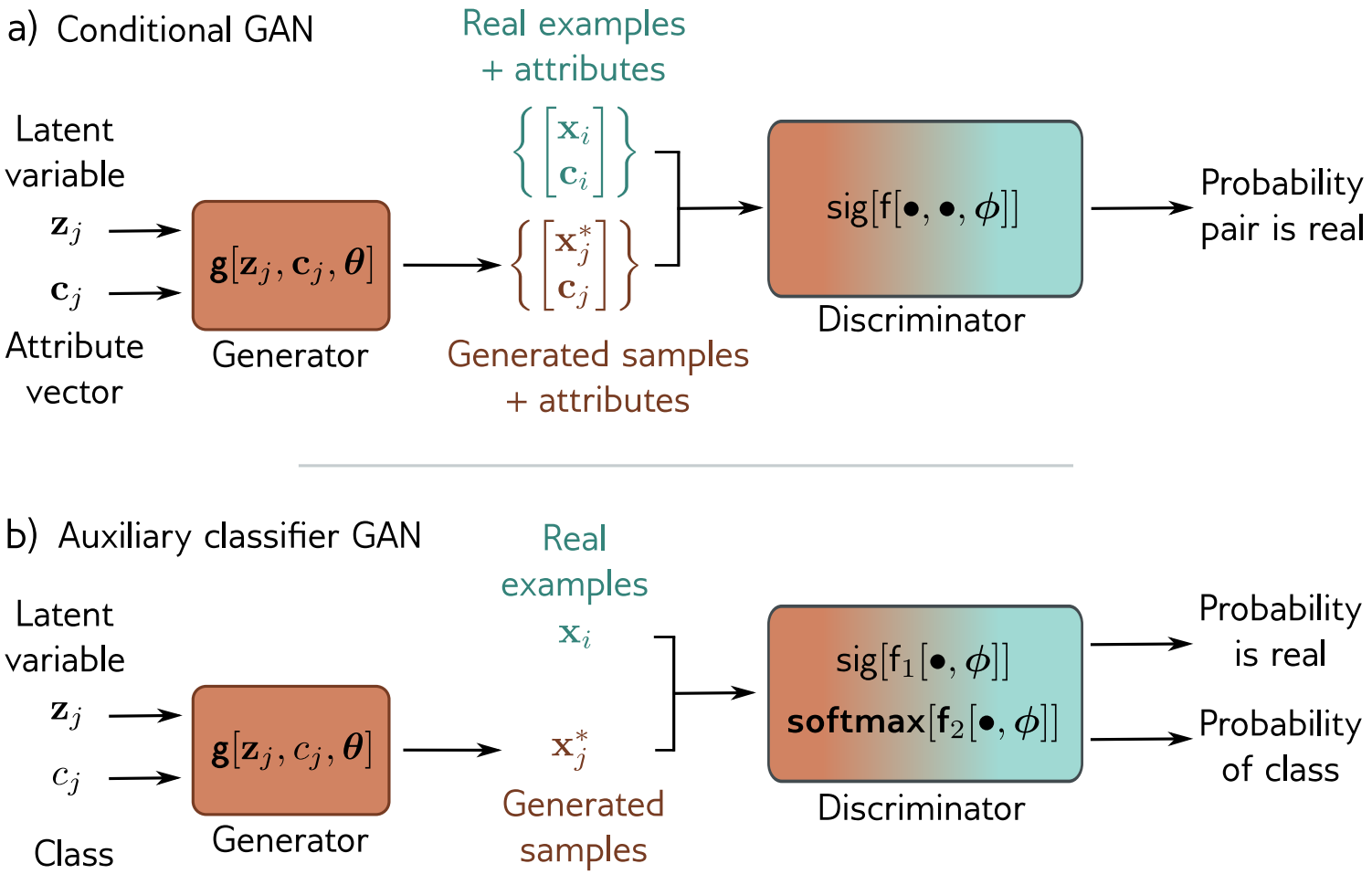
Slide credit: Simon Prince

Generative adversarial models

- GAN loss function
- DCGAN results and problems
- Tricks for improving performance
- Conditional GANs
- Image translation models

Slide credit: Simon Prince

Conditional GAN models



Slide credit: Simon Prince

ACGAN results

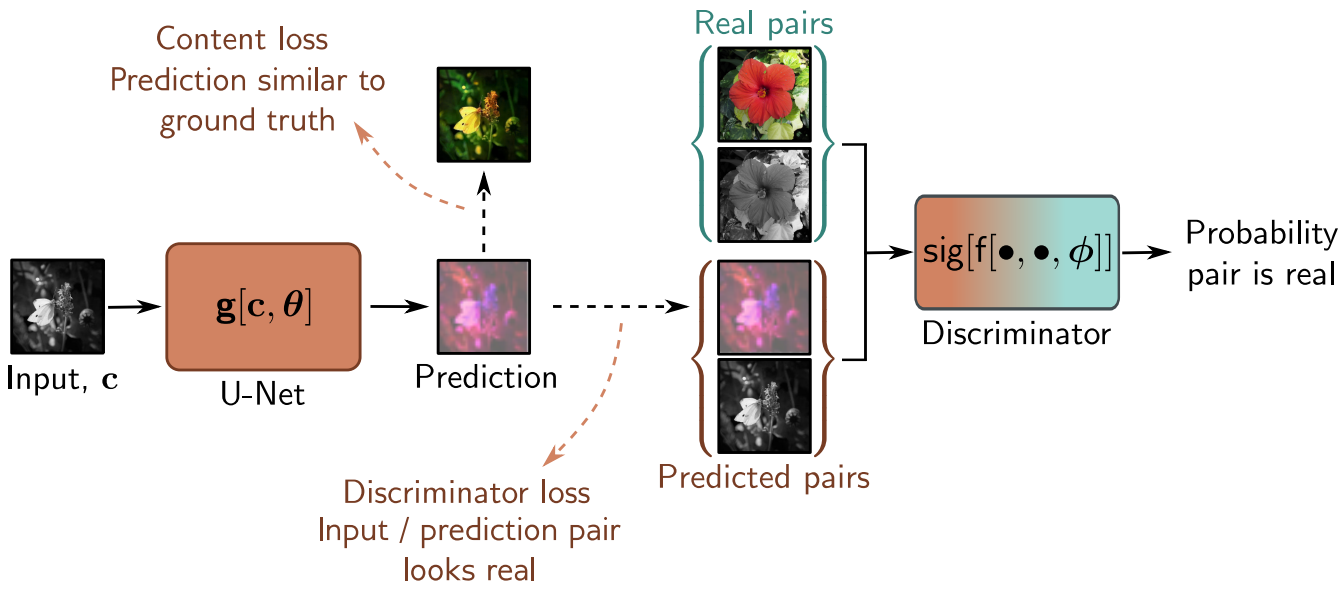
Slide credit: Simon Prince

Generative adversarial models

- GAN loss function
- DCGAN results and problems
- Tricks for improving performance
- Conditional GANs
- Image translation models

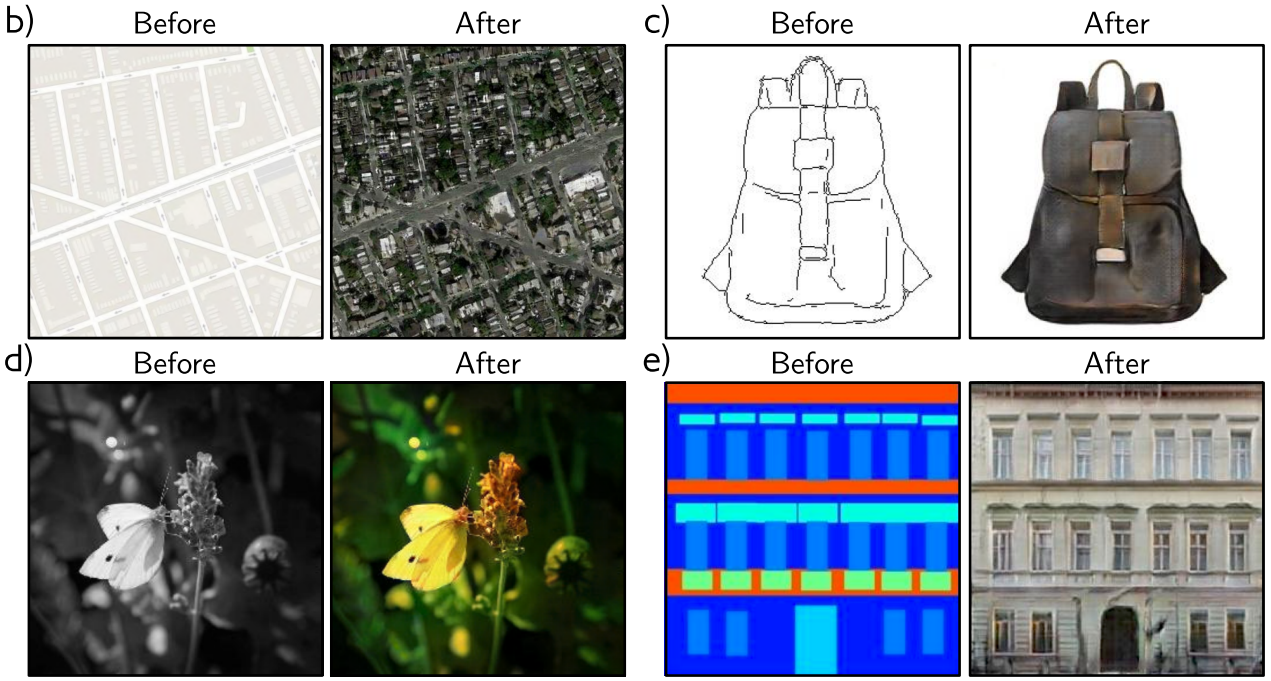
Slide credit: Simon Prince

Image translation: Pix2Pix



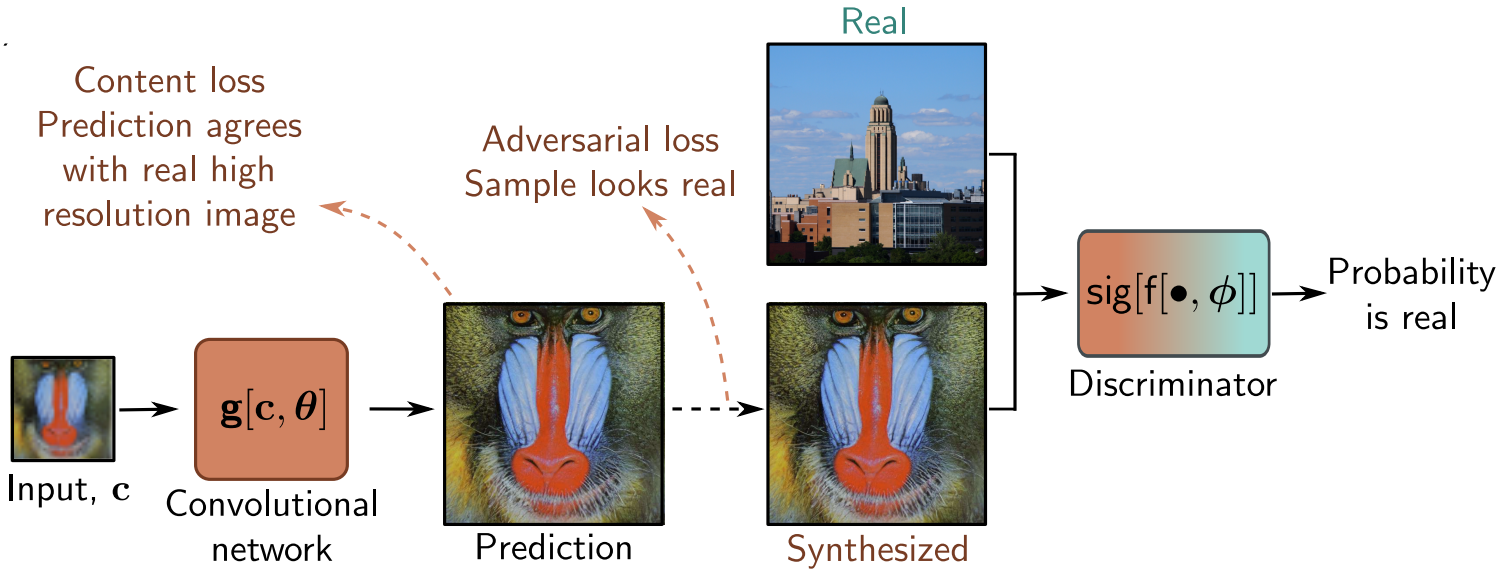
Slide credit: Simon Prince

Image translation: Pix2Pix



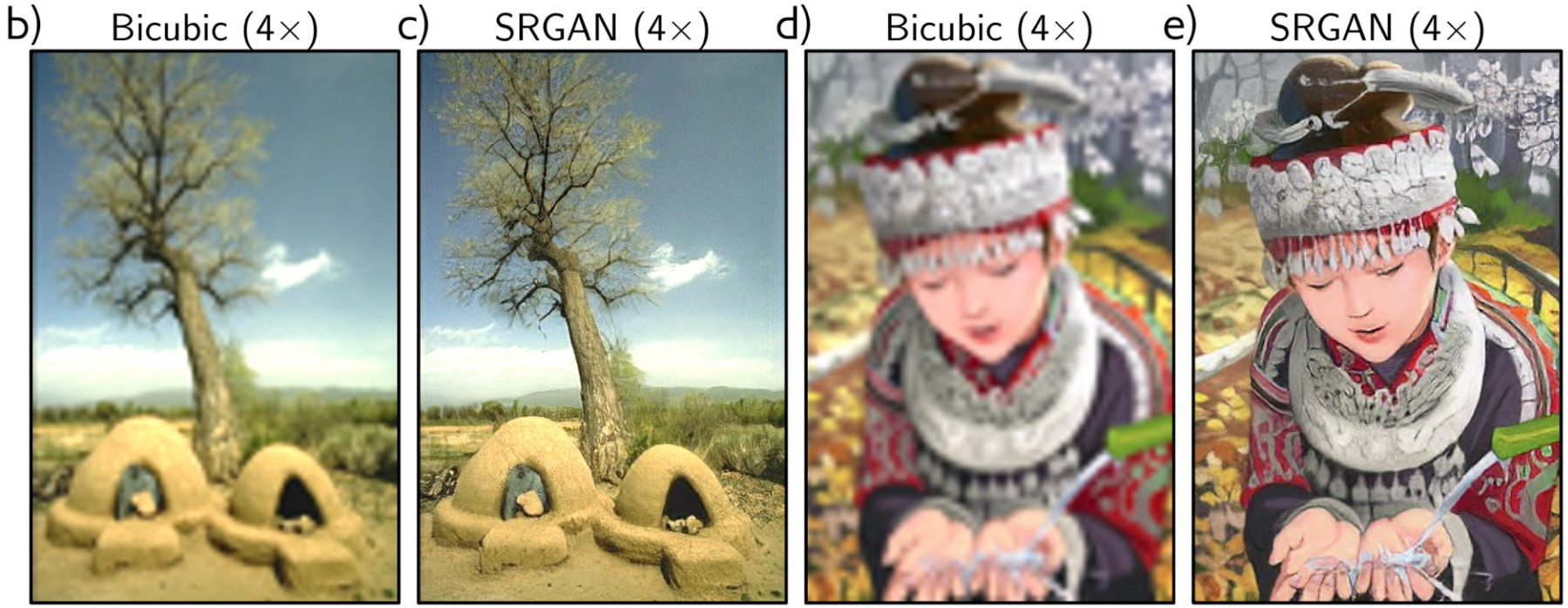
Slide credit: Simon Prince

Image translation: SRGAN



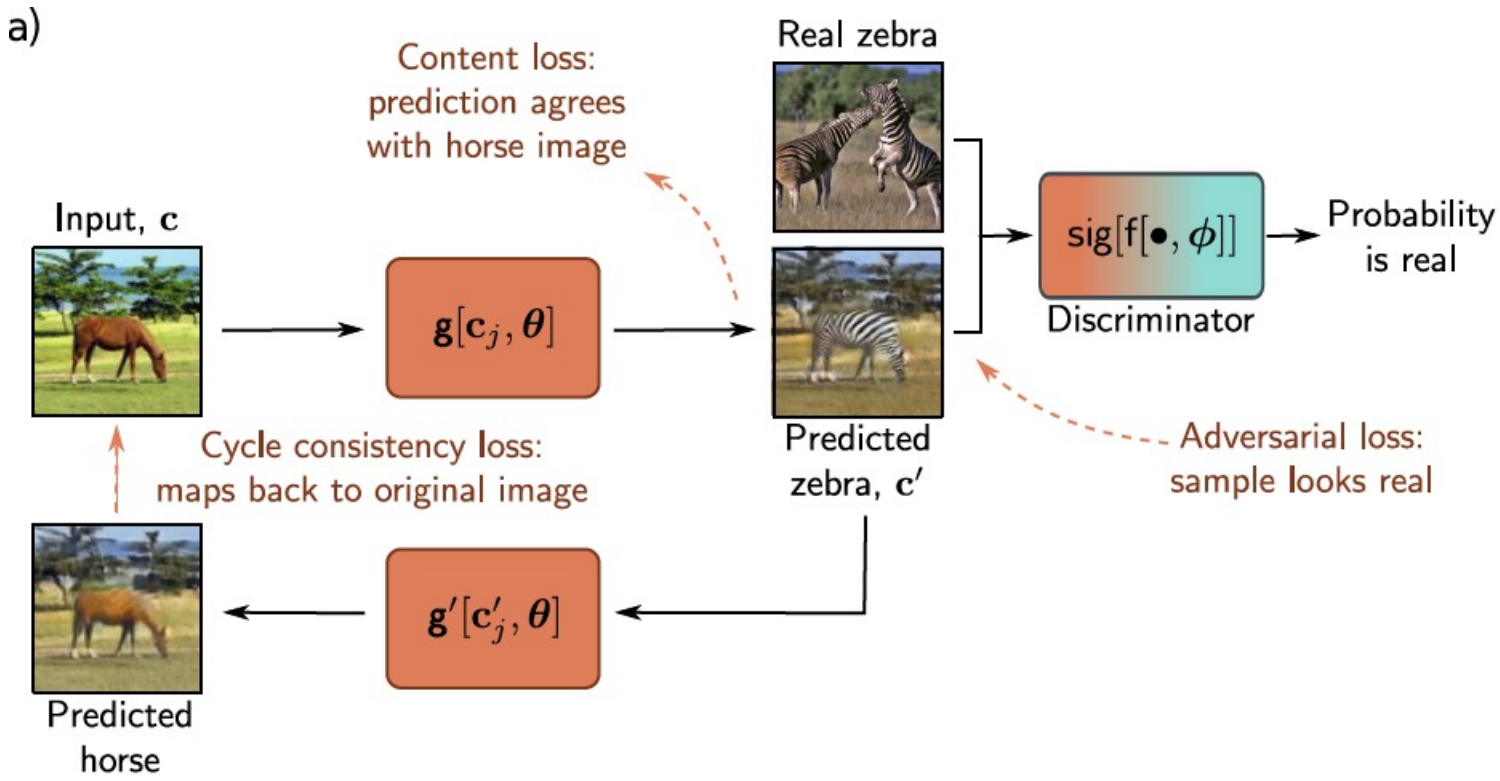
Slide credit: Simon Prince

Image translation: SRGAN



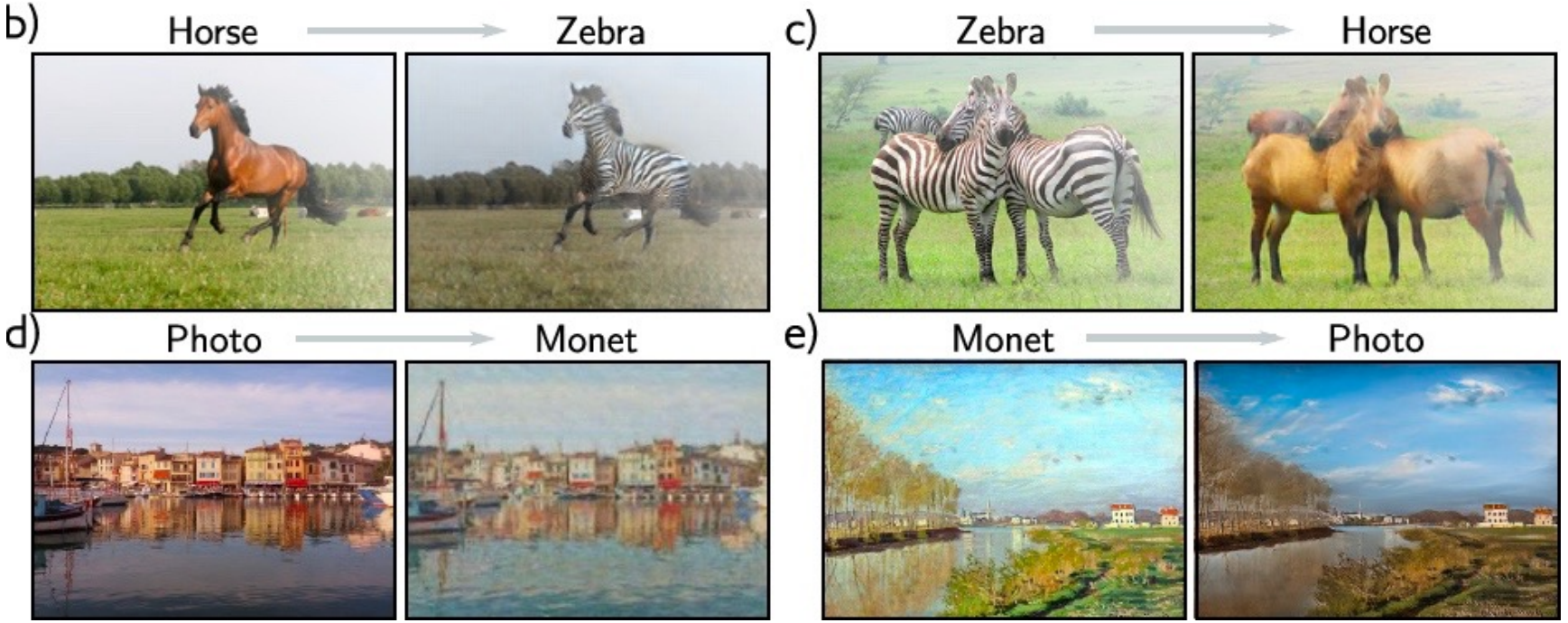
Slide credit: Simon Prince

Image translation: CycleGAN



Slide credit: Simon Prince

Image translation: CycleGAN



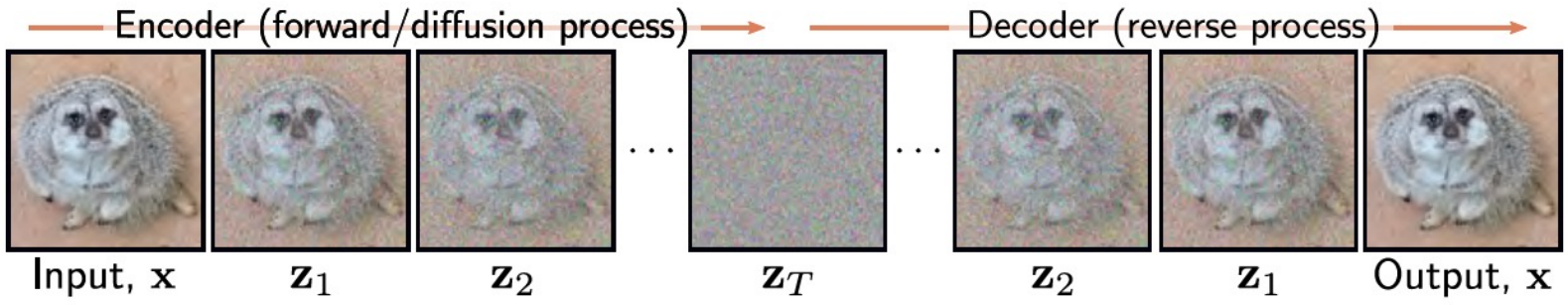
Slide credit: Simon Prince

Diffusion Model

Video Generation – OpenAI Sora

Denoising Diffusion models

- Forward process/diffusion gradually adds noise to data
- Backward process generates data by denoising



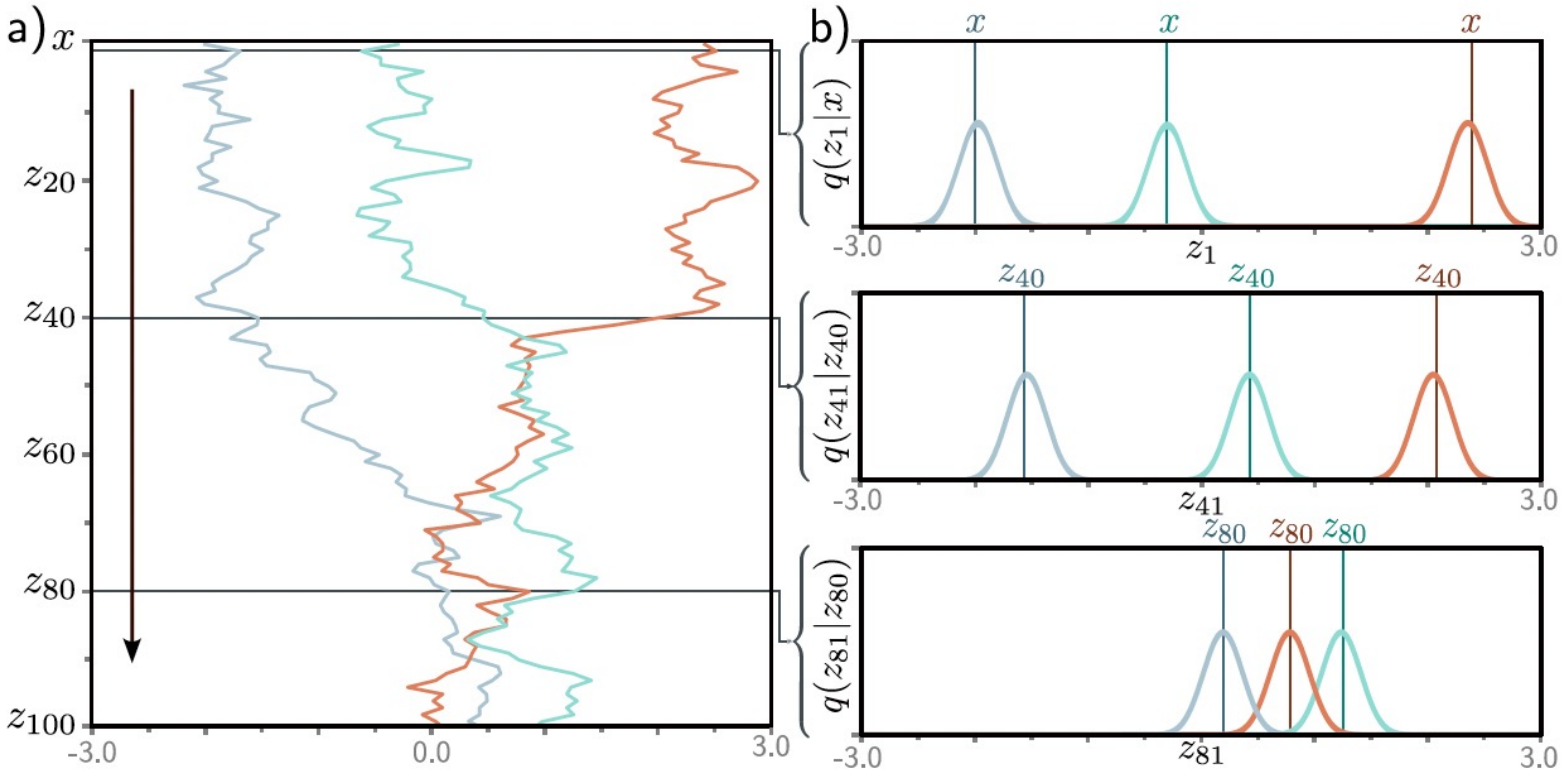
Forward process/diffusion

$$\begin{aligned}\mathbf{z}_1 &= \sqrt{1-\beta_1}\cdot\mathbf{x} + \sqrt{\beta_1}\cdot\boldsymbol{\epsilon}_1 \\ \mathbf{z}_t &= \sqrt{1-\beta_t}\cdot\mathbf{z}_{t-1} + \sqrt{\beta_t}\cdot\boldsymbol{\epsilon}_t \qquad \forall\, t \in 2,\ldots,T,\end{aligned}$$

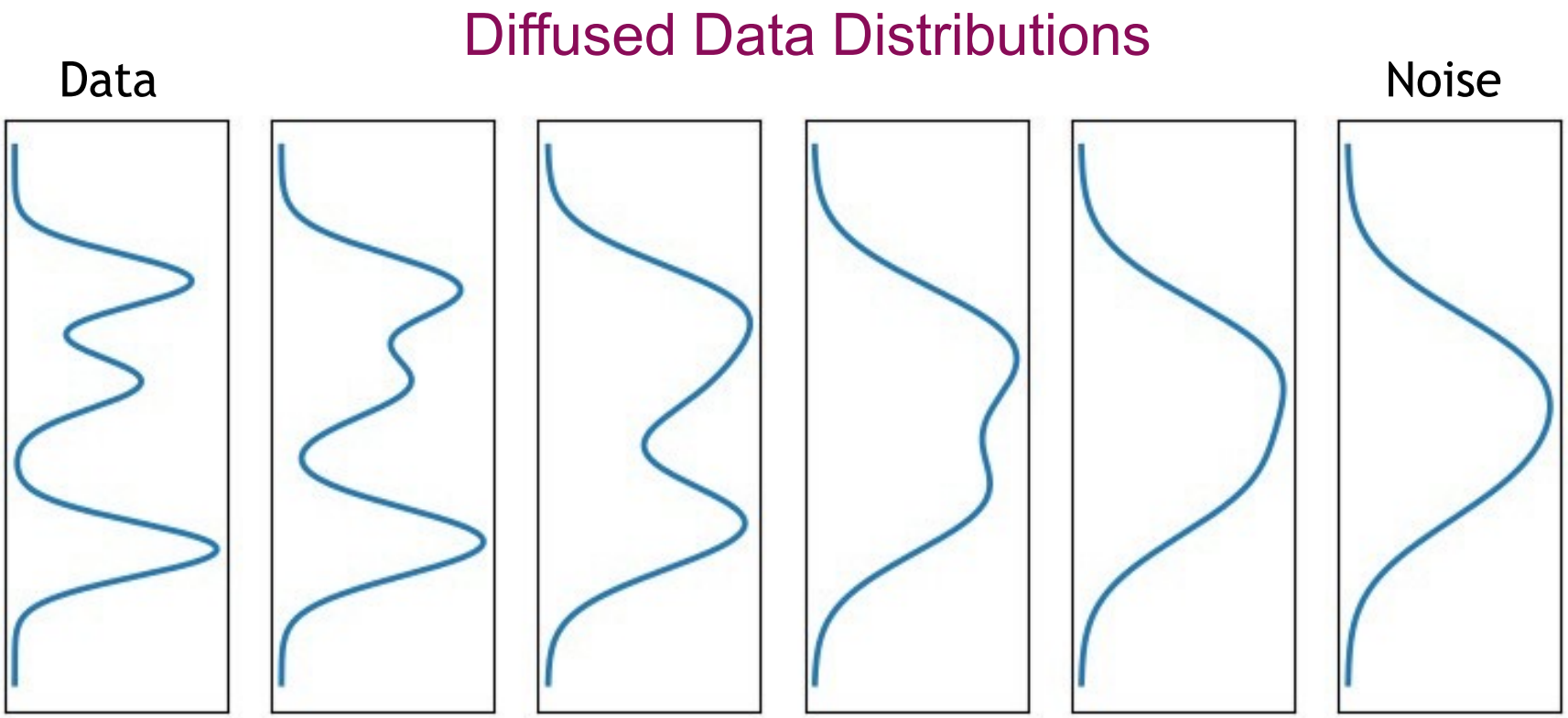
$$\begin{aligned}q(\mathbf{z}_1|\mathbf{x}) &= \text{Norm}_{\mathbf{z}_1}\left[\sqrt{1-\beta_1}\mathbf{x}, \beta_1\mathbf{I}\right] \\ q(\mathbf{z}_t|\mathbf{z}_{t-1}) &= \text{Norm}_{\mathbf{z}_t}\left[\sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I}\right] \qquad \forall\, t \in \{2,\ldots,T\}.\end{aligned}$$

$$q(\mathbf{z}_{1\ldots T}|\mathbf{x}) = q(\mathbf{z}_1|\mathbf{x})\prod_{t=2}^{T} q(\mathbf{z}_t|\mathbf{z}_{t-1}).$$

Forward process



What happens to data distribution in forward process?



Diffusion kernel

- Can we directly compute z_t without computing previous z?

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

where $\alpha_t = \prod_{s=1}^{t} 1 - \beta_s$. We can equivalently write this in probabilistic form:

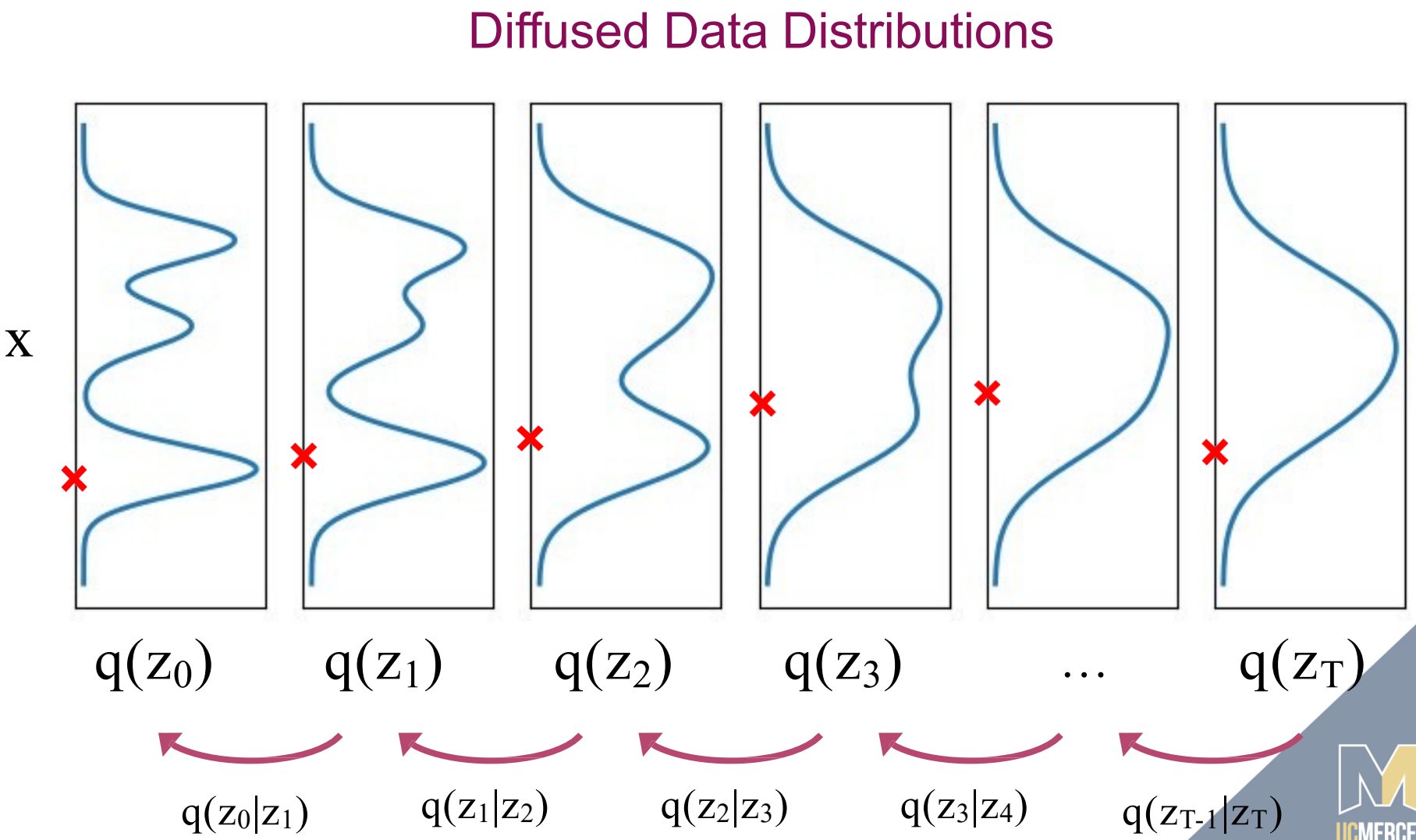
$$q(\mathbf{z}_t|\mathbf{x}) = \text{Norm}_{\mathbf{z}_t}\left[\sqrt{\alpha_t} \cdot \mathbf{x}, (1-\alpha_t)\mathbf{I}\right].$$

- Marginal distribution

$$q(\mathbf{z}_t) = \int q(\mathbf{z}_t|\mathbf{x}) Pr(\mathbf{x}) d\mathbf{x}.$$

- Quiz: What is the distribution of $\mathbf{z}_T$?

Generative learning by denoising



Conditional distribution in reverse process

- To reverse the process, we apply Bayes' rule

$$q(\mathbf{z}_{t-1}|\mathbf{z}_t) = \frac{q(\mathbf{z}_t|\mathbf{z}_{t-1})q(\mathbf{z}_{t-1})}{q(\mathbf{z}_t)}.$$

- This is intractable since we cannot compute marginal $q(z_t)$
- KEY assumption: Approximate conditional distribution using a Gaussian distribution

- Note: closed form Gaussian if conditioned on X

$$q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) = \text{Norm}_{\mathbf{z}_{t-1}}\left[\frac{(1-\alpha_{t-1})}{1-\alpha_t}\sqrt{1-\beta_t}\mathbf{z}_t + \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}, \frac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t}\mathbf{I}\right]$$

Decoder model (reverse process)

- Let a neural network predict mean of Gaussian

$$
\begin{aligned}
Pr(\mathbf{z}_T) &= \text{Norm}_{\mathbf{z}_T}[\mathbf{0}, \mathbf{I}] \\
Pr(\mathbf{z}_{t-1}|\mathbf{z}_t, \boldsymbol{\phi}_t) &= \text{Norm}_{\mathbf{z}_{t-1}}\Big[\mathbf{f}_t[\mathbf{z}_t, \boldsymbol{\phi}_t], \sigma_t^2\mathbf{I}\Big] \\
Pr(\mathbf{x}|\mathbf{z}_1, \boldsymbol{\phi}_1) &= \text{Norm}_{\mathbf{x}}\Big[\mathbf{f}_1[\mathbf{z}_1, \boldsymbol{\phi}_1], \sigma_1^2\mathbf{I}\Big],
\end{aligned}
$$

- What about variance of Gaussian? – Fixed schedule

Training

- To train the model, we maximize the log-likelihood of the training data $\{x_i\}$ with respect to the parameters

$$\hat{\phi}_{1\ldots T} = \underset{\phi_{1\ldots T}}{\operatorname{argmax}} \left[\sum_{i=1}^{I} \log \left[Pr(\mathbf{x}_i | \phi_{1\ldots T}) \right] \right]$$

Evidence lower bound (ELBO)

$$
\begin{aligned}
\log\left[Pr(\mathbf{x}|\boldsymbol{\phi}_{1\ldots T})\right] &= \log\left[\int Pr(\mathbf{x}, \mathbf{z}_{1\ldots T}|\boldsymbol{\phi}_{1\ldots T}) d\mathbf{z}_{1\ldots T}\right] \\
&= \log\left[\int q(\mathbf{z}_{1\ldots T}|\mathbf{x}) \frac{Pr(\mathbf{x}, \mathbf{z}_{1\ldots T}|\boldsymbol{\phi}_{1\ldots T})}{q(\mathbf{z}_{1\ldots T}|\mathbf{x})} d\mathbf{z}_{1\ldots T}\right] \\
&\geq \int q(\mathbf{z}_{1\ldots T}|\mathbf{x}) \log\left[\frac{Pr(\mathbf{x}, \mathbf{z}_{1\ldots T}|\boldsymbol{\phi}_{1\ldots T})}{q(\mathbf{z}_{1\ldots T}|\mathbf{x})}\right] d\mathbf{z}_{1\ldots T}.
\end{aligned}
$$

$$
= \mathbb{E}_{q(\mathbf{z}_1|\mathbf{x})}\left[\log\left[Pr(\mathbf{x}|\mathbf{z}_1, \boldsymbol{\phi}_1)\right]\right] - \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{z}_t|\mathbf{x})}\left[\mathrm{D}_{KL}\left[q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x})\middle|\middle|Pr(\mathbf{z}_{t-1}|\mathbf{z}_t, \boldsymbol{\phi}_t)\right]\right]
$$

UCMERCED

Analyzing ELBO

$$\mathbb{E}_{q(\mathbf{z}_1|\mathbf{x})}\Big[\log\left[Pr(\mathbf{x}|\mathbf{z}_1,\boldsymbol{\phi}_1)\right]\Big] - \sum_{t=2}^{T}\mathbb{E}_{q(\mathbf{z}_t|\mathbf{x})}\Big[\mathrm{D}_{KL}\Big[q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})\Big|\Big|Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\boldsymbol{\phi}_t)\Big]\Big]$$

$$Pr(\mathbf{x}|\mathbf{z}_1,\boldsymbol{\phi}_1) = \mathrm{Norm}_{\mathbf{x}}\Big[\mathbf{f}_1[\mathbf{z}_1,\boldsymbol{\phi}_1],\sigma_1^2\mathbf{I}\Big]$$

$$Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\boldsymbol{\phi}_t) = \mathrm{Norm}_{\mathbf{z}_{t-1}}\Big[\mathbf{f}_t[\mathbf{z}_t,\boldsymbol{\phi}_t],\sigma_t^2\mathbf{I}\Big]$$

$$q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x}) = \mathrm{Norm}_{\mathbf{z}_{t-1}}\left[\frac{(1-\alpha_{t-1})}{1-\alpha_t}\sqrt{1-\beta_t}\mathbf{z}_t + \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}, \frac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t}\mathbf{I}\right]$$

UCMERCED

Diffusion Loss Function

$$L[\boldsymbol{\phi}_{1\ldots T}] = \sum_{i=1}^{I}\Bigg(\overbrace{-\log\Big[\text{Norm}_{\mathbf{x}_i}\left[\mathbf{f}_1[\mathbf{z}_{i1},\boldsymbol{\phi}_1],\sigma_1^2\mathbf{I}\right]\Big]}^{\text{reconstruction term}} + \sum_{t=2}^{T}\frac{1}{2\sigma_t^2}\Bigg\|\underbrace{\frac{1-\alpha_{t-1}}{1-\alpha_t}\sqrt{1-\beta_t}\mathbf{z}_{it}+\frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}_i}_{\text{target, mean of } q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})} - \underbrace{\mathbf{f}_t[\mathbf{z}_{it},\boldsymbol{\phi}_t]}_{\text{predicted } \mathbf{z}_{t-1}}\Bigg\|^2\Bigg), \tag{18.29}$$

where $\mathbf{x}_i$ is the i^{th} data point, and $\mathbf{z}_{it}$ is the associated latent variable at diffusion step t.

Reparameterization of target

- Target (mean) can be reparameterized using noise

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

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

- Reparameterization of network

$$\mathbf{f}_t[\mathbf{z}_t, \boldsymbol{\phi}_t] = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t].$$

- Simple loss function for diffusion model:

$$\begin{aligned} L[\boldsymbol{\phi}_{1\ldots T}] &= \sum_{i=1}^{I}\sum_{t=1}^{T} \left\| \mathbf{g}_t[\mathbf{z}_{it}, \boldsymbol{\phi}_t] - \boldsymbol{\epsilon}_{it} \right\|^2 \\ &= \sum_{i=1}^{I}\sum_{t=1}^{T} \left\| \mathbf{g}_t\left[\sqrt{\alpha_t} \cdot \mathbf{x}_i + \sqrt{1-\alpha_t} \cdot \boldsymbol{\epsilon}_{it}, \boldsymbol{\phi}_t\right] - \boldsymbol{\epsilon}_{it} \right\|^2 \end{aligned}$$

Diffusion Model Training

Algorithm 18.1: Diffusion model training

Input: Training data $\mathbf{x}$
Output: Model parameters $\boldsymbol{\phi}_t$
repeat
  for $i \in \mathcal{B}$ **do** // For every training example index in batch
    $t \sim \text{Uniform}[1, \ldots T]$ // Sample random timestep
    $\boldsymbol{\epsilon} \sim \text{Norm}[\mathbf{0}, \mathbf{I}]$ // Sample noise
    $\ell_i = \left\| \mathbf{g}_t\left[\sqrt{\alpha_t}\mathbf{x}_i + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}, \boldsymbol{\phi}_t\right] - \boldsymbol{\epsilon} \right\|^2$ // Compute individual loss
  Accumulate losses for batch and take gradient step
until converged

UCMERCED

Diffusion Model Inference

Algorithm 18.2: Sampling

Input: Model, $\mathbf{g}_t[\bullet, \boldsymbol{\phi}_t]$

Output: Sample, $\mathbf{x}$

$\mathbf{z}_T \sim \text{Norm}_{\mathbf{z}}[\mathbf{0}, \mathbf{I}]$ // Sample last latent variable

for $t = T \ldots 2$ **do**

$\hat{\mathbf{z}}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$ // Predict previous latent variable

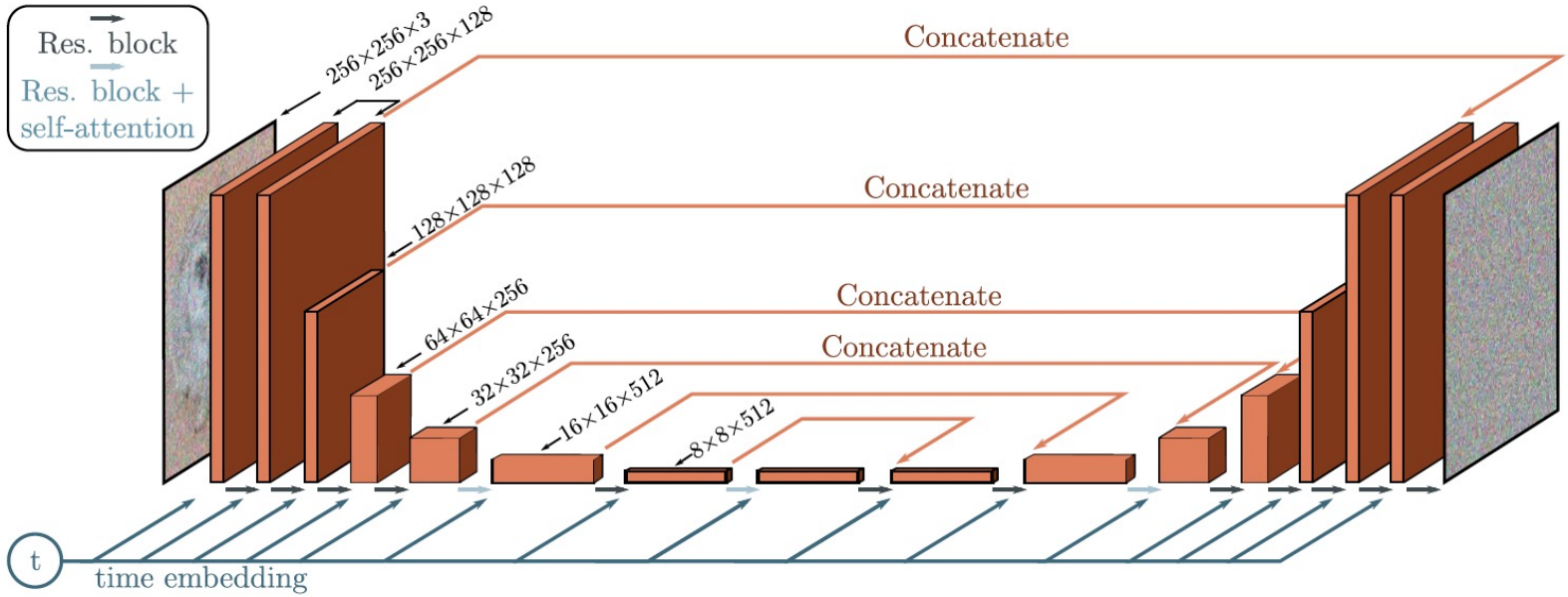
$\boldsymbol{\epsilon} \sim \text{Norm}_{\boldsymbol{\epsilon}}[\mathbf{0}, \mathbf{I}]$ // Draw new noise vector

$\mathbf{z}_{t-1} = \hat{\mathbf{z}}_{t-1} + \sigma_t\boldsymbol{\epsilon}$ // Add noise to previous latent variable

$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}\sqrt{1-\beta_1}}\mathbf{g}_1[\mathbf{z}_1, \boldsymbol{\phi}_1]$ // Generate sample from $\mathbf{z}_1$ without noise

Implementation

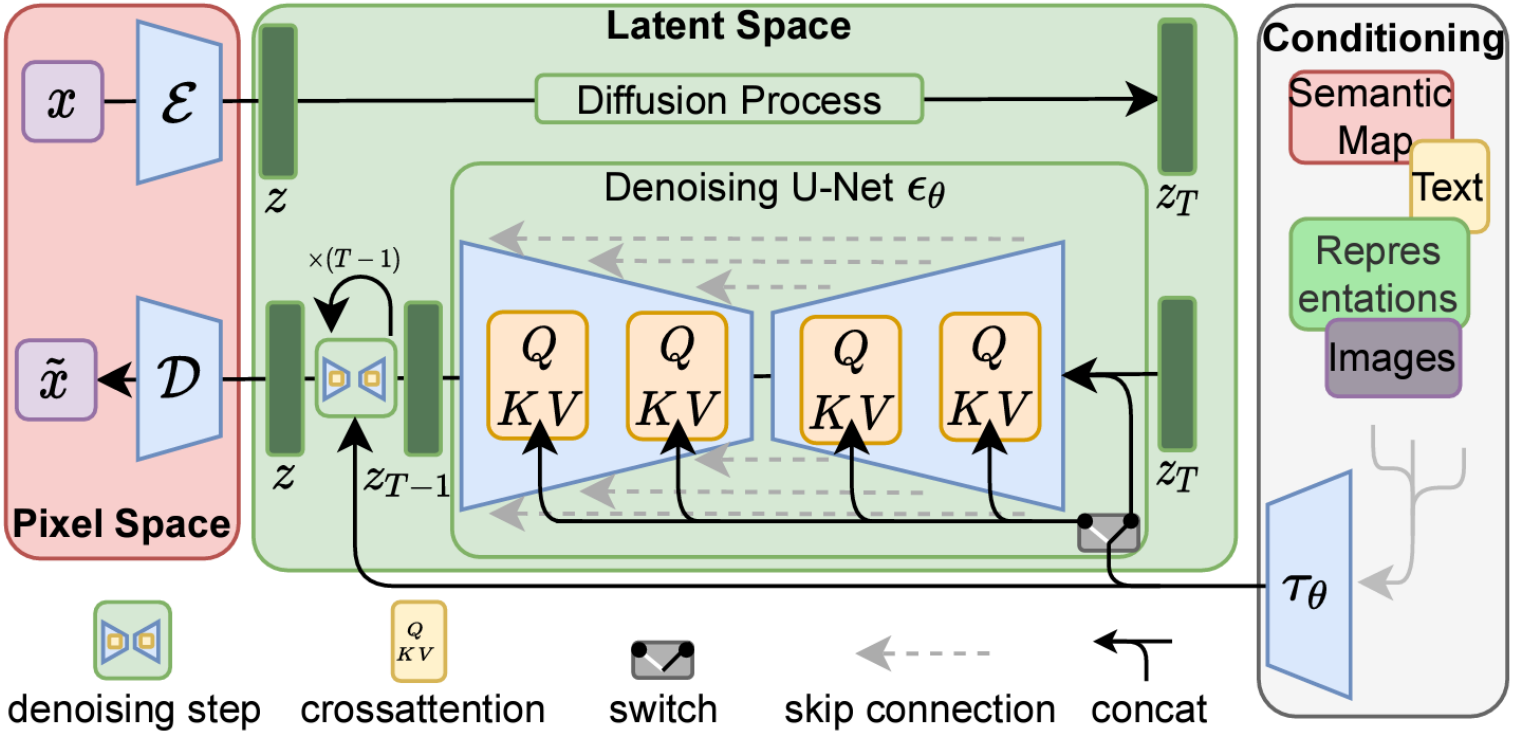
- U-Net based denoising network
- Noisy image as input
- Noise as output



UCMERCED

Latent diffusion model

- Reduce dimensionality of image by VAE
- Diffusion process in latent space of VAE
- Conditional generation using text, image, semantic map etc.



Parameter-efficient Conditonal Generation

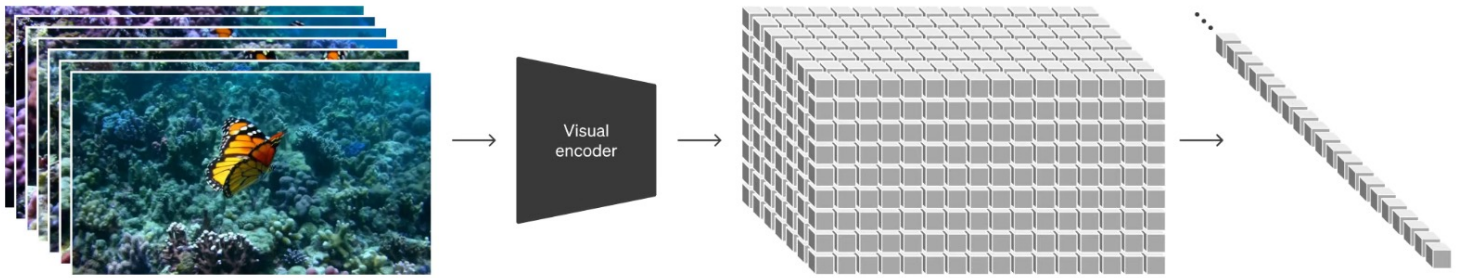
- ControlNet: Controlling diffusion model with various control

- Key idea: Zero convolution

UCMERCED

How Sora works?

- Latent diffusion model + U-ViT



Turning video into patches and compressed

Diffusion model for video

UCMERCED