Agenda

Last class:

- Generative RNN
  - Training using teacher-forcing
  - Generating new sequences (temperature setting)

Today:

- Mid-Course Survey
- Progress Meeting
- Generative Adversarial Networks
- Adversarial Examples
Mid-Course Survey

https://forms.gle/WpcG8nirrNw9inXx9

- The survey is completely anonymous
- Only 5 questions – your response can be as short or as long as you want
- Help me improve your APS360 learning experience
- Use the survey to tell me anything you want
Project Progress Meeting
Why?

- So you have some one-on-one time with a ML researcher
- So that you’re on the right track
- So that everyone on your team is contributing
Scheduling

- Email your TA with several times that you are available
- If you really can’t find a time before July 15, please email me
- Send your TA your project proposal
Before the meeting. . .

- You should have all your data collected and cleaned
- You should be able to show your data to your TA, to give them some intuition about your problem
- There should be some code that the TA can look at

In the past, teams that have at least a baseline model implemented benefited most from the progress meeting.
Generative Models
Supervised vs Unsupervised Learning

From week 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

Learning to generate new data is an *unsupervised learning* task

- Yes, there is a loss function
- There is an auxiliary task that we know the answer to
- But there is no “label” or “ground truth” with respect to the **actual task** that we want to accomplish.
Example

Q: Are these supervised or unsupervised learning tasks?

- Task 1: Predict the next character given all the previous characters in a “Trump tweet”
- Task 2: Generate a new “Trump tweet”
Q: Are these supervised or unsupervised learning task?

- Task 1: Predict the next character given all the previous characters in a “Trump tweet”
- Task 2: Generate a new “Trump 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 Models

- 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
Review Autoencoders - Code!
Autoencoder outputs are Blurry

These faces are generated using a variant of autoencoders
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 – e.g. average the background

Can we use a better loss function?
Generative Adversarial Network
Generative Adversarial Network

Idea: train two networks

- **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 Architecture

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

- **Discriminator network:** try to distinguish between real and fake images
  - Input: an image
  - Output: a binary label (real vs fake)
Training GANs: two-player (minmax) game

Play a minmax game:

- the discriminator will try to do the best job it can
- the generator is set to make the discriminator as wrong as possible
Loss function for min-max game

Tune \textbf{discriminator} weights to:

- \textbf{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?
Loss function for min-max game

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?

Tune **generator** weights to:

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

Alternate between:

- Training the discriminator
- Training the generator
Caveat before we look at code

- Can work very well, but very 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 $x \geq 0$
- $f(x) = \alpha x$ if $x < 0$

You’ve implemented this in assignment 1.

**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

\[
\text{#images} = \text{batch of images} \\
\text{#batch_size} = \text{images.size(0)} \\
\text{noise} = \text{torch.randn(batch_size, 100)} \\
\text{fake_images} = \text{generator(noise)} \\
\text{inputs} = \text{torch.cat([images, fake_images])} \\
\text{labels} = \text{torch.cat([torch.zeros(batch_size)), \# real} \\
\text{torc.ones(batch_size)]) \# fake \\
\text{d_outputs} = \text{discriminator(inputs)} \\
\text{d_loss} = \text{criterion(d_outputs, labels)} \\
\text{(Labels: real=0, fake=1)}
\]
Training the Generator

\[
\begin{align*}
\text{noise} &= \text{torch.randn(batch\_size, 100)} \\
\text{fake\\_images} &= \text{generator(noise)} \\
\text{outputs} &= \text{discriminator(fake\\_images)} \\
\text{generator.zero\_grad}() \\
\text{g\\_loss} &= \text{criterion(outputs, torch.zeros(batch\_size))}
\end{align*}
\]

(Labels: real=0, fake=1)
Let’s run the code!
Mode Collapse

- Mode = “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
Batch Normalization

Normalization on input data helps training. But what about the hidden activations?

- **Training time**: normalize activations based on mini-batch statistics, and keep track of those statistics
- **Test time**: normalize activations based on saved statistics
Balance between Generator and Discriminator

If the discriminator is too good, then the generator will not learn

- 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
GAN now

CycleGAN

CartoonGAN

Figure 5. Some results of different artistic styles generated by CartoonGAN. (a) Input real-world photos. (b) Makoto Shinkai style. (c) Miyazaki Hayao style.

Adversarial Examples
What is this a picture of?

“panda” 57.7% confidence

+ ε

= “gibbon” 99.3% confidence
What is this a picture of?

“pig” + 0.005 x = “airliner”
Adversarial attack

**Goal:** Choose a small perturbation $\epsilon$ on an image $x$ so that a neural network $f$ misclassifies $x + \epsilon$.

**Approach:**
Use the same optimization process to choose $\epsilon$ to minimize the probability that

$$f(x + \epsilon) = \text{correct class}$$

We are treating $\epsilon$ as the **parameters**.
Targeted vs Non-Targeted Adversarial Attack

Non-targeted attack
Minimize the probability that $f(x + \epsilon) = \text{correctclass}$

Targeted attack
Maximize the probability that $f(x + \epsilon) = \text{targetclass}$
White-box Adversarial Attack

- Assumes that the model is known
- We need to know the architectures and weights of \( f \) to optimize \( \epsilon \)
Black-box Adversarial Attack

- Don’t know the architectures and weights of $f$ to optimize $\epsilon$
- Substitute model mimicking target model with known, differentiable function
  - adversarial attacks often transfer across models!
Printed Objects

https://openai-public.s3-us-west-2.amazonaws.com/blog/2017-07/robust-adversarial-examples/iphone.mp4
3D Objects

https://www.youtube.com/watch?v=piYnd_wYIT8
Printed Pictures

https://www.youtube.com/watch?v=MIbFvK2S9g8&feature=youtu.be
Defenses Against Adversarial Attack

- Active area of research
Failed Defenses

- Generative pre-training
- Adding noise at test time
- Averaging many models
- Weight decay
- Adding noise at training time
- Adding adversarial noise at training time
- Dropout