Machine Learning

Lecture 17: Deep Generative Models

Fall 2023

Instructor: Xiaodong Gu

Unsupervised Learning

Clustering
(Learn data similarity)

- K-means
- Gaussion Mixture

Generative
(learn data distribution)

- GAN
- VAE
- Diffusion

Dimensional Reduction
(Learn data compression)

- Principal component analysis
- Auto-encoder

Generation Problems

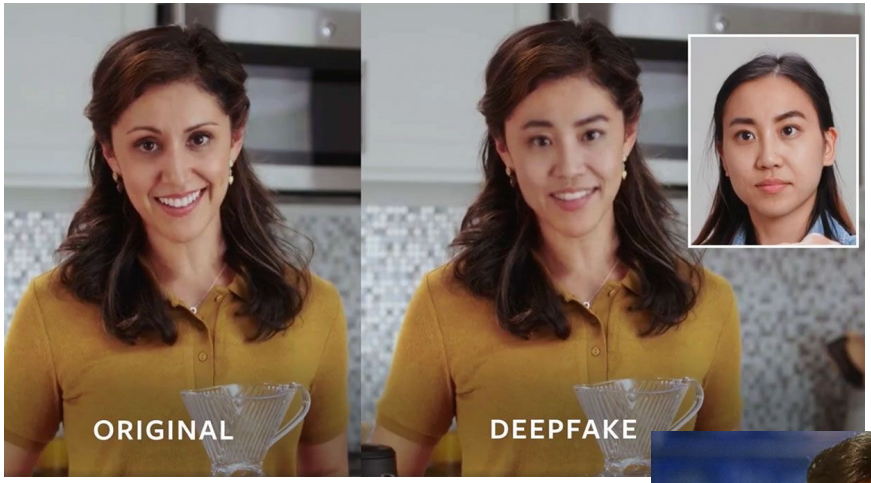
Deep Fake of Nvidia's CEO Jensen Huang at GTC

Example: stylized movie

AI-generated movie pictures in the style of 1980s

Example: Deep Fake

Which face is real?



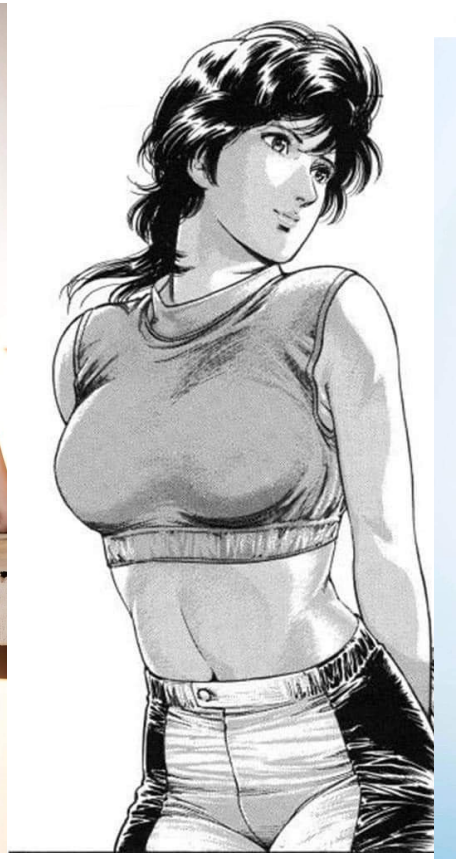
Example: Image Restoration/Colorization

1990s with super-resolution

with colorization

Example:漫画变真人






Example: DALL·E 2

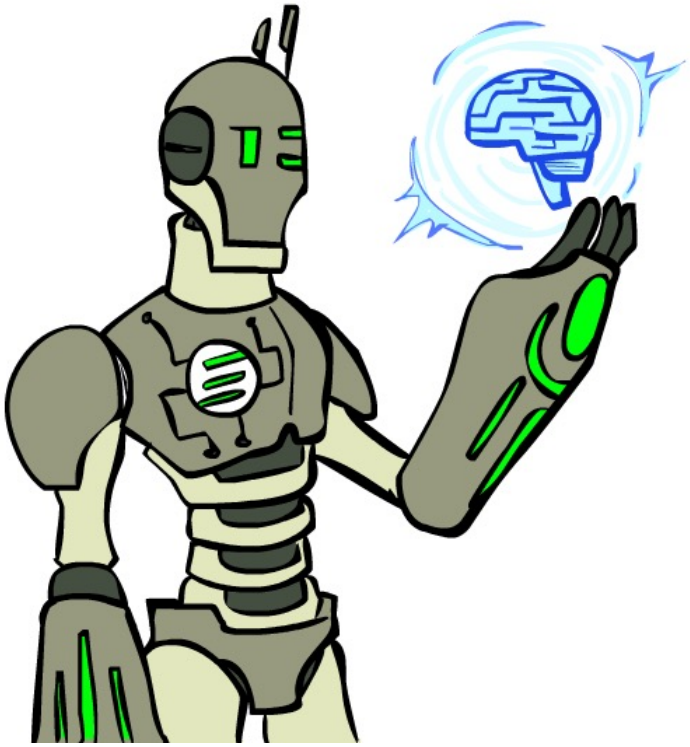
https://openai.com/dall-e-2/

Example: DALL·E 2

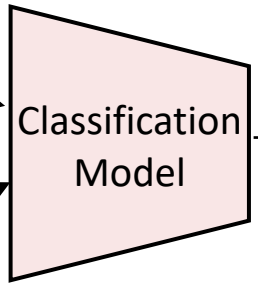
Today

Deep Generative Models

- Concept
- **GAN**
- VAE
- Diffusion

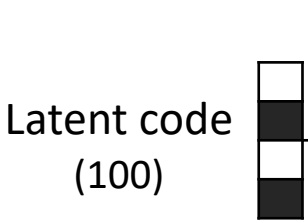


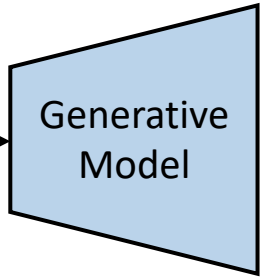
Classification vs. Generation



Man?
Woman?

learns how to classify input to its class.





What is the goal of generation tasks?

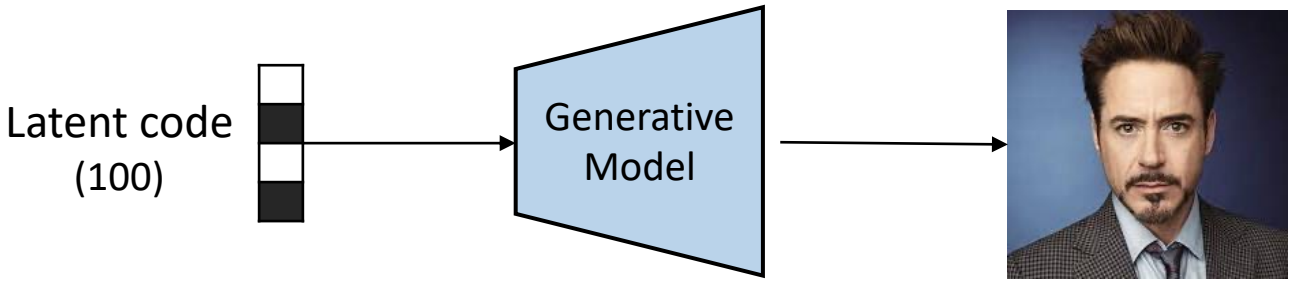
Review: Language Model

- A probabilistic model of how likely a given string appear in a given "language".

$$p(x) = p(w_1, w_2, \ldots, w_N)$$

P_1=P("我爱机器学习")
P_2=P("我爱学习机器")
P_3=P("机器我爱学习")
P_4=P("爱我机学习器")

Chinese: $P_1 > P_2 > P_3 > P_4$



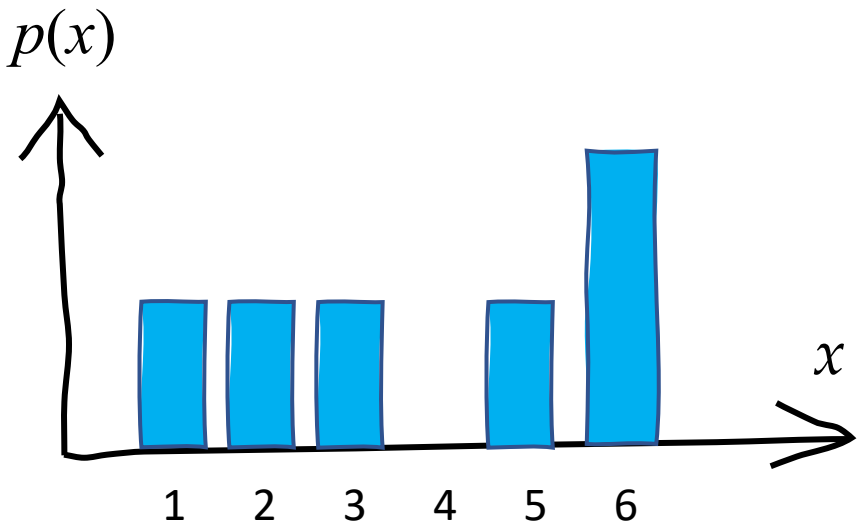
Generation = learns the probability distribution of training data.

Review: Probability Distribution

Random variable

Probability mass function

x	1	2	3	4	5	6
p(x)	1/6	1/6	1/6	0	1/6	2/6



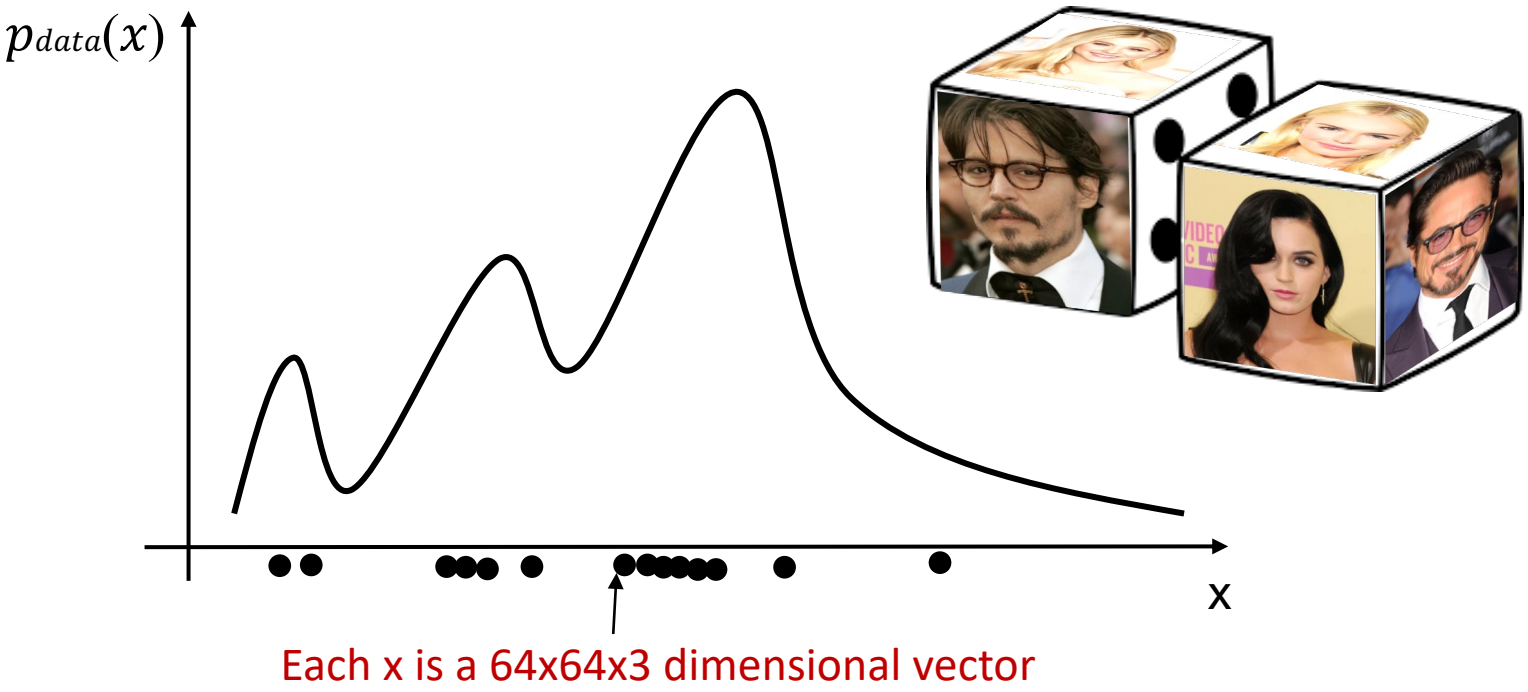
Probability Distributions for Images

- What if $\boldsymbol{x}$ is actual images in the training data?
- At this point, $\boldsymbol{x}$ can be represented as a (for example) 64×64×3 dimensional vector.

Probability Distributions for Images

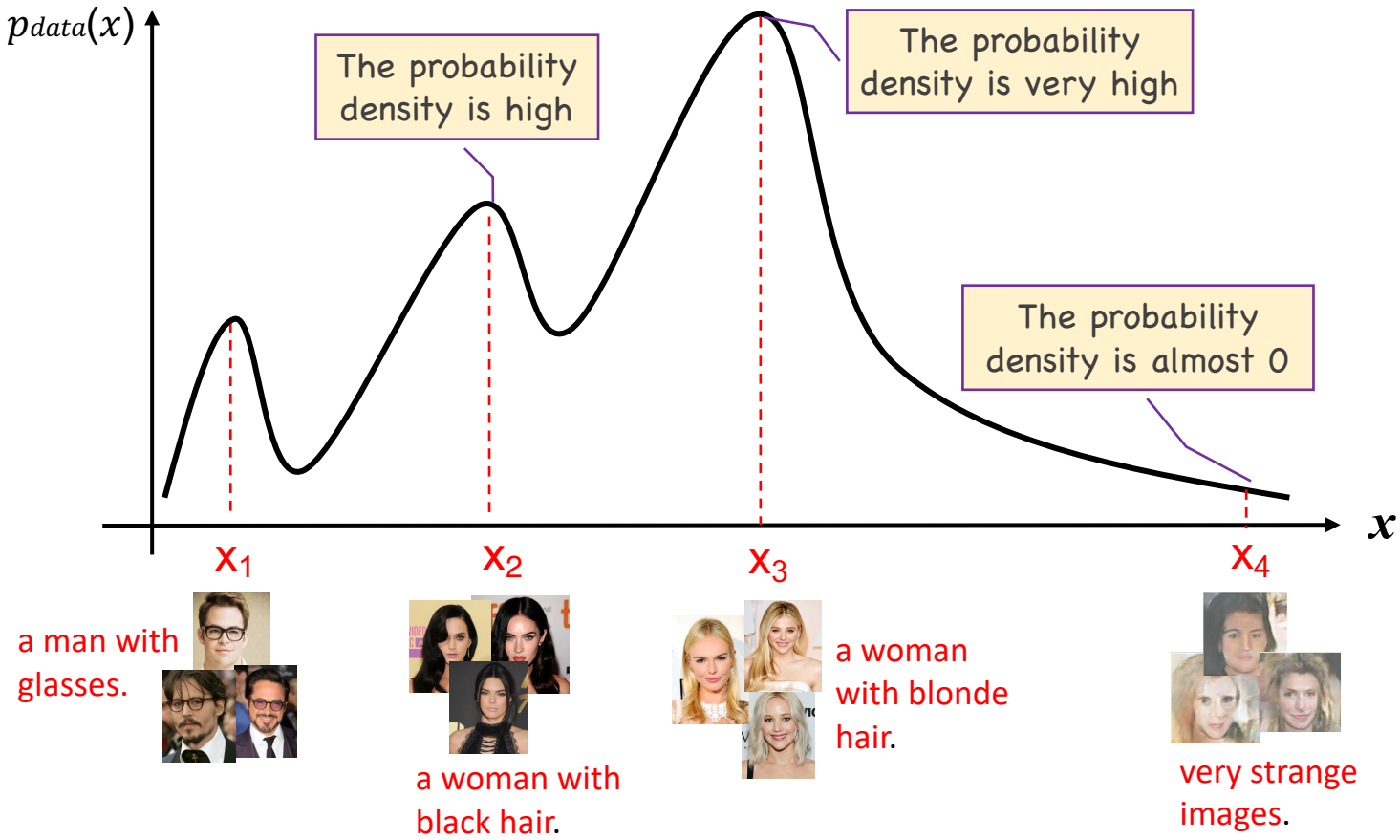
- There is a $p_{\text{data}}(\boldsymbol{x})$ that represents the distribution of actual images.

Probability density function



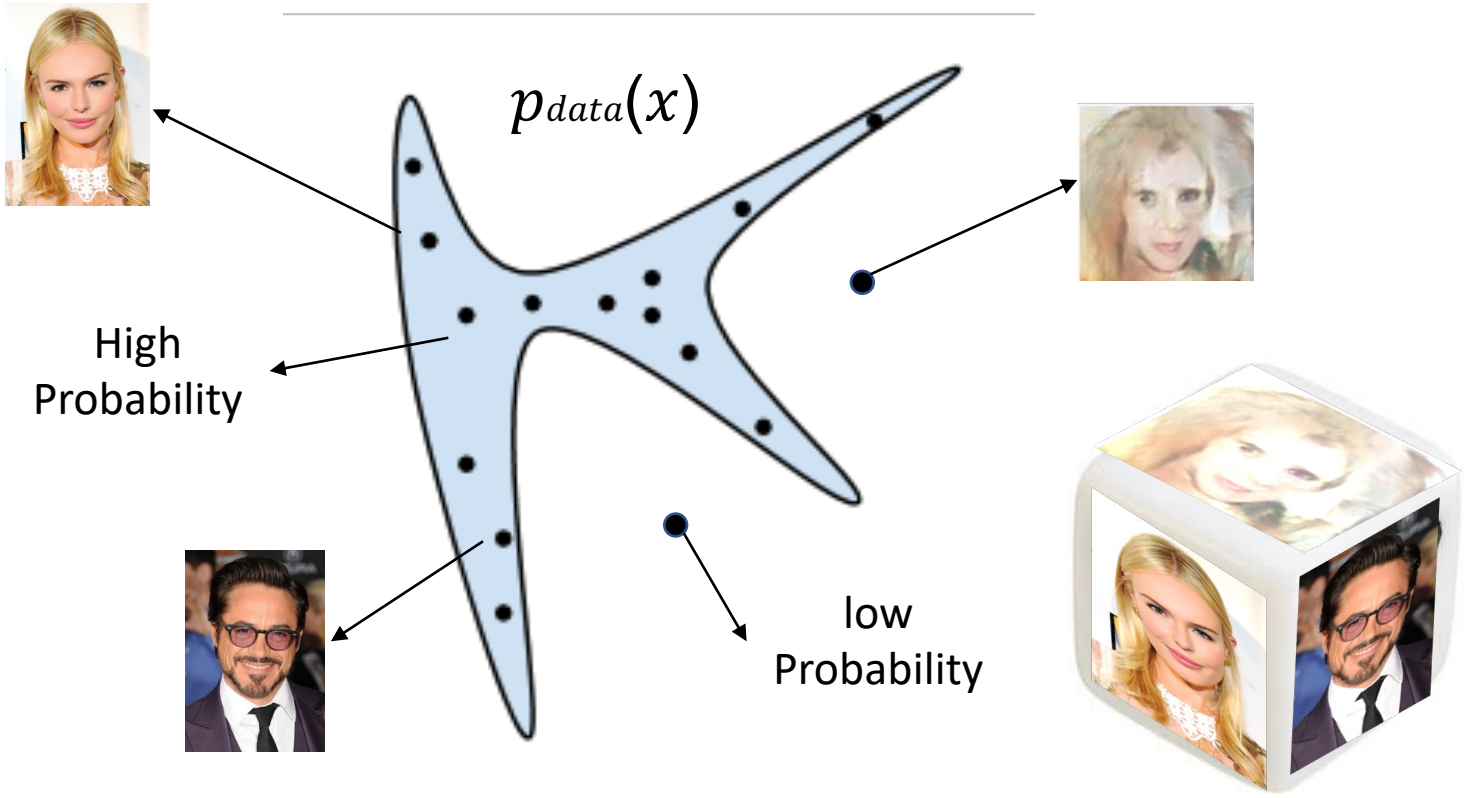
Probability Distributions for Images

- Let's take an example with an image dataset of western stars.



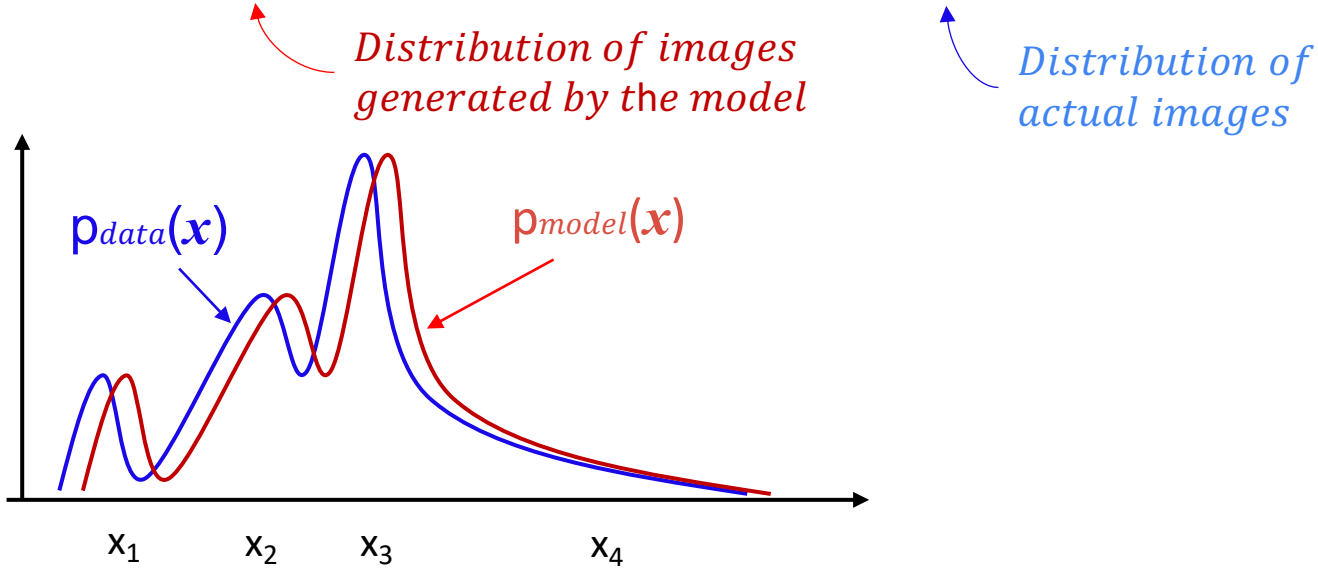
Probability Distributions for Images

- We have a data distribution $P_{data}(x)$



Generative Model

- **Goal**: find a $p_{model}(x)$ that approximates $p_{data}(x)$ well.

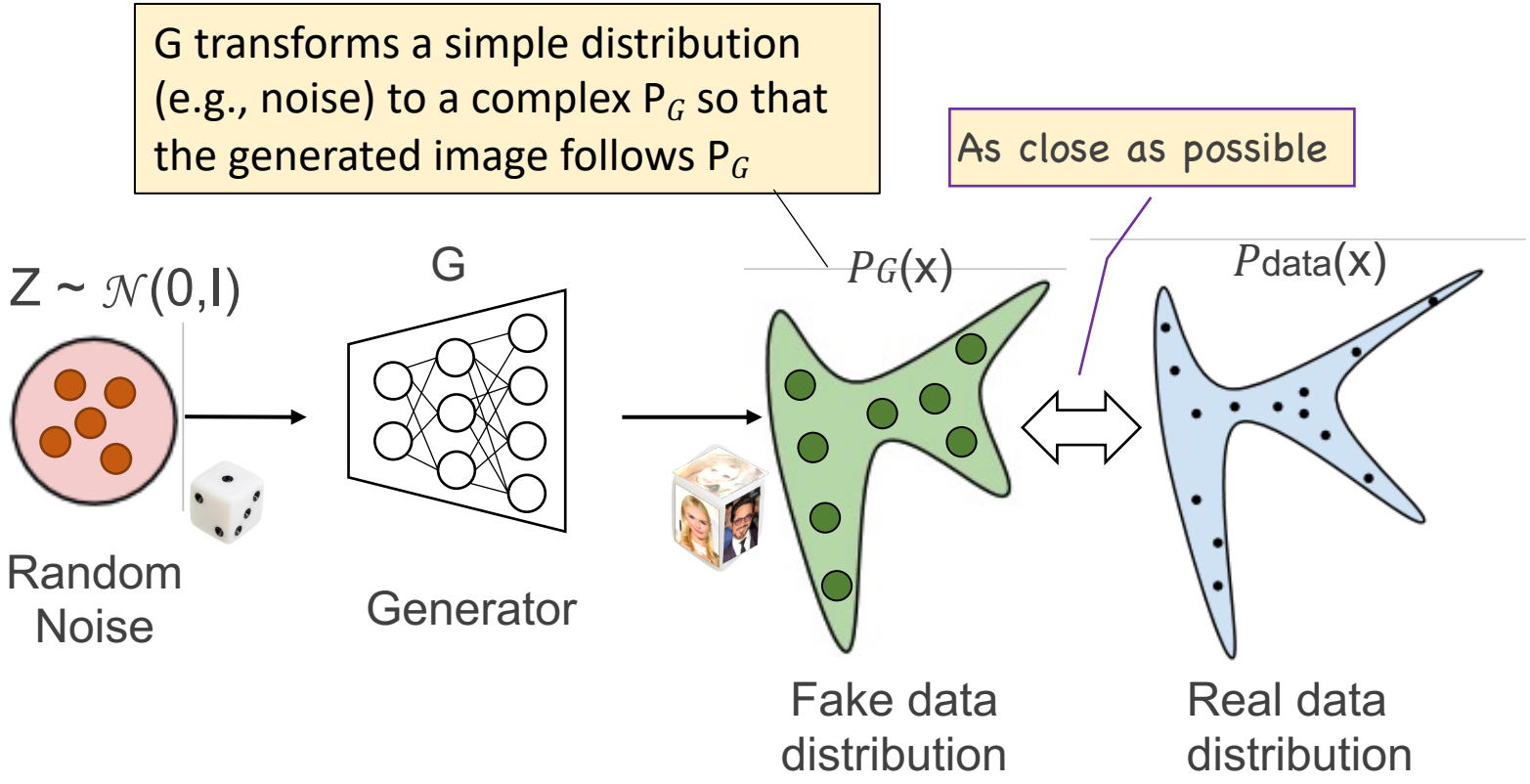


Q1: How to generate an image that follows a distribution $P_{model}(x)$?
In other words, how to represent $P_{model}(x)$?

Q2: How can we learn a $P_{model}(x)$ that is close to $P_{data}(x)$?

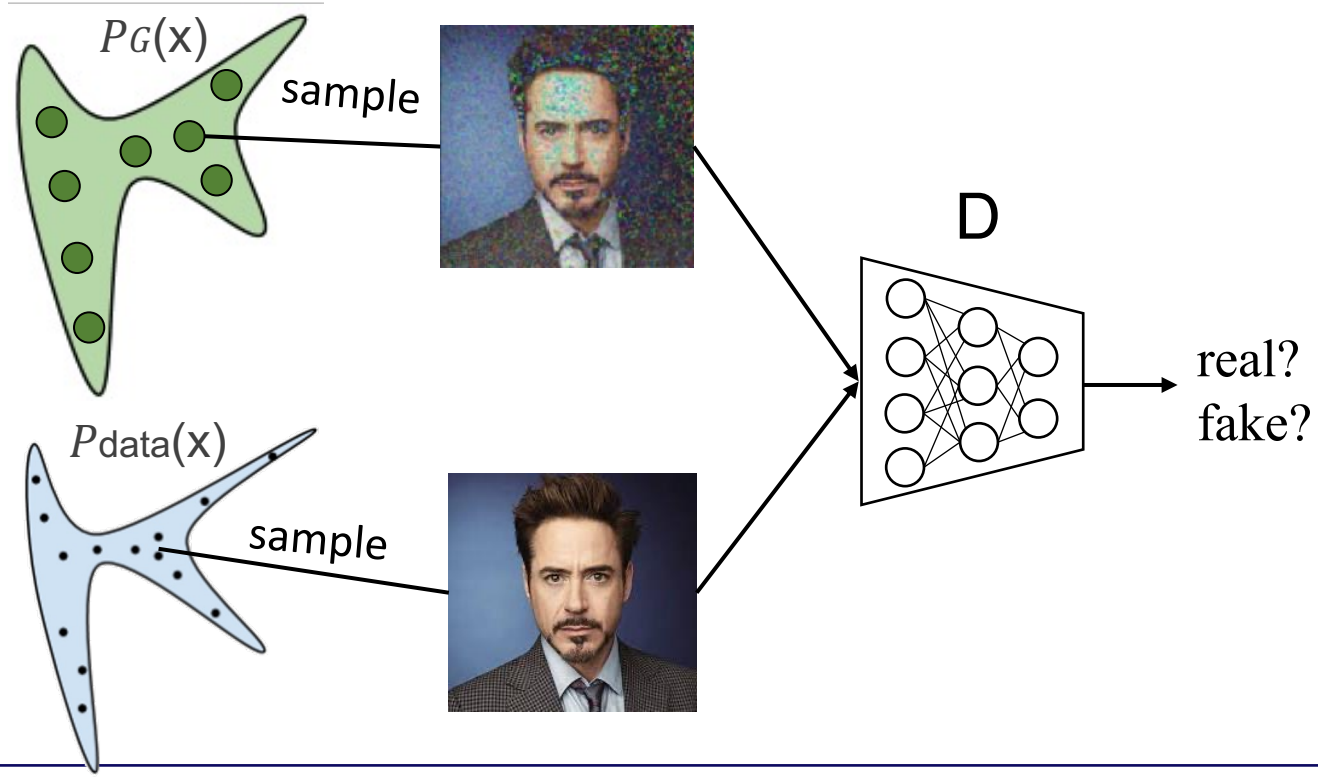
How to represent $p_{model}(x)$?

- **Generator G:** a neural network that generates an image x with a probability $\boldsymbol{p_{model}(x)}$ given a random code.



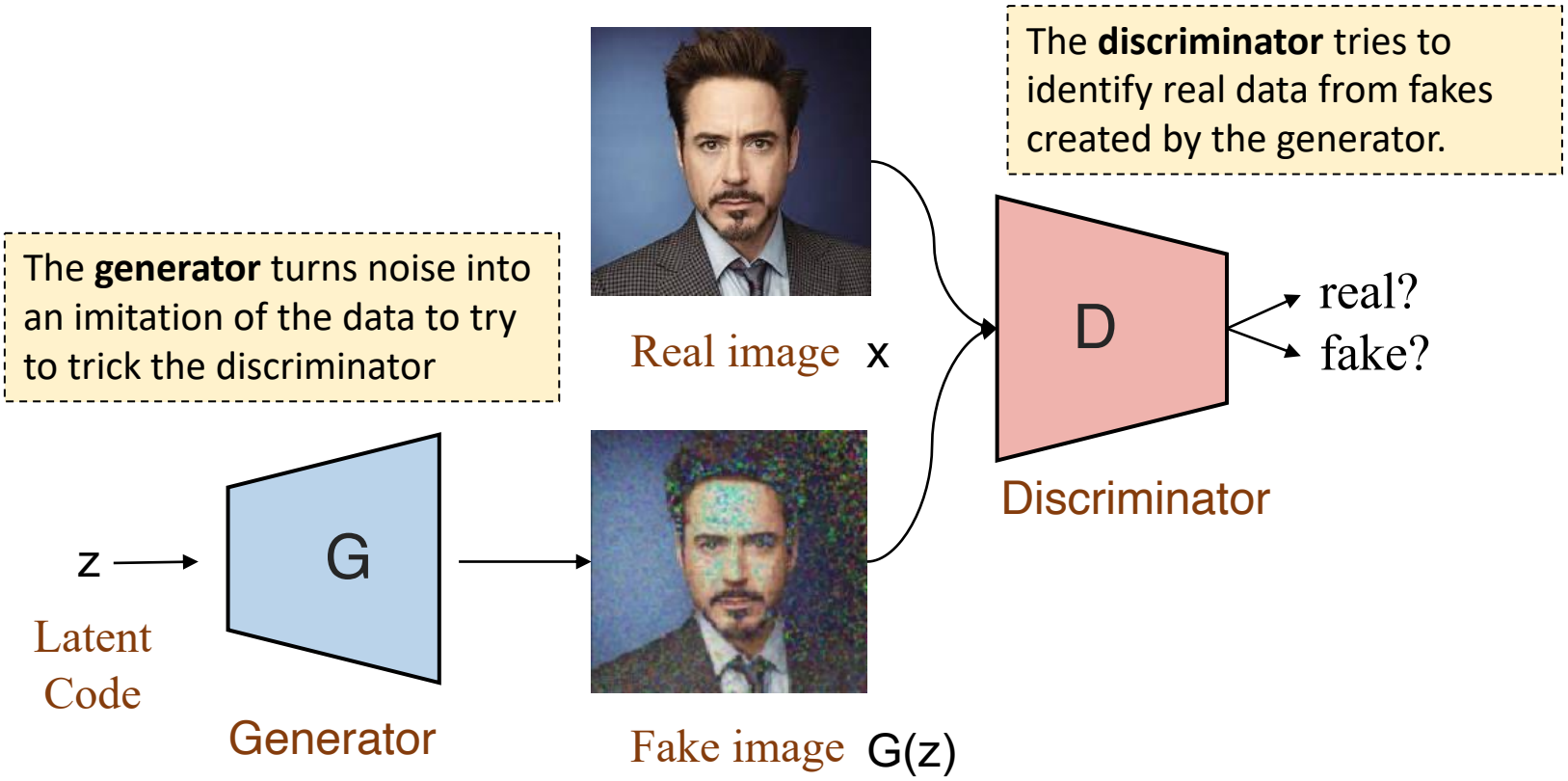
How to make $p_{\text{model}}(x) \approx p_{\text{data}}(x)$?

- **Discriminator D**: a neural network classifier that predicts whether a given image is sampled from $p_{\text{model}}(x)$ (fake) or $p_{\text{data}}(x)$ (real). If undistinguishable, then $p_{\text{model}}(x) \approx p_{\text{data}}(x)$



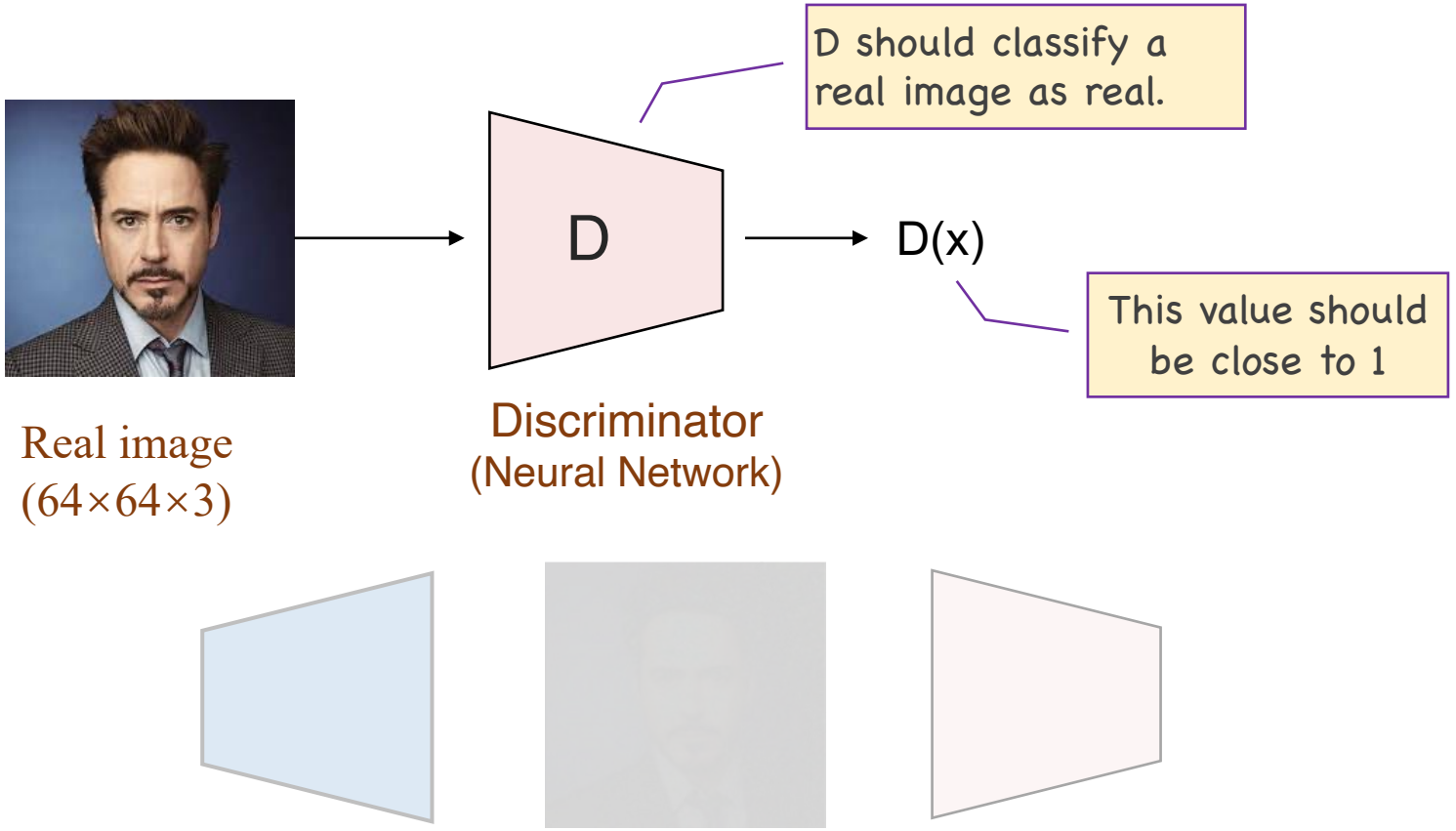
Generative Adversarial Network (GAN)

- A generative model by having **two** neural networks **compete** each other.



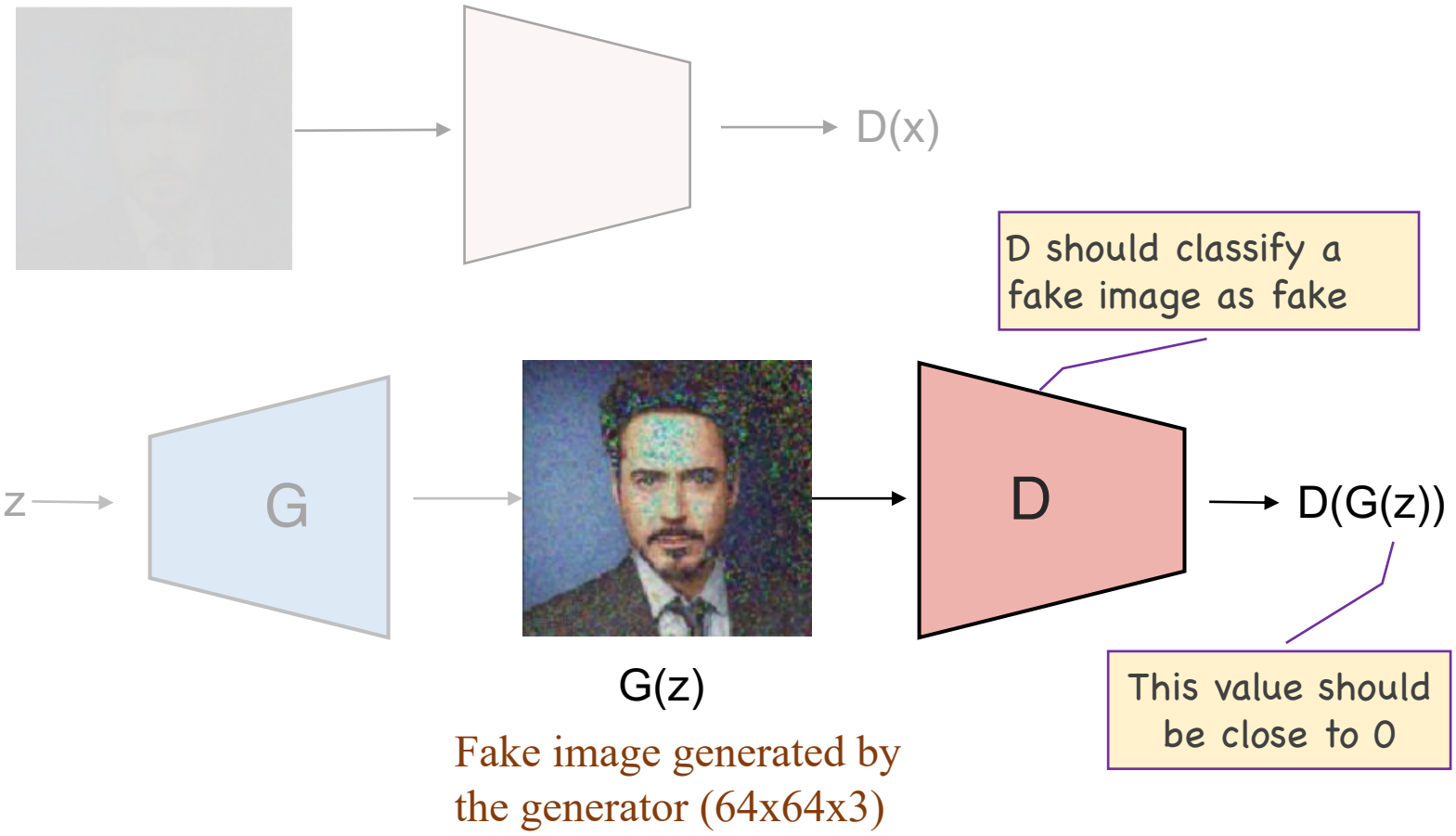
Objective of Discriminator

- The discriminator tries to identify the synthesized instances



Objective of Discriminator

- The discriminator tries to identify the synthesized instances



Objective of Discriminator

Recall: binary cross-entropy loss:
$l(h(x), y) = -y \log h(x) - (1-y)\log(1-h(x))$

- The objective function can be formulated as the **binary cross-entropy**:

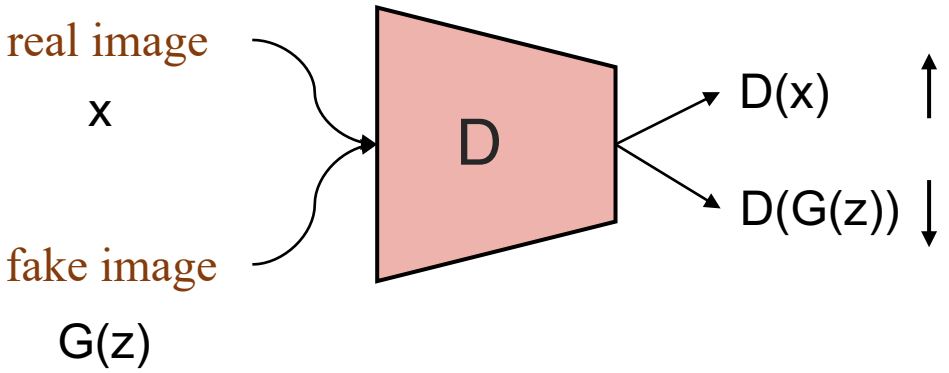
Sample x from real data distribution

Sample x from model distribution

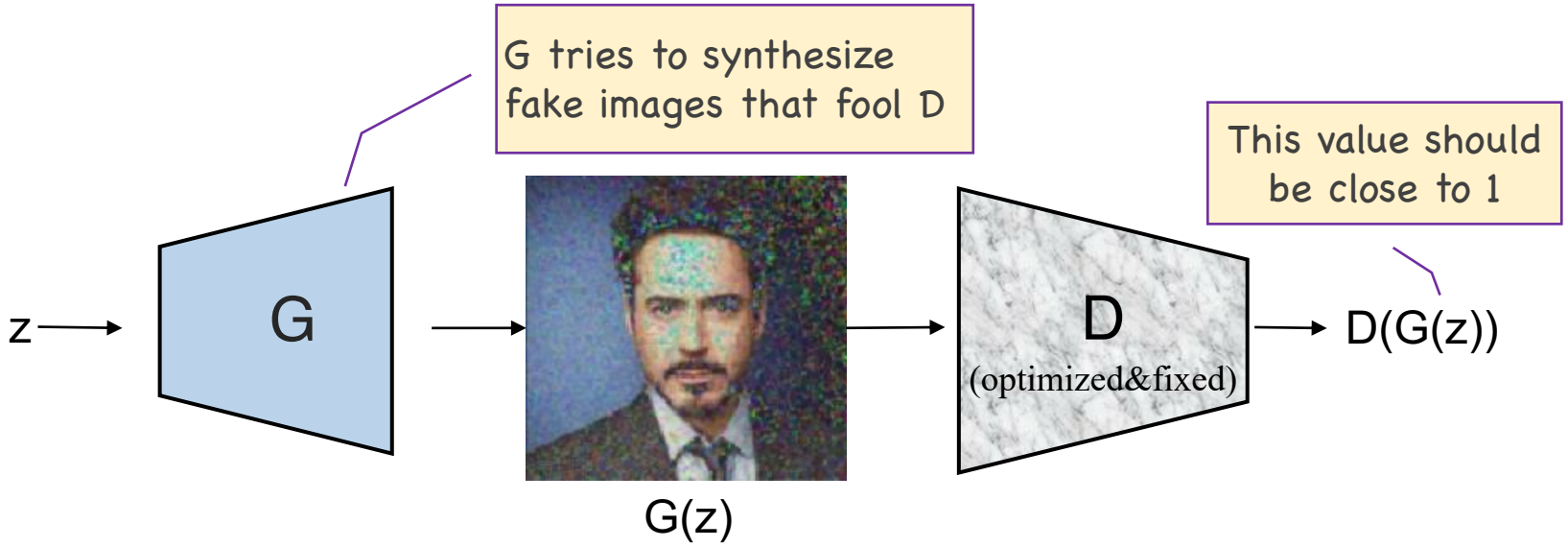
$$D^* = \arg\max_D E_{x\sim P_{data}}[\log D(x)] + E_{x\sim P_G}[\log(1-D(x)]$$

$$= \arg\max_D E_{x\sim P_{data}}[\log D(x)] + E_{z\sim P_z(z)}[\log(1-D(G(z))]$$

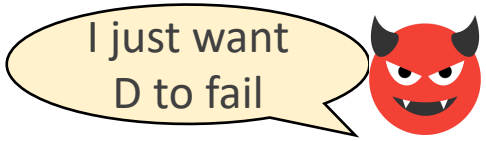
Sample z from unit Gaussian



Objective of Generator



G has an opposite (adversarial) objective to D !

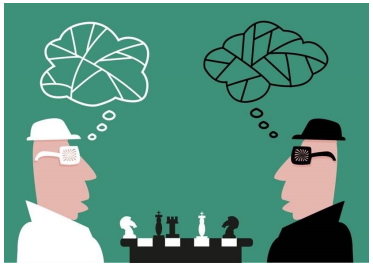


G is independent of this part

$$G^* = \text{argmin}_G \; \mathbf{E}_{x\sim P_{data}}[\log D(x)] + \mathbf{E}_{z\sim P_z(z)}[\log(1-D(G(z)))]$$

$$= \text{argmin}_G \; \mathbf{E}_{z\sim P_z(z)}[\log(1-D(G(z)))]$$

Objective of GAN



Adversarial Objectives for G and D

$$\min_G \max_D V(G,D) = E_{x\sim P_{data}}[\log D(x)] + E_{z\sim P_z(z)}[\log(1-D(G(z))]$$

D: By maximizing V(G, D), I can distinguish real and fake, I am getting stronger 😏

D

G: No matter how strong you are, I can minimize V(G,D) to fool you. I am also stronger ✌️.

G

D: Though you are strong, I can be much stronger by maximizing V(G, D) . 🤗

⋮

Global Optimum: G reproduces the true data distribution

The Algorithm

GAN

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

A PyTorch Implementation

```python
import torch
import torch.nn as nn


D = nn.Sequential(
    nn.Linear(784, 128),
    nn.ReLU(),
    nn.Linear(128, 1),
    nn.Sigmoid())

G = nn.Sequential(
    nn.Linear(100, 128),
    nn.ReLU(),
    nn.Linear(128, 784),
    nn.Tanh())

```

```python
criterion = nn.BCELoss()

d_optimizer = torch.optim.Adam(D.parameters(), lr=0.01)
g_optimizer = torch.optim.Adam(G.parameters(), lr=0.01)

# Assume x be real images of shape (batch_size, 784)
# Assume z be random noise of shape (batch_size, 100)

while True:
    # train D
    loss = criterion(D(x), 1) + criterion(D(G(z)), 0)
    loss.backward()
    d_optimizer.step()

    # train G
    loss = criterion(D(G(z)), 1)
    loss.backward()
    g_optimizer.step()
```

TIME for Coding

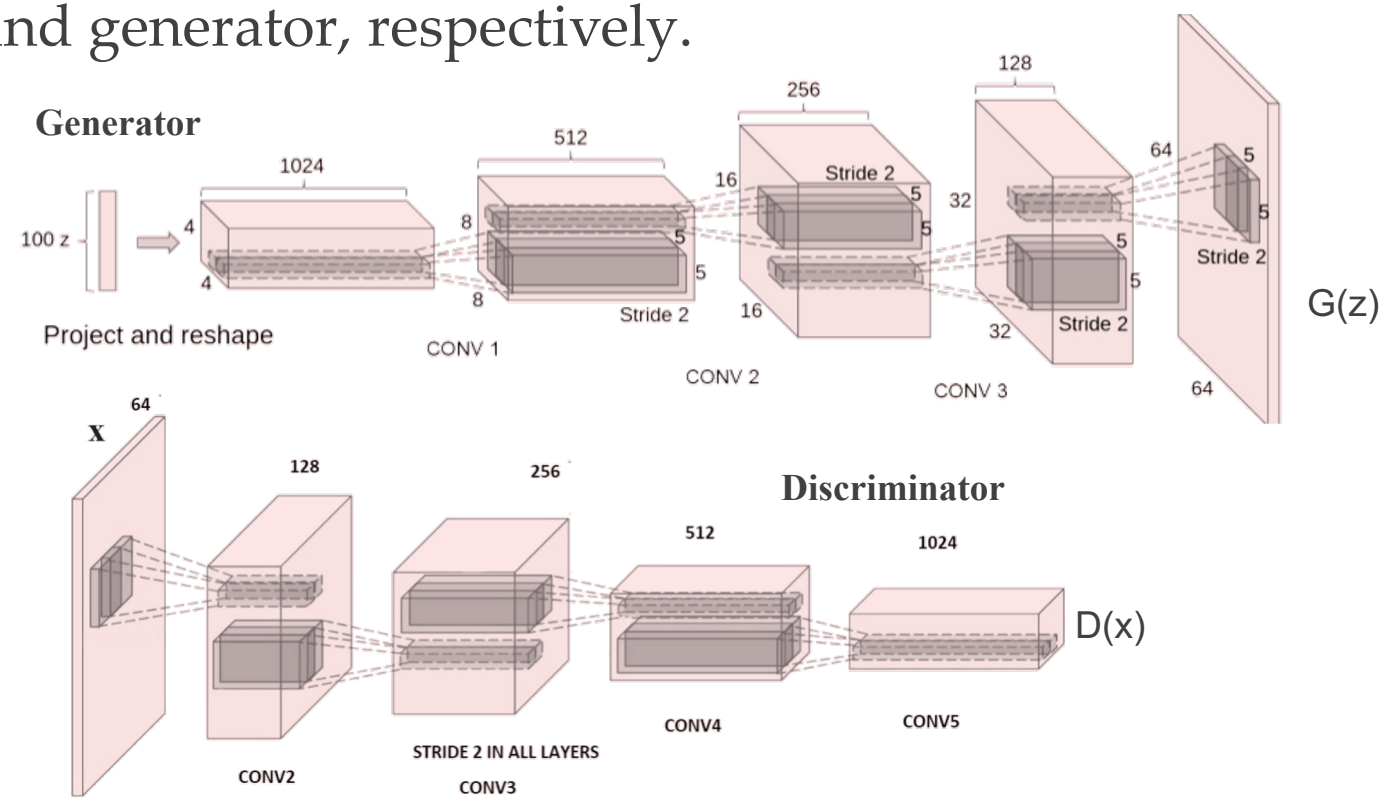
Tutorial: Implement GAN with PyTorch

https://www.kaggle.com/code/chandraprajapati/image-generation-using-gan-anime-face

Some GANs

Deep Convolutional GANs (DCGAN)

- Use **convolution** and **deconvolution** for the discriminator and generator, respectively.

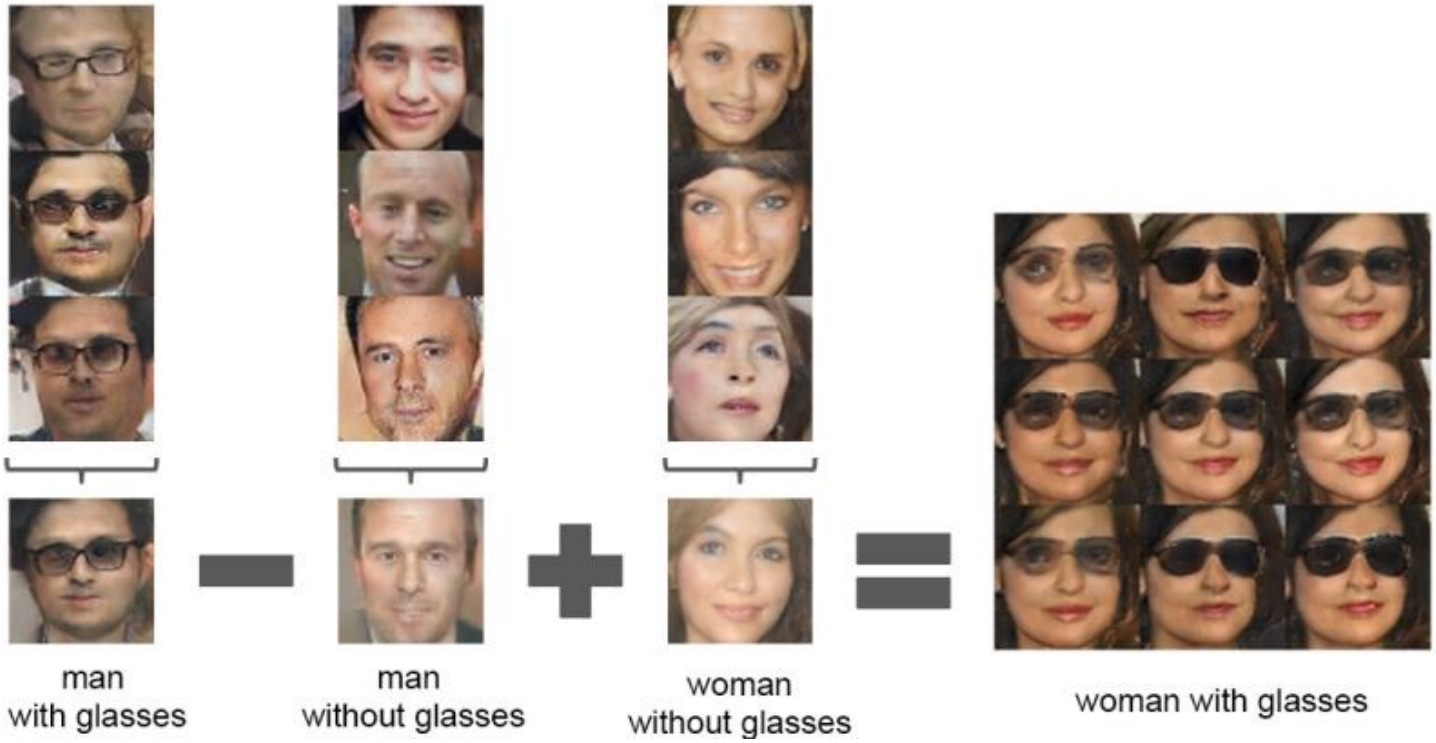


Tutorial: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

Radford et al. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015

Deep Convolutional GANs (DCGAN)

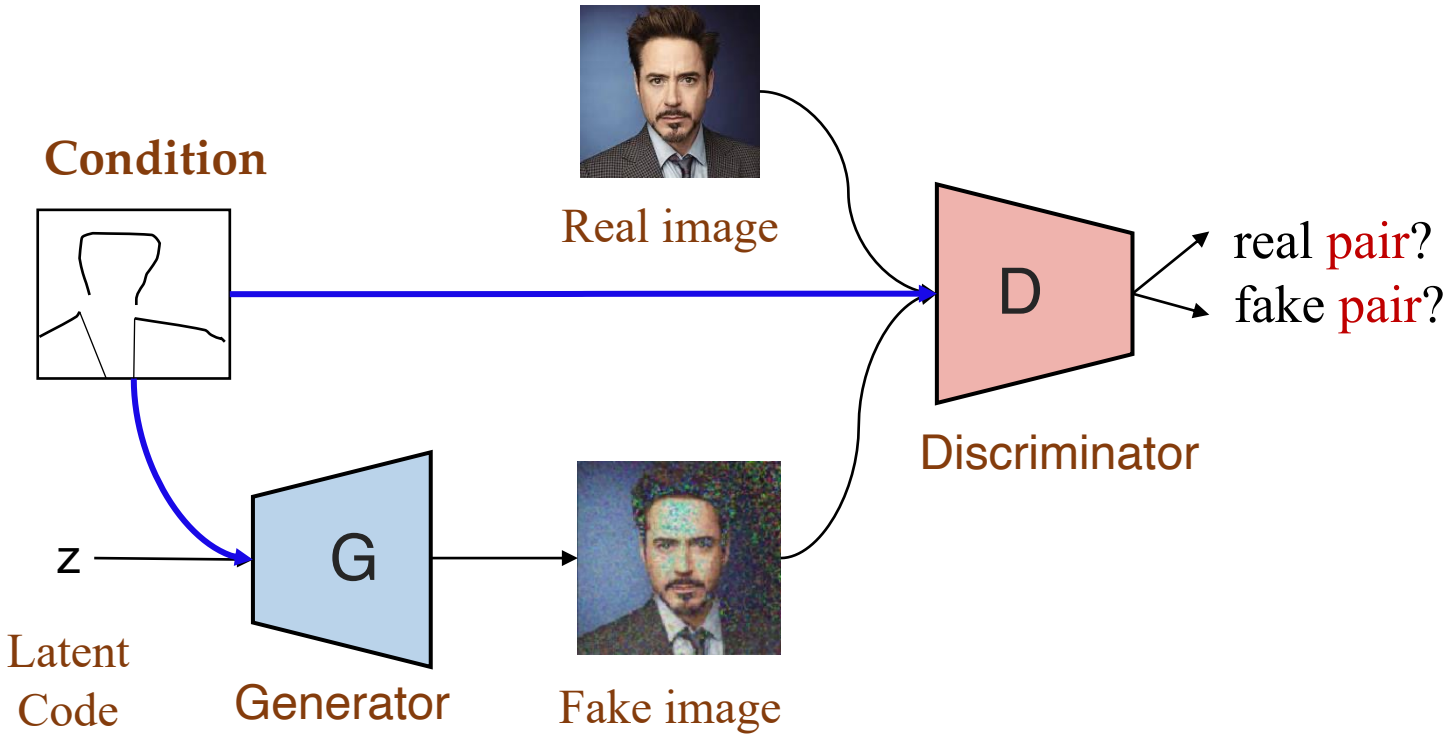
Latent vector arithmetic



Tutorial: https://machinelearningmastery.com/how-to-interpolate-and-perform-vector-arithmetic-with-faces-using-a-generative-adversarial-network/

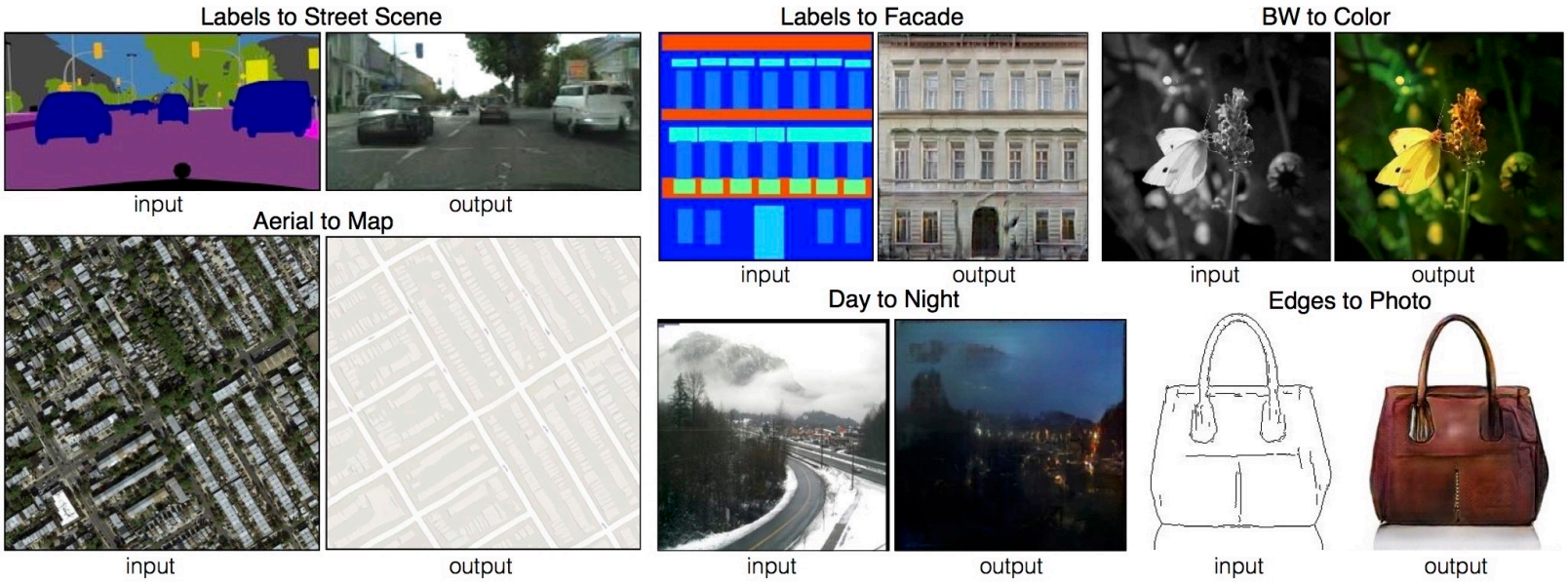
Conditional GANs

- What if we want to control the nature of the output, by **conditioning** on a label?



Conditional GANs

- **Results** – Image-to-Image Translation

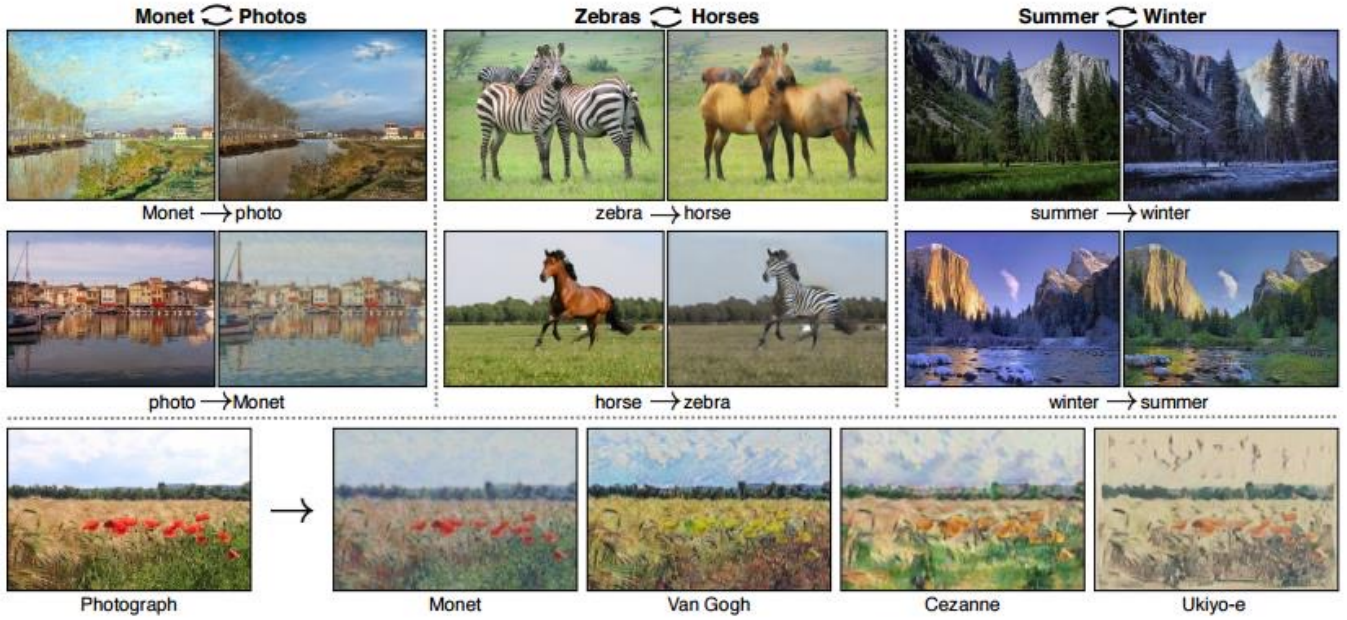


Limitation: Supervised Learning on Parallel Images

CycleGAN: Domain Transformation

- A GAN model that transfers an image from a **source domain** A to a **target domain** B in the absence of paired examples.

Example: Unpaired Image-to-Image Translation

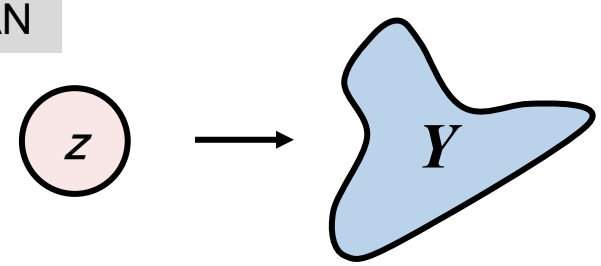


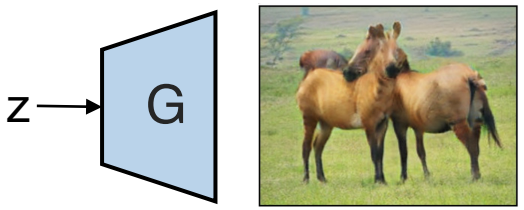
Jun-Yan Zhu et al. Unpaired Image-to-Image Translation using Cycle Consistent Adversarial Networks, 2017

CycleGAN

Distribution transformations

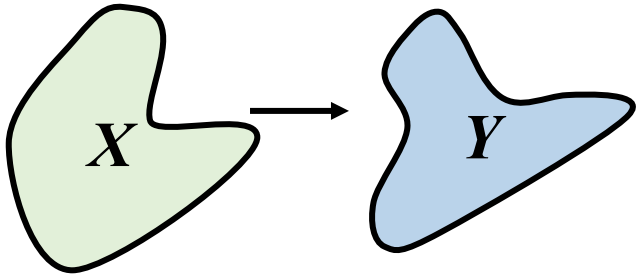
GAN

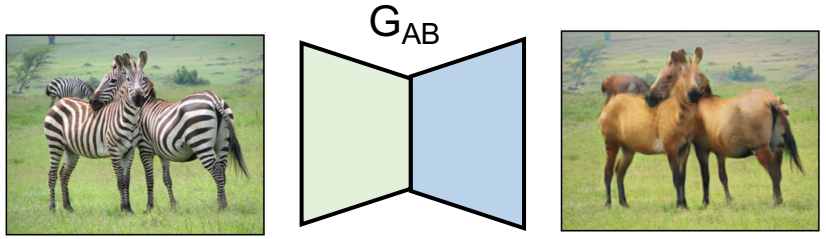




Gaussian noise → target data manifold

CycleGAN





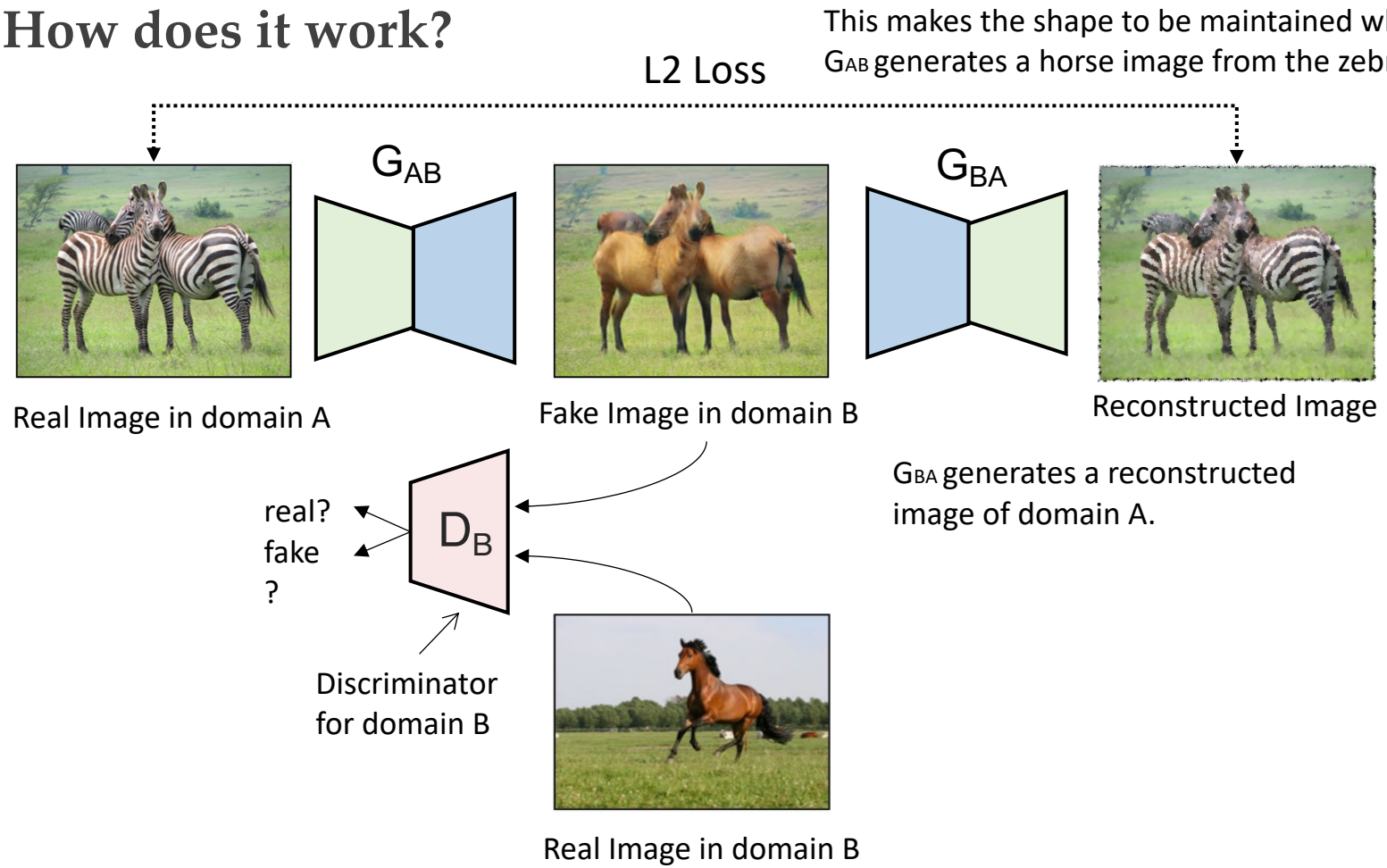
Real Image in domain A

Generated Image in domain B

data manifold X → data manifold Y

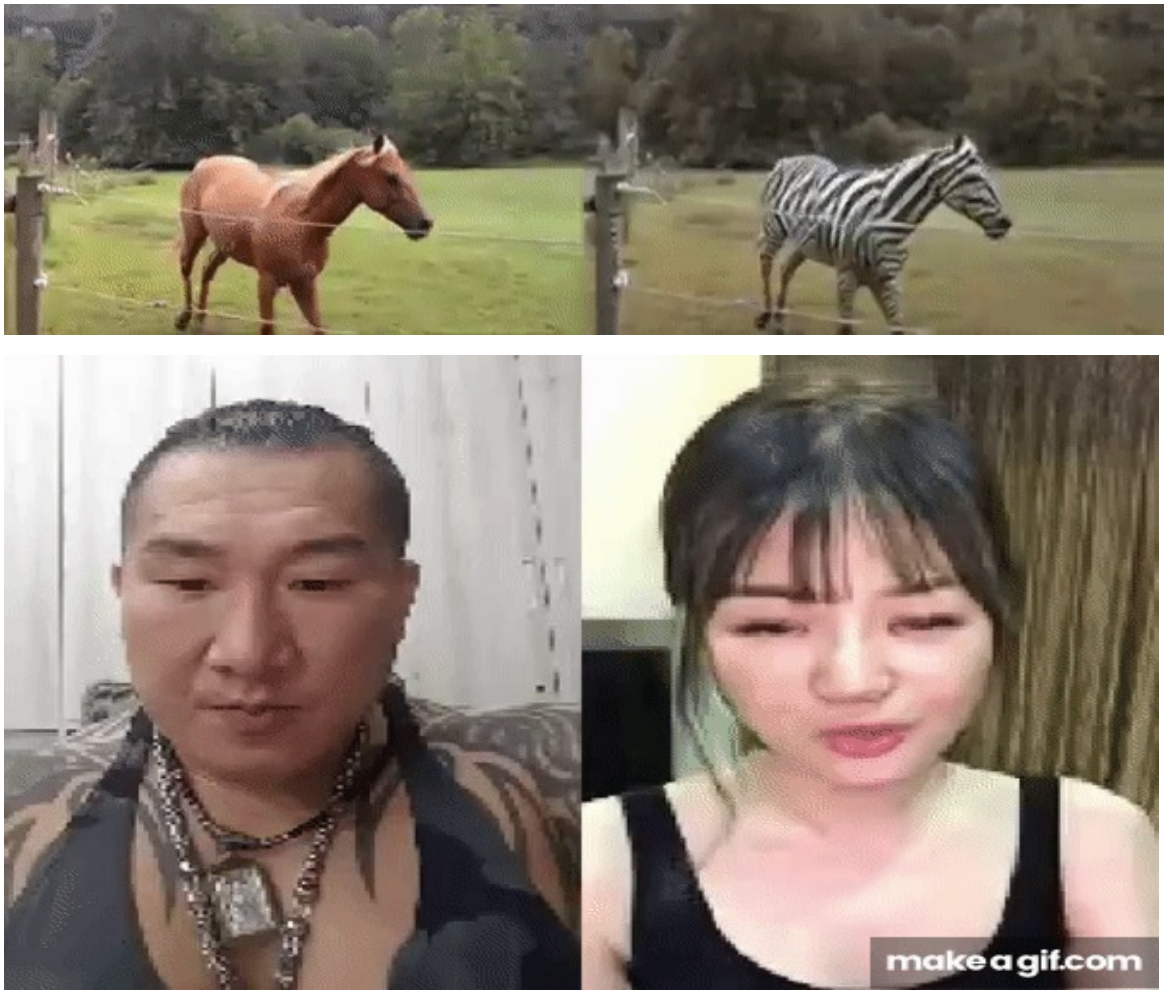
CycleGAN

How does it work?



CycleGAN

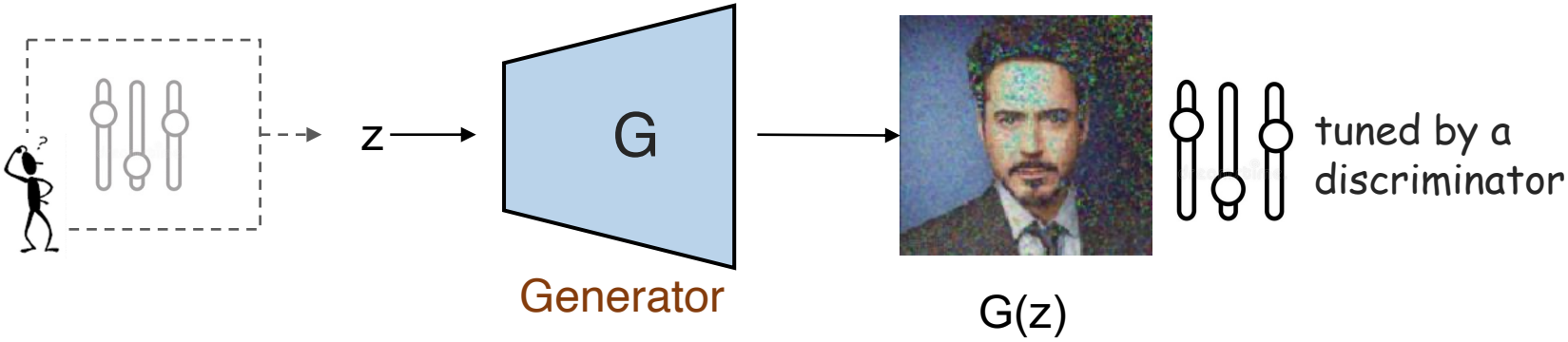
Results



CycleGAN

Results – Amini to Obama

GAN and Autoencoder



GAN: I can generate images that look real because I tune G(z) by a discriminator.

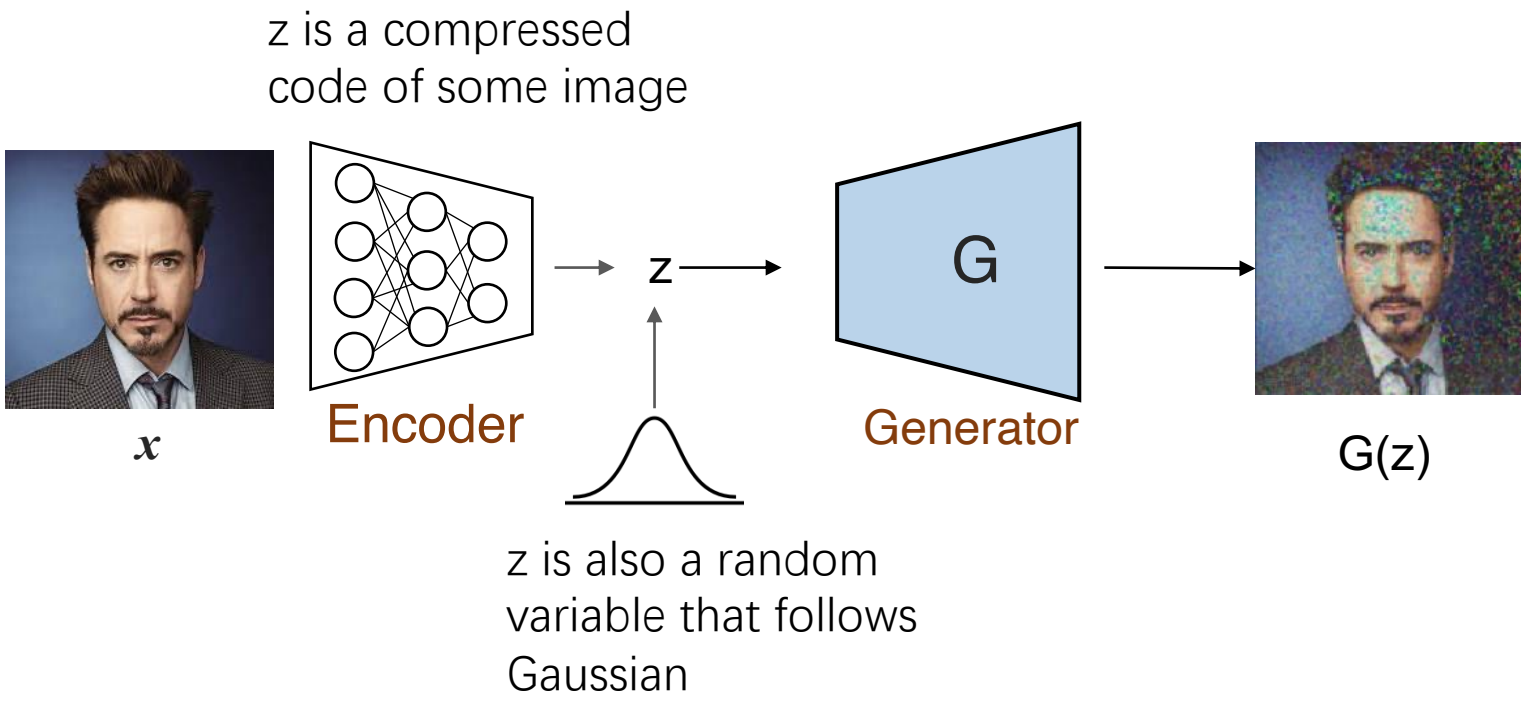
Can we control the random vector z instead? …

Autoencoder: how about letting z be a compressed code of some real image, then G(z) must look real.

But then z is intractable (purely random) and cannot be used for generation. …

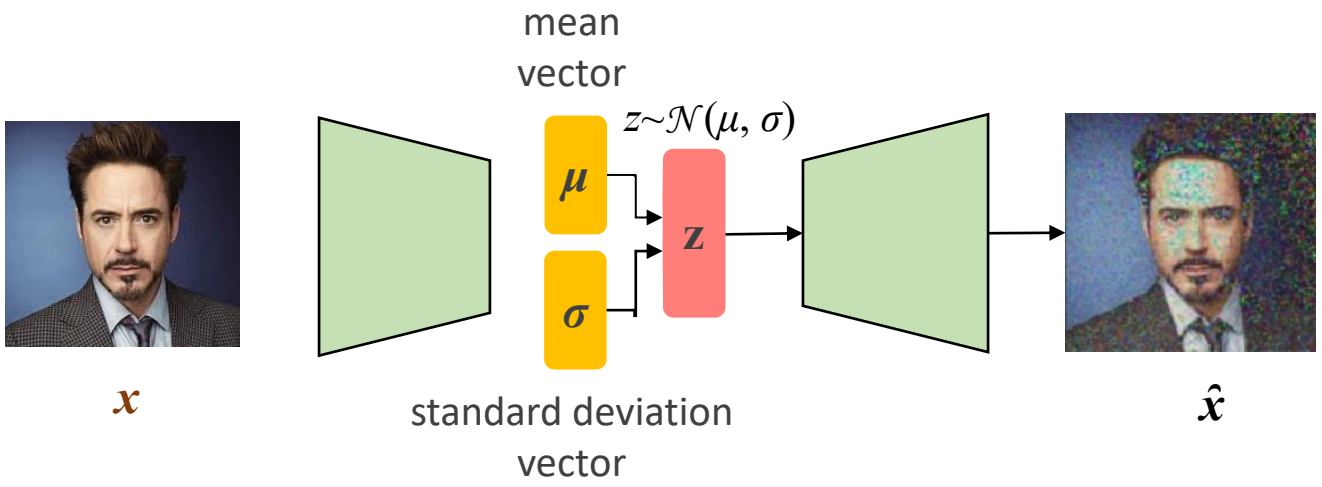
Variational Autoencoder (VAE)

Idea: can we let z be a latent code of real images and meanwhile force it to follow some distributions e.g., unit Gaussian ?



Variational Autoencoder

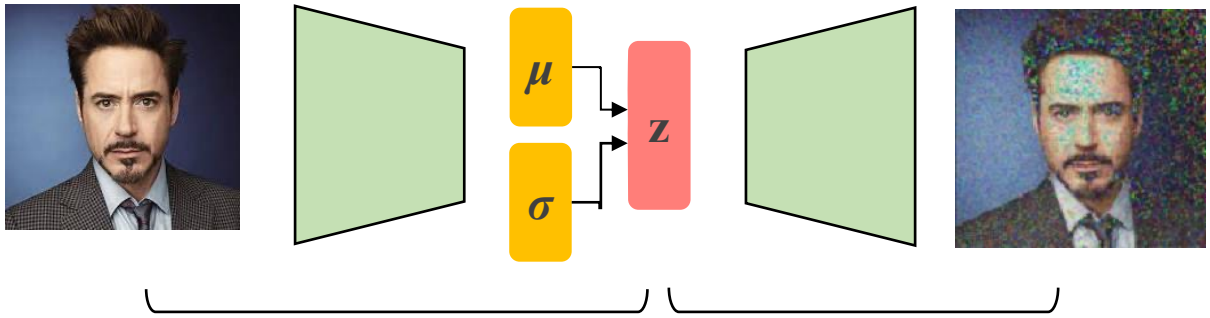
- An **autoencoder** where the latent code is sampled from a predicted Gaussian distribution.



Variational autoencoders are a probabilistic twist on autoencoders!
Sample from the mean and standard deviation to compute latent sample

Training VAE

Sampling is nondifferentiable
⇒ Reparameterization trick: $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\varepsilon}$



Encoder computes: $\mathbf{q}_{\phi}(\mathbf{z}|\mathbf{x})$ Decoder computes: $\mathbf{p}_{\theta}(\mathbf{x}|\mathbf{z})$

maximize the log likelihood

Inferred latent distribution

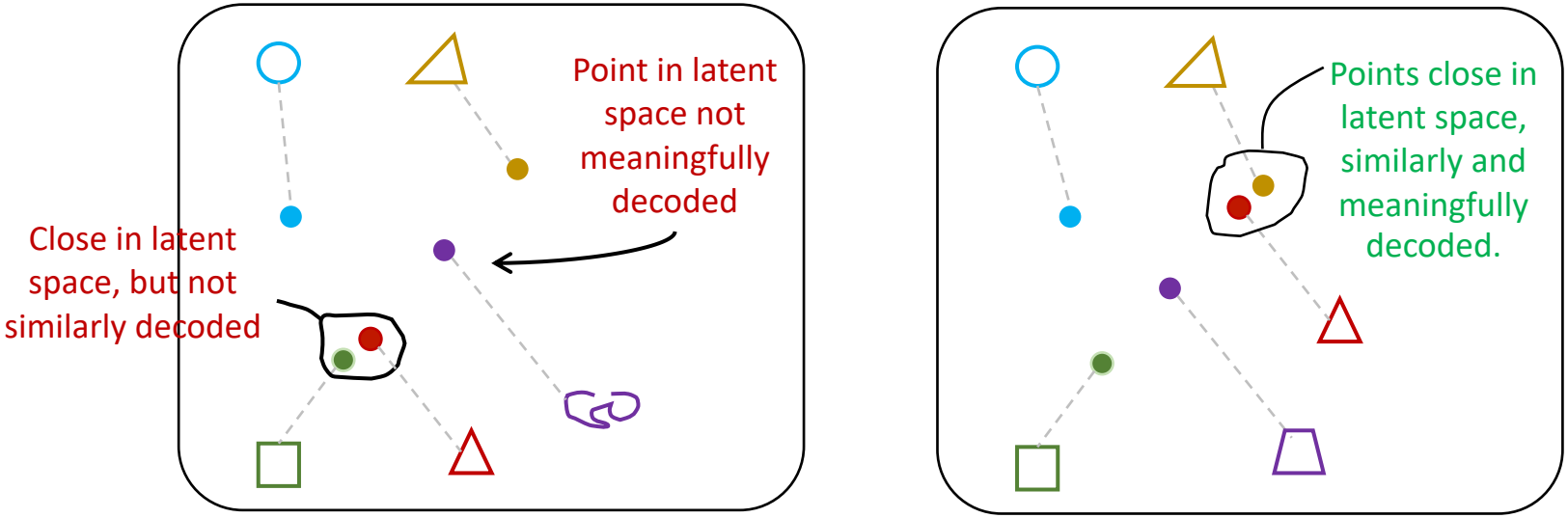
Prior dist. of z, commonly unit Gaussian

$$L(\theta, \phi|x) = \underbrace{-\mathbf{E}_{z\sim q_{\phi(z|x)}}[\log p_{\theta}(x|z)]}_{\text{Reconstruction Loss}} + \underbrace{\mathbf{KL}(q_{\phi}(z|x) \,||\, p(z))}_{\text{Regularization}}$$

Why Regularization?

What properties do we want to achieve from regularization?

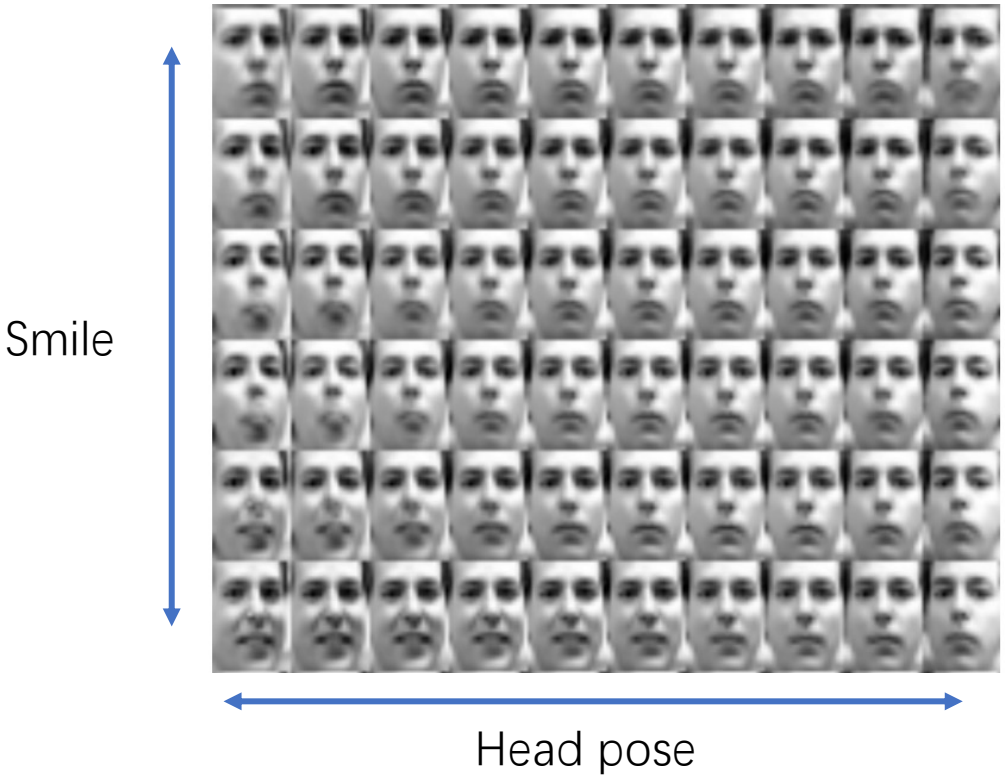
1. **Continuity**: points that are close in latent space → similar contents after decoding
2. **Completeness**: sampling from latent space → always "meaningful" content after decoding



without regularization (AE) **with regularization (VAE)**

VAEs: Latent Perturbation

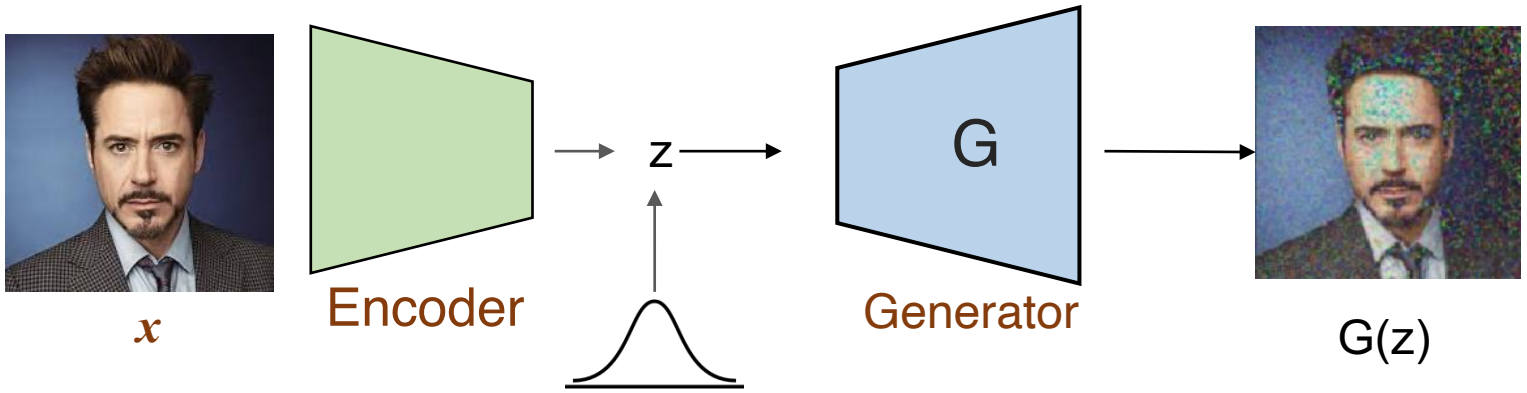
Slowly increase or decrease a single latent variable



Different dimensions of z encodes different interpretable latent features.

Limitations of VAEs (and also GANs)

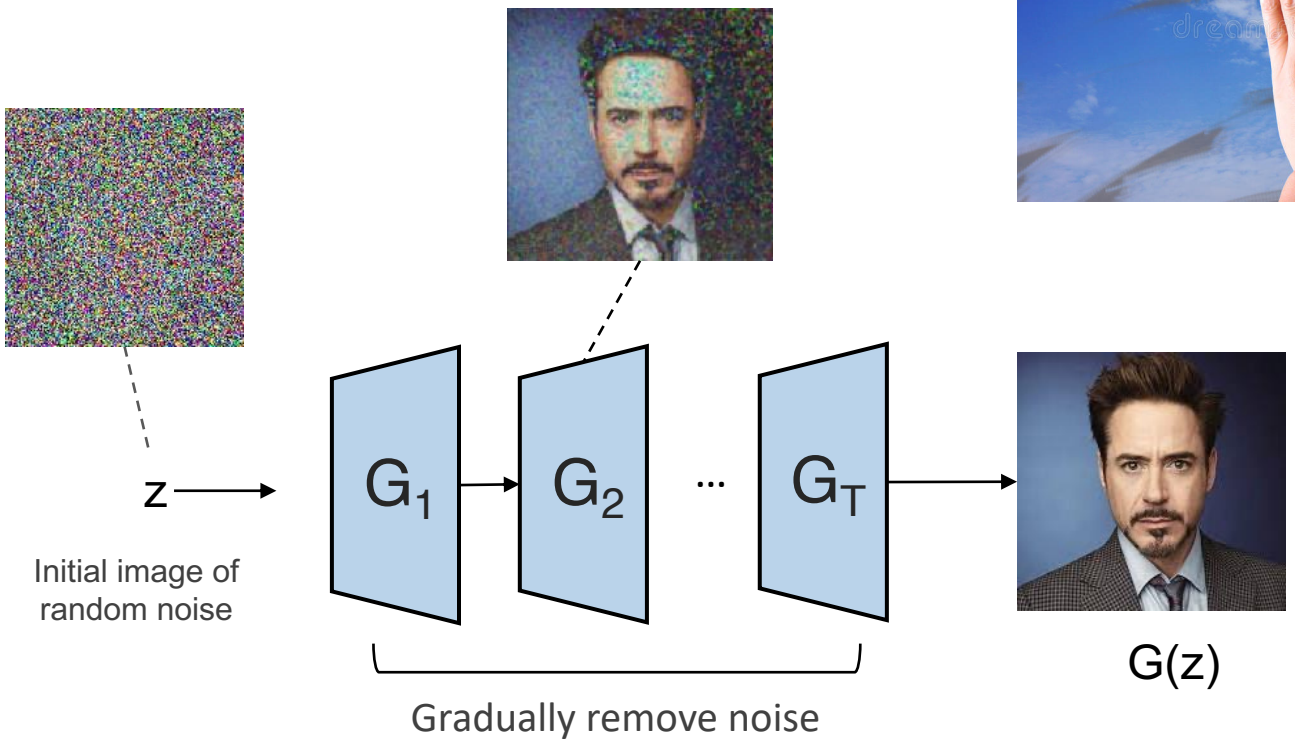
- The latent code z has a different (much smaller) size to the original image.
- The content is generated all at once.



Can we encode and generate images gradually from coarse to fine ?

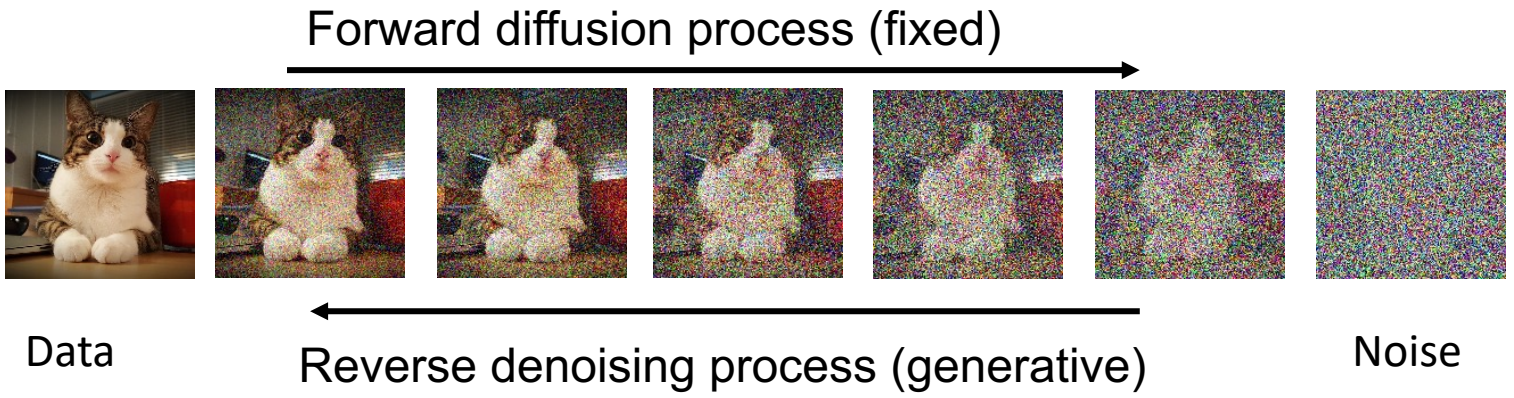
Denoising Diffusion Model

- Learning to generate by denoising



Denoising Diffusion Model

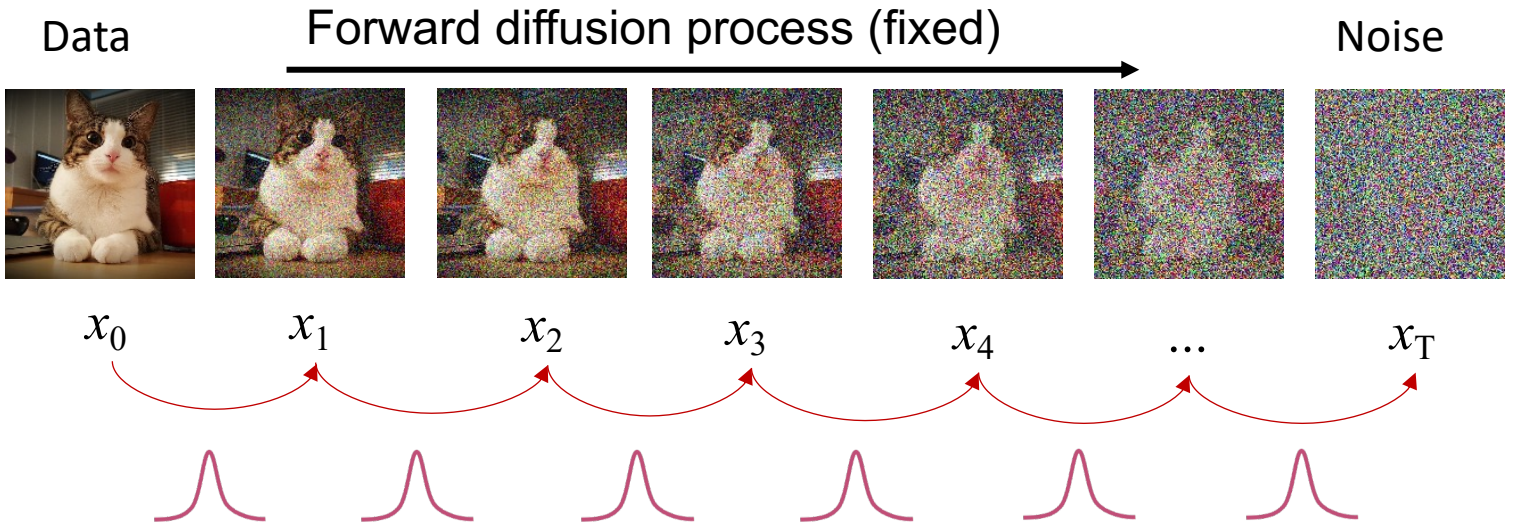
- Two processes:
  - **Forward diffusion process**: gradually adds noise to input (x→z)
  - **Reverse denoising process**: learns to generate data by denoising (z→x)



Denoising Diffusion Probabilistic Models. NeurIPS 2020
Denoising Diffusion Implicit Models. ICLR 2021
High-Resolution Image Synthesis with Latent Diffusion Models. CVPR 2022
Understanding Diffusion Models: A Unified Perspective

Forward Diffusion Process

T steps: each step (x_t) adds a fixed gaussian noise (β_t) to the previous step (x_{t-1}).

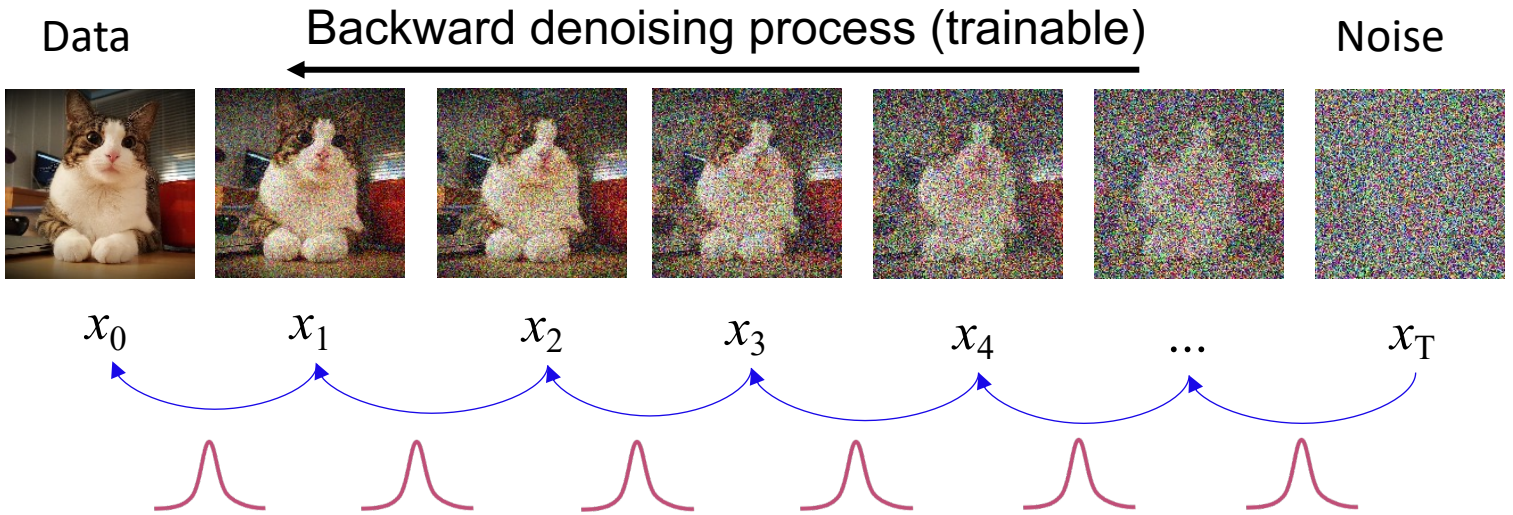


$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x_t}; \sqrt{1-\beta_t}\mathbf{x_{t-1}}, \beta_t\mathbf{I})$$

$$\text{Let } \bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s) \Longrightarrow \quad x_t \sim \; q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}))$$

Backward Denoising Process

T steps: each step (x_{t-1}) removes some noise from the previous step (x_t).



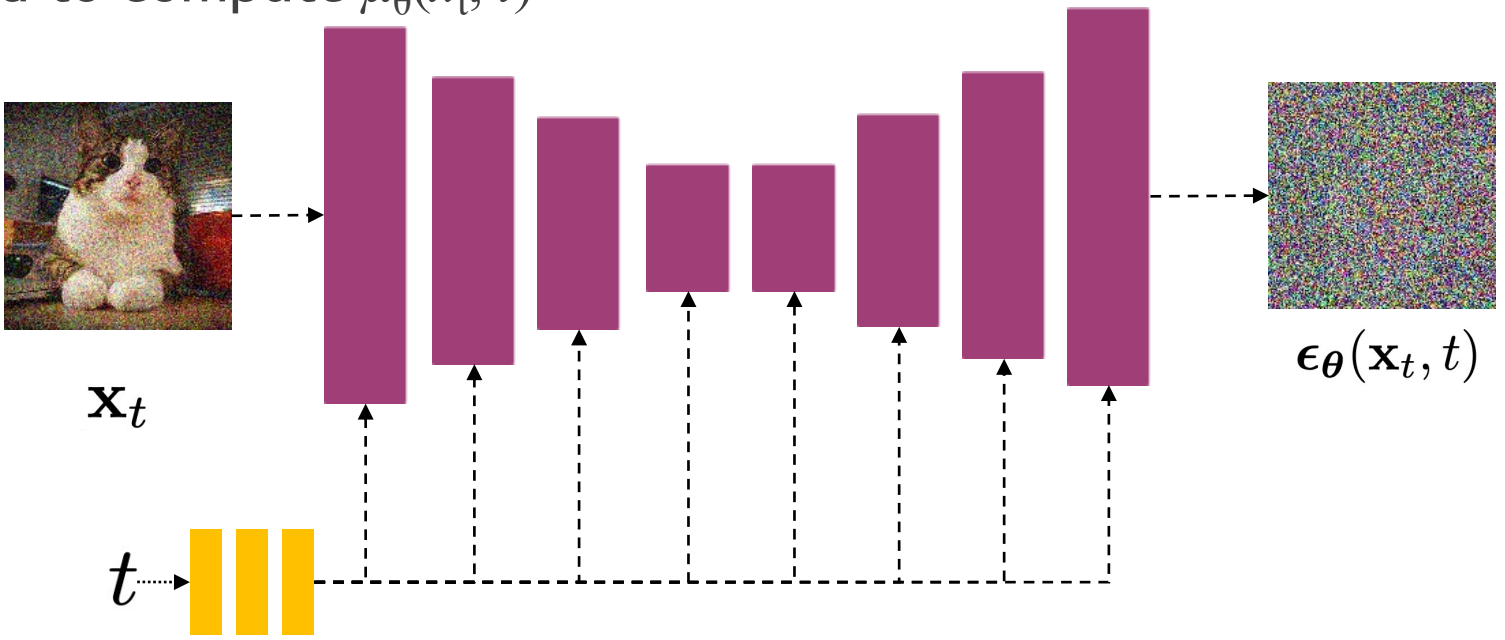
The distribution (μ and σ^2) of x_{t-1} can be estimated based on the previous image (x_t) using a neural network:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I}) \qquad p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$$

Trainable network (U-net, Denoising Autoencoder)

Architecture

Often use U-Net architectures with ResNet blocks and self-attention layers to represent $\varepsilon_\theta(x_t, t)$, which can further be used to compute $\mu_\theta(x_t, t)$



$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\epsilon_\theta(\mathbf{x}_t, t)\right)$$

Algorithm

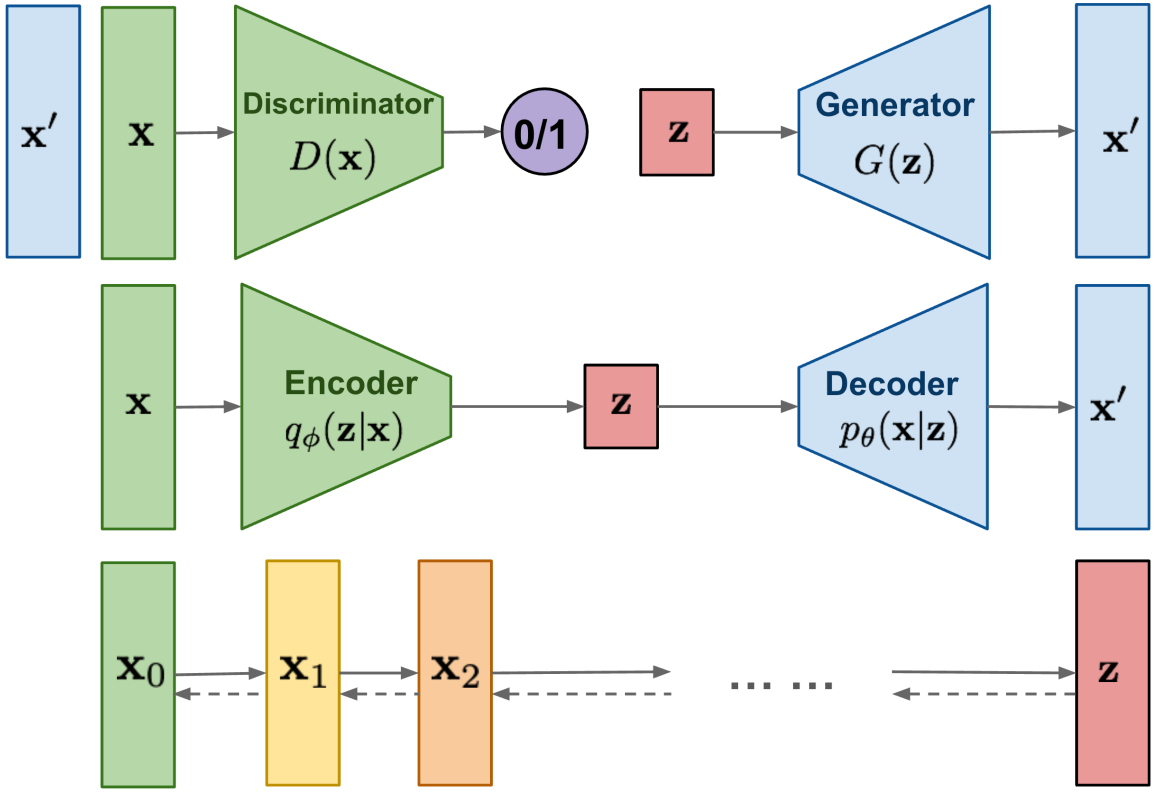
Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

GAN vs. VAE vs. Diffusion



What's Next?

Reinforcement Learning