Generative Models

HESAM HOSSEINI

SUMMER 2024

Are these faces is real?

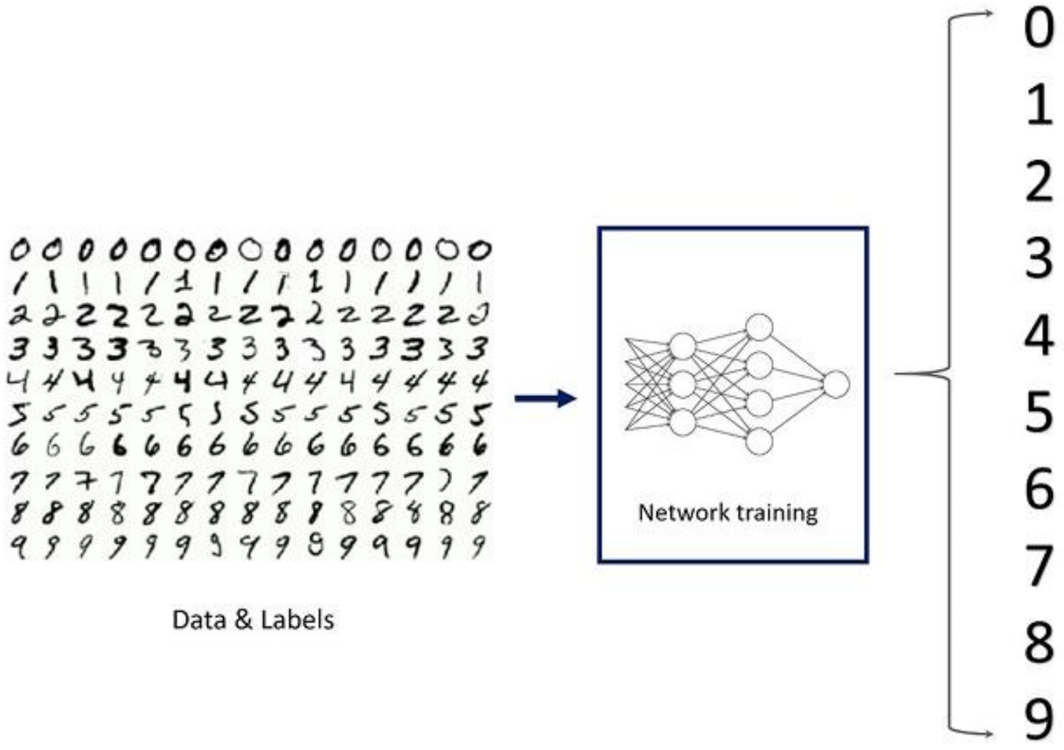
Review: Supervised Learning

Supervised Learning

Data: (x, y) where x is data, y is label

Goal: Learn a function to map x→ y

Examples: Classification, Object Detection, Semantic segmentation, Image captioning



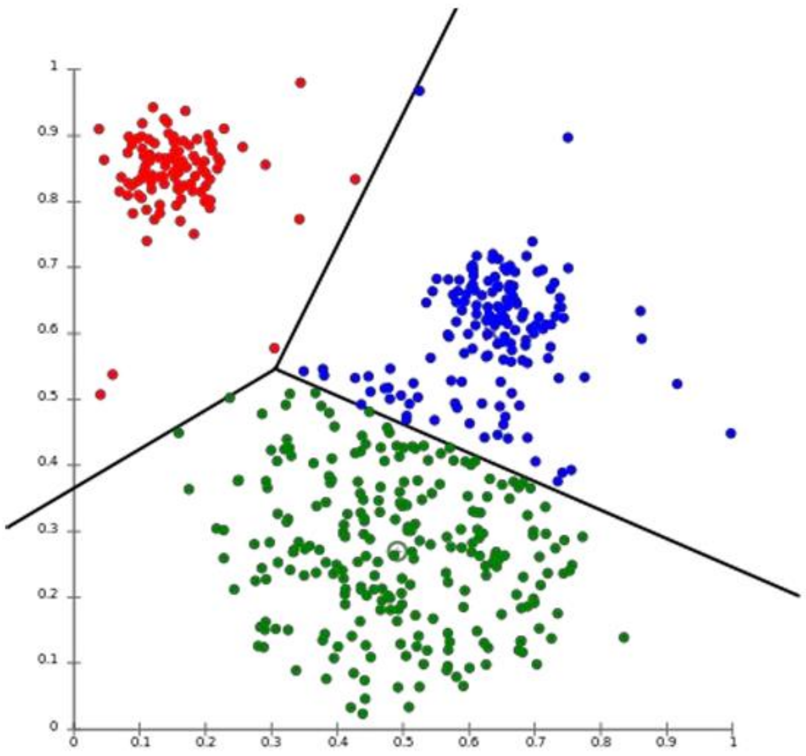
Review: Unsupervised Learning

Supervised Learning

Data: x, NO labels!!

Goal: : Learn some underlying hidden structure of the data

Examples: Clustering, Dimensionality reduction, Feature learning, Density estimation



K-means clustering

Generative Models

Given training data, generate new samples from same distribution

Training data $\sim p_{data}(x)$ Generated samples $\sim p_{model}(x)$

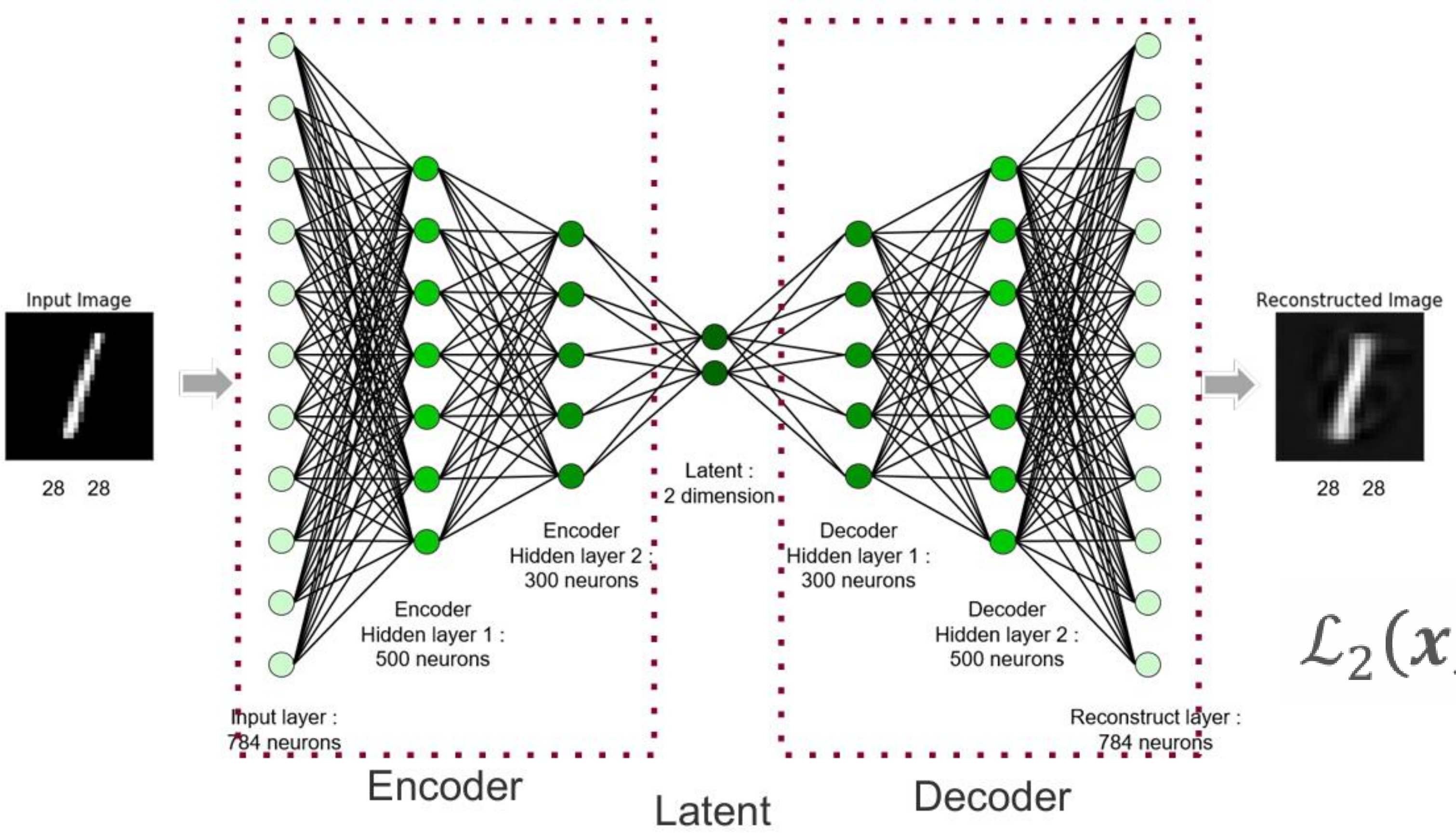
Want to: learn $p_{model}(x)$ similar to $p_{data}(x)$

Addresses density estimation which is a core problem in unsupervised learning

Two approach

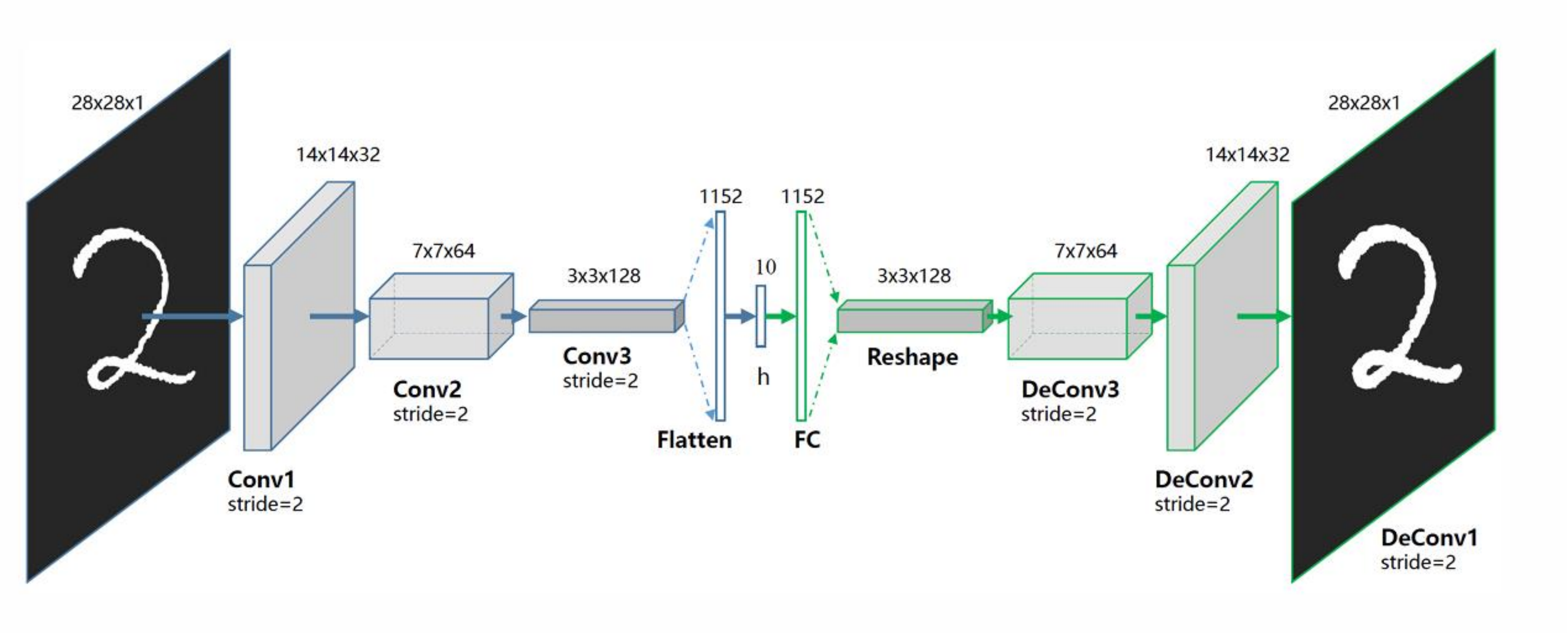
- Explicit density estimation: explicitly define and solve for $p_{model}(x)$
- Implicit density estimation: learn model that can sample from $p_{model}(x)$without explicitly defining it

Autoencoders



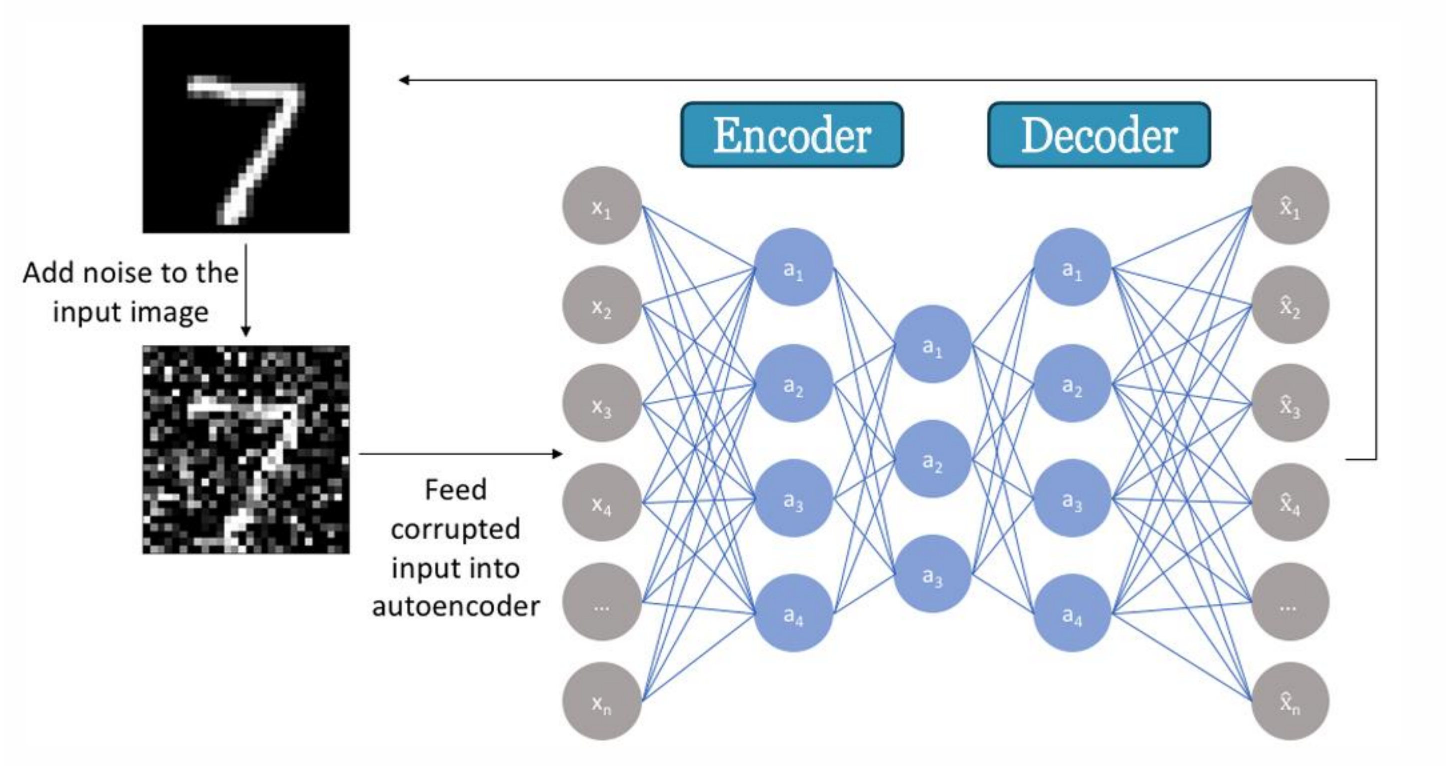
$$\mathcal{L}_2(\boldsymbol{x}, \boldsymbol{x}') = \sum_i \|\boldsymbol{x}_i - \boldsymbol{x}_i'\|_2^2$$

Deep Convolutional Autoencoder (CAE)



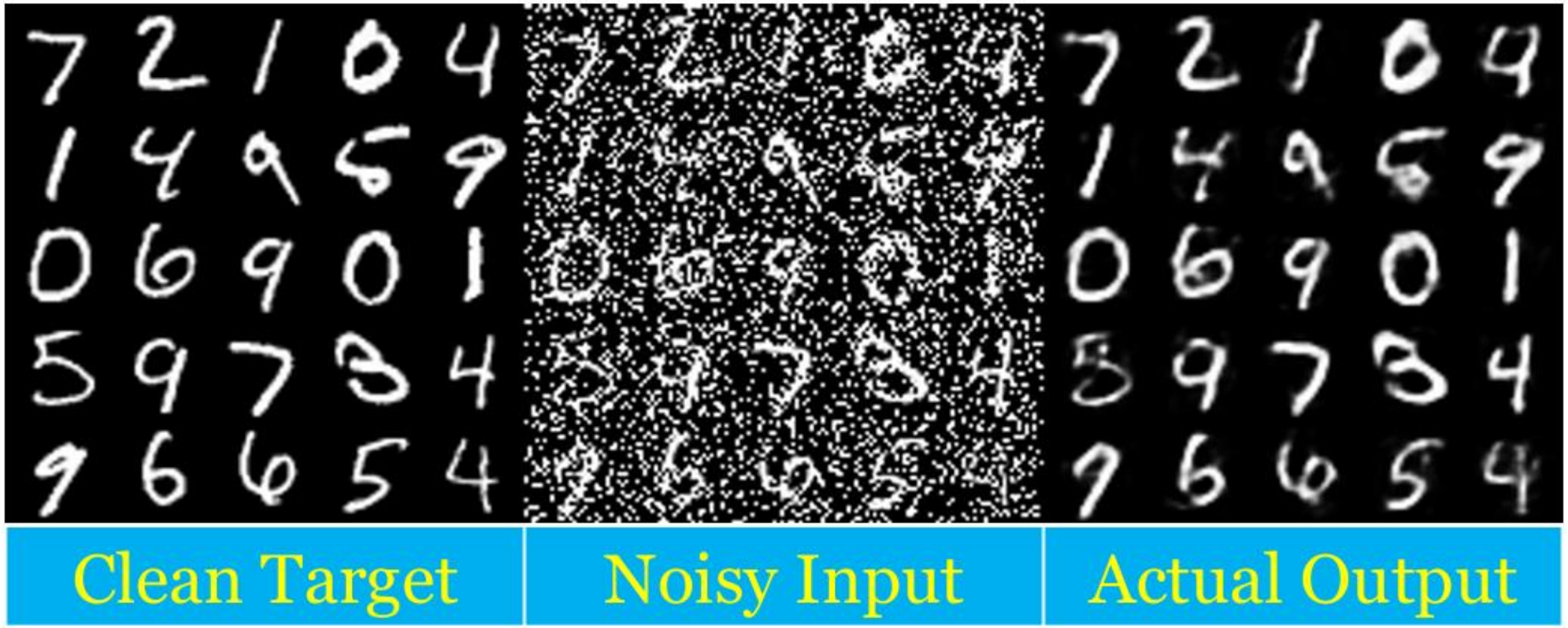
Denoising Autoencoder (DAE)

Learn robust feature to able reconstruct data from an input of corrupted data (input: noisy data, output: clean data)



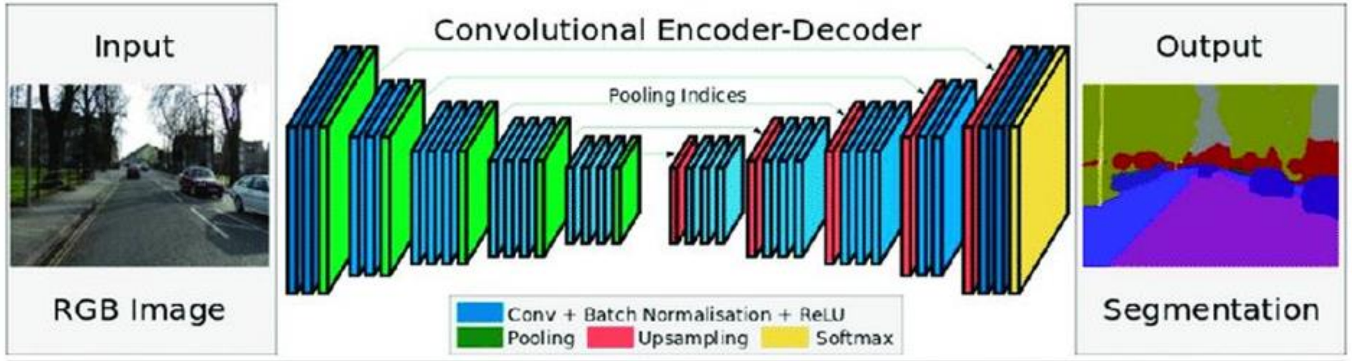
Denoising Autoencoder (DAE)

Example:

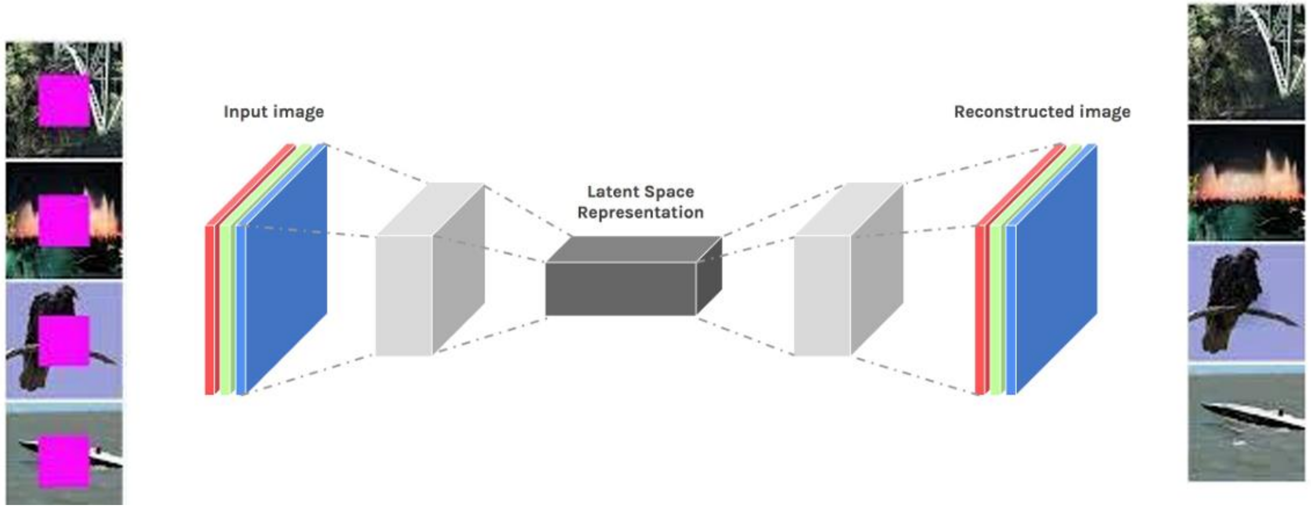


Autoencoder Application

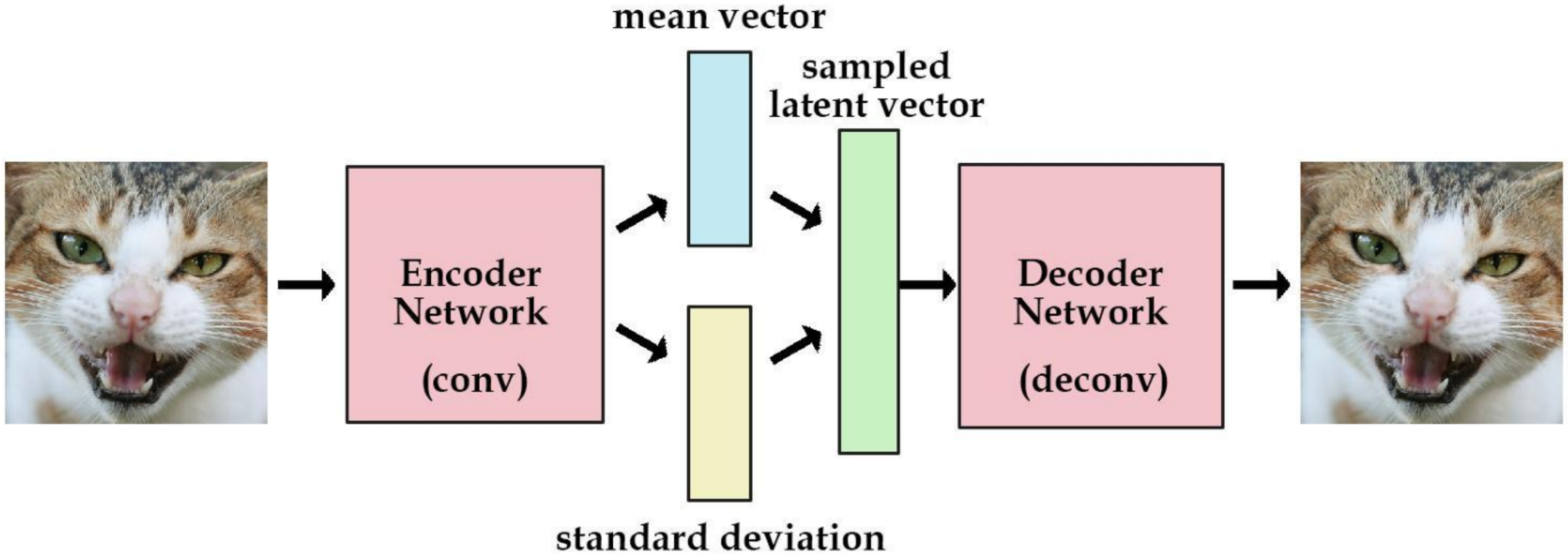
Semantic Segmentation



Neural Inpainting



Variational Autoencoders (VAE)



$$l_i(\theta, \phi) = -E_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] + KL(q_\theta(z|x_i)||p(z))$$

e.g. log-likelihood, $\|x - \hat{x}\|^2$ Reconstruction loss

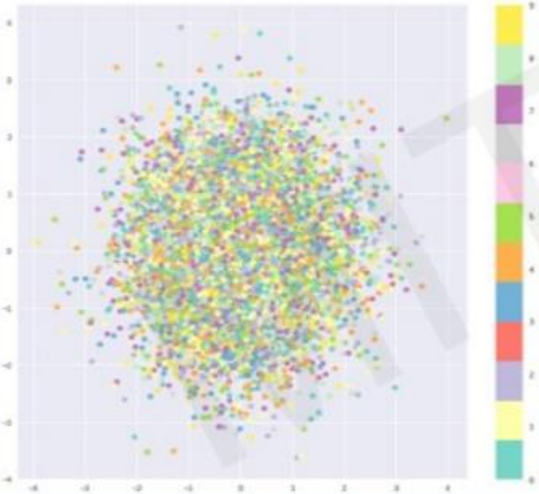
(regularization term) Stay close to normal(0,1)

Priors on the latent distribution

$$D\left(q_{\phi}(z|x) \parallel p(z)\right)$$

Inferred latent distribution

Fixed prior on latent distribution



Common choice of prior – Normal Gaussian:

$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalize the network when it tries to "cheat" by clustering points in specific regions (i.e., by memorizing the data)

Priors on the latent distribution

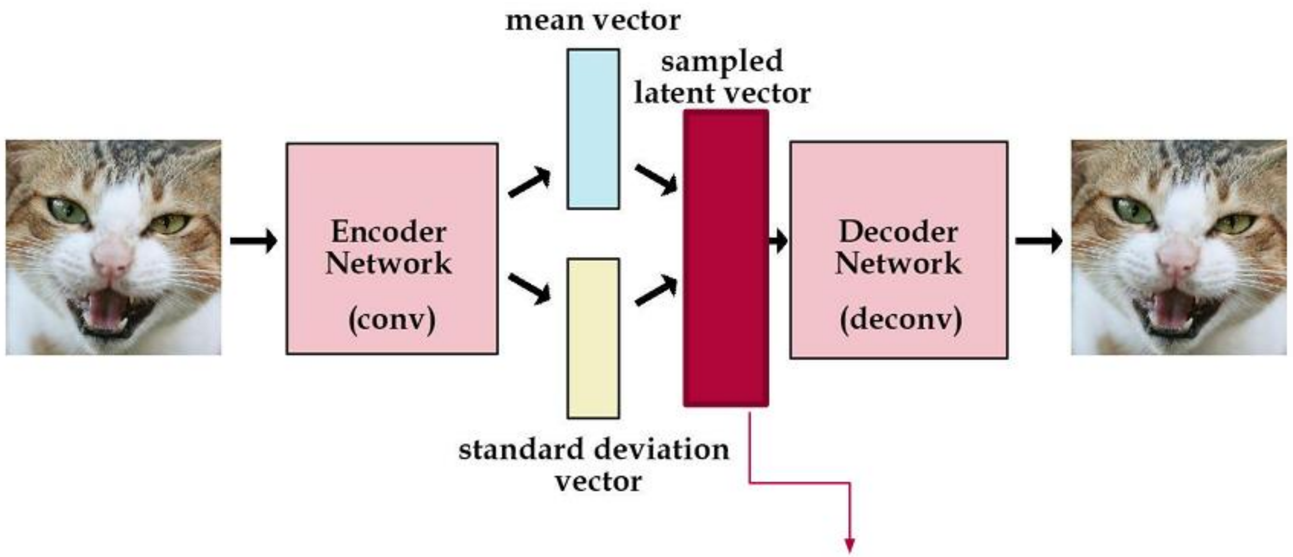
What properties do we want to achieve from regularization?

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

Regularized

VAE

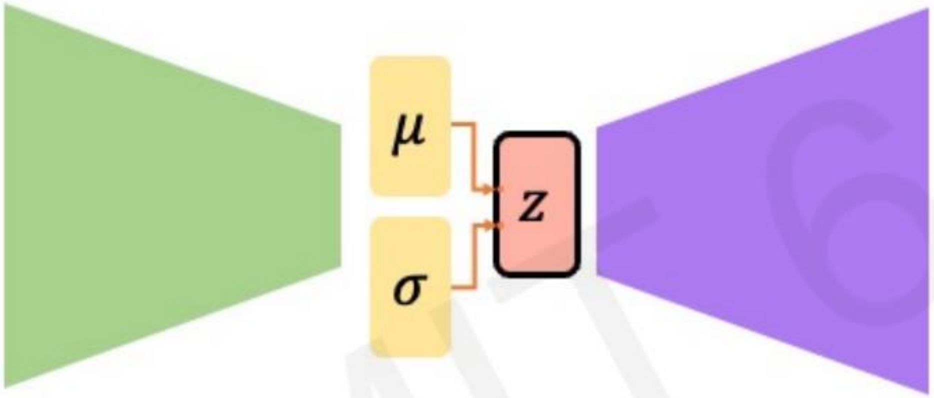
Problem: We cannot backpropagate gradients through sampling layers!



$Z = \mu + \sigma \Theta \varepsilon$

Where $\varepsilon \sim \text{normal}(0,1)$

Reparameterization trick



Key Idea:

$$\cancel{z \sim \mathcal{N}(\mu, \sigma^2)}$$

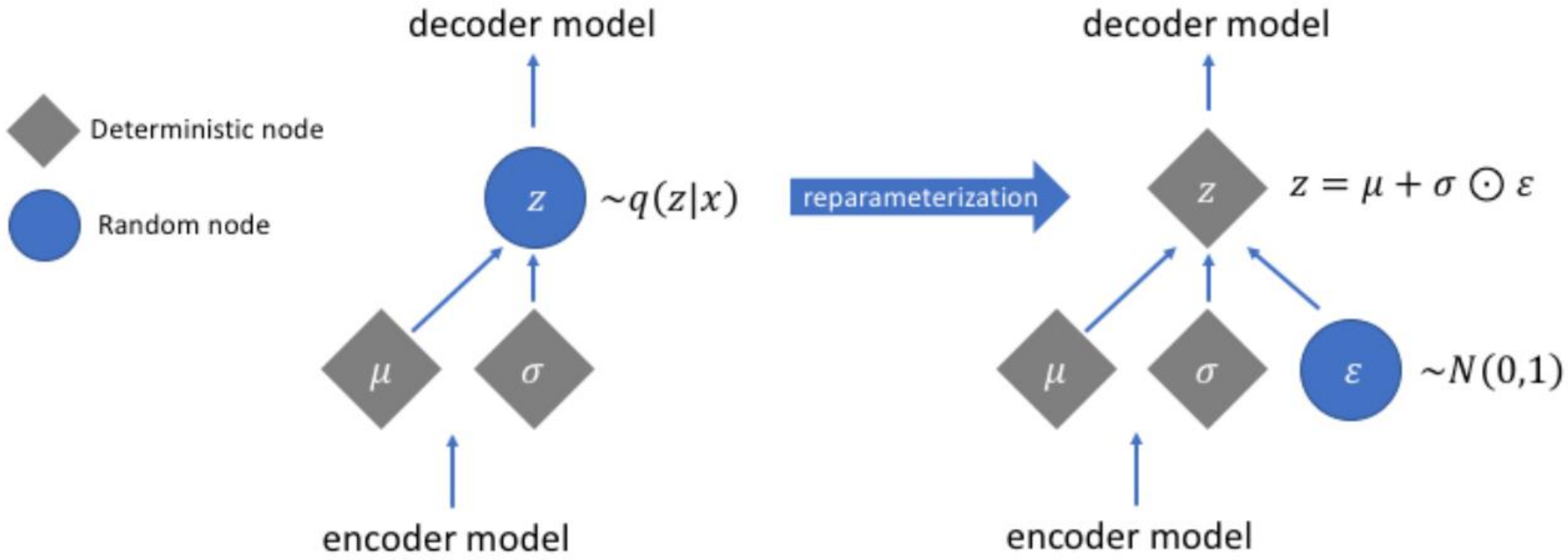
Consider the sampled latent vector $\boldsymbol{z}$ as a sum of

- a fixed $\boldsymbol{\mu}$ vector,
- and fixed $\boldsymbol{\sigma}$ vector, scaled by random constants drawn from the prior distribution

$$\Rightarrow z = \mu + \sigma \odot \varepsilon$$

where $\varepsilon \sim \mathcal{N}(0,1)$

Reparameterization trick



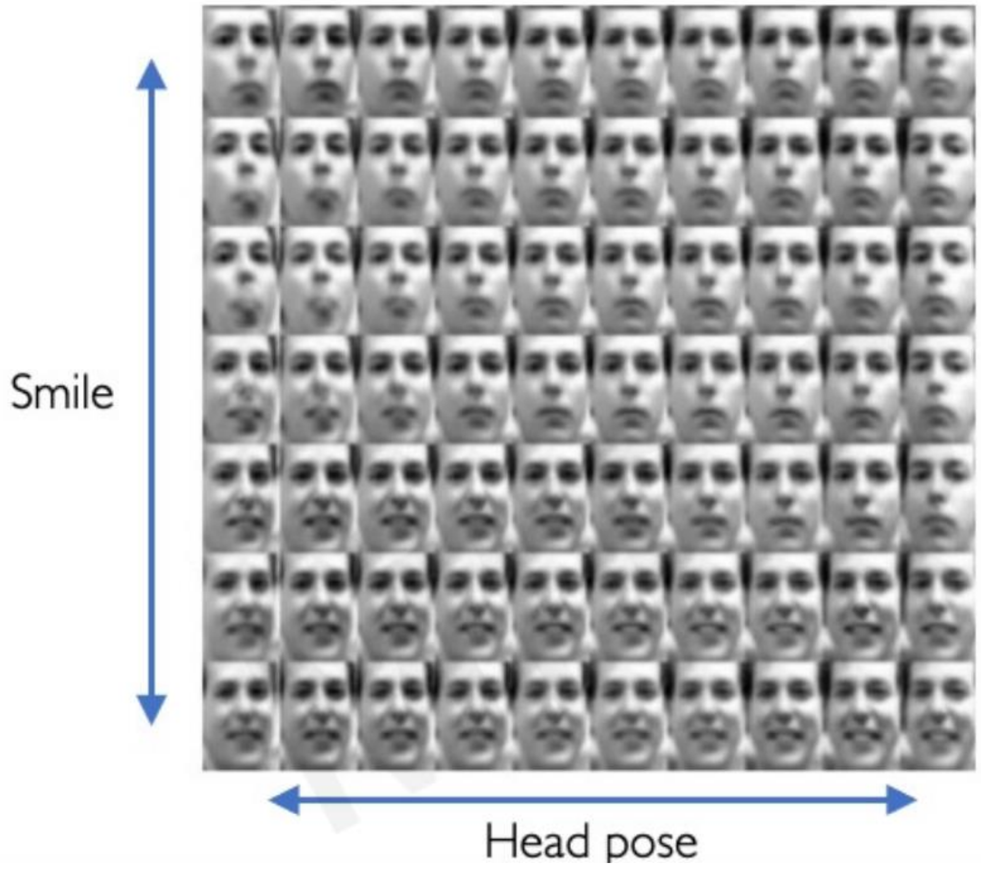
VAEs: Latent perturbation

Slowly increase or decrease a **single latent variable**
Keep all other variables fixed

Head pose

Different dimensions of z encodes **different interpretable latent features**

VAEs: Latent perturbation



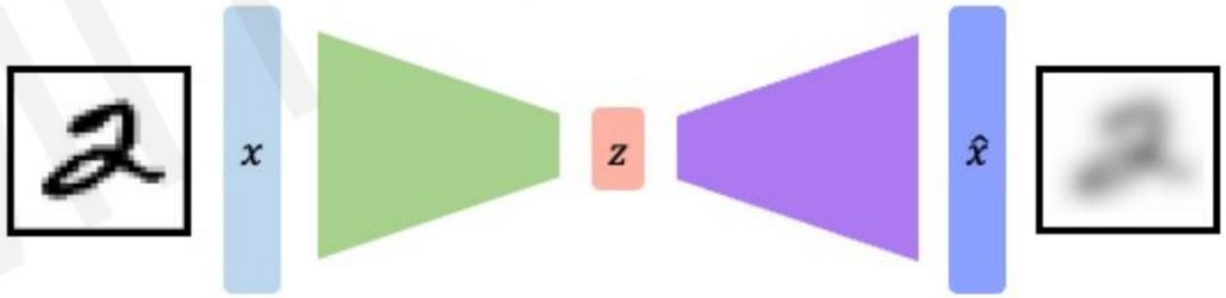
Ideally, we want latent variables that are uncorrelated with each other

Enforce diagonal prior on the latent variables to encourage independence

Disentanglement

VAE summary

1. Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)
3. Reparameterization trick to train end-to-end
4. Interpret hidden latent variables using perturbation
5. Generating new examples



Vector Quantized VAE (VQ-VAE)

Backbone of original Dall-E model

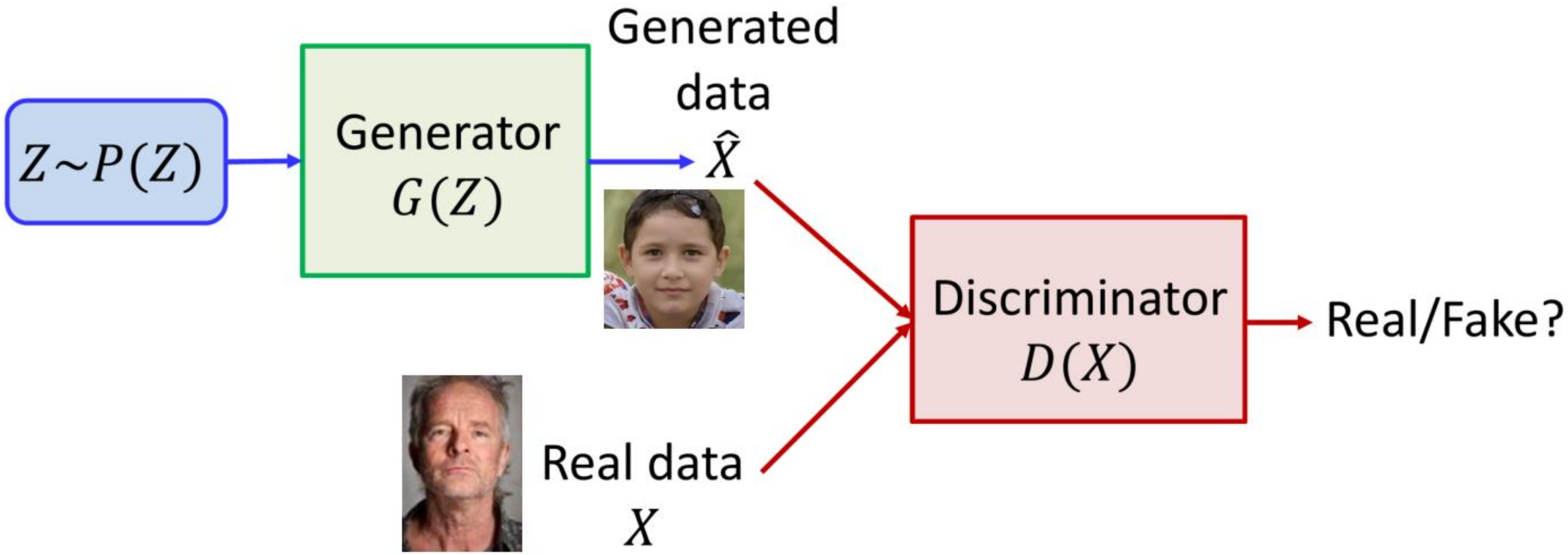
Generative Adversarial Network (GAN)

Generative models: Learn a generative model, which generate data similar to the training data .

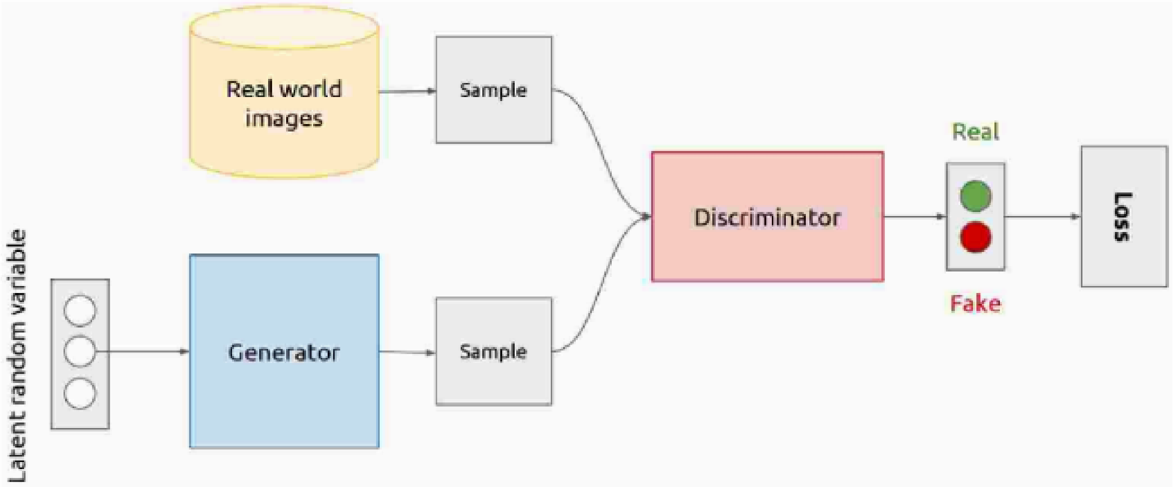
Adversarial training: Trained in an adversarial setting, GANS are made up of two competing networks (adversaries) that are trying beat each other. A «game» is being played between the two networks.

Network: Use Deep Neural Networks

GAN Block diagram



Train GAN

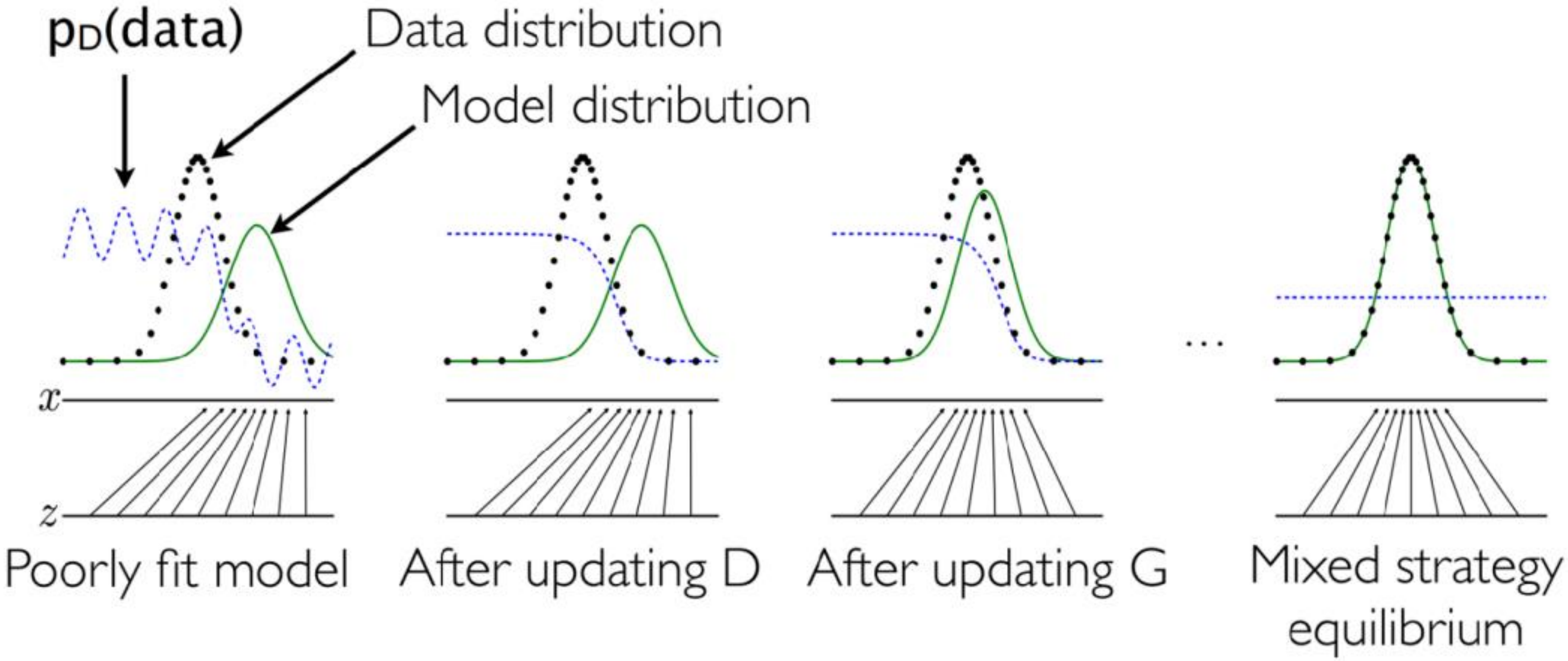


$$D_{ideal}(x) = \begin{cases} 1 & \text{if x is real} \\ 0 & \text{if x is fake} \end{cases}$$

Discriminator: maximize classification for a given generator

Generator: degrade classification of a given discriminator

GAN Learning Process



Min-Max Game Between Discriminator and Generator

Performance of discriminator

High values to **real** samples **Low** values to **fake** samples

$$V(G, D) = \mathbb{E}_{X \sim P_{data}}[\log(D(X))] + \mathbb{E}_{Z \sim P_z} \log(1 - D(G(Z))]$$

$$\min_{G_\theta} \max_{D_\phi} V(G_\theta, D_\theta)$$

Discriminator: Assign high value to real & low value to fake for a fixed Generator $\max_{D_\phi} V(G_\theta, D_\theta)$

Generator: Attempt to "fool" the discriminator D into assigning high values for fake.

$$\min_{G} \mathbb{E}_{Z \sim P_z} \log(1 - D(G(Z))]$$

Training alg

for number of training iterations **do**

for k steps **do**

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

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

Optimal Discriminator

Let's find optimal state:

$$\min_G \max_D L(D,G) = E_{x\sim p_r(x)}[logD(\boldsymbol{x})] + E_{z\sim p_z(z)}\left[log\left(1 - D\left(G(z)\right)\right)\right]$$

$$\min_G \max_D L(D,G) = E_{x\sim p_r(x)}[logD(\boldsymbol{x})] + E_{x\sim p_g(x)}[log(1 - D(\boldsymbol{x}))]$$

$$L(D,G) = \int_x \left(p_r(\boldsymbol{x})logD(\boldsymbol{x}) + p_g(\boldsymbol{x})log\left(1 - D(\boldsymbol{x})\right)\right)dx$$

Using Calculus of variation:

$$\frac{p_r(x)}{D(x)} - \frac{p_g(x)}{1 - D(x)} = 0 \Rightarrow D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)} \in [0,1]$$

Optimal Generator

- Optimal Discriminator is:

$$D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

- For optimal Generator we should have: $p_r(x) = p_g(x)$, then optimal discriminator becomes 0.5 (Nash equilibrium)

GAN Training Issues: Loss Saturation

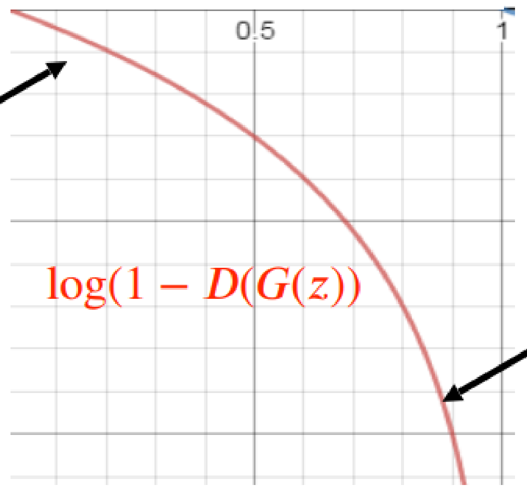
$$\min_{\theta} \mathbb{E}_{Z\sim P_z} \log(1 - D(G_\theta(Z)]$$

Minimize log-probability of generator samples labeled "fake"

Low discriminator score (low-quality samples)

High discriminator score (high-quality samples)

Want to learn from confidently rejected sample but gradients here are small



These samples already fool the discriminator so we don't need large gradients here

GAN Training Issues: Loss Saturation

$$\min_{\theta} \mathbb{E}_{Z \sim P_z} \log(1 - D(G_\theta(Z)]$$

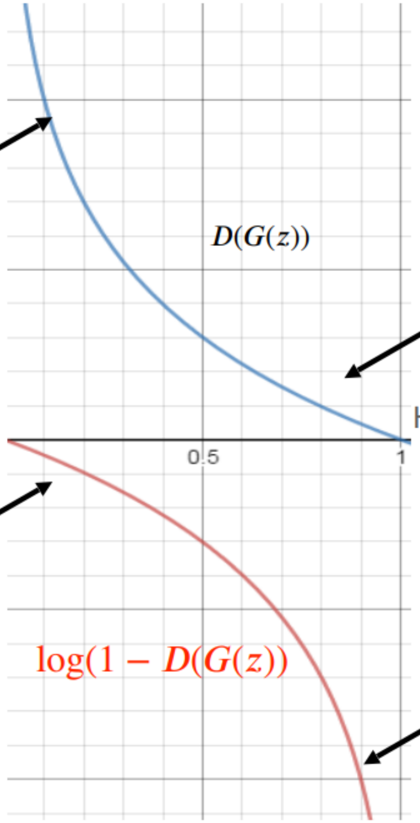
Minimize log-probability of generator samples labeled “fake”

$$\max_{\theta} \mathbb{E}_{Z \sim P_z} \log(D(G_\theta(Z)]$$

Maximize log-probability of generator samples labeled “real”

Large gradients for low-quality samples

Small gradients for high-quality samples



Low discriminator score (low-quality samples)

High discriminator score (high-quality samples)

Want to learn from confidently rejected sample but gradients here are small

These samples already fool the discriminator so we don’t need large gradients here

Maximizes the probability that the image is real.

NSGAN training algorithm

- Update discriminator:
  - Repeat for k steps:
    - Sample mini-batch of noise samples $z_1, \ldots, z_m$ and mini-batch of real samples $x_1, \ldots, x_m$
    - Update parameters of D by stochastic gradient ascent on

$$\frac{1}{m}\sum_{m}[\log D(x_m) + \log(1 - D(G(z_m)))]$$

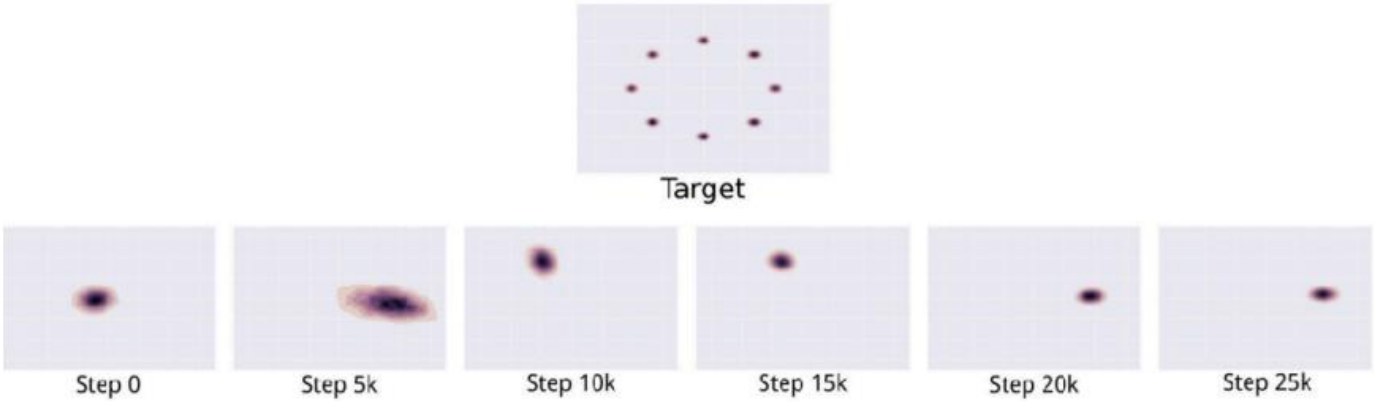
- Update generator:
  - Sample mini-batch of noise samples $z_1, \ldots, z_m$
  - Update parameters of G by stochastic gradient ascent on

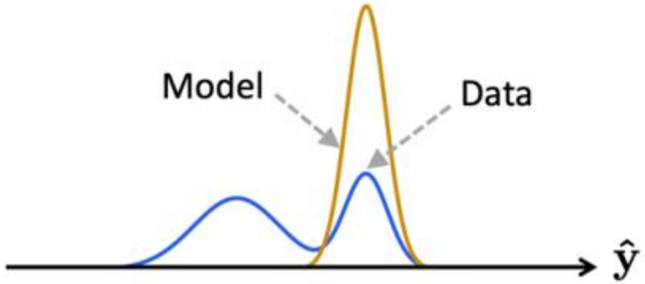
$$\frac{1}{m}\sum_{m}\log D(G(z_m))$$

- Repeat until happy with results

Mode Collapse

Generator ends up modeling only a small subset of the training dat Does not learn the true distribution





Problems with GAN

Stability Training

- Parameters can oscillate or diverge, generator loss does not correlate with sample quality
- Behavior very sensitive to hyperparameter selection

Mode Collapse:

- Generator ends up modeling only a small subset of the training dat

Vanishing Gradients:

- Typical discriminator is too strong. Thus gradient of the loss function drops down to close to zero and the learning becomes super slow or even jammed