CSC321 Neural Networks and Machine Learning

Lecture 12

April 1, 2020

Agenda

- Generative Adversarial Networks
- Course Evaluation

Exam Logistics

We're still working on the exam logistics, here's what we currently have in mind:

- Exam paper will be downloadable on April 16th (most likely via a link on Markus). There will be different versions of the exam
- We will log the time that you access your exam script
- We will expect you save your exam script locally and cut internet
- You will have 2 hours to write the exam (e.g. on a piece of paper, type your answers . . .)
- You will write a declaration stating the time you start/complete the exam, and that you did not use any unauthorized aid
- You will have 30 minutes to upload your solutions

Mock Exam

We'll hold a 30 minute mock exam available between April 9th-April 11th.

- Download the exam script (e.g. from the link on Markus)
- We will log the time that you access your exam script (you won't see this)
- The mock test should take ~30 minutes
- Upload your solutions on Markus

Exam Review Sessions and Office Hours

Student-run review sessions:

https://docs.google.com/spreadsheets/d/1jBhAVD-Miux7GBiPrDZXT_vPFPv6kSsTh6_aJjHd678/edit#gid=0

Instructor Exam Office Hours (Starting April 6th):

- **Lisa**: MWF 4:00pm-5:00pm (but will cut short if there are no questions)
- **Pouria**: TTh 12:00pm-1:00pm (but will cut short if there are no questions)

CSC321 Social: Evening of Go

Learn Go with Lisa: Friday, April 3rd, 4pm-5pm on Bb Collaborate

AlphaGo Documentary:
https://www.youtube.com/watch?v=WXuK6gekU1Y

AlphaGo/Reinforcement Learning Discussions: Friday, April 3rd, 8pm-9pm

CSC321 “Exam Jam”

Is there interest in having something on April 3rd or 4th?

Generative Adversarial Networks

Supervised vs Unsupervised Learning

From lecture 1:

- **Supervised Learning**: learning a function that maps an input to an output based on example input-output pairs
- **Unsupervised Learning**: learning the structure of some (unlabelled) data

Question:

Is the tweet generation task from Tutorial 11 a *supervised* or *unsupervised* learning problem?

Example

Q: Are these supervised or unsupervised learning task?

- Task 1: Predict the next word given all the previous words in a "happy tweet"
- Task 2: Generate a new "happy tweet"

Example

Q: Are these supervised or unsupervised learning task?

- Task 1: Predict the next word given all the previous words in a "happy tweet"
- Task 2: Generate a new "happy tweet"

Task 1 is supervised, and is an example of a **discriminative model**

Task 2 is unsupervised, and is an example of a **generative model**.

Generative Model

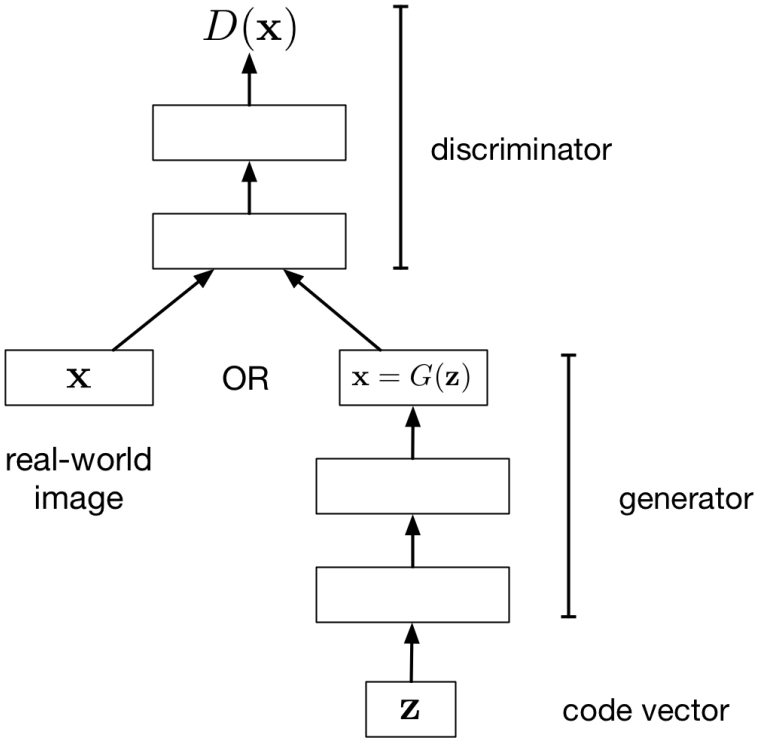
- A **generative model** learns the *structure* of a set of input data, and can be used to **generate** new data
- Examples:
  - Autoencoders
  - RNN for text generation

Autoencoders uses MSELoss

- Blurry images, blurry backgrounds
- Why? Because the loss function used to train an autoencoder is the **mean square error loss** (MSELoss)
- To minimize the MSE loss, autoencoders predict the "average" pixel

Can we use a better loss function?

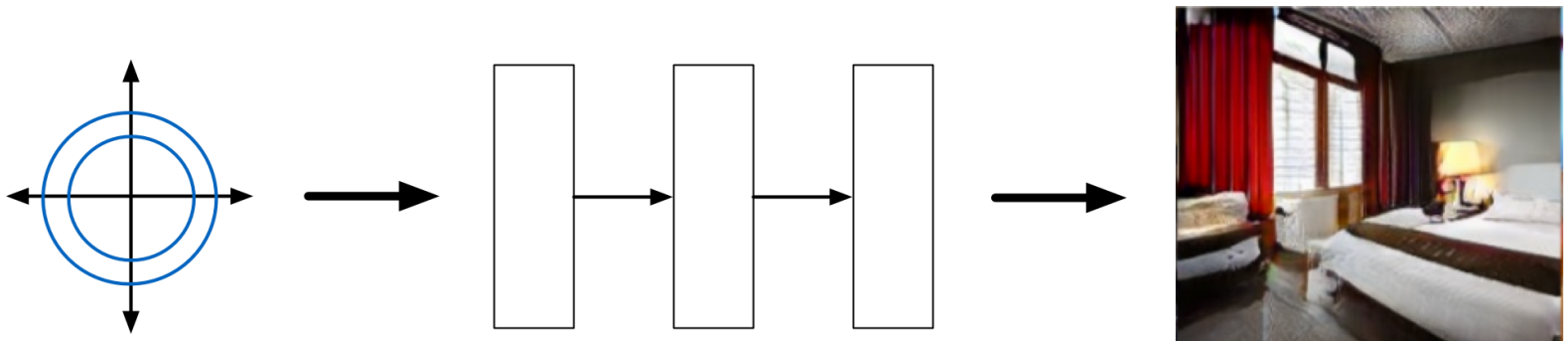
Generative Adversarial Network



- **Generator network**: try to fool the discriminator by generating real-looking images
- **Discriminator network**: try to distinguish between real and fake images

The loss function of the generator (the model we care about) is defined by the discriminator!

GAN Generator



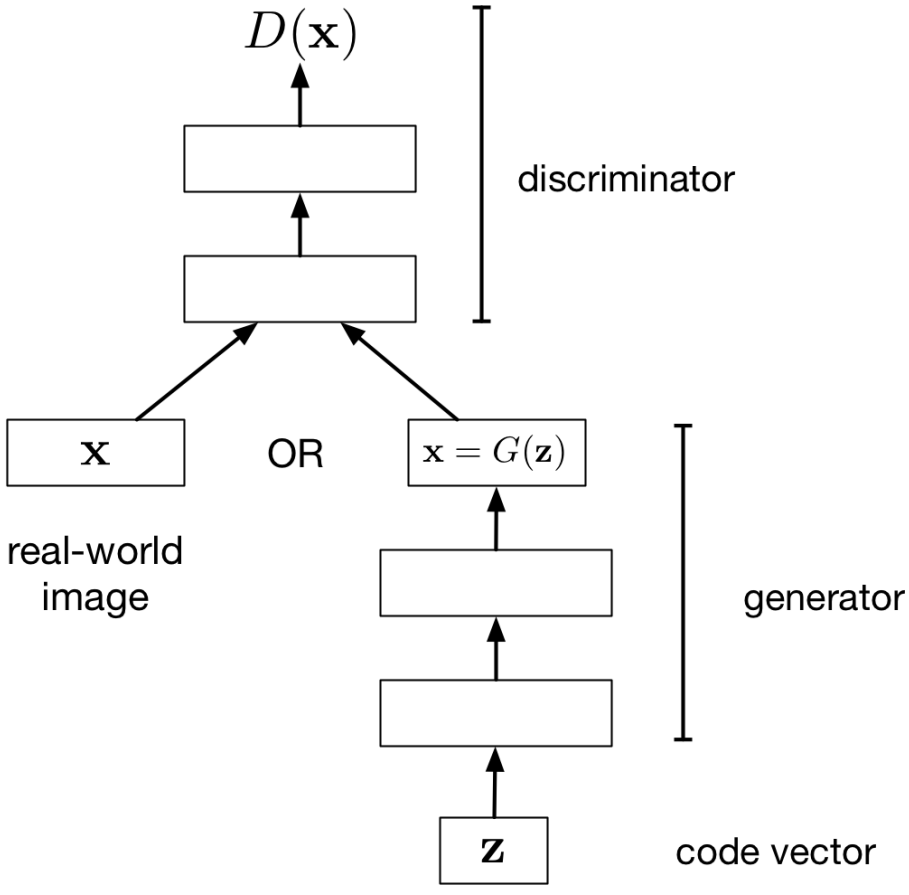
Each dimension of the code vector is sampled independently from a simple distribution, e.g. Gaussian or uniform.

This is fed to a (deterministic) generator network.

The network outputs an image.

- Generator Input: a random noise vector (Q: Why do we need to input noise?)
- Generator Output: a generated image

GAN Architecture



- Discriminator Input: an image
- Discriminator Output: a binary label (real vs fake)

GAN Loss function notation

Discriminator:

- D – the discriminator neural network
- θ – the trainable parameters of the discriminator (we'll write D_θ if we want to make the dependency clear)
- x – an image (either real or fake)
- $D(x)$ or $D_\theta(x)$ – the discriminator's determination of whether the image is real ($1 =$ real, $0 =$ fake)

Generator:

- G – the generator neural netowrk
- ϕ – the trainable parameters of the discriminator (we'll write G_ϕ if we want to make the dependency clear)
- z – a random noise vector
- $G(z)$ or $G_\phi(z)$ – a generated image

GAN Loss function notation

Discriminator:

- D – the discriminator neural network
- θ – the trainable parameters of the discriminator (we'll write D_θ if we want to make the dependency clear)
- x – an image (either real or fake)
- $D(x)$ or $D_\theta(x)$ – the discriminator's determination of whether the image is real ($1 =$ real, $0 =$ fake)

Generator:

- G – the generator neural netowrk
- ϕ – the trainable parameters of the discriminator (we'll write G_ϕ if we want to make the dependency clear)
- z – a random noise vector
- $G(z)$ or $G_\phi(z)$ – a generated image

Q: What does $D(G(z))$ mean?

GAN: Optimizing the generator

Tune **generator** weights to:

- **maximize** the probability that. . .
  - discriminator labels a generated image as real
  - Q: What loss function should we use?

GAN: Optimizing the generator

Tune **generator** weights to:

- **maximize** the probability that. . .
  - discriminator labels a generated image as real
  - Q: What loss function should we use?

We wish to tune ϕ to increase $D_\theta(G_\phi(z))$

GAN: Optimizing the generator

Tune **generator** weights to:

- **maximize** the probability that. . .
  - discriminator labels a generated image as real
  - Q: What loss function should we use?

We wish to tune ϕ to increase $D_\theta(G_\phi(z))$

$$\min_\phi \mathbf{E}_z \left[\log\left(1 - D_\theta(G_\phi(z))\right)\right]$$

GAN: Optimizing the discriminator

Tune **discriminator** weights to:

- **maximize** the probability that the
  - discriminator labels a real image as real
  - discriminator labels a generated image as fake
  - Q: What loss function should we use?

GAN: Optimizing the discriminator

Tune **discriminator** weights to:

- **maximize** the probability that the
  - discriminator labels a real image as real
  - discriminator labels a generated image as fake
  - Q: What loss function should we use?

We wish to tune θ to:

- decrease $D_\theta(G_\phi(z))$
- increase $D_\theta(x)$, where $x \sim \mathcal{D}$ (the data distribution)

GAN: Optimizing the discriminator

Tune **discriminator** weights to:

- **maximize** the probability that the
  - discriminator labels a real image as real
  - discriminator labels a generated image as fake
  - Q: What loss function should we use?

We wish to tune θ to:

- decrease $D_\theta(G_\phi(z))$
- increase $D_\theta(x)$, where $x \sim \mathcal{D}$ (the data distribution)

$$\max_\theta \mathbf{E}_{x \sim \mathcal{D}} \left[\log D_\theta(x)\right] + \mathbf{E}_z \left[\log\left(1 - D_\theta(G_\phi(z))\right)\right]$$

GAN Optimization Problem

We wish to optimize

$$\min_{\phi} \max_{\theta} \mathbf{E}_{x \sim \mathcal{D}} \left[\log D_\theta(x) \right] + \mathbf{E}_z \left[\log \left(1 - D_\theta(G_\phi(z)) \right) \right]$$

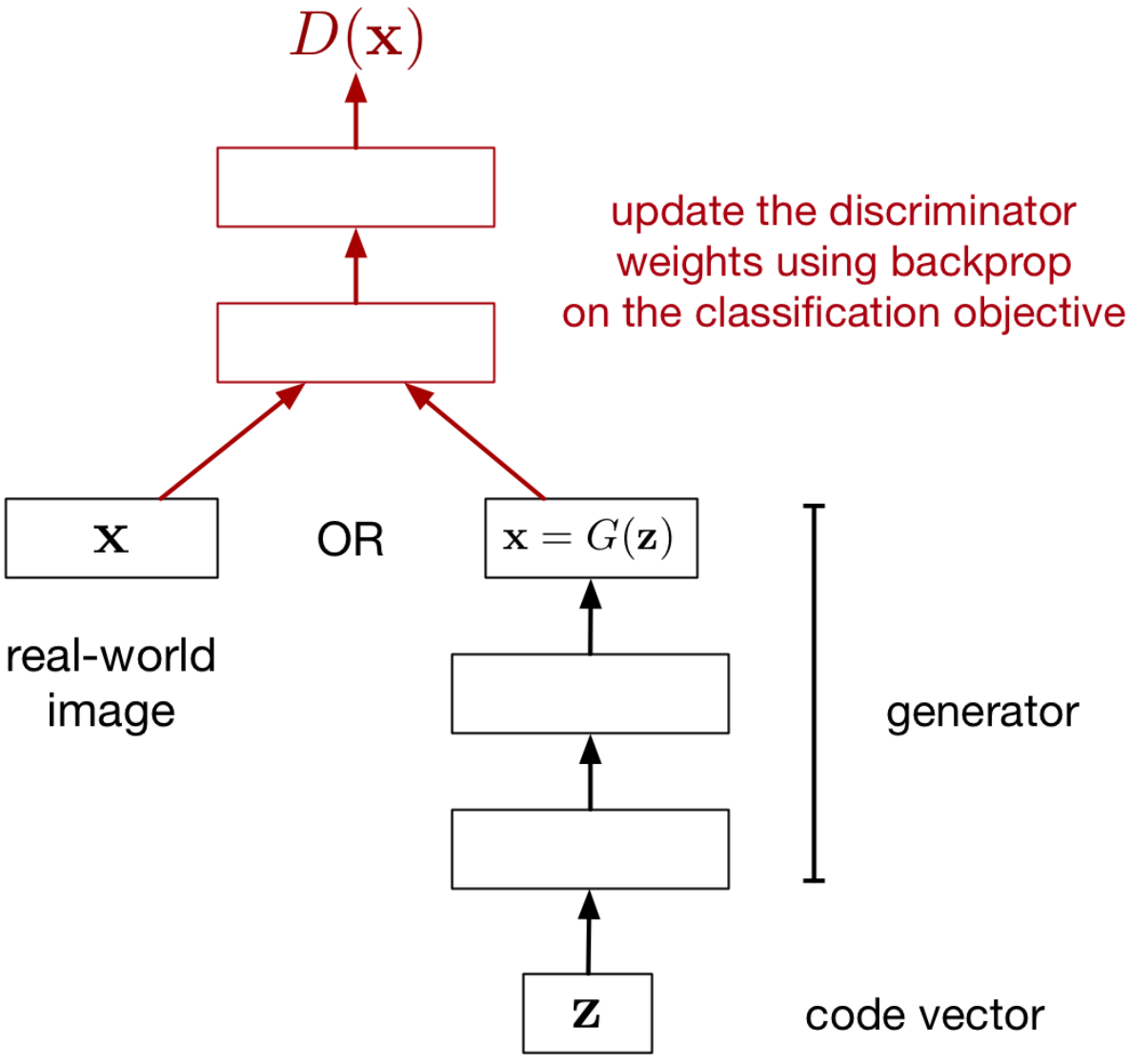
This is called the *minimax formulation* since the generator and discriminator are playing a zero-sum game against each other

Training

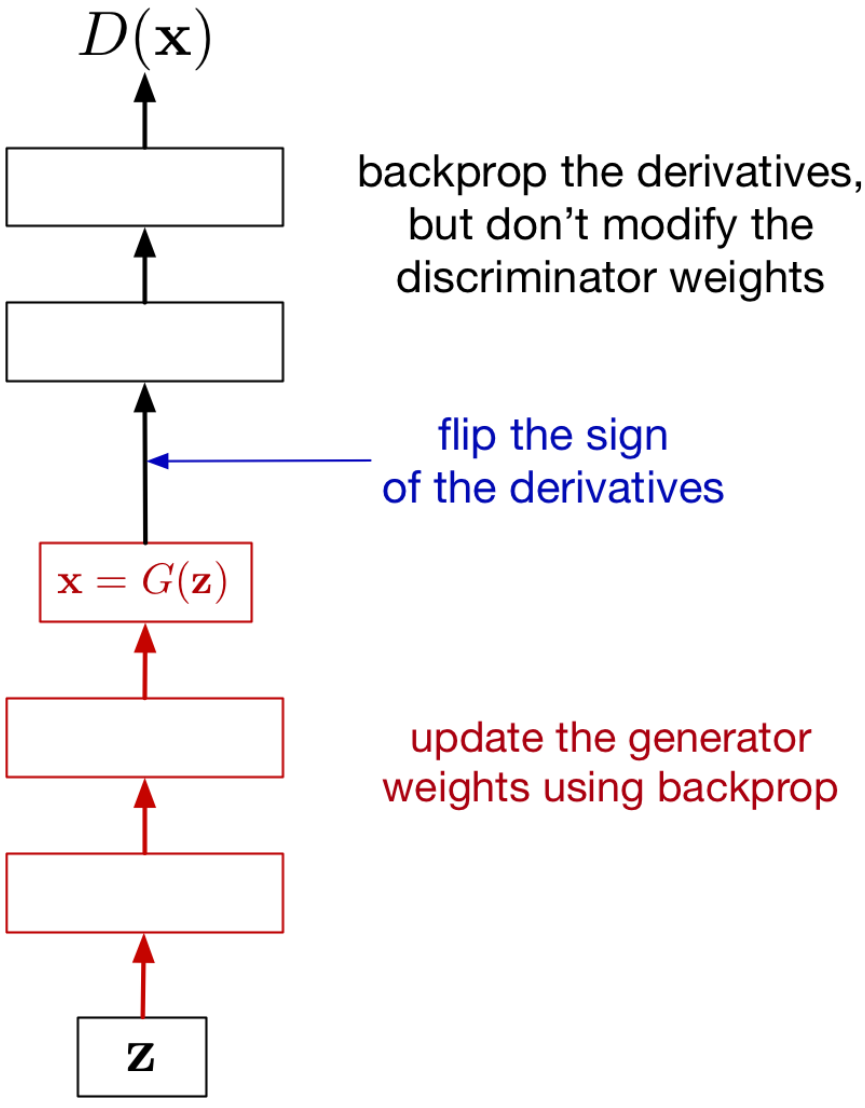
Alternate between:

- Training the discriminator
- Training the generator

Updating the discriminator

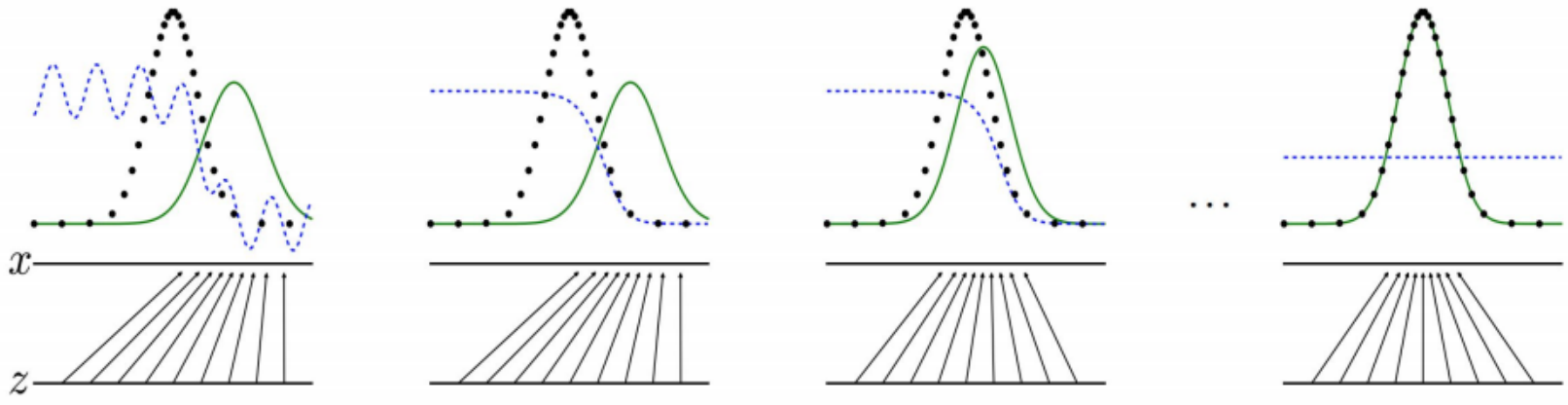


Updating the generator



GAN Alternating Training Visualized

Black dots is the data distribution $\mathcal{D}$, green line is the generator distribution $G(z)$, and blue dotted line is the discriminator:



1. The distributions $G(z)$ and $\mathcal{D}$ are quite different
2. The discriminator is updated to be able to better distinguish real vs fake
3. The generator is updated to be better match $\mathcal{D}$
4. If training is successful, $G(z)$ is indistinguisable from $\mathcal{D}$

A better cost function

- We introduced the minimax cost function for the generator:

$$\min_{\phi} \mathbf{E}_z \left[\log \left(1 - D_\theta(G_\phi(z)) \right) \right]$$

- One problem with this loss function is *saturation*
- Recall from classification. When the prediction is really wrong
  - "Logistic + square 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 generator cost function

Original minimax cost:

$$\min_{\phi} \mathbf{E}_z \left[\log \left(1 - D_\theta(G_\phi(z)) \right) \right]$$

Modified generator cost:

$$\min_{\phi} \mathbf{E}_z \left[-\log D_\theta(G_\phi(z)) \right]$$

Caveat before demo

- Can work very well and produces crisp, high-res images, but **difficult to train**!
- Difficult to numerically see whether there is progress
  - Plotting the "training curve" (discriminator/generator loss) doesn't help much
- Takes a long time to train (a long time before we see progress)
- To make the GAN train faster, we'll use:
  - LeakyReLU Activations instead of ReLU
  - Batch Normalization (later)

GAN: Discriminator

```
class Discriminator(nn.Module):
  def __init__(self):
    super().__init__()
    self.model = nn.Sequential(
      nn.Linear(28*28, 300),
      nn.LeakyReLU(0.2, inplace=True),
      nn.Linear(300, 100),
      nn.LeakyReLU(0.2, inplace=True),
      nn.Linear(100, 1))
  def forward(self, x):
    x = x.view(x.size(0), -1)
    out = self.model(x)
    return out.view(x.size(0))
```

Leaky Relu activation

Like a relu, but "leaks" data:

- $f(x) = x$ if $x >= 0$
- $f(x) = \alpha x$ if $x < 0$

Reason:

- Always have some information pass through in the forwards pass
- Always have some information pass back in the backwards pass
- Better weight updates during the backwards pass

GAN: Generator

```
class Generator(nn.Module):
  def __init__(self):
    super().__init__()
    self.model = nn.Sequential(
      nn.Linear(100, 300),
      nn.LeakyReLU(0.2, inplace=True),
      nn.Linear(300, 28*28),
      nn.Sigmoid())

  def forward(self, x):
    out = self.model(x).view(x.size(0), 1, 28, 28)
    return out
```

Training the Discriminator

```
#images = batch of images
#batch_size = images.size(0)
noise = torch.randn(batch_size, 100)
fake_images = generator(noise)
inputs = torch.cat([images, fake_images])
labels = torch.cat([torch.zeros(batch_size)), # real
                    torch.ones(batch_size)])  # fake
d_outputs = discriminator(inputs)
d_loss = criterion(d_outputs, labels)
```

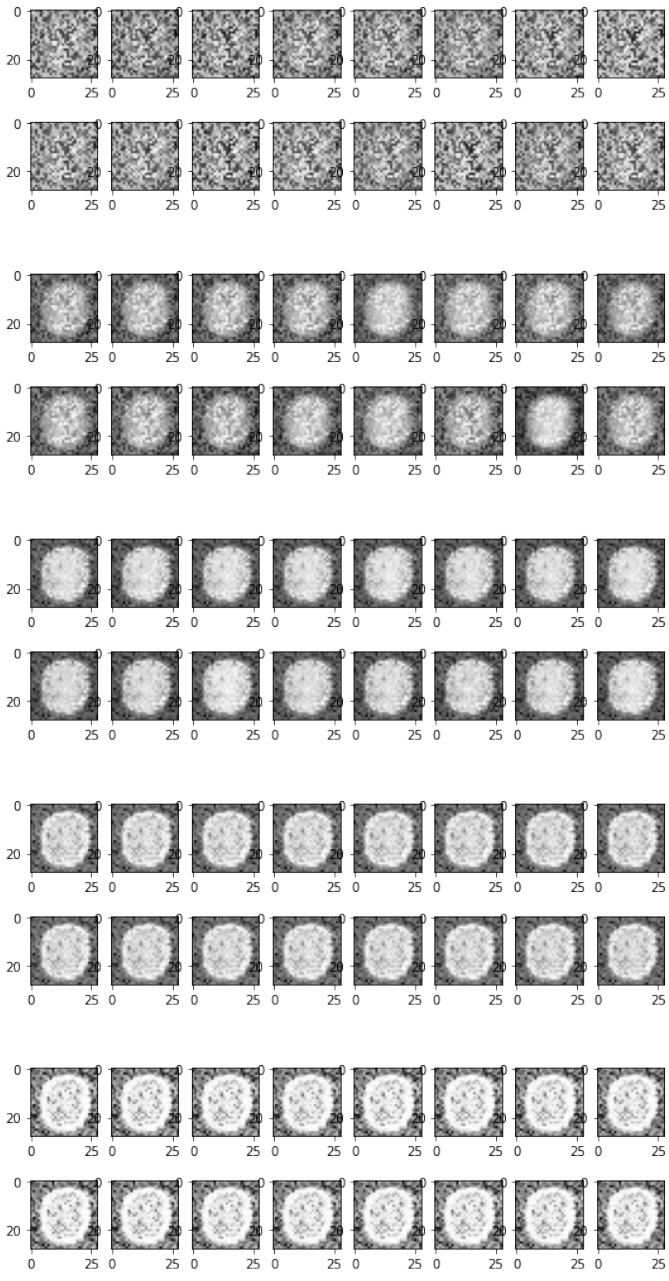
Note: The labels (`real=0`, `fake=1`) are opposite from our mathematical derivation. This version turns out to perform better

Training the Generator

```
noise = torch.randn(batch_size, 100)
fake_images = generator(noise)
outputs = discriminator(fake_images)
generator.zero_grad()
g_loss = criterion(outputs, torch.zeros(batch_size))
```

(Labels: `real=0`, `fake=1`)

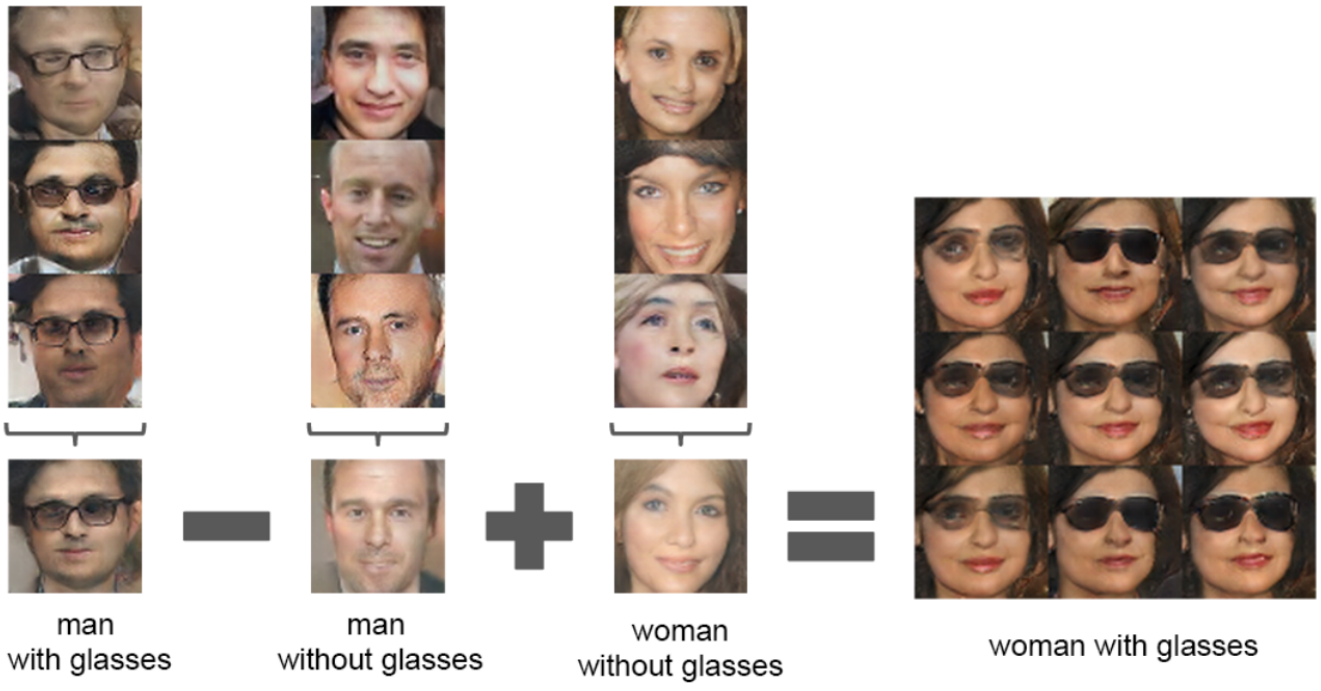
Let's look at the code!



GAN: Interpolation in z

Radford et al. (2016) https://arxiv.org/pdf/1511.06434.pdf

GAN: Vector Arithmetics in z



Radford et al. (2016) https://arxiv.org/pdf/1511.06434.pdf

GAN Samples (2019)

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 fidel

Mode Collapse

We don't actually know how well a GAN is modelling the distribution. One prominent issue is *mode collapse*

- The word "Mode" means "average"
- GAN model learns to generate one type of input data (e.g. only digit 1)
- Generating anything else leads to detection by discriminator
- Generator gets stuck in that local optima

Balance between Generator and Discriminator

If the discriminator is too good, then the generator will not learn due to **saturation**:

- Remember that we are using the discriminator like a "loss function" for the generator
- If the discriminator is too good, small changes in the generator weights won't change the discriminator output
- If small changes in generator weights make no difference, then we can't incrementally improve the generator

Wasserstein GAN (not on exam)

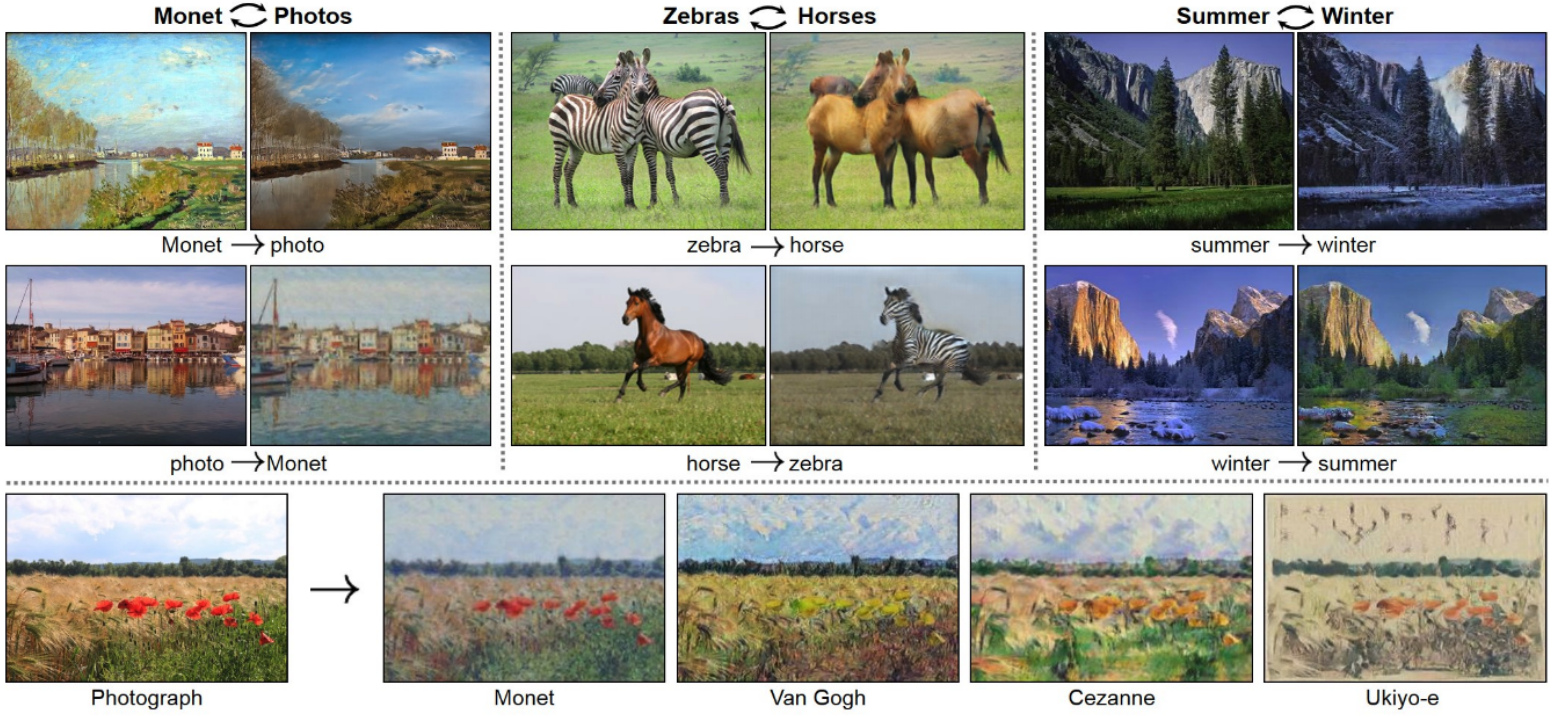
Idea: Use a different loss function.

`Arjovsky et al. (2017) Wasserstein GAN. https://arxiv.org/a`

- Use the *Wasserstein distance* between the generator distribution and the data distribution
- Reduces mode collapse, better measurement of progress

Style Transfer with Cycle GAN

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

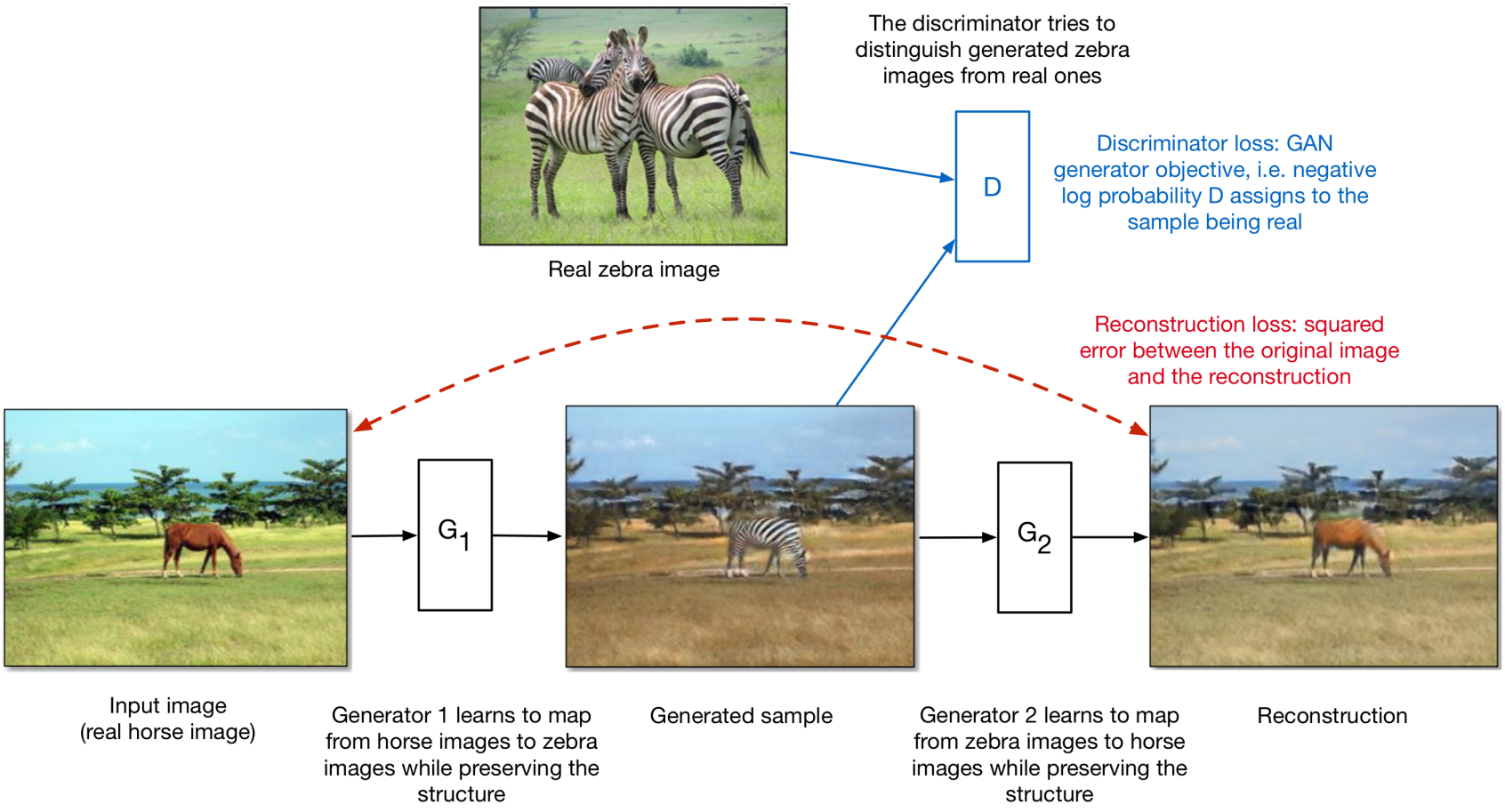


Data: Two unrelated collections of images, one for each style

Cycle GAN Idea

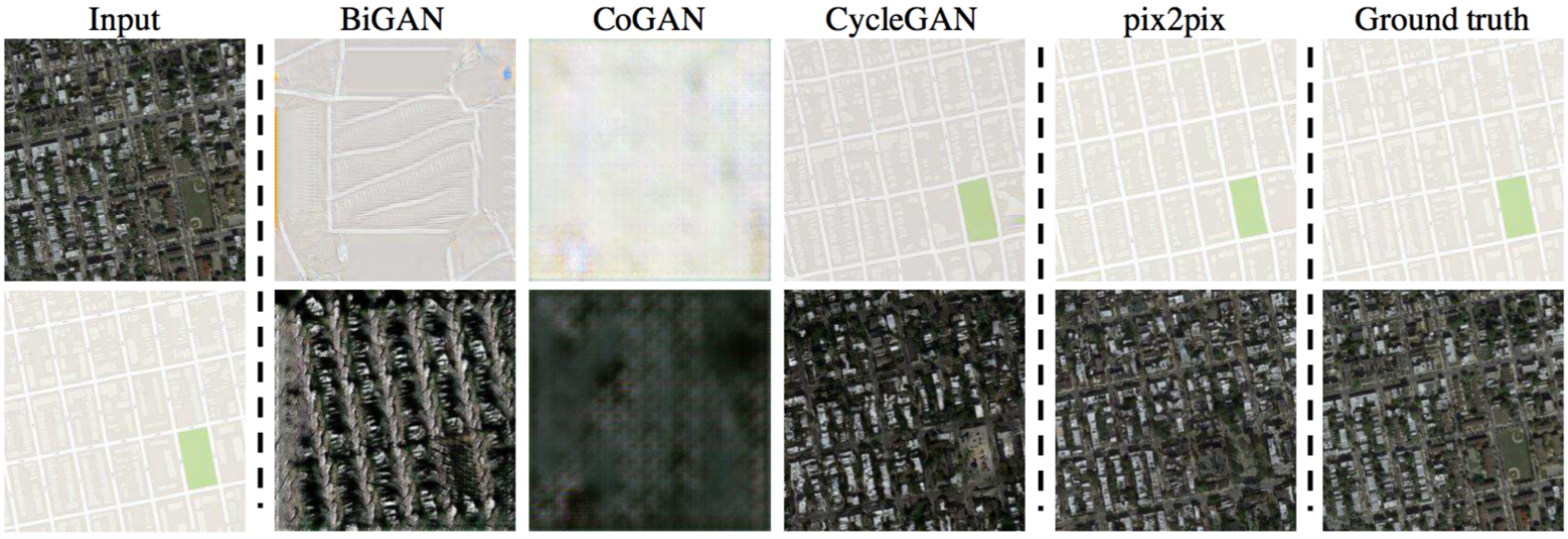
- If we had paired data (same content in both styles), this would be a supervised learning problem. But this is hard to find.
- The CycleGAN architecture learns to do it from unpaired data.
  - Train two different generator nets to go from style 1 to style 2, and vice versa.
  - Make sure the generated 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.

Cycle GAN Architecture

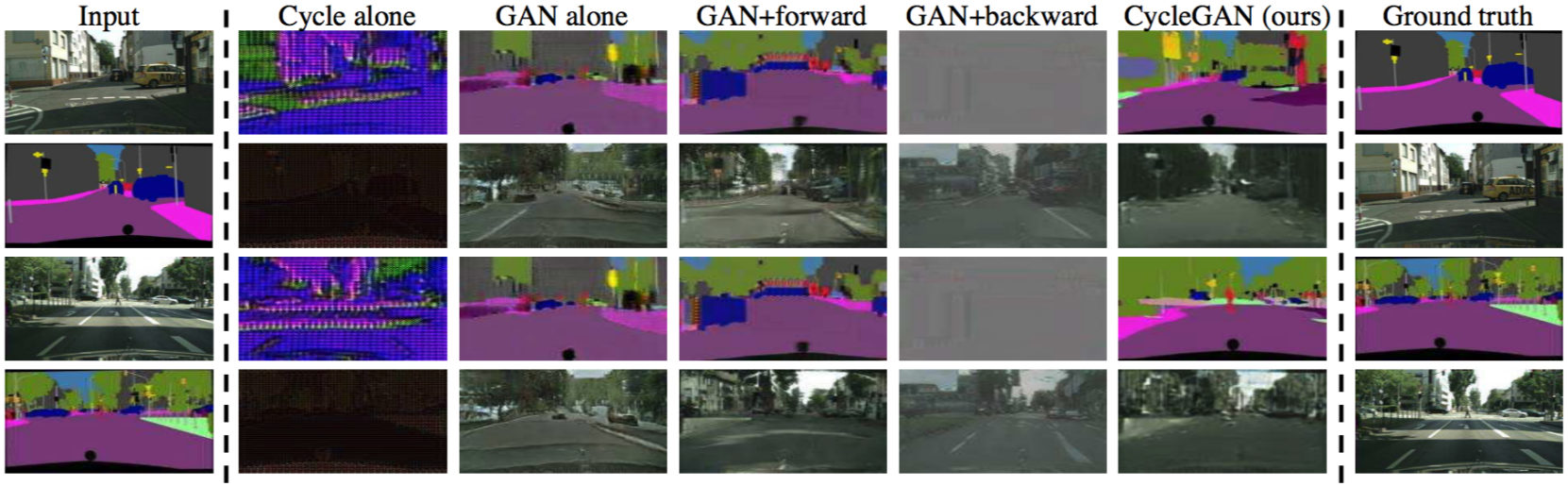


Total loss = discriminator loss + reconstruction loss

Cycle GAN: Aerial photos and maps



Cycle GAN: Road scenes and semantic segmentation



Course Evaluations