Introduction
Spectrum of Low-Labeled Learning

Supervised Learning
- Train Input: \( \{X, Y\} \)
- Learning output: \( f : X \rightarrow Y, P(y|x) \)
- e.g. classification

Unsupervised Learning
- Input: \( \{X\} \)
- Learning output: \( P(x) \)
- Example: Clustering, density estimation, etc.
Traditional unsupervised learning methods:

- **Modeling** $P(x)$
- **Comparing/Grouping**
- **Representation Learning**

- Density estimation
- Clustering
- Principal Component Analysis

Similar in deep learning, but **from neural network/learning perspective**

What to Learn?
Discriminative vs. Generative Models

- Discriminative models model $P(y|x)$
  - Example: Model this via neural network, SVM, etc.

- Generative models model $P(x)$

---

**Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks**
**Discriminative vs. Generative Models**

- Discriminative models model $P(y|x)$
  - Example: Model this via neural network, SVM, etc.

- Generative models model $P(x)$
  - We can parameterize our model as $P(x, \theta)$ and use maximum likelihood to optimize the parameters given an unlabeled dataset:
    
    $$
    \theta^* = \arg \max_{\theta} \prod_{i=1}^{m} p_{\text{model}} \left( x^{(i)} ; \theta \right) 
    $$
    
    $$
    = \arg \max_{\theta} \log \prod_{i=1}^{m} p_{\text{model}} \left( x^{(i)} ; \theta \right) 
    $$
    
    $$
    = \arg \max_{\theta} \sum_{i=1}^{m} \log p_{\text{model}} \left( x^{(i)} ; \theta \right) . 
    $$

- They are called generative because they can often generate samples
  - Example: Multivariate Gaussian with estimated parameters $\mu, \sigma$

*Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks*
Generative Models

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks
PixelRNN & PixelCNN
Generative Models

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks
We can use chain rule to decompose the joint distribution

- Factorizes joint distribution into a product of conditional distributions
  - Similar to Bayesian Network (factorizing a joint distribution)
  - Similar to language models!
- Requires some ordering of variables (edges in a probabilistic graphical model)
- We can estimate this conditional distribution as a neural network

\[
p(x) = \prod_{i=1}^{n^2} p(x_i | x_1, \ldots, x_{i-1})
\]

Oord et al., Pixel Recurrent Neural Networks
\[ p(s) = p(w_1, w_2, \ldots, w_n) \]
\[ = p(w_1) p(w_2 \mid w_1) p(w_3 \mid w_1, w_2) \ldots p(w_n \mid w_{n-1}, \ldots, w_1) \]
\[ = \prod_{i} p(w_i \mid w_{i-1}, \ldots, w_1) \]

Modeling language as a sequence
Language modeling involves estimating a probability distribution over sequences of words.

\[ p(s) = p(w_1, w_2, \ldots, w_n) = \prod_i p(w_i \mid w_{i-1}, \ldots, w_1) \]

RNNs are a family of neural architectures for modeling sequences.
\[ p(x) = \prod_{i=1}^{n^2} p(x_i | x_1, \ldots, x_{i-1}) \]

\[ p(x) = p(x_1) \prod_{i=2}^{n^2} p(x_i | x_1, \ldots, x_{i-1}) \]
$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_1) \prod_{i=1}^{n^2} p(x_i|x_1, \ldots, x_{i-1})$

- **Training:**
  - We can train similar to language models: Teacher/student forcing
  - Maximum likelihood approach

- **Downsides:**
  - Slow sequential generation process
  - Only considers few context pixels

*Oord et al., Pixel Recurrent Neural Networks*
**Idea:** Represent conditional distribution as a convolution layer!

- Considers larger context (receptive field)

- Practically can be implemented by applying a mask, zeroing out “future” pixels

- Faster training but still slow generation
  - Limited to smaller images

_Oord et al., Conditional Image Generation with PixelCNN Decoders_
Example Results: Image Completion (PixelRNN)

Oord et al., Conditional Image Generation with PixelCNN Decoders
Example Images (PixelCNN)

Oord et al., Conditional Image Generation with PixelCNN Decoders
Generative Adversarial Networks (GANs)
Generative Models

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks
Implicit generative models do not actually learn an explicit model for $p(x)$

Instead, learn to \textit{generate samples} from $p(x)$

- Learn good feature representations
- Perform data augmentation
- Learn world models (a simulator!) for reinforcement learning

How?

- \textbf{Learn to sample} from a neural network output
- \textbf{Adversarial training} that uses one network’s predictions to train the other (dynamic loss function!)
- \textbf{Lots of tricks} to make the optimization more stable
We would like to sample from $p(x)$ using a neural network.

**Idea:**
- Sample from a simple distribution (Gaussian)
- Transform the sample to $p(x)$
- Input can be a vector with (independent) Gaussian random numbers
- We can use a CNN to generate images!
**Goal:** We would like to generate *realistic* images. How can we drive the network to learn how to do this?

**Idea:** Have *another* network try to distinguish a real image from a generated (fake) image

**Why?** Signal can be used to determine how well it’s doing at generation
Generative Adversarial Networks (GANs)

Vector of Random Numbers

Generator

Discriminator

Cross-entropy (Real or Fake?) We know the answer (self-supervised)

Mini-batch of real & fake data

Question: What loss functions can we use (for each network)?

- **Generator**: Update weights to improve realism of generated images
- **Discriminator**: Update weights to better discriminate
Since we have two networks competing, this is a mini-max two player game

- Ties to game theory
- Not clear what (even local) Nash equilibria are for this game
Since we have two networks competing, this is a mini-max two player game.

- Ties to game theory
- Not clear what (even local) Nash equilibria are for this game

The full mini-max objective is:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log (1 - D(G(z)))]$$
\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]
\]

- where \(D(x)\) is the discriminator outputs probability ([0,1]) of real image
- \(x\) is a real image and \(G(z)\) is a generated image

- The discriminator wants to maximize this:
  - \(D(x)\) is pushed up (to 1) because \(x\) is a real image
  - \(1 - D(G(z))\) is also pushed up to 1 (so that \(D(G(z))\) is pushed down to 0)
  - In other words, discriminator wants to classify real images as real (1) and fake images as fake (0)
\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]
\]

- where \(D(x)\) is the discriminator outputs probability ([0,1]) of real image
- \(x\) is a real image and \(G(z)\) is a generated image

The generator wants to minimize this:
- \(1 - D(G(z))\) is pushed down to 0 (so that \(D(G(z))\) is pushed up to 1)
- This means that the generator is fooling the discriminator, i.e. succeeding at generating images that the discriminator can’t discriminate from real
Since we have two networks competing, this is a mini-max two player game

- Ties to game theory
- Not clear what (even local) Nash equilibria are for this game

The full mini-max objective is:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}}(x) [\log D(x)] + \mathbb{E}_{z \sim p_z}(z) [\log (1 - D(G(z)))]$$

**Generator minimizes**

- where $D(x)$ is the discriminator outputs probability ([0,1]) of real image
- $x$ is a real image and $G(z)$ is a generated image

**Sample from fake**

- How well discriminator does (0 for fake)
Since we have two networks competing, this is a mini-max two player game
- Ties to game theory
- Not clear what (even local) Nash equilibria are for this game

The full mini-max objective is:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log (1 - D(G(z)))]$$

**Discriminator maximizes**
- Sample from real
- How well discriminator does (1 for real)
- where $D(x)$ is the discriminator outputs probability ([0,1]) of real image

**Sample from fake**
- How well discriminator does (0 for fake)
- $x$ is a real image and $G(z)$ is a generated image

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks
Generative Adversarial Networks (GANs)

Vector of Random Numbers

Generator

Discriminator

Mini-batch of real & fake data

Cross-entropy (Real or Fake?)
We know the answer (self-supervised)

\[ \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right) . \]

Generator Loss

\[ \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D \left( x^{(i)} \right) + \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right) \right] . \]

Discriminator Loss
The generator part of the objective does not have good gradient properties

\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log (1 - D(G(z)))]
\]

- High gradient when \(D(G(z))\) is high (that is, discriminator is wrong)
- We want it to improve when samples are bad (discriminator is right)

Alternative objective, maximize:

\[
\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log D_{\theta_d}(G_{\theta_g}(z))
\]
Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, $k$, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations do
  for $k$ steps do
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Sample minibatch of $m$ examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
    • Update the discriminator by ascending its stochastic gradient:
      \[ \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D(x^{(i)}) + \log \left(1 - D(G(z^{(i)}))\right) \right]. \]
  end for
  • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  • Update the generator by descending its stochastic gradient:
    \[ \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D(G(z^{(i)}))\right). \]
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Goodfellow, NeurIPS 2016 Generative Adversarial Nets
Generative Adversarial Networks (GANs)

- Vector of Random Numbers
- Generator
- Discriminator
- Cross-entropy (Real or Fake?)

At the end, we have:
- An *implicit* generative model!
- Features from discriminator

Mini-batch of real & fake data

An implicit generative model!
Features from discriminator

We know the answer (self-supervised)
Early Results

- Low-resolution images but look decent!
- Last column are nearest neighbor matches in dataset
GANs are very difficult to train due to the mini-max objective

Advancements include:
- More stable architectures
- Regularization methods to improve optimization
- Progressive growing/training and scaling
Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.
Training GANs is difficult due to:

- Minimax objective – For example, what if generator learns to memorize training data (no variety) or only generates part of the distribution?
- Mode collapse – Capturing only some modes of distribution

Several theoretically-motivated regularization methods
- Simple example: Add noise to real samples!

\[
\lambda \cdot \mathbb{E}_{x \sim P_{real}, \delta \sim N_d(0, cI)} \left[ \left\| \nabla_x D_\theta(x + \delta) \right\| - k \right]^2
\]

*Kodali et al., On Convergence and Stability of GANs (also known as How to Train your DRAGAN)*
Generative Adversarial Nets: Convolutional Architectures

Samples from the model look much better!

Radford et al, ICLR 2016
Generative Adversarial Nets: Convolutional Architectures

Interpolating between random points in latent space

Radford et al, ICLR 2016
Figure 4: Samples from our model with truncation threshold 0.5 (a-c) and an example of class leakage in a partially trained model (d).
Video Generation

https://www.youtube.com/watch?v=PCBTZh41Ris
A few other examples:

- Deep nostalgia: https://www.myheritage.com/deep-nostalgia
- High-resolution outputs: https://compvis.github.io/taming-transformers/
GANs

Don't work with an explicit density function
Take game-theoretic approach: learn to generate from training distribution through 2-player game

Pros:
- Beautiful, state-of-the-art samples!

Cons:
- Trickier / more unstable to train
- Can't solve inference queries such as p(x), p(z|x)

Active areas of research:
- Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
- Conditional GANs, GANs for all kinds of applications
Mode Collapse

• Optimization of GANs is tricky
  – Not guaranteed to find Nash equilibrium

• Large number of methods to combat:
  – Use history of discriminators
  – Regularization
  – Different divergence measures
Application: Data Augmentation

Low-Shot Learning from Imaginary Data, Yu-Xiong Wang, Ross Girshick, Martial Hebert, Bharath Hariharan
Application: Domain Adaptation

- **Idea:** Train a model on *source* data and adapt to *target* data using unlabeled examples from target data.
Approach

Pre-training
- Source images + labels
- Source CNN
- Classifier
  - class label

Adversarial Adaptation
- Source images
- Target images
- Source CNN
- Target CNN
- Discriminator
  - domain label

Testing
- Target image
- Target CNN
- Classifier
  - class label

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST $\rightarrow$ USPS $[773 \rightarrow 1105]$</th>
<th>USPS $\rightarrow$ MNIST $[105 \rightarrow 773]$</th>
<th>SVHN $\rightarrow$ MNIST $[14\times15 \rightarrow 773]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source only</td>
<td>$0.752 \pm 0.016$</td>
<td>$0.571 \pm 0.017$</td>
<td>$0.601 \pm 0.011$</td>
</tr>
<tr>
<td>Gradient reversal</td>
<td>$0.771 \pm 0.018$</td>
<td>$0.730 \pm 0.020$</td>
<td>$0.739$ [16]</td>
</tr>
<tr>
<td>Domain confusion</td>
<td>$0.791 \pm 0.005$</td>
<td>$0.665 \pm 0.033$</td>
<td>$0.681 \pm 0.003$</td>
</tr>
<tr>
<td>CoGAN</td>
<td>$0.912 \pm 0.008$</td>
<td>$0.891 \pm 0.008$</td>
<td>did not converge</td>
</tr>
<tr>
<td>ADDA (Ours)</td>
<td>$0.894 \pm 0.002$</td>
<td>$0.901 \pm 0.008$</td>
<td>$0.760 \pm 0.018$</td>
</tr>
</tbody>
</table>

Table 2: Experimental results on unsupervised adaptation among MNIST, USPS, and SVHN.
Aside: Other ways to Align

[GANIN et al., JMLR 2016]
Generative Adversarial Networks (GANs) can produce amazing images!

Several drawbacks
- High-fidelity generation heavy to train
- Training can be unstable
- No explicit model for distribution

Larger number of extensions:
- GANs conditioned on labels or other information
- Adversarial losses for other applications