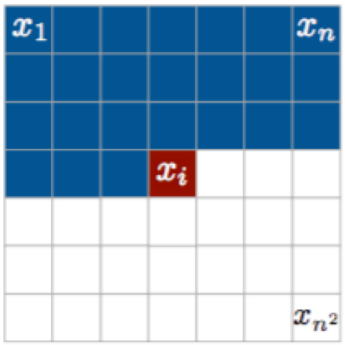
CSC413/2516 Lecture 9: Autoregressive Models and GANs

Jimmy Ba

Overview

- We've already looked at a few autoregressive models in this course:
  - Neural language models from Lecture 3
  - RNN language models (and decoders) from Lecture 7
  - Transformer decoders from Lecture 8
- We can push this further, and generate very long sequences.



Treat an image as a very long sequence using raster scan order

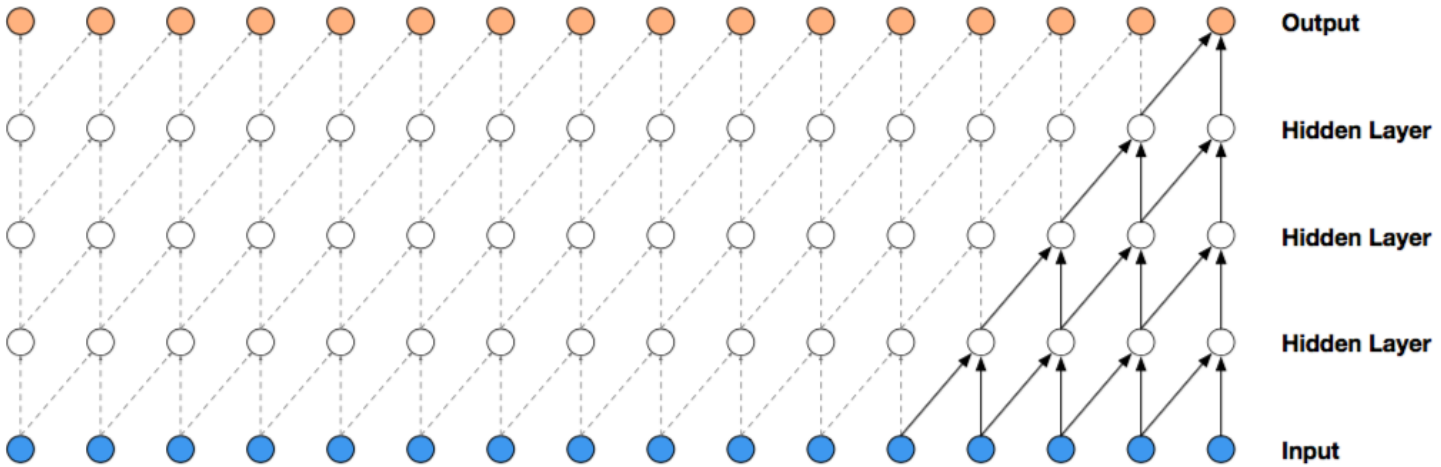
A speech signal can be represented as a waveform, with at least 16,000 samples per second.

- Problem:
  - Training an RNN to generate these sequences requires a sequential computation $>$ 10,000 time steps.
  - Transformers are too expensive to train on 10,000 time steps.

Causal Convolution

Idea 1: causal convolution

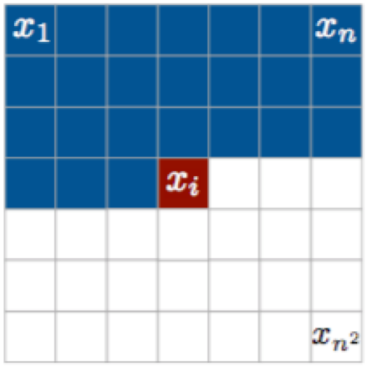
- For RNN language models, we used the training sequence as both the inputs and the outputs to the RNN.
  - We made sure the model was causal: each prediction depended only on inputs earlier in the sequence.
- We can do the same thing using a convolutional architecture.



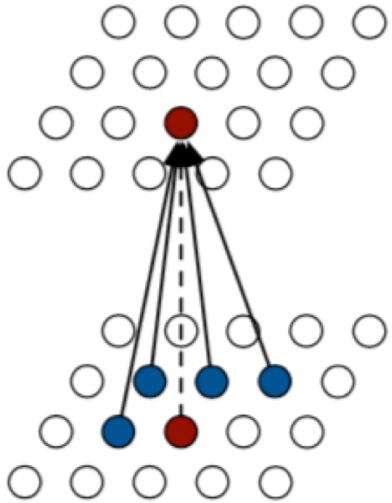
- No for loops! Processing each input sequence just requires a series of convolution operations.

Causal Convolution

Causal convolution for images:

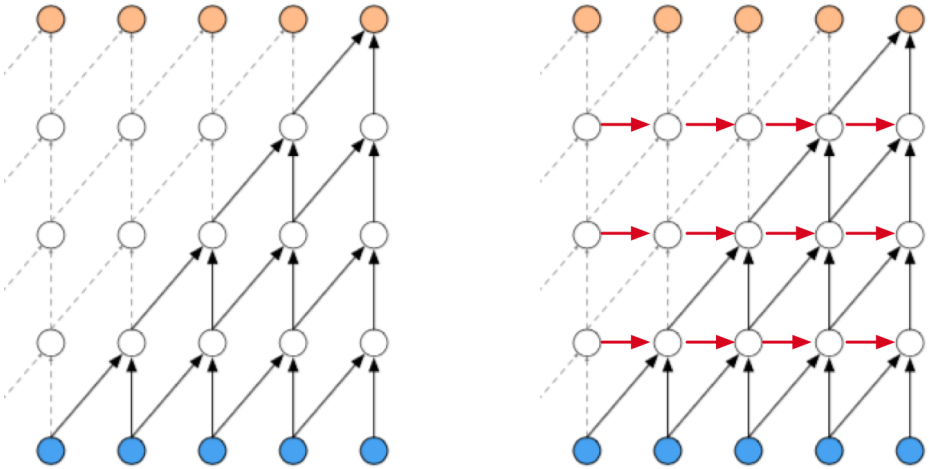


The image is treated as a very long sequence of pixels using raster scan order.



We can restrict the connectivity pattern in each layer to make it causal. This can be implemented by clamping some weights to zero.

CNN vs. RNN



- We can turn a causal CNN into an RNN by adding recurrent connections. Is this a good idea?
  - The RNN has a memory, so it can use information from all past time steps. The CNN has a limited context.
  - But training the RNN is very expensive since it requires a for loop over time steps. The CNN only requires a series of convolutions.
  - Generating from both models is very expensive, since it requires a for loop. (Whereas generating from a GAN or a reversible model is very fast.)

PixelCNN and PixelRNN

- Van den Oord et al., ICML 2016, "Pixel recurrent neural networks"
- This paper introduced two autoregressive models of images: the PixelRNN and the PixelCNN. Both generated amazingly good high-resolution images.
- The output is a softmax over 256 possible pixel intensities.
- Completing an image using an PixelCNN:

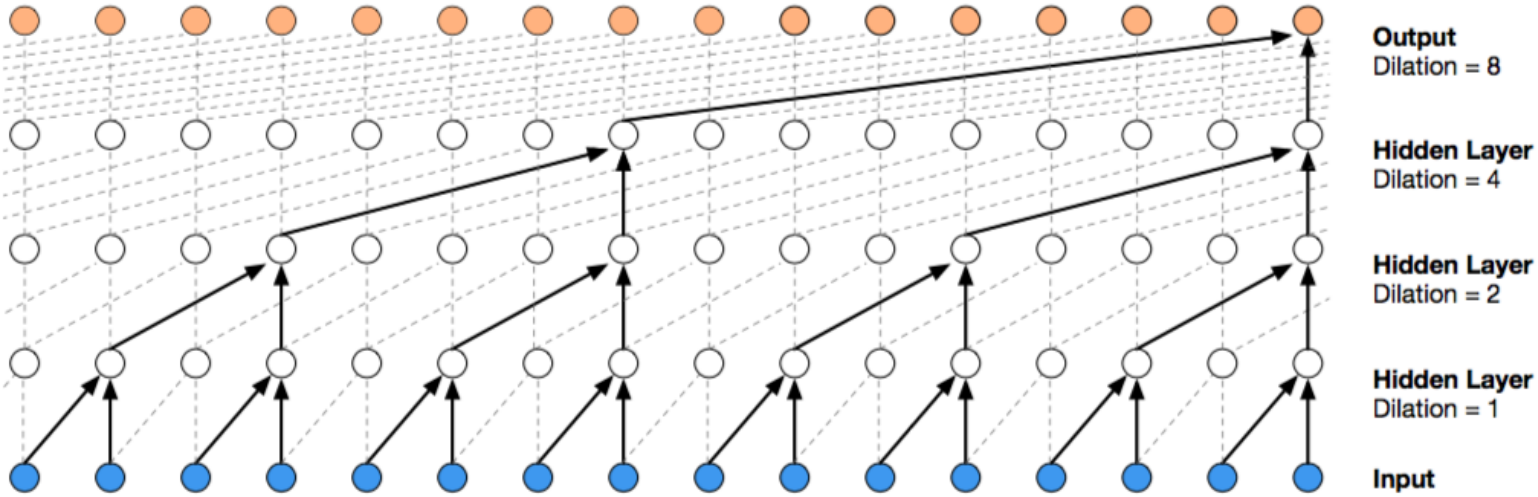
PixelCNN and PixelRNN

Samples from a PixelRNN trained on ImageNet:

Dilated Convolution

Idea 2: dilated convolution

- The advantage of RNNs over CNNs is that their memory lets them learn arbitrary long-distance dependencies.
- But we can dramatically increase a CNN's receptive field using dilated convolution.



WaveNet

- WaveNet is an autoregressive model for raw audio based on causal dilated convolutions.
  - van den Oord et al., 2016. "WaveNet: a generative model for raw audio".
- Audio needs to be sampled at at least 16k frames per second for good quality. So the sequences are very long.
- WaveNet uses dilations of $1, 2, \ldots, 512$, so each unit at the end of this block as a receptive field of length 1024, or 64 milliseconds.
- It stacks several of these blocks, so the total context length is about 300 milliseconds.
- `https://deepmind.com/blog/wavenet-generative-model-raw-audio/`

After the break

After the break: **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

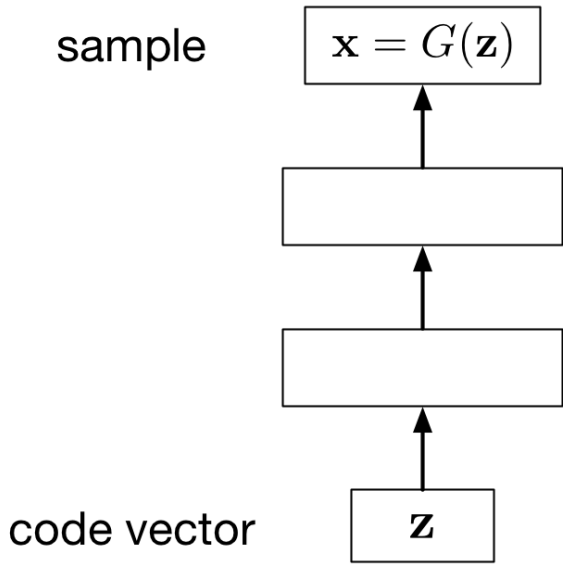
Four modern approaches to generative modeling:

- Autoregressive models (Lectures 3, 7, and 8)
- Generative adversarial networks (this lecture)
- Reversible architectures (next lecture)
- Variational autoencoders (next lecture)

All four approaches have different pros and cons.

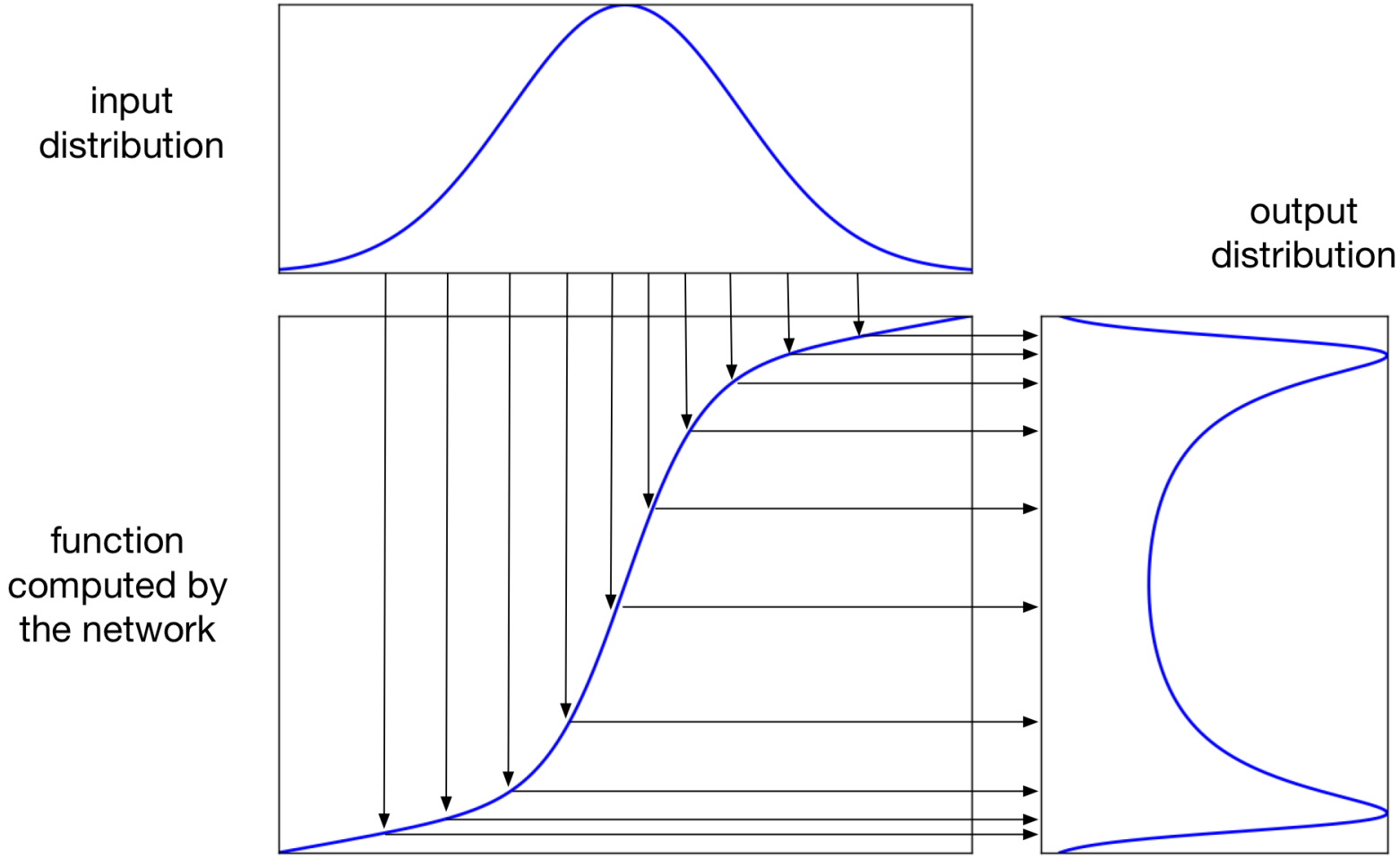
Generator Networks

- Autoregressive models explicitly predict a distribution at each step.
- Another approach to generative modeling is to train a neural net to produce approximate samples from the 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

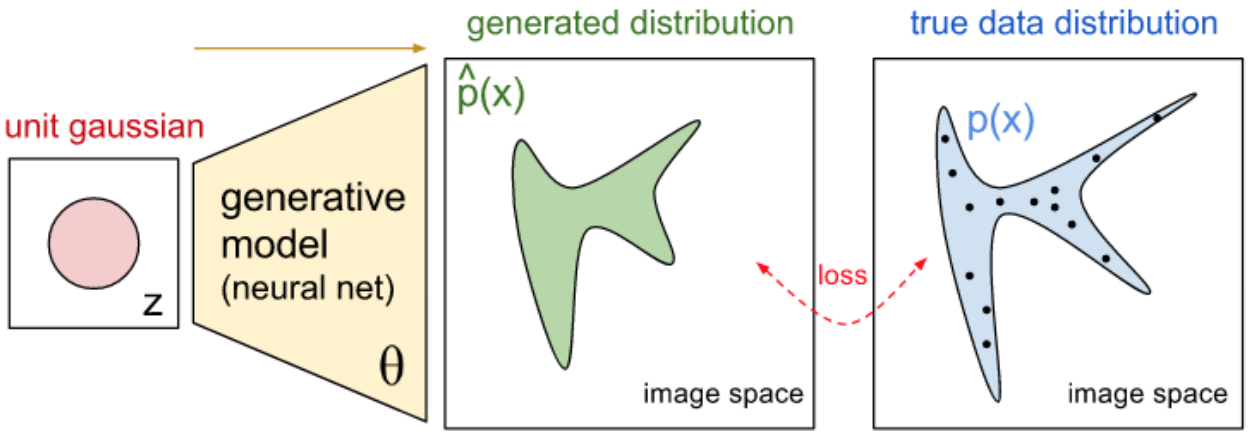


Generator Networks

A 1-dimensional example:

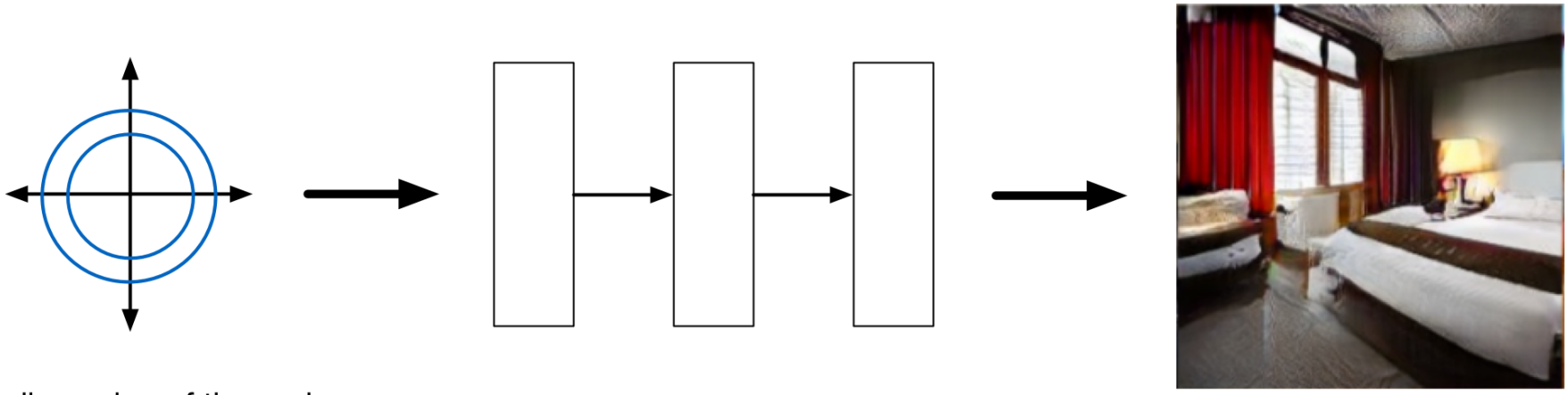


Generator Networks



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

Generator Networks



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.

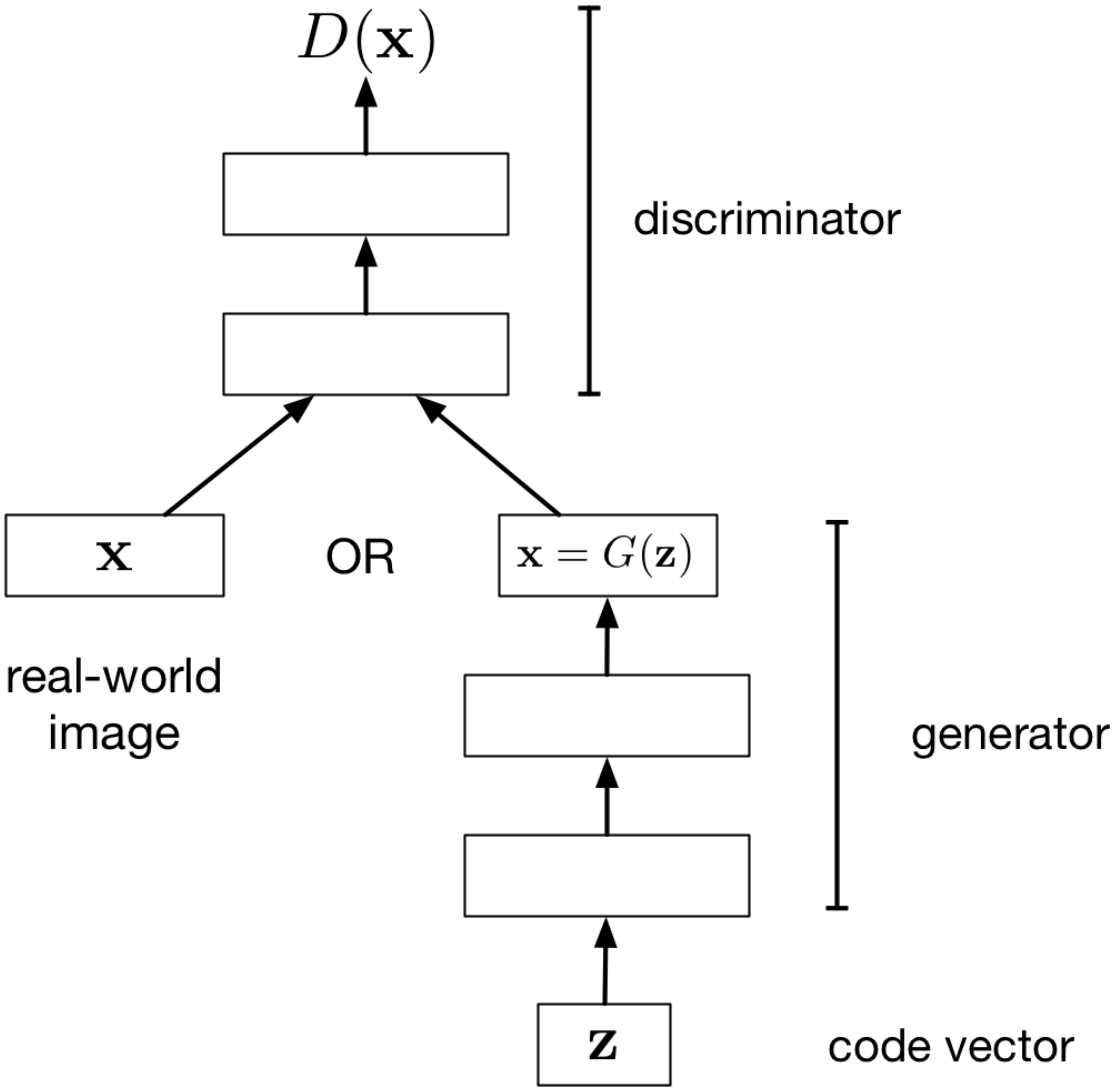
Generative Adversarial Networks

- Implicit generative models learn a mapping from random noise vectors to things that look like, e.g., images
- 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

Generative Adversarial Networks

- Implicit generative models learn a mapping from random noise vectors to things that look like, e.g., images
- 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

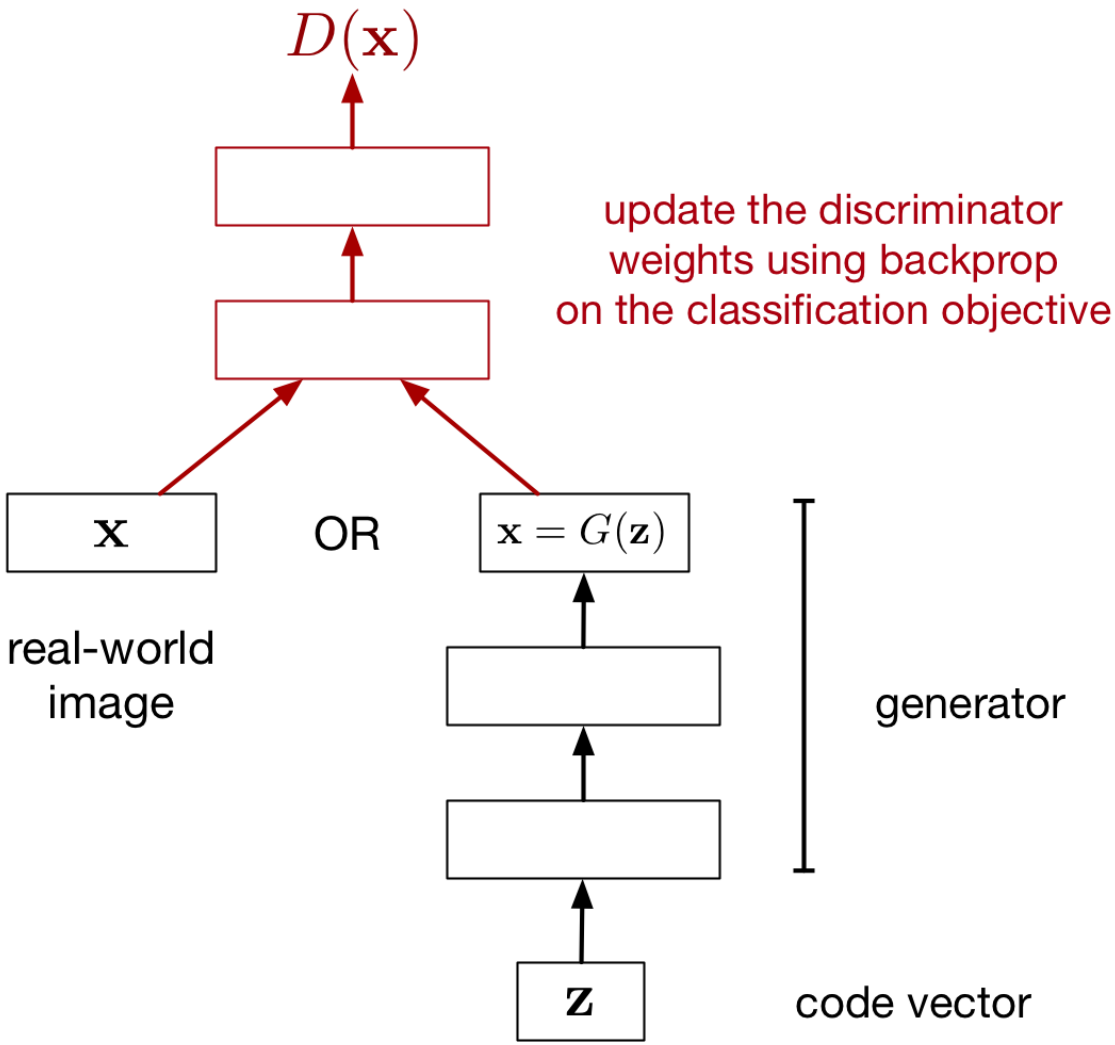
$$\begin{aligned}\mathcal{J}_G &= -\mathcal{J}_D \\ &= \text{const} + \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]\end{aligned}$$

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

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

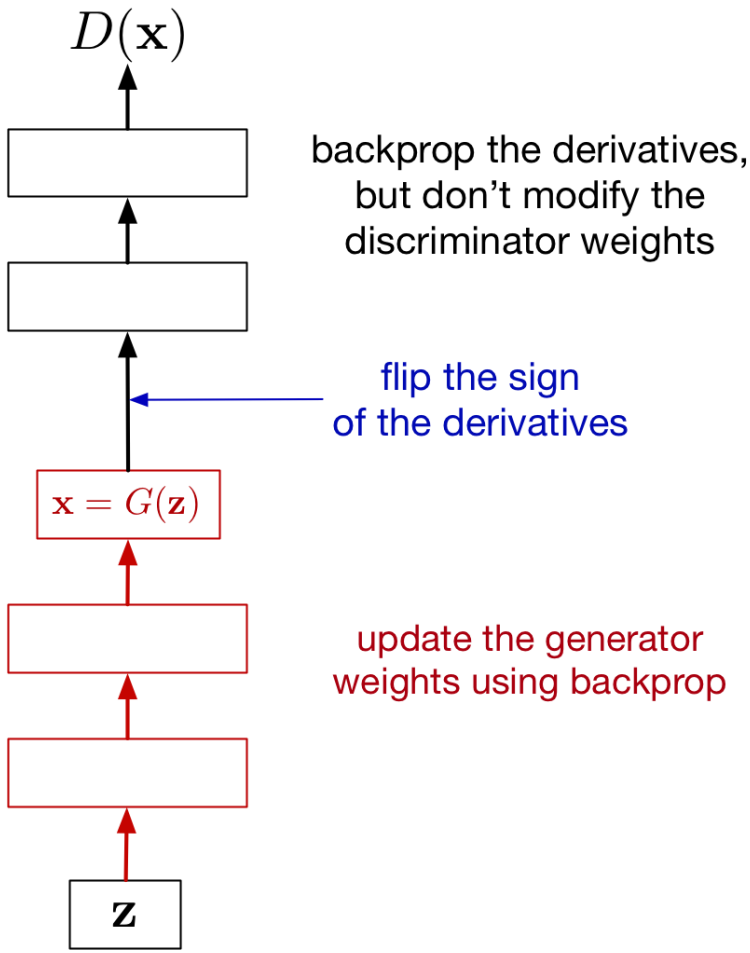
Generative Adversarial Networks

Updating the discriminator:



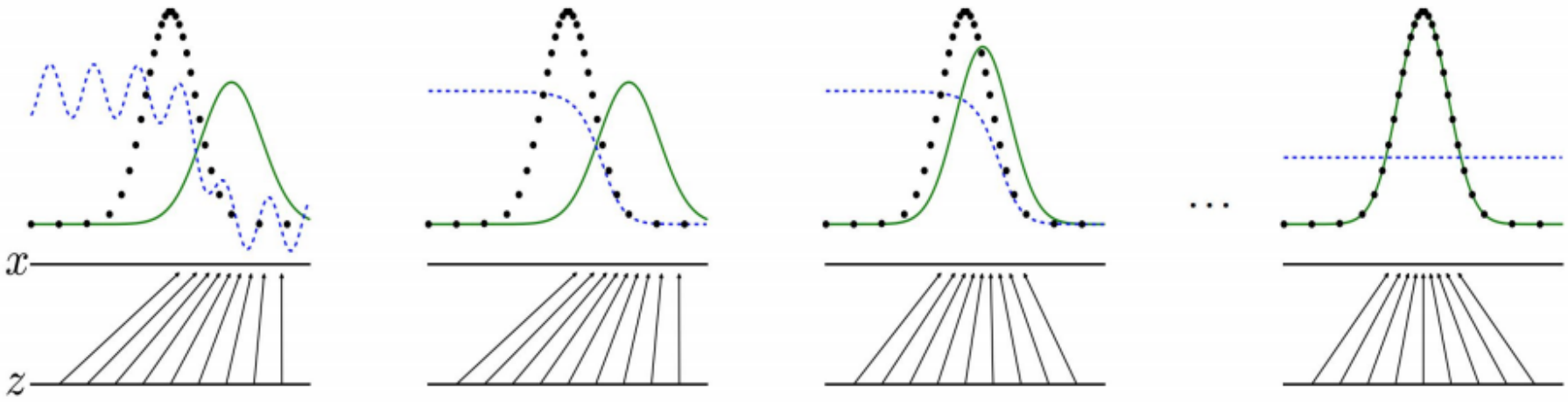
Generative Adversarial Networks

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

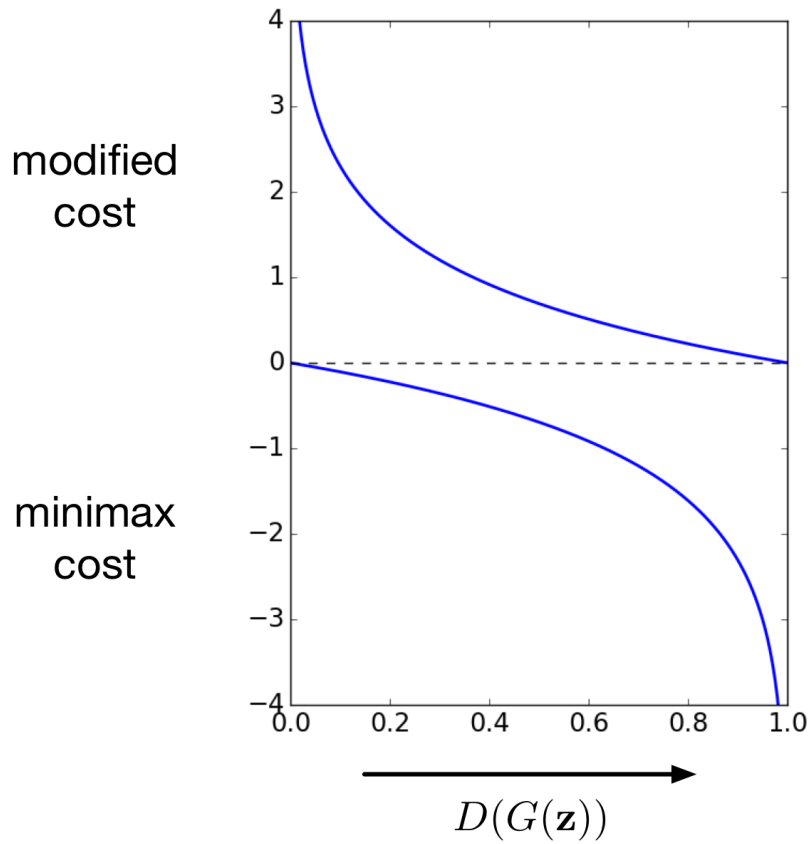
- Original minimax cost:

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

- 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

ImageNet object categories (by BigGAN, a much larger model with a bunch more engineering tricks):

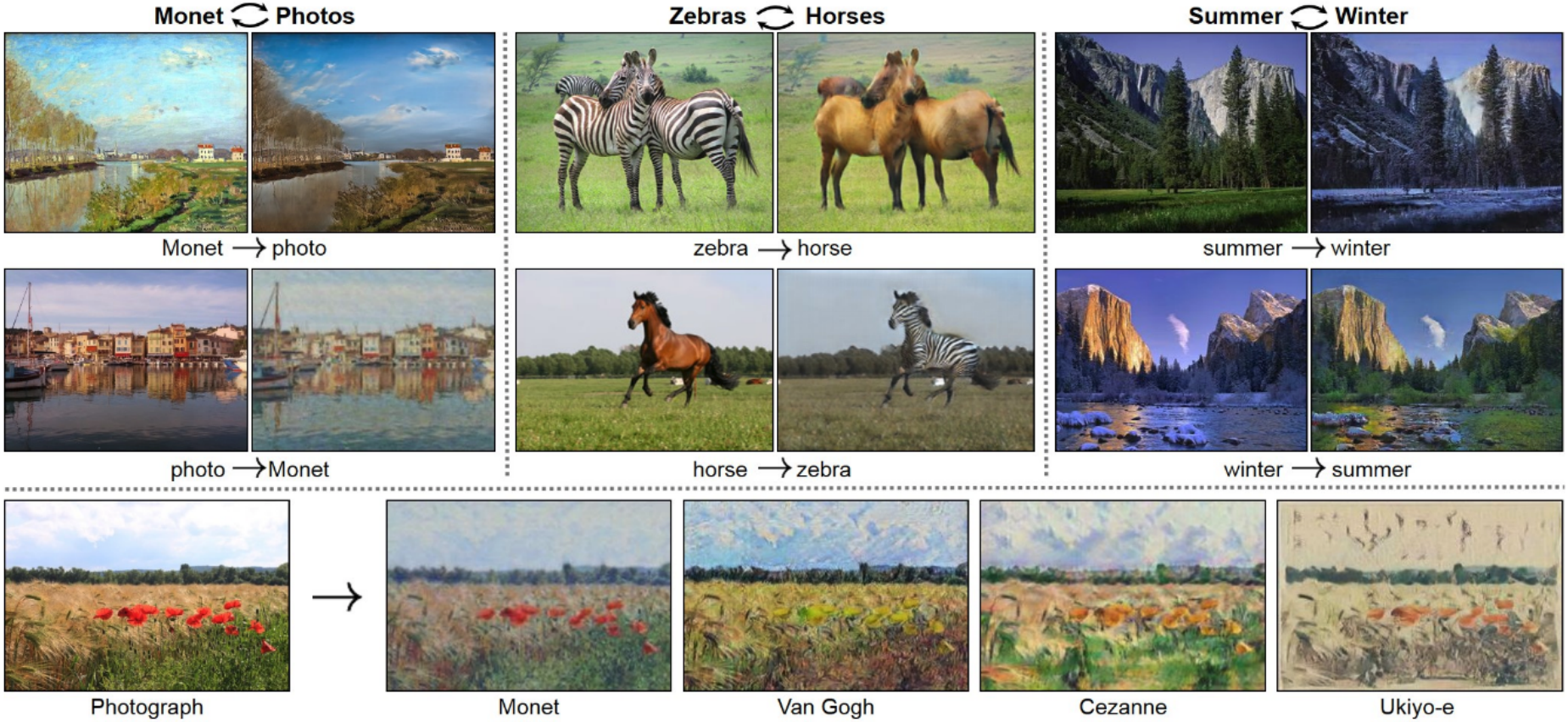
Brock et al., 2019. Large scale GAN training for high fidelity natural image synthesis.

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.

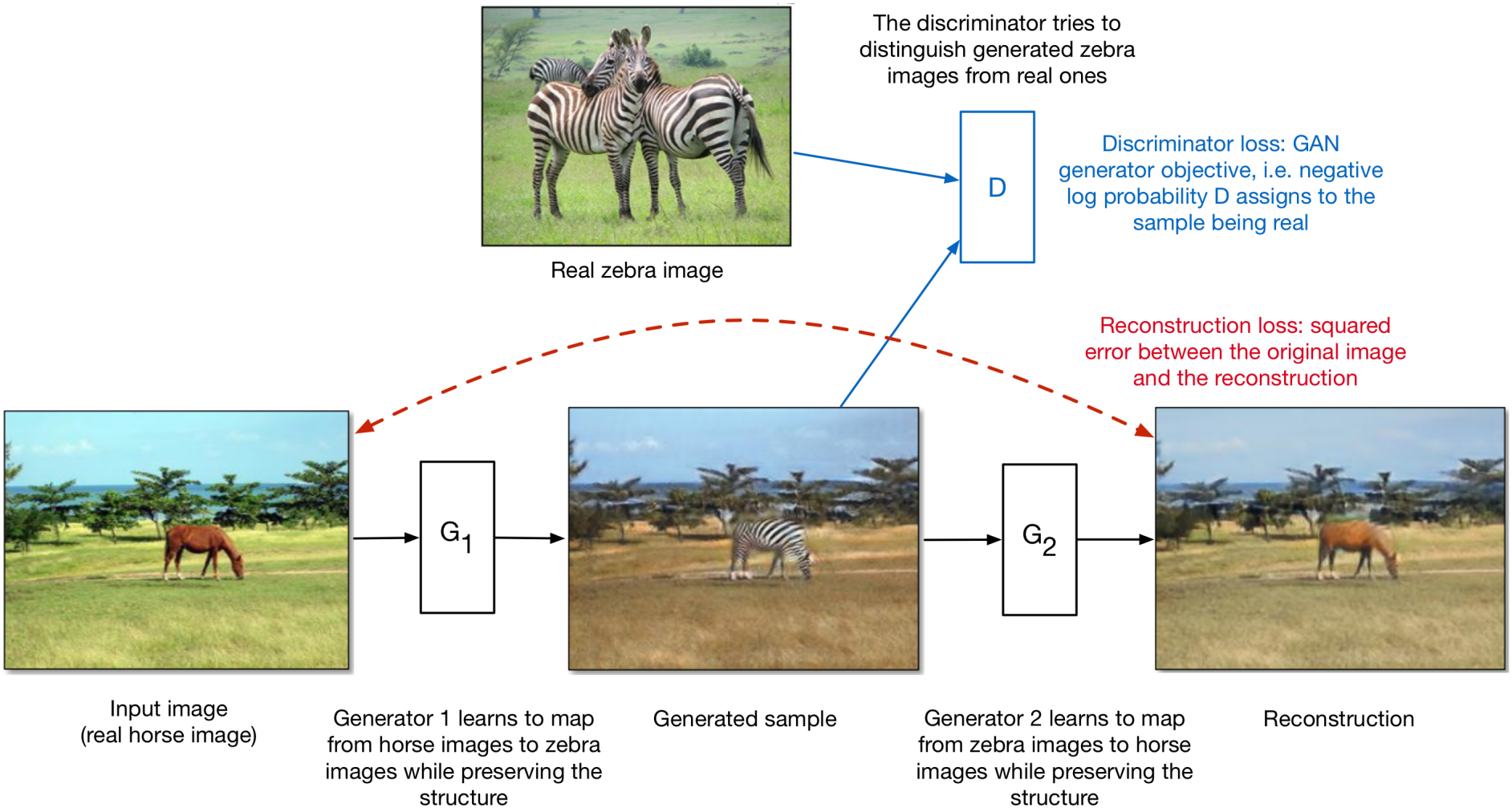


Data: Two unrelated collections of images, one for each style

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.

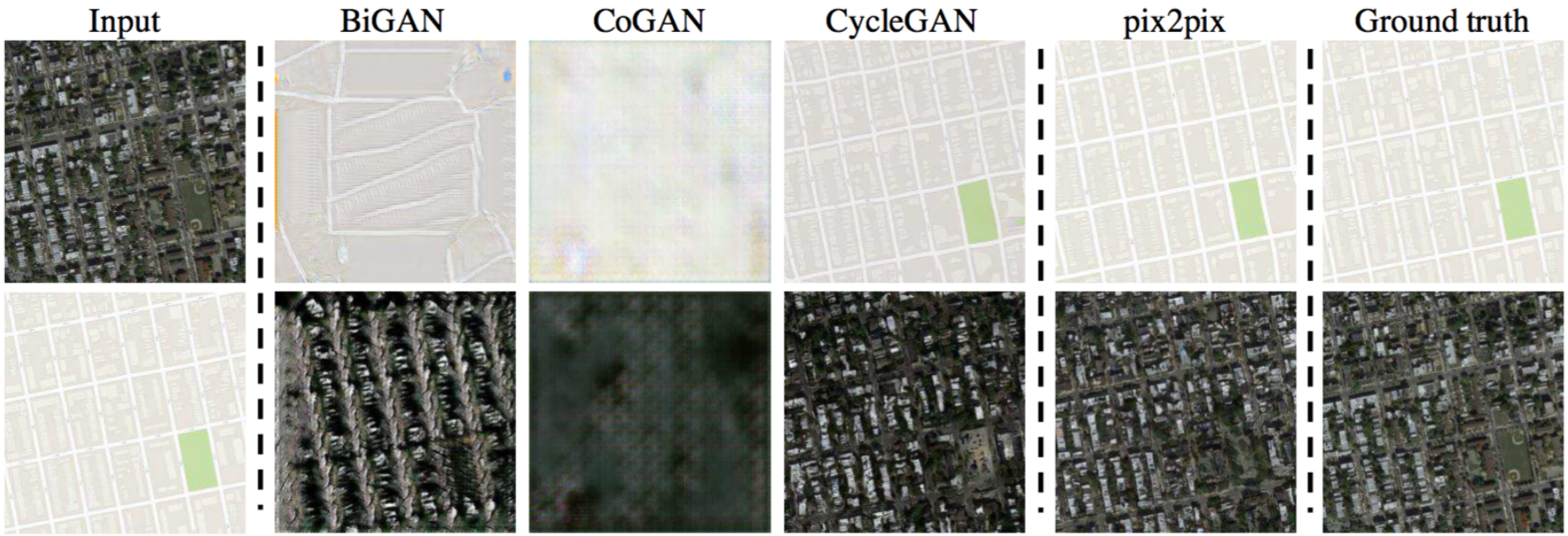
CycleGAN



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):