Generative Adversarial Networks (GANs)

GANs

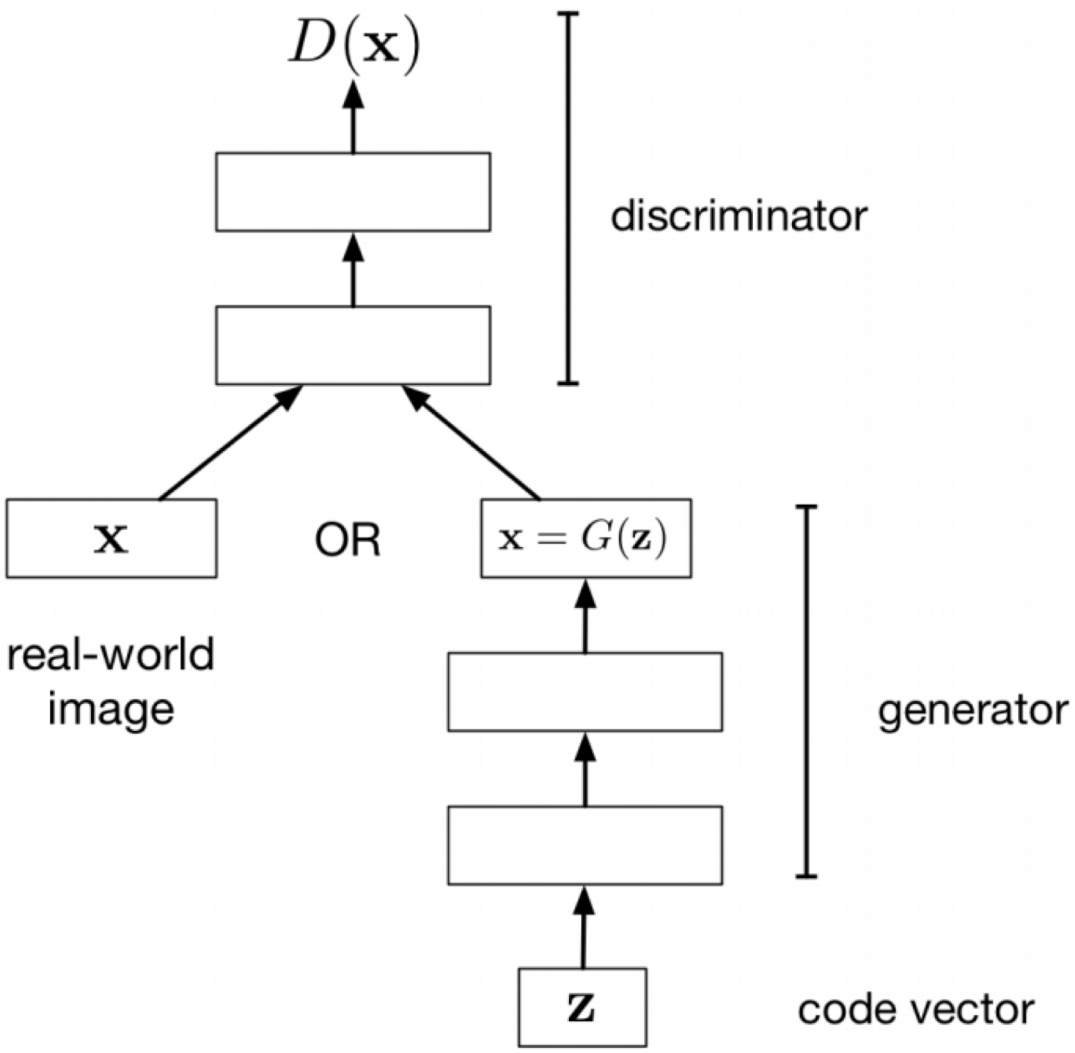
- Generative
  - Generates samples instead of discriminates
    - Models p(x) instead of p(y|x)
- Adversarial
  - Trained in an “Adversarial” setting (more on this to come)
- Networks
  - Using a neural network

GANs Overview

- Implicit data generation – create samples without any explicit data
- If we have some mechanism of evaluating the quality of implicitly generated data, then we can compute a gradient for training a network

- The big idea of GANs is we train two networks simultaneously –
  - The generator network generates real looking samples
  - The discriminator network tries to figure out whether or not a sample from came from the generator network or a training set
- The generator network tries to fool the discriminator

Adversarial Training



GAN Training

- Discriminator – Predicts the probability of being true data
- Generator – Creates samples from the distribution of interest

- Two cost functions –
  - Discriminator cost function:

$$J_D = \mathbb{E}_{x\sim\mathcal{D}}\big[-\log\big(D(x)\big)\big] + \mathbb{E}_z\left[-\log\left(1 - D\big(G(z)\big)\right)\right]$$

  - Generator cost function:

$$J_G = -J_D$$

Minimax Formulation of GAN Training

- Discriminator cost function:

$$J_D = \mathbb{E}_{x\sim\mathcal{D}}\big[-\log\big(D(x)\big)\big] + \mathbb{E}_z\left[-\log\left(1 - D\big(G(z)\big)\right)\right]$$

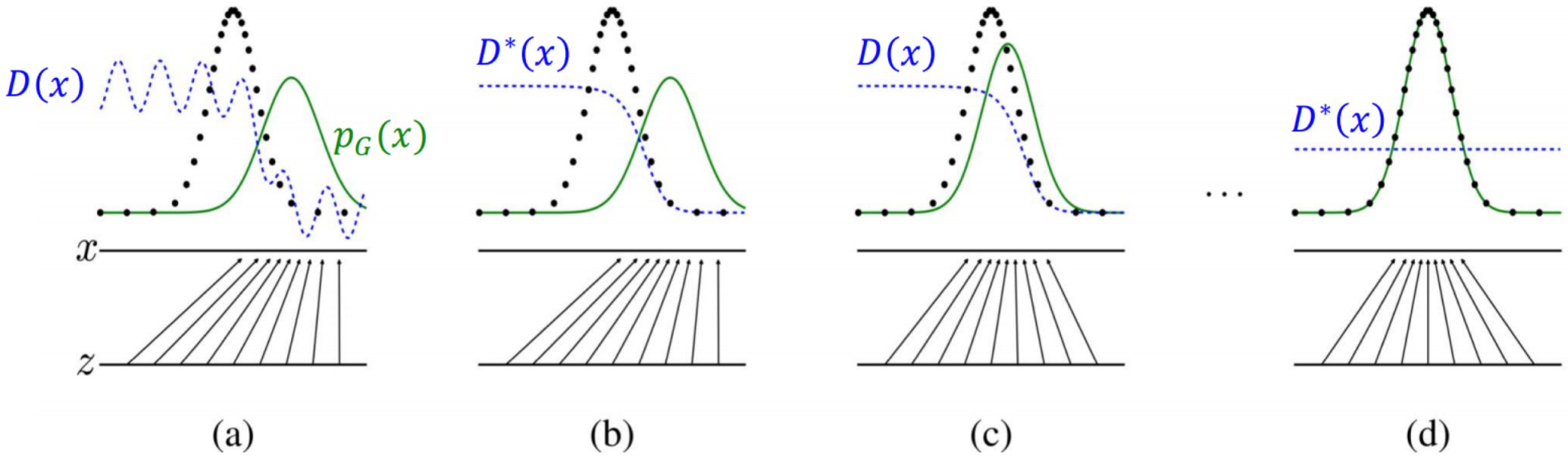
- Generator cost function:

$$J_G = -J_D$$

- Minimax Formulation:

$$\max_G \min_D J_D$$

Training the GAN



Problem with the Minimax Formulation

- Since $J_G = -J_D$

$$J_G = \mathbb{E}_{x \sim \mathcal{D}}\left[\log\left(D(x)\right)\right] + \mathbb{E}_z\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

- There is a problem when the discriminator is much better than the generator

$$D\left(G(z)\right) = 0$$

$$\mathbb{E}_z\left[\log\left(1 - D\left(G(z)\right)\right)\right] = 0$$

- There is no gradient to improve the generator

A Better Cost Function

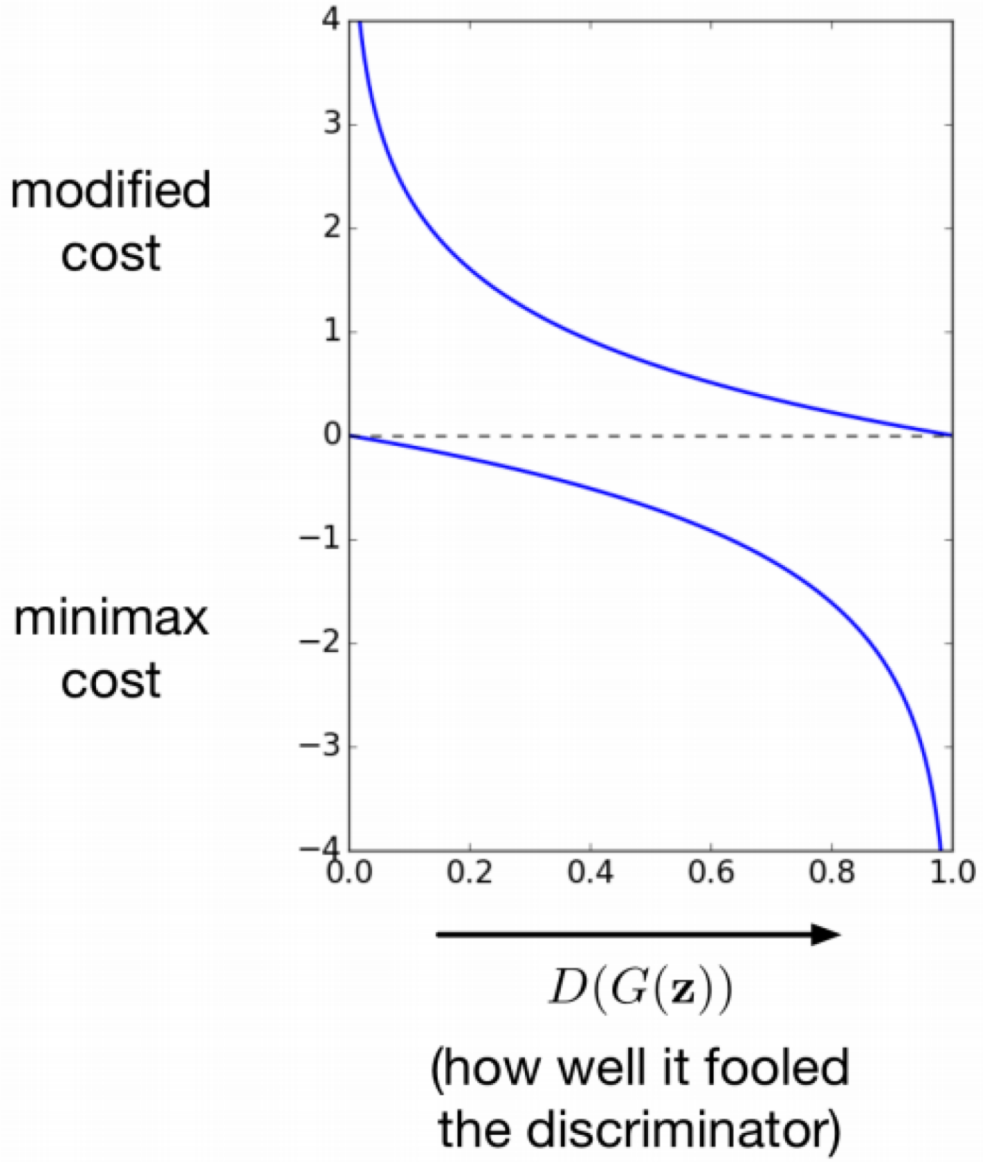
- Original cost function –

$$J_G = \mathbb{E}_{x\sim\mathcal{D}}\left[\log\left(D(x)\right)\right] + \mathbb{E}_z\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

- New cost function –

$$J_G = \mathbb{E}_z\left[\log\left(D\left(G(z)\right)\right)\right]$$

- Change is to the target class



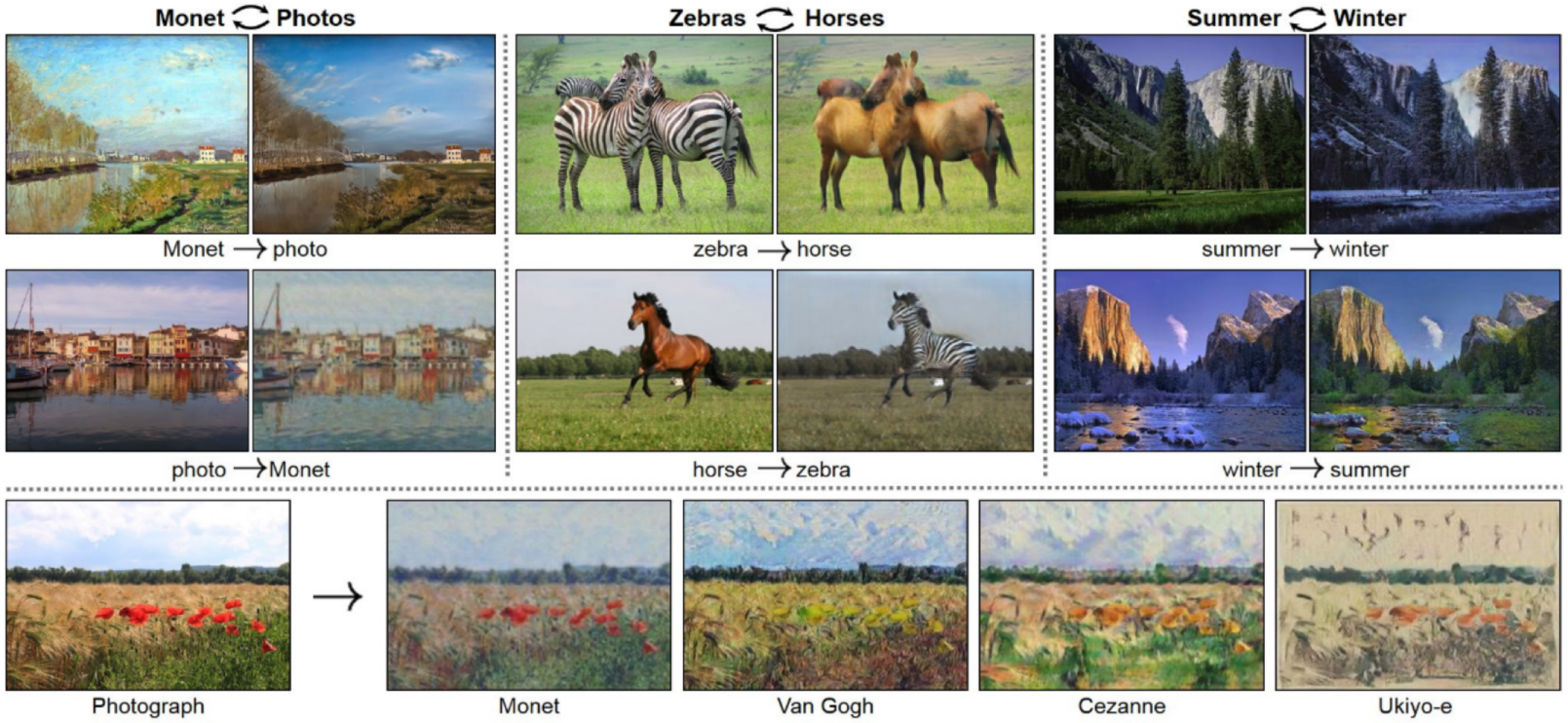
GANs

- Since their original formulation in 2014, hundreds of papers have introduced various training techniques and architectures

- Many common architectures focus on image generation and rely on the DC-GAN architecture
  - De-Convolution Generative Adversarial Network
  - Generator and Discriminator are convolutional neural networks

CycleGAN

Style Transfer: Change the style of an image while preserving its content



Data: Two **unrelated** collections of images, one set for each style

CycleGANs

- If we have paired data, this just becomes a supervised learning problem
  - Often times, it is hard or impossible to find paired data
- The CycleGAN architecture learns style transfer from unpaired data
  - Train two different generator networks to go from style 1 to style 2 and vice versa
  - Make sure the generated samples of style 2 are indistinguishable from real images in the generator set
  - Make sure the generators are cycle consistent – mapping from style 1 to style 2 and back should give you back almost the original image

The discriminator tries to distinguish generated zebra images from real ones

D

Discriminator loss: GAN generator objective, i.e. negative log probability D assigns to the sample being real

Real zebra image

Reconstruction loss: squared error between the original image and the reconstruction

G_1

G_2

Input image (real horse image)

Generator 1 learns to map from horse images to zebra images while preserving the structure

Generated sample

Generator 2 learns to map from zebra images to horse images while preserving the structure

Reconstruction

Total loss = discriminator loss + reconstruction loss

CycleGAN – Style Transfer Between Photos and Maps