Lecture 11

Deep learning

1

Deep Generative Models

- Learns a probability distribution over multiple variables.
- Two main categories:
  - Models that allow probability density function to be evaluated **explicitly**.
    - Tractable density: PixelRNN, PixelCNN
    - Approximate density: Variational Autoencoder, Boltzmann Machines.
  - Models that support operations that **implicitly** require knowledge of the probability density function.
    - GANs

X_1 10 20 50
X_2 500 400 350
X_3 2 4 6

Goodfellow, Bengio, Courville 2016

2

Why Generative Models

- Many tasks intrinsically require realistic generation of samples from some distribution.
- Can be incorporated into reinforcement learning in several ways:
  - Generative models of time-series data can be used to simulate possible futures.
  - Might be used to enable learning in an imaginary environment, where mistaken actions do not cause real damage to the agent.
- Can be training with missing data (for example, datasets with missing labels) such as in semi-supervised learning (combines a small amount of labeled data with a large amount of unlabeled data).

Ian Goodfellow **NIPS 2016 Tutorial: Generative Adversarial Networks**

3

Why Generative Models

- Examples of some of these tasks that require the generation of good samples include:
  - Single image super-resolution: In this task, the goal is to take a low-resolution image and synthesize a high-resolution equivalent.
  - Tasks where the goal is to create art.
  - Image-to-image translation applications (convert aerial photos into maps, convert sketches to images)

Ian Goodfellow **NIPS 2016 Tutorial: Generative Adversarial Networks**

4

Generative Adversarial Networks

- **Goal:** we want to sample from a complex distribution that we don't know.

Generative Adversarial Networks

- Two-player game
  - **Generator:** tries to fool another player (discriminator) by generating 'fake' samples from the "unknown" distribution.
  - **Discriminator:** tries to distinguish between real and fake samples.

- Both models are trained jointly

Ian Goodfellow **NIPS 2016 Tutorial: Generative Adversarial Networks**

Generative Adversarial Networks

- Two-player game
  - **Generator:** tries to fool another player (discriminator) by generating 'fake' samples from the "unknown" distribution.
  - **Discriminator:** tries to distinguish between real and fake samples.

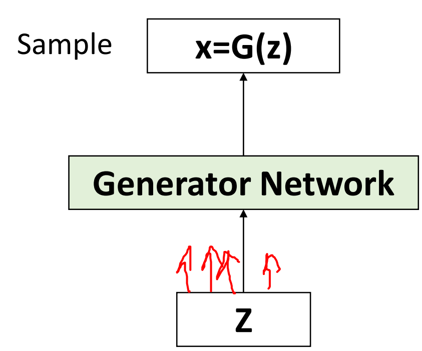
- Both models are trained jointly

Ian Goodfellow **NIPS 2016 Tutorial: Generative Adversarial Networks**

7

Generator

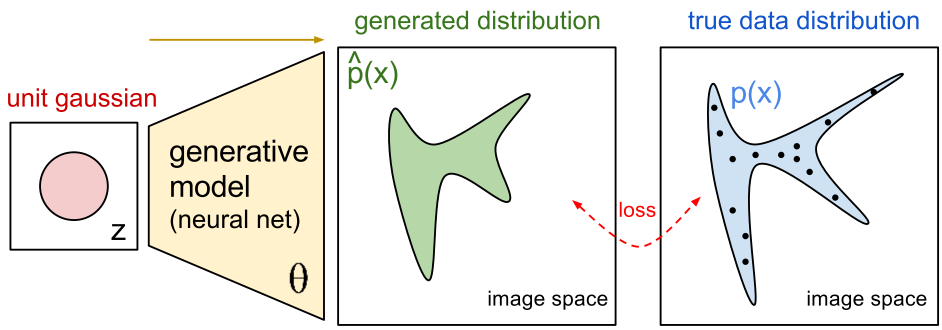
- Start by sampling the code vector **z** from a simple prior distribution (e.g. Gaussian)
- The generator network computes a differentiable function *G* mapping **z** to an **x** in data space.
- The generator is defined by a function *G* that takes $\boldsymbol{z}$ as input and uses $\boldsymbol{\theta}^{(G)}$ as parameters.
- Typically, a deep neural network is used to represent *G*.
- Cost function of the discriminator is $\boldsymbol{J}^{(G)}$



Ian Goodfellow **NIPS 2016 Tutorial: Generative Adversarial Networks**

8

GAN

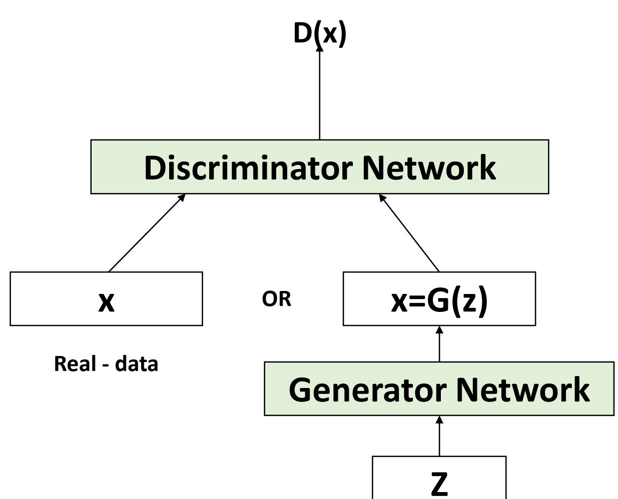


https://openai.com/blog/generative-models/

9

Discriminator

- Examines samples to determine whether they are real or fake.
- The discriminator learns using traditional supervised learning techniques, dividing inputs into two classes (real or fake). It produces 1 for real images and 0 for fake images.
- The discriminator is a function *D* that takes $\boldsymbol{x}$ as input and uses $\boldsymbol{\theta}^{(\boldsymbol{D})}$ as parameters.
- Cost function of the discriminator is $\boldsymbol{J}^{(\boldsymbol{D})}$



Ian Goodfellow **NIPS 2016 Tutorial: Generative Adversarial Networks**

10

The Training Process

- The training process consists of simultaneous SGD.
- On each step, two minibatches are sampled:
  - a minibatch of $\boldsymbol{x}$ values from the dataset
  - a minibatch of $\boldsymbol{z}$ values drawn from the model's prior distribution over latent variables.
- All of the different games designed for GANs so far use the same cost for the discriminator, $\boldsymbol{J^{(D)}}$. They differ only in terms of the cost used for the generator, $\boldsymbol{J^{(G)}}$.
- The cost function for discriminator is the standard cross-entropy that is used for a standard binary classifier. The difference it is that it is trained on two minibatches

$$J^{(D)}\left(\theta^{(D)}, \theta^{(G)}\right) = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}} \log D(x) - \frac{1}{2}\mathbb{E}_z \log(1 - D(G(z)))$$

Ian Goodfellow **NIPS 2016 Tutorial: Generative Adversarial Networks**

11

The Training Process

- Because the GAN framework can naturally be analyzed with the tools of game theory, we call GANs "adversarial"
- We can also think of them as cooperative, in the sense that the shares the information about the accuracy with the generator.
- From this point of view, the discriminator is more like a teacher instructing the generator in how to improve than an adversary.

Ian Goodfellow **NIPS 2016 Tutorial: Generative Adversarial Networks**

12

The Cost Function for the Generator

- A complete specification of the game requires that we specify a cost function also for the generator.
- The simplest version of the game is a **zero-sum game (minimax games)**, in which the sum of all player's costs is always zero.

$$J^{(G)} = -J^{(D)}$$

- Solution for minimax games involves minimization (MIN player tries to minimize MAX payoff) and maximization (MAX tries to maximize his payoff)

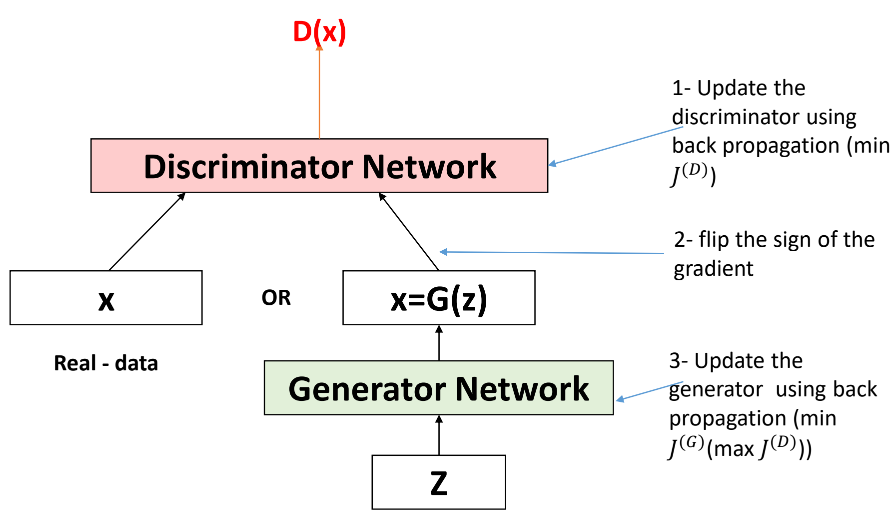
$$\theta^{(G)*} = \underset{\theta^{(G)}}{\operatorname{argmax}} \min_{\theta^{(D)}} J^{(D)} \;\; Or$$
$$\theta^{(G)*} = \underset{\theta^{(G)}}{\operatorname{argmin}} \max_{\theta^{(D)}} -J^{(D)}$$

-

Ian Goodfellow **NIPS 2016 Tutorial: Generative Adversarial Networks**

13

The Training Process



Ian Goodfellow **NIPS 2016 Tutorial: Generative Adversarial Networks**

14

Heuristic, non-saturating game

- The cost used for the generator in the minimax game is useful for theoretical analysis, but does not perform especially well in practice.
- In the minimax game, the discriminator minimizes a cross-entropy, but the generator maximizes the same cross-entropy.
- This is unfortunate for the generator, because when the discriminator successfully rejects generator samples with high confidence, the generator's gradient vanishes.

Ian Goodfellow **NIPS 2016 Tutorial: Generative Adversarial Networks**

15

Heuristic, non-saturating game

- To solve this problem, one approach is to continue to use a modified cost function for the generator.
- Instead of flipping the sign on the discriminator's cost to obtain a cost for the generator, we flip the target used to construct the cross-entropy cost.
- The cost for the generator then becomes:

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_z \log(D(G(z)))$$

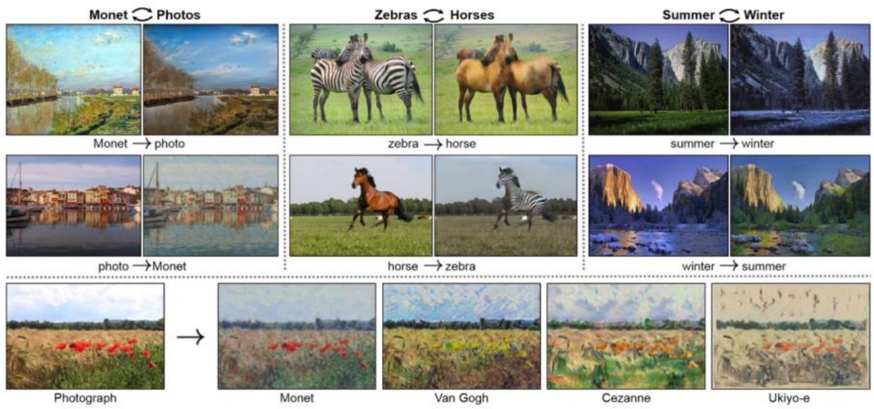
- In the minimax game, the generator minimizes the log-probability of the discriminator being correct.
- In this game, the generator maximizes the log-probability of the discriminator being mistaken.

Ian Goodfellow **NIPS 2016 Tutorial: Generative Adversarial Networks**

16

Cycle GANs

- Style transfer problem (Image-to-image translation): change the style of an image while preserving the content



https://junyanz.github.io/CycleGAN/

17

Cycle GANs

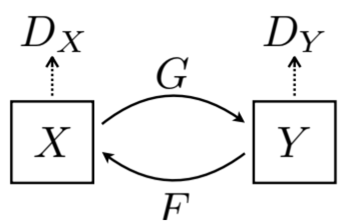
- The goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs.
- For many tasks, paired training data will not be available
- Cycle GANs: learning technique to translate an image from a source domain X to a target domain Y in the absence of paired examples
- Tries to learn a mapping function $G: X \rightarrow Y$ such that the distribution of images from G(X) is indistinguishable from the distribution Y using an adversarial loss.

https://junyanz.github.io/CycleGAN/

18

Cycle GANs

- Train two different generator nets to go from domain (style) X to domain (style) Y, and vice versa.
- The idea is to make sure the two generators are cycle-consistent: mapping from style 1 to style 2 and back again should give you almost the original image.



https://junyanz.github.io/CycleGAN/