CS480/680: Introduction to Machine Learning

Lecture 14: Generative Adversarial Networks

Hongyang Zhang

UNIVERSITY OF WATERLOO

July 21, 2025

Generative Models

- In generative modeling, we'd like to train a network that models a distribution:
  - For example, LLMs want to model the distribution of natural language.
  - We want to design a generative model to generate images.

Mathematical Model

- Given training data $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \sim p_{data}(\mathbf{x})$, the true data density
- Parameterize $p_{\boldsymbol{\theta}}(\mathbf{x})$, the data density estimated by model
- Estimate $\boldsymbol{\theta}$ by minimizing some "distance" between p_{data} (the unknown data density) and $p_{\boldsymbol{\theta}}$:

$$\min_{\boldsymbol{\theta}} \ \text{dist}(p_{data} \| p_{\boldsymbol{\theta}})$$

- After training, can generate new data $\mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x})$
- Need a training set from p_{data} and an explicit form of $p_{\boldsymbol{\theta}}$

How to sample a new data: Push-forward Maps

Theorem: Representation through push-forward

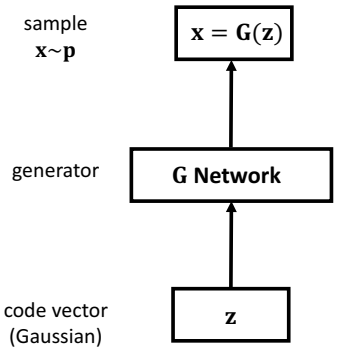
Let r be any continuous distribution on $\mathbb{R}^h$. For any distribution p on $\mathbb{R}^d$, there exist push-forward maps $\mathbf{G} : \mathbb{R}^h \to \mathbb{R}^d$ such that

$$\mathbf{z} \sim r \implies \mathbf{G}(\mathbf{z}) \sim p$$

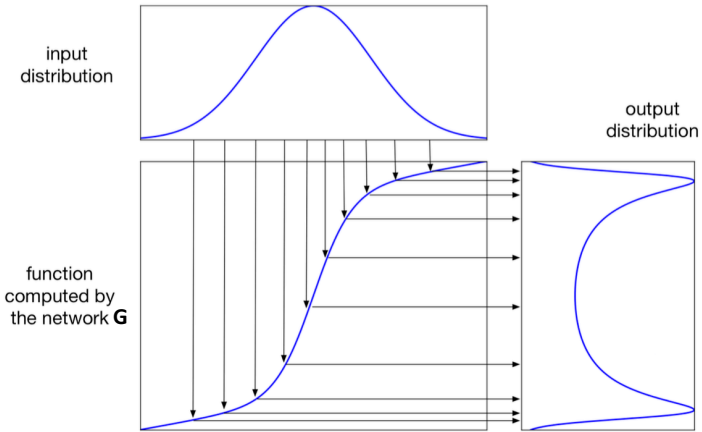
- W.l.o.g. we may simply take r to be standard Gaussian noise

Generating Samples

- Start by sampling the code vector z from a simple distribution (e.g., Gaussian)
- GAN computes a differentiable function $\mathbf{G}$ mapping $\mathbf{z}$ to an $\mathbf{x}$ in data space
  - $\mathbf{G}$ maps one distribution to another

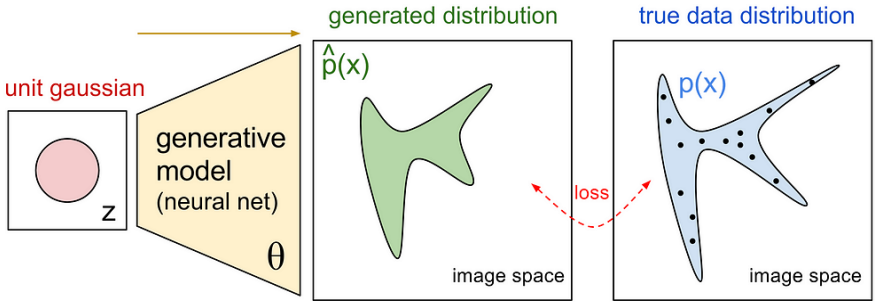


Generating Samples—A 1-dimensional example



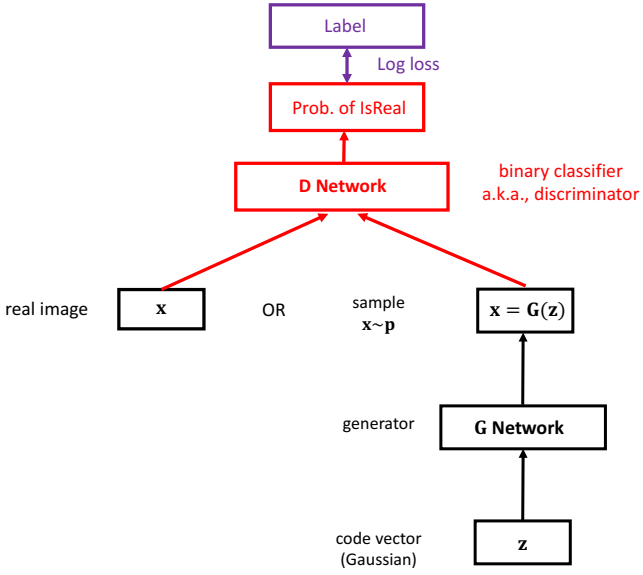
- G maps a sample from one distribution to a sample from another distribution
- The remaining question is how we can learn the G network

How to learn G network?

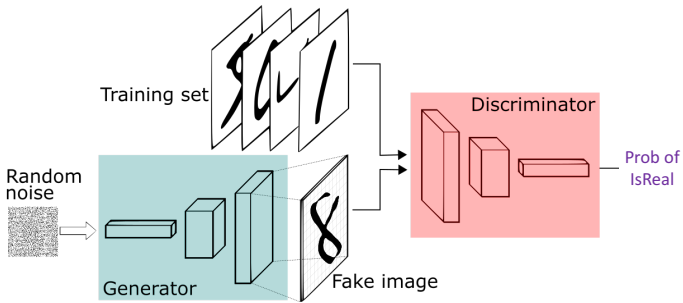


- How can we define the loss to distinguish two distributions?
  - Use a discriminator/binary classifier!

Generative Adversarial Networks



Generative Adversarial Networks



- Zero-sum game between the two parties:
  - Discriminator's goal: distinguish real images from fake images
  - Generator's goal: generate images that look like real one to confuse discriminator

Recall how we derive Log Loss — Page 2, Lecture 4

- Let $\mathcal{Y} = \{0, 1\}$; Let's directly learn confidence $D(\mathbf{x}) := \Pr(\mathsf{Y} = 1 | \mathsf{X} = \mathbf{x})$
- Given $(\mathbf{x}_i, \mathsf{y}_i), i = 1, \ldots, n$, assume independence:

$$\Pr(\mathsf{Y}_1 = \mathsf{y}_1, \ldots, \mathsf{Y}_n = \mathsf{y}_n | \mathsf{X}_1 = \mathbf{x}_1, \ldots, \mathsf{X}_n = \mathbf{x}_n) = \prod_{i=1}^{n} \Pr(\mathsf{Y}_i = \mathsf{y}_i | \mathsf{X}_i = \mathbf{x}_i)$$

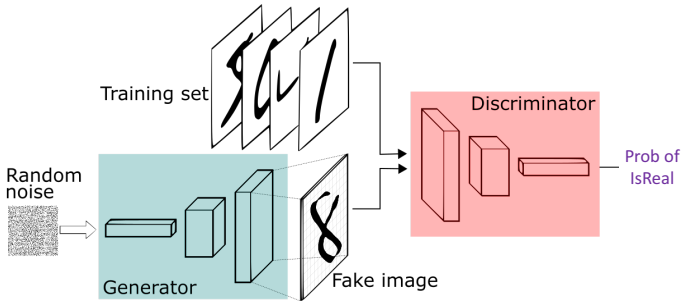
$$\overset{\mathcal{Y}=\{0,1\}}{=} \prod_{i=1}^{n} [D(\mathbf{x})]^{\mathsf{y}_i} [1 - D(\mathbf{x})]^{1-\mathsf{y}_i}$$

- Maximizing the likelihood: $\max_D \prod_{i=1}^{n} [D(\mathbf{x})]^{\mathsf{y}_i} [1 - D(\mathbf{x})]^{1-\mathsf{y}_i}$
  or minimizing the minus log (log loss):

$$\min_D \frac{1}{n} \sum_{i=1}^{n} [-\mathsf{y}_i \log D(\mathbf{x}) - (1 - \mathsf{y}_i) \log(1 - D(\mathbf{x}))]$$

$$= \min_D \mathbb{E}_{(\mathbf{x},\mathsf{y})} [-\mathsf{y} \log D(\mathbf{x}) - (1 - \mathsf{y}) \log(1 - D(\mathbf{x}))]$$

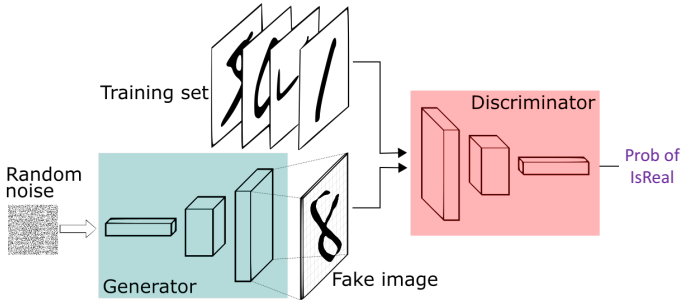
Discriminator's Goal



- For a fixed generator G, minimize a log loss over D (output prob. of IsReal):
  - If $\mathbf{x}$ is real, minimize $-\log D(\mathbf{x})$; if $\mathbf{x}$ is fake, minimize $-\log(1 - D(\mathbf{x}))$
  - Assume that $\mathbf{x}$ being from real/fake distribution is with equal chance:

$$\min_D \underbrace{-\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})]}_{\mathbf{x} \text{ is real with one half prob}} \underbrace{-\frac{1}{2}\mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,\mathbf{I})}[\log(1 - D(G(\mathbf{z})))]}_{\mathbf{x} \text{ is fake with another half prob}}$$

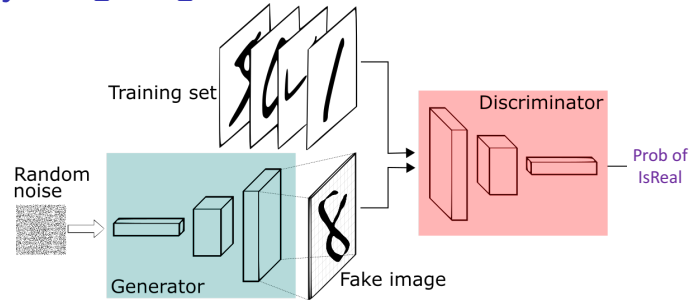
Generator's Goal



- For a fixed discriminator D, maximize a log loss (due to zero-sum game) over G:
  - If $\mathbf{x}$ is real, maximize $-\log D(\mathbf{x})$; if $\mathbf{x}$ is fake, maximize $-\log(1 - D(\mathbf{x}))$
  - Assume that $\mathbf{x}$ being from real/fake distribution is with equal chance:

$$\max_G -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] - \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,\mathbf{I})}[\log(1 - D(G(\mathbf{z})))]$$

Putting Everything Together



- Learn G and D simultaneously by a minimax problem:

$$\max_G \min_D -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] - \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,\mathbf{I})}[\log(1 - D(G(\mathbf{z})))]$$

- Replace expectation with empirical expectation:

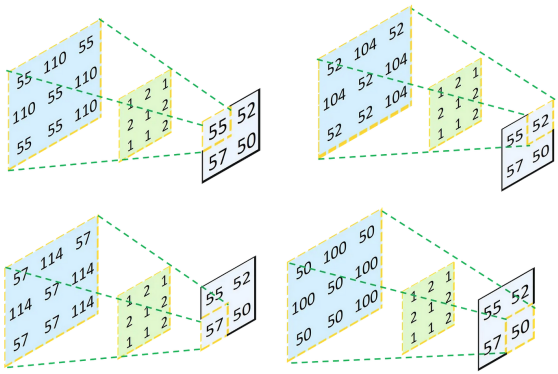
$$\min_G \max_D \hat{\mathbb{E}}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] + \hat{\mathbb{E}}_{\mathbf{z}\sim\mathcal{N}(0,\mathbf{I})}[\log(1 - D(G(\mathbf{z})))]$$

Solver

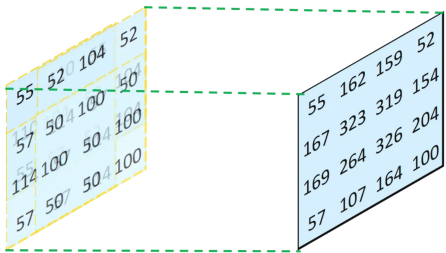
$$\min_G \max_D V(G,D) := \mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,\mathbf{I})}[\log(1 - D(G(\mathbf{z})))]$$

- Solved by alternative minimization-maximization:
  - G step: Fix D and update G by one-step gradient descent
  - D step: Fix G and update D by one-step gradient ascent
  - Repeat until the algorithm reaches an approximate equilibrium

Deconvolution (Transposed Convolution)



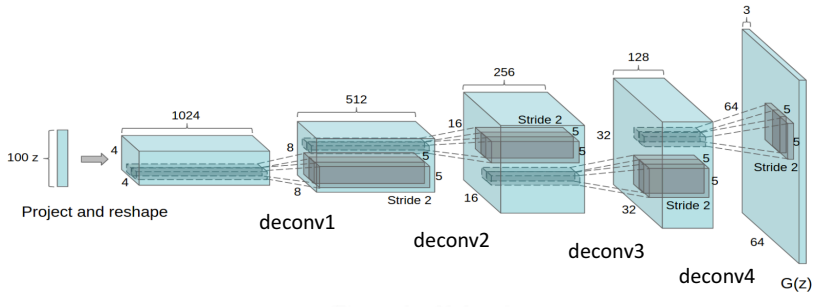
Multiply Each Element in the Input Layer by Each Value in the Kernel



Combine All Four Resulting Layers Together And Sum the Overlapped Values (Image by Author)

A small-size (e.g., 2×2) matrix $\rightarrow$ A large-size (e.g., 4×4) matrix

Network Architecture



Why does GAN work?

Proposition 1: solution of D^*

Let $p_g(\mathbf{x})$ be the density of $\mathbf{x}$ estimated by the generator G. For G fixed, the optimal discriminator D is $D^*_G(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x})+p_g(\mathbf{x})}$.

Proof of Proposition 1:

$$\begin{aligned} V(G,D) &:= \mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,\mathbf{I})}[\log(1-D(G(\mathbf{z})))] \\ &= \int_{\mathbf{x}} p_{data}(\mathbf{x}) \log D(\mathbf{x}) d\mathbf{x} + \int_{\mathbf{z}} p_{\mathbf{z}}(\mathbf{z}) \log(1-D(G(\mathbf{z}))) d\mathbf{z} \\ &= \int_{\mathbf{x}} \underbrace{p_{data}(\mathbf{x}) \log(D(\mathbf{x})) + p_g(\mathbf{x}) \log(1-D(\mathbf{x}))}_{f(D(\mathbf{x}))} d\mathbf{x}. \end{aligned}$$

For any fixed $\mathbf{x}$, taking derivative $= 0$, $D^*_G(\mathbf{x}) := \text{argmax}_{D(\mathbf{x})} f(D(\mathbf{x})) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x})+p_g(\mathbf{x})}$.

Why does GAN work?—Cont'

Theorem 1: solution of G^*

The global minimum of $\min_G \max_D V(G, D)$ is achieved if and only if $p_g = p_{data}$. The optimal objective value is $-\log 4$.

Therefore, the generator is able to learn the data distribution p_{data} exactly if we can solve $\min_G \max_D V(G, D)$ exactly.

Proof of Theorem 1

By proposition 1,

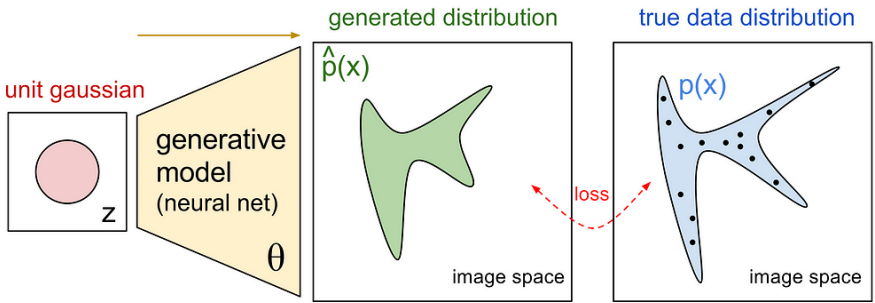
$$\begin{aligned} V(G, D_G^*) &= \mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D_G^*(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,\mathbf{I})}[\log(1 - D_G^*(G(\mathbf{z})))] \\ &= \mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D_G^*(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim p_g}[\log(1 - D_G^*(\mathbf{x}))] \\ &= \mathbb{E}_{\mathbf{x}\sim p_{data}}\left[\log \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}\right] + \mathbb{E}_{\mathbf{x}\sim p_g}\left[\log \frac{p_g(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}\right]. \end{aligned}$$

By definition, $KL(P||Q) = \mathbb{E}_{\mathbf{x}\sim P}\left[\log \frac{p(\mathbf{x})}{q(\mathbf{x})}\right]$. So

$$\begin{aligned} V(G, D_G^*) &= -\log 4 + KL\left(p_{data}\middle\|\frac{p_{data} + p_g}{2}\right) + KL\left(p_g\middle\|\frac{p_{data} + p_g}{2}\right) \\ &= -\log 4 + 2 \cdot JSD(p_{data}||p_g) \geq -\log 4, \end{aligned}$$

where JSD is the Jensen–Shannon divergence (distance between two distributions). The equality holds iff $p_{data} = p_g$.

Why does GAN work?



- Thus GAN is minimizing the Jensen–Shannon divergence between generated and real data distributions.


Questions
?
?
Answers
?