Generative Adversarial Networks

Overview

- In generative modeling, we'd like to train a network that models a distribution, such as a distribution over images.
- One way to judge the quality of the model is to sample from it.
- This field has seen rapid progress:

2009

2015

2018

Overview

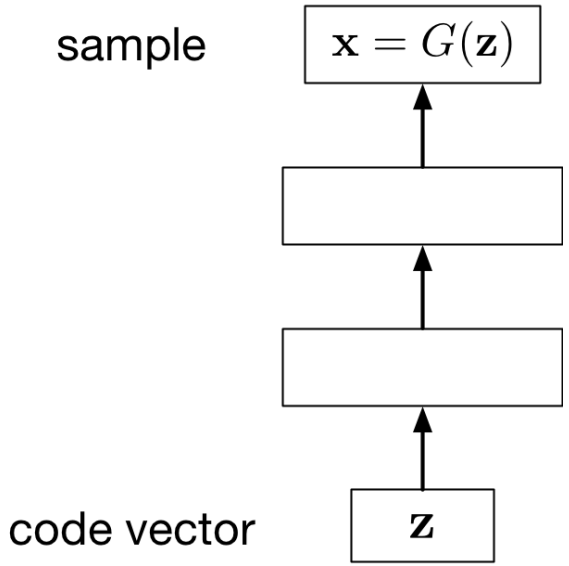
Four modern approaches to generative modeling:

- Generative adversarial networks (today)
- Reversible architectures (next lecture)
- Autoregressive models (Lecture 7, and next lecture)
- Variational autoencoders (CSC412)

All four approaches have different pros and cons.

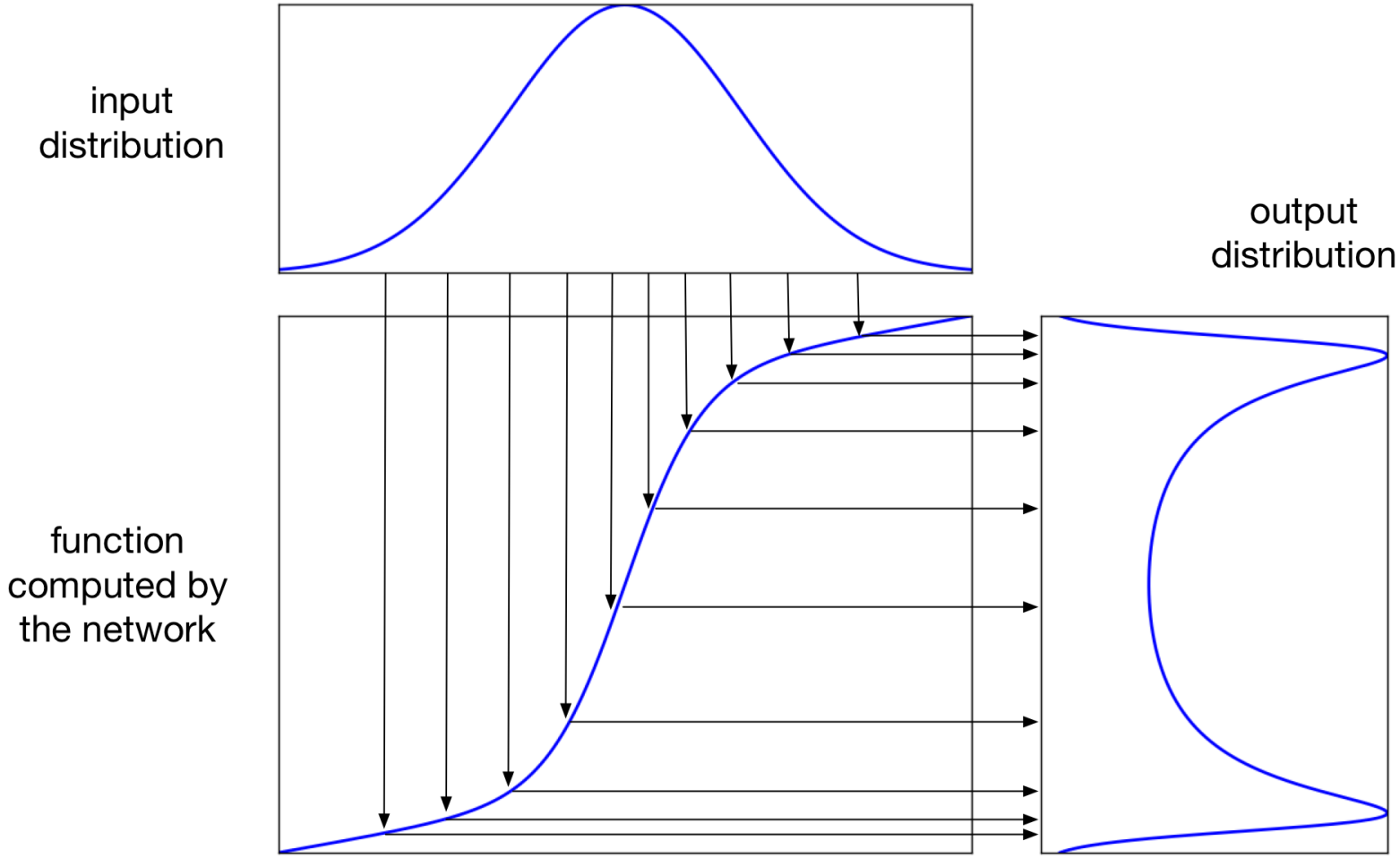
Implicit Generative Models

- Implicit generative models implicitly define a probability distribution
- Start by sampling the code vector $\mathbf{z}$ from a fixed, simple distribution (e.g. spherical Gaussian)
- The generator network computes a differentiable function G mapping $\mathbf{z}$ to an $\mathbf{x}$ in data space

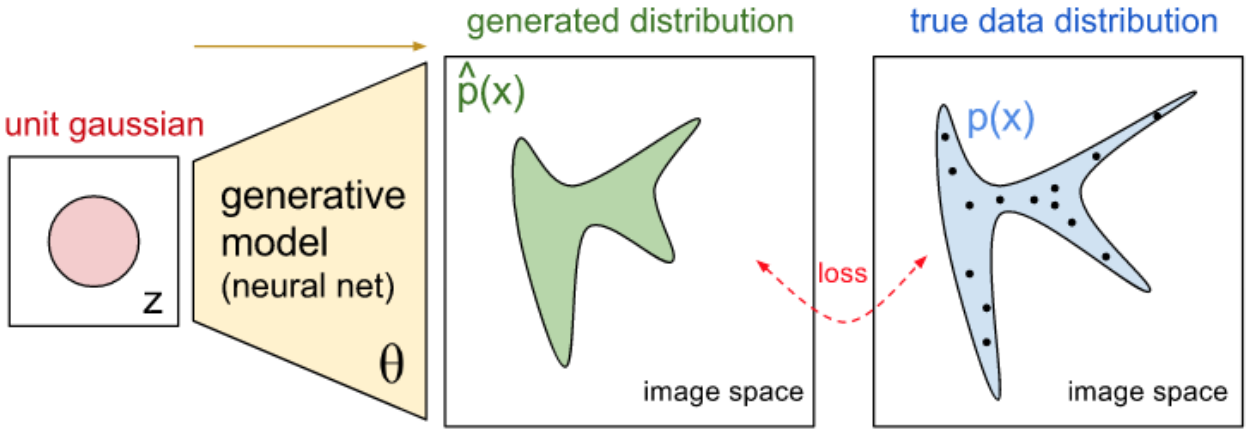


Implicit Generative Models

A 1-dimensional example:

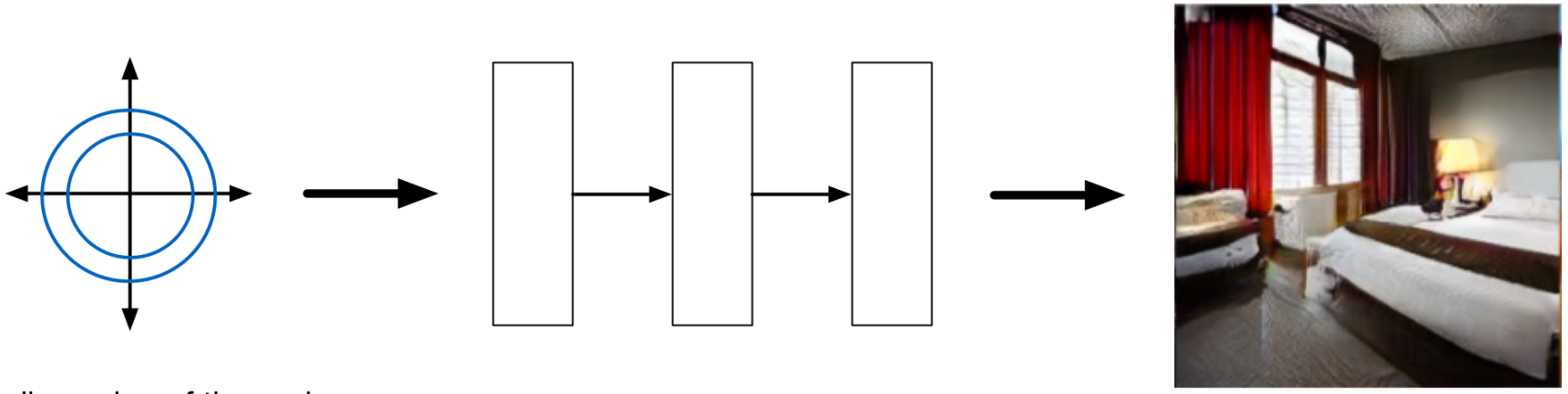


Implicit Generative Models



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

Implicit Generative Models



Each dimension of the code vector is sampled independently from a simple distribution, e.g. Gaussian or uniform.

This is fed to a (deterministic) generator network.

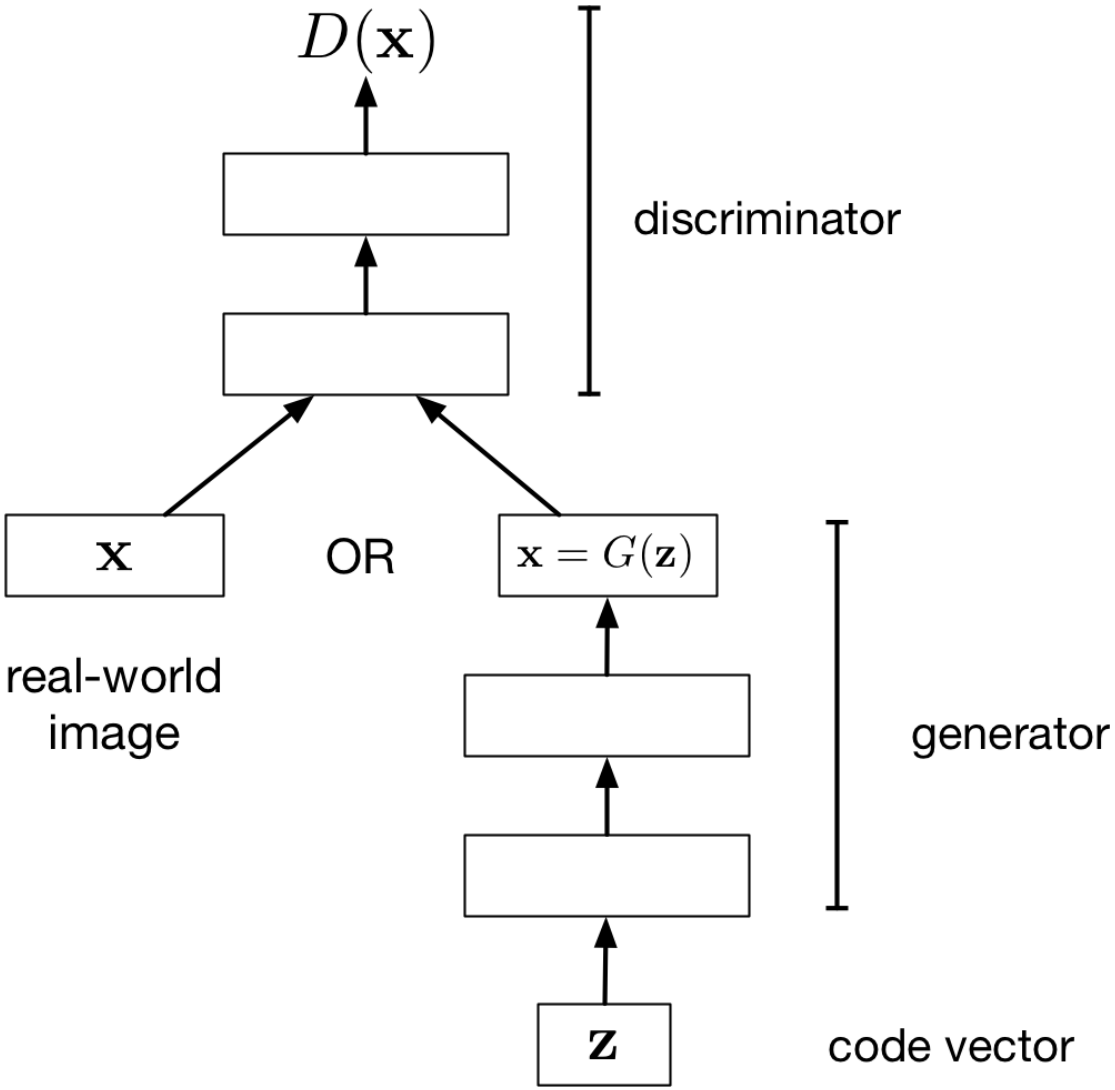
The network outputs an image.

This sort of architecture sounded preposterous to many of us, but amazingly, it works.

Generative Adversarial Networks

- The advantage of implicit generative models: if you have some criterion for evaluating the quality of samples, then you can compute its gradient with respect to the network parameters, and update the network's parameters to make the sample a little better
- The idea behind Generative Adversarial Networks (GANs): train two different networks
  - The generator network tries to produce realistic-looking samples
  - The discriminator network tries to figure out whether an image came from the training set or the generator network
- The generator network tries to fool the discriminator network

Generative Adversarial Networks



Generative Adversarial Networks

- Let D denote the discriminator's predicted probability of being data
- Discriminator's cost function: cross-entropy loss for task of classifying real vs. fake images

$$\mathcal{J}_D = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[-\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}}[-\log(1 - D(G(\mathbf{z})))]$$

- One possible cost function for the generator: the opposite of the discriminator's

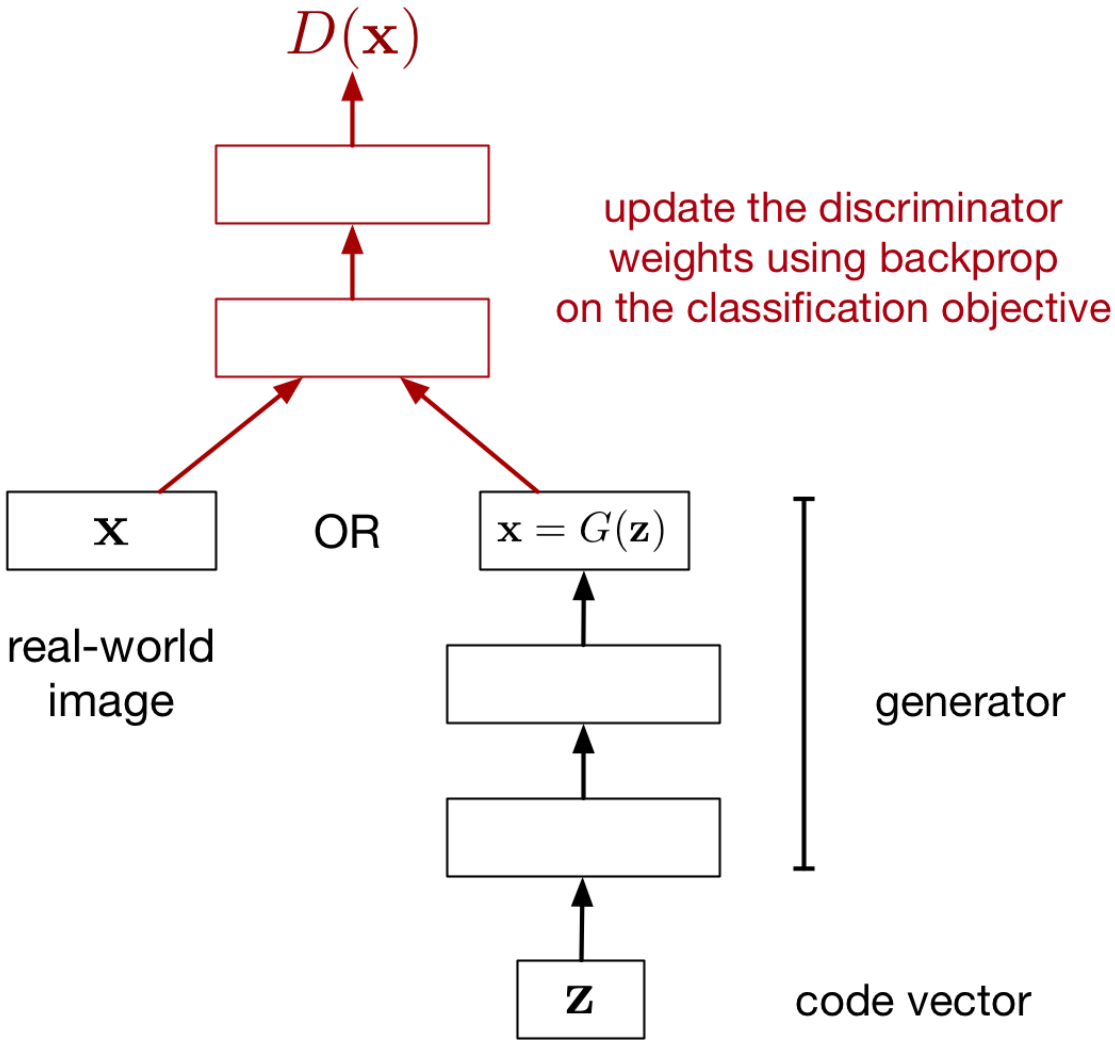
$$\begin{aligned}\mathcal{J}_G &= -\mathcal{J}_D \\ &= \text{const} + \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]\end{aligned}$$

- This is called the minimax formulation, since the generator and discriminator are playing a zero-sum game against each other:

$$\max_G \min_D \mathcal{J}_D$$

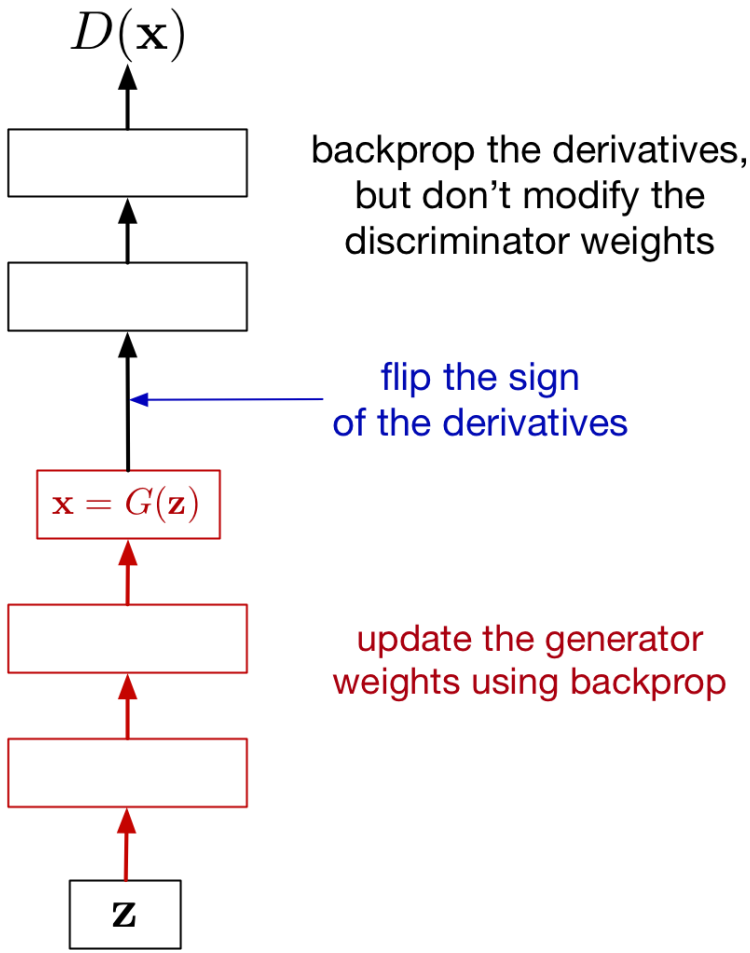
Generative Adversarial Networks

Updating the discriminator:



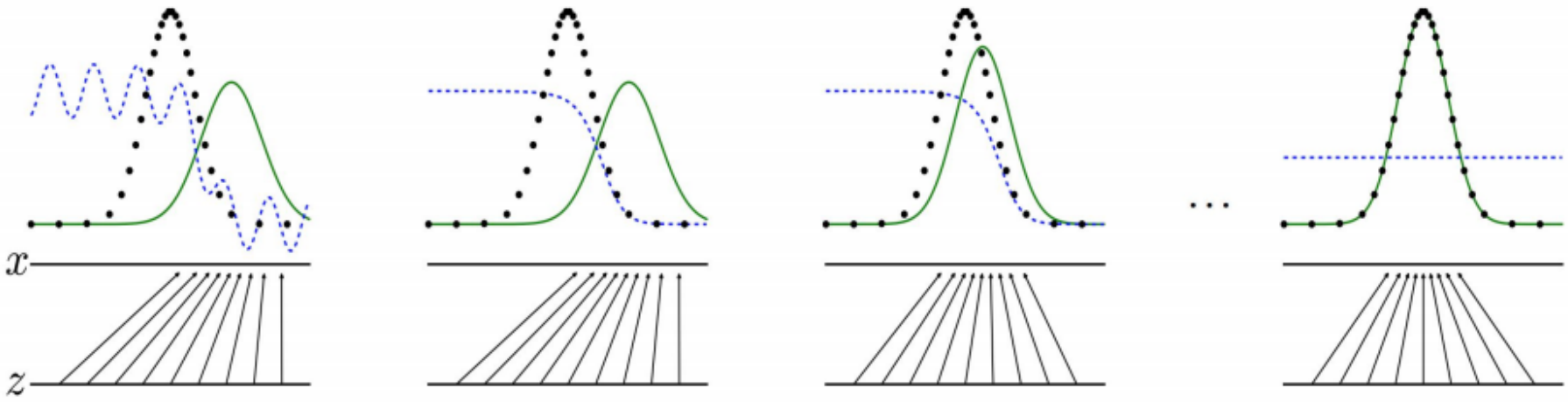
Generative Adversarial Networks

Updating the generator:



Generative Adversarial Networks

Alternating training of the generator and discriminator:



A Better Cost Function

- We introduced the minimax cost function for the generator:

$$\mathcal{J}_G = \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

- One problem with this is saturation.
- Recall from our lecture on classification: when the prediction is really wrong,
  - "Logistic + squared error" gets a weak gradient signal
  - "Logistic + cross-entropy" gets a strong gradient signal
- Here, if the generated sample is really bad, the discriminator's prediction is close to 0, and the generator's cost is flat.

A Better Cost Function

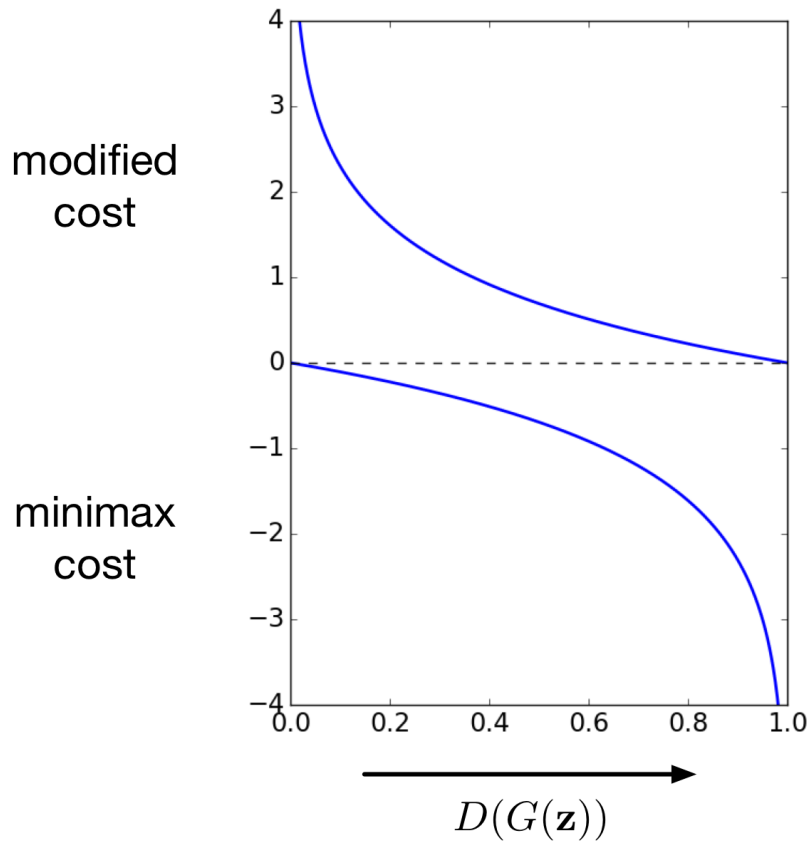
- Original minimax cost:

$$\mathcal{J}_G = \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

- Modified generator cost:

$$\mathcal{J}_G = \mathbb{E}_{\mathbf{z}}[-\log D(G(\mathbf{z}))]$$

- This fixes the saturation problem.



Generative Adversarial Networks

- Since GANs were introduced in 2014, there have been hundreds of papers introducing various architectures and training methods.
- Most modern architectures are based on the Deep Convolutional GAN (DC-GAN), where the generator and discriminator are both conv nets.
- GAN Zoo: `https://github.com/hindupuravinash/the-gan-zoo`
  - Good source of horrible puns (VEEGAN, Checkhov GAN, etc.)

GAN Samples

Celebrities:

Karras et al., 2017. Progressive growing of GANs for improved quality, stability, and variation

GAN Samples

Bedrooms:

Karras et al., 2017. Progressive growing of GANs for improved quality, stability, and variation

GAN Samples

Objects:

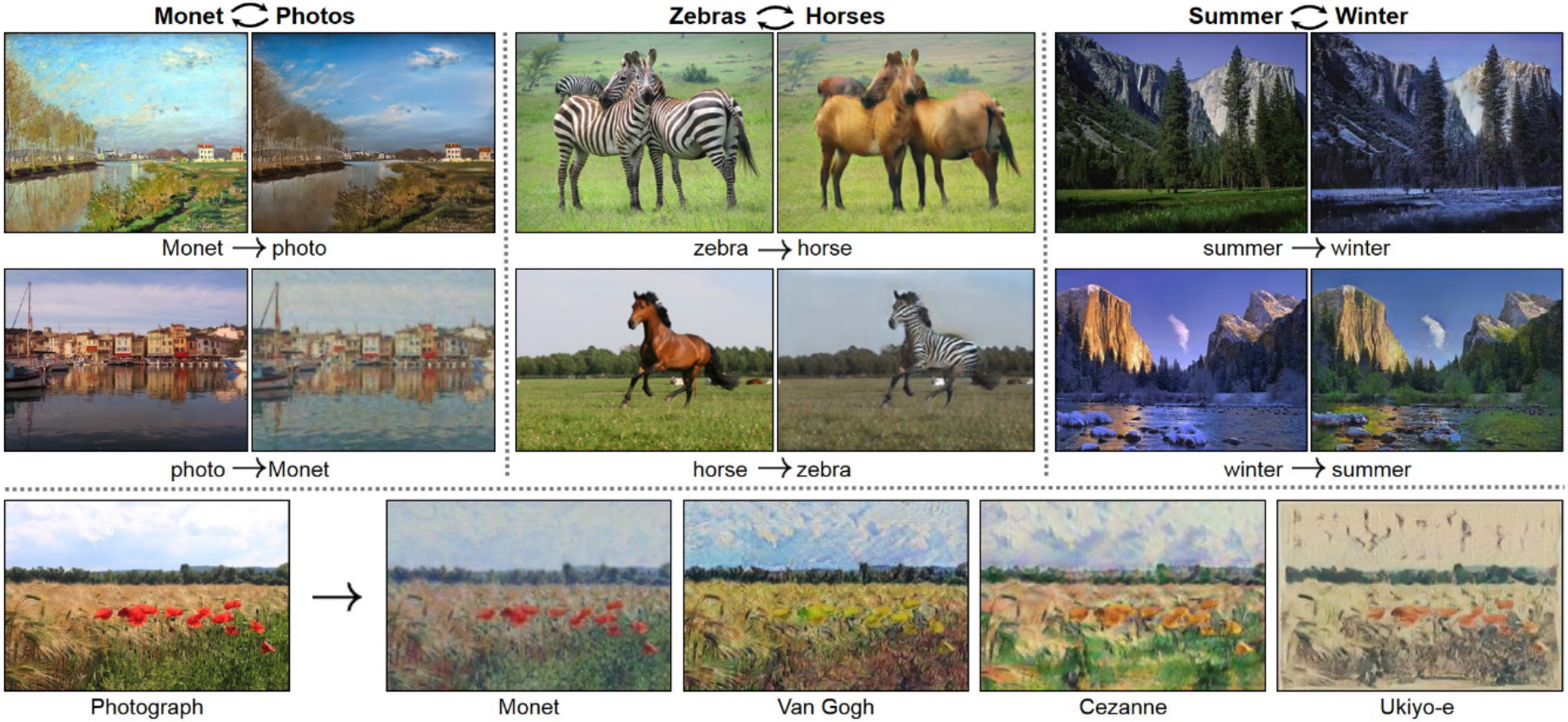
Karras et al., 2017. Progressive growing of GANs for improved quality, stability, and variation

GAN Samples

- GANs revolutionized generative modeling by producing crisp, high-resolution images.
- The catch: we don't know how well they're modeling the distribution.
  - Can't measure the log-likelihood they assign to held-out data.
  - Could they be memorizing training examples? (E.g., maybe they sometimes produce photos of real celebrities?)
  - We have no way to tell if they are dropping important modes from the distribution.
  - See Wu et al., "On the quantitative analysis of decoder-based generative models" for partial answers to these questions.

CycleGAN

Style transfer problem: change the style of an image while preserving the content.



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

CycleGAN

- If we had paired data (same content in both styles), this would be a supervised learning problem. But this is hard to find.
- The CycleGAN architecture learns to do it from unpaired data.
  - Train two different generator nets to go from style 1 to style 2, and vice versa.
  - Make sure the generated samples of style 2 are indistinguishable from real images by a discriminator net.
  - Make sure the generators are cycle-consistent: mapping from style 1 to style 2 and back again should give you almost the original image.

CycleGAN

Real zebra image

The discriminator tries to distinguish generated zebra images from real ones

D

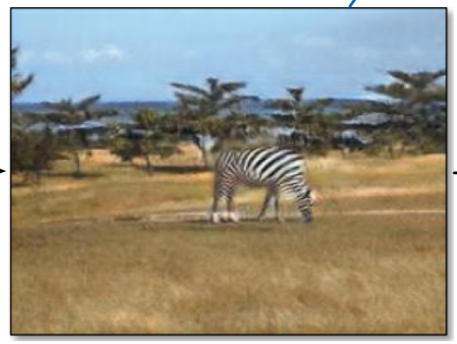
Discriminator loss: GAN generator objective, i.e. negative log probability D assigns to the sample being real

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

Input image (real horse image)

G_1

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



Generated sample

G_2

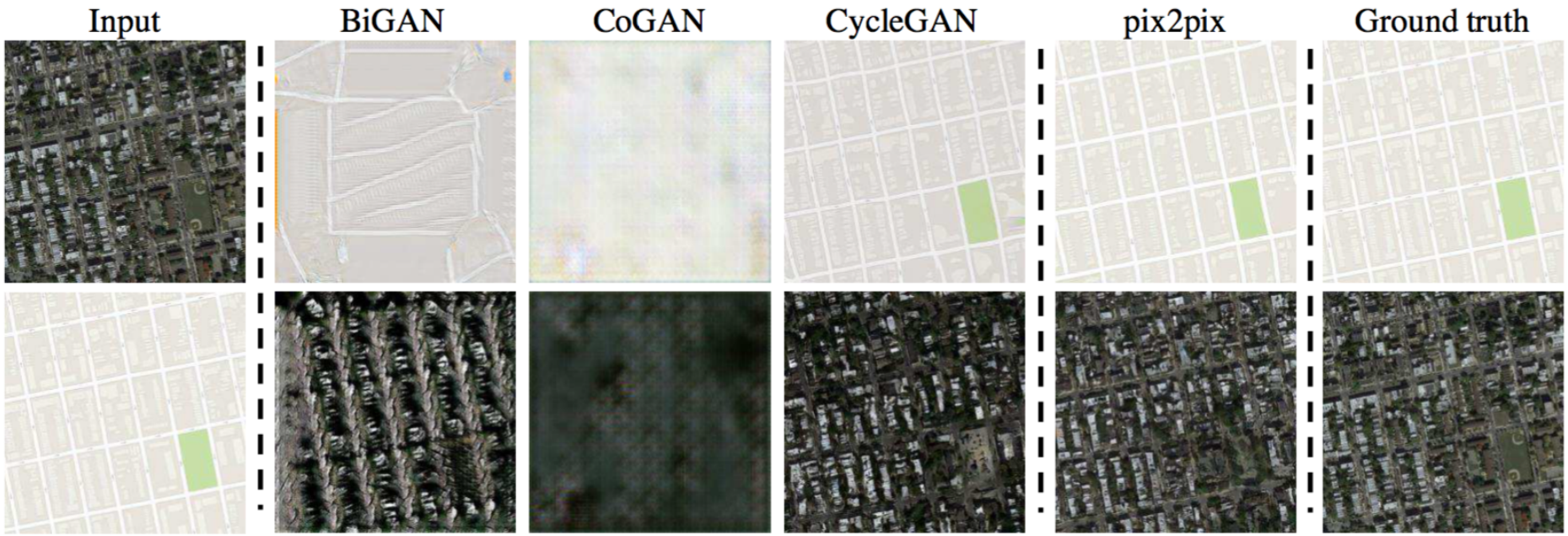
Generator 2 learns to map from zebra images to horse images while preserving the structure

Reconstruction

Total loss = discriminator loss + reconstruction loss

CycleGAN

Style transfer between aerial photos and maps:



CycleGAN

Style transfer between road scenes and semantic segmentations (labels of every pixel in an image by object category):