Topics:

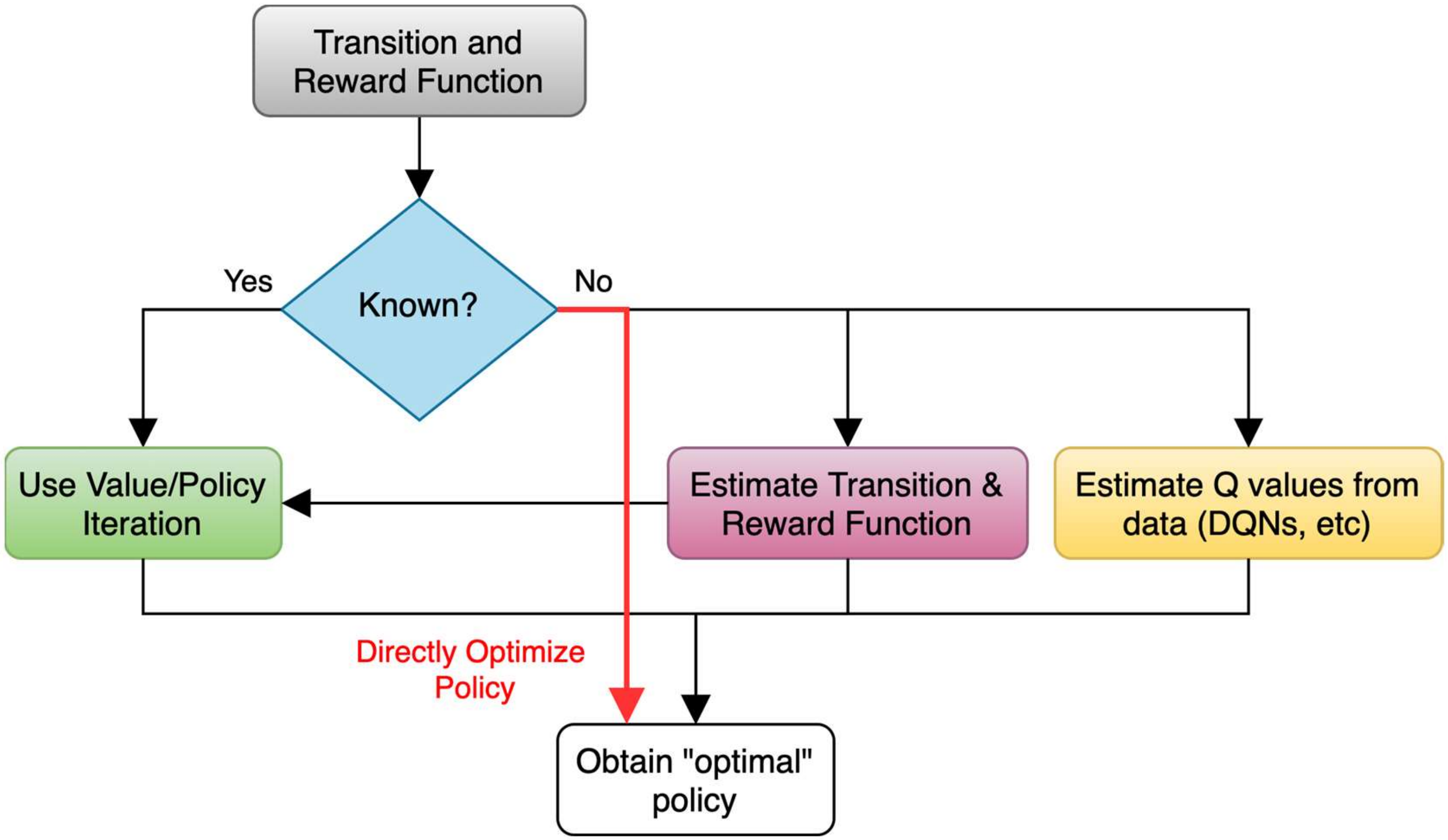
- Generative Models / Generative Adversarial Networks

CS 4644-DL / 7643-A
ZSOLT KIRA

- **Projects!**
  - Due May 1rd (May 3th with grace period)
  - Cannot extend due to grade deadlines!

- Outline of rest of course:

W14: Apr 11	Generative Adversarial Networks (GANs).	•NIPS 2016 Tutorial: Generative Adversarial Networks
W14: Apr 13	Guest Lecture by Ishan Misra (Meta) on Self-Supervised Learning	
W15: Apr 18	Variational Autoencoders (VAEs)	•Tutorial on Variational Autoencoders
W15: Apr 20	Guest Lecture by Joanne Truong on Embodied AI	



Overview

Georgia Tech

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)}[\mathcal{R}(\tau)]$$

$$= \nabla_\theta \int \pi_\theta(\tau)\mathcal{R}(\tau)d\tau$$

Expectation as integral

$$= \int \nabla_\theta \pi_\theta(\tau)\mathcal{R}(\tau)d\tau$$

Exchange integral and gradient

$$= \int \nabla_\theta \pi_\theta(\tau) \cdot \frac{\pi_\theta(\tau)}{\pi_\theta(\tau)} \cdot \mathcal{R}(\tau)d\tau$$

$$= \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)\mathcal{R}(\tau)d\tau$$

$$\nabla_\theta \log \pi(\tau) = \frac{\nabla_\theta \pi(\tau)}{\pi(\tau)}$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)\mathcal{R}(\tau)]$$

Deriving The Policy Gradient

Georgia Tech

Actor-critic

- In general, replacing the policy evaluation or the "critic" leads to different flavors of the actor-critic
  - REINFORCE: $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)\mathcal{R}(s,a)\right]$
  - Q – Actor Critic $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)Q^{\pi_\theta}(s,a)\right]$
  - Advantage Actor Critic: $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)A^{\pi_\theta}(s,a)\right]$

$= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$

"how much better is an action than expected?

Summary

- **Policy gradients**: very general but suffer from high variance so requires a lot of samples. **Challenge**: sample-efficiency
- **Q-learning**: does not always work but when it works, usually more sample-efficient. **Challenge**: exploration

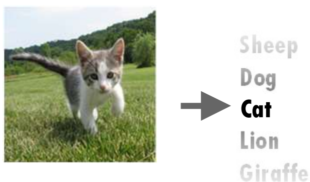
- Guarantees:
  - **Policy Gradients**: Converges to a local minima of J(θ), often good enough!
  - **Q-learning**: Zero guarantees since you are approximating Bellman equation with a complicated function approximator

Introduction

Spectrum of Low-Labeled Learning

Supervised Learning

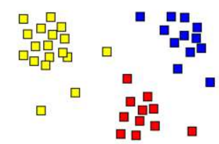
- Train Input: $\{X, Y\}$
- Learning output: $f : X \rightarrow Y$, $P(y|x)$
- e.g. classification



Less Labels →

Unsupervised Learning

- Input: $\{X\}$
- Learning output: $P(x)$
- Example: Clustering, density estimation, etc.



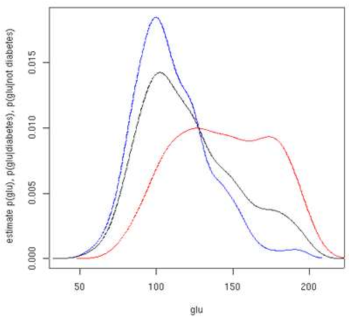
Supervised Learning

Input		Task		Output	Type
x	→	Classification	→	y	Discrete
x	→	Regression	→	y	Continuous

Unsupervised Learning

Input		Task		Output	Type
x	→	Clustering	→	c	Discrete
x	→	Dimensionality Reduction	→	z	Continuous
x	→	Density Estimation	→	p(x)	On simplex

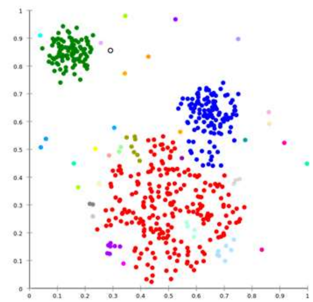
Traditional unsupervised learning methods:



Density estimation

Modeling $P(x)$

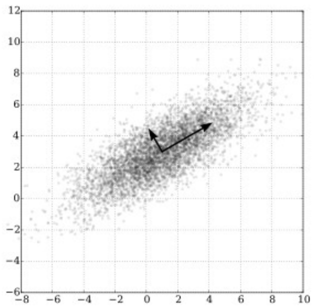
Deep Generative Models



Clustering

Comparing/ Grouping

Metric learning & clustering



Principal Component Analysis

Representation Learning

Almost all deep learning!

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

What to Learn?

Georgia Tech

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

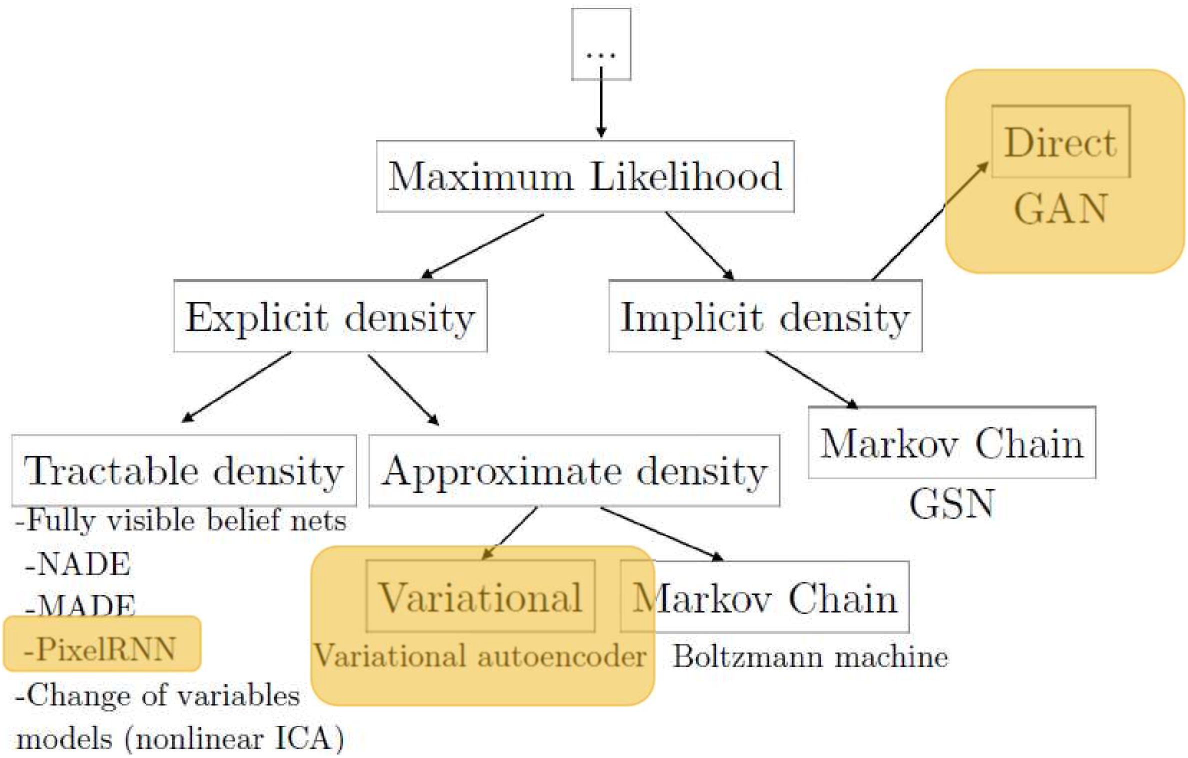
- Discriminative models model $P(y|x)$
  - Example: Model this via neural network, SVM, etc.

- 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:

$$\begin{aligned}\boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\arg\max} \prod_{i=1}^{m} p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) \\ &= \underset{\boldsymbol{\theta}}{\arg\max} \log \prod_{i=1}^{m} p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) \\ &= \underset{\boldsymbol{\theta}}{\arg\max} \sum_{i=1}^{m} \log p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right).\end{aligned}$$

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

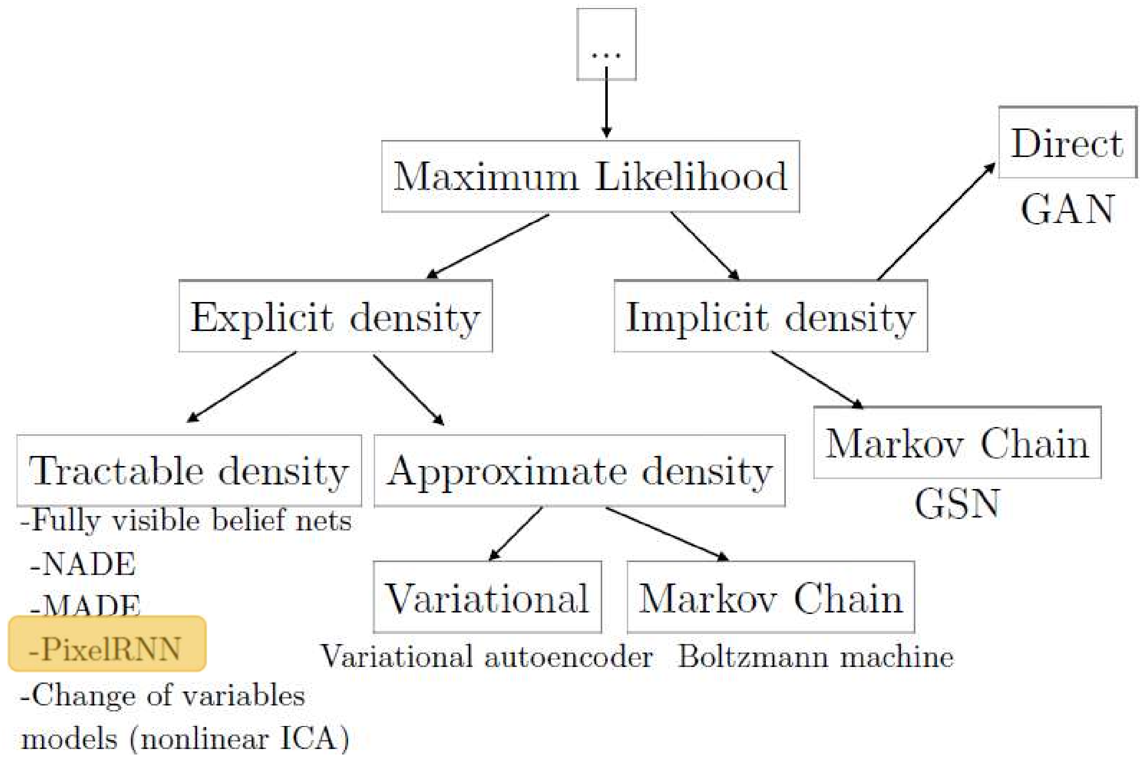
Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks



Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks

Generative Models

PixelRNN &
PixelCNN



Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks

Generative Models

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!

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

- Requires some *ordering* of variables (edges in a probabilistic graphical model)
- We can estimate this conditional distribution as a neural network

Oord et al., Pixel Recurrent Neural Networks

$$
\begin{aligned}
p(\boldsymbol{s}) &= p(w_1, w_2, \ldots, w_n) \\
&= p(w_1)\, p(w_2 \mid w_1)\, p(w_3 \mid w_1, w_2) \cdots p(w_n \mid w_{n-1}, \ldots, w_1) \\
&= \prod_i p(\underbrace{w_i}_{\text{next word}} \mid \underbrace{w_{i-1}, \ldots, w_1}_{\text{history}})
\end{aligned}
$$

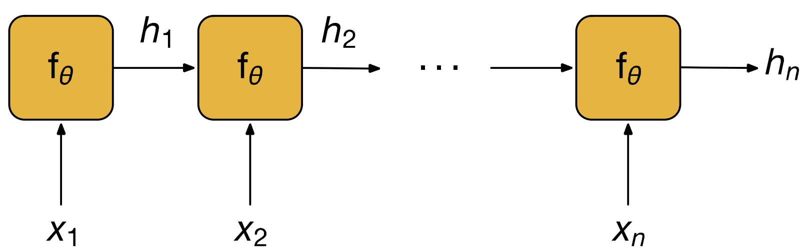
Modeling language as a sequence

Georgia Tech

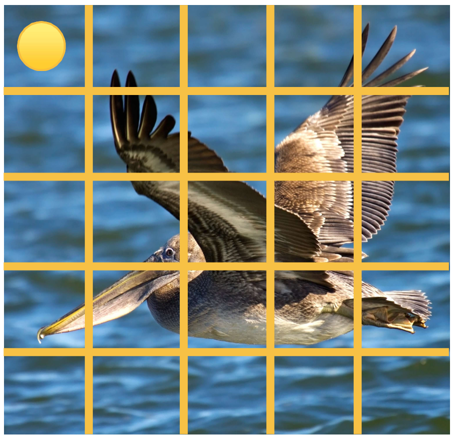
- Language modeling involves estimating a probability distribution over sequences of words.

$$p(\boldsymbol{s}) = p(w_1, w_2, \ldots, w_n) = \prod_i p(\underbrace{w_i}_{\text{next word}} \mid \underbrace{w_{i-1}, \ldots, w_1}_{\text{history}})$$

- RNNs are a family of neural architectures for modeling sequences.



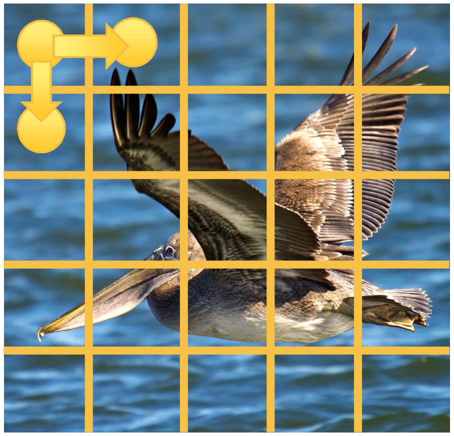
Language Models as an RNN

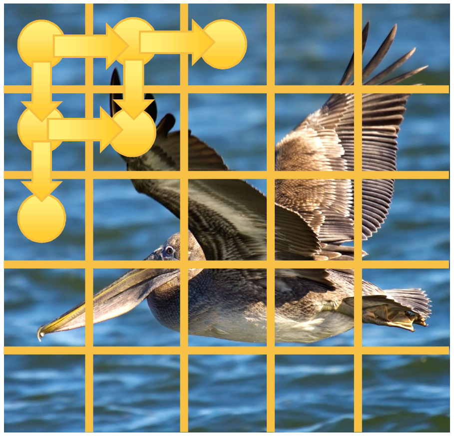


$$p(x) = \prod_{i=1}^{n^2} p(x_i | x_1, \dots, x_{i-1})$$

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

Oord et al., Pixel Recurrent Neural Networks



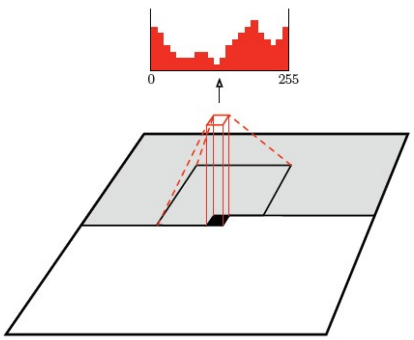


$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_1)\prod_{i=1}^{n^2} p(x_i|x_1, \dots, x_{i-1})$$

- 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

Factorized Models for Images



1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0

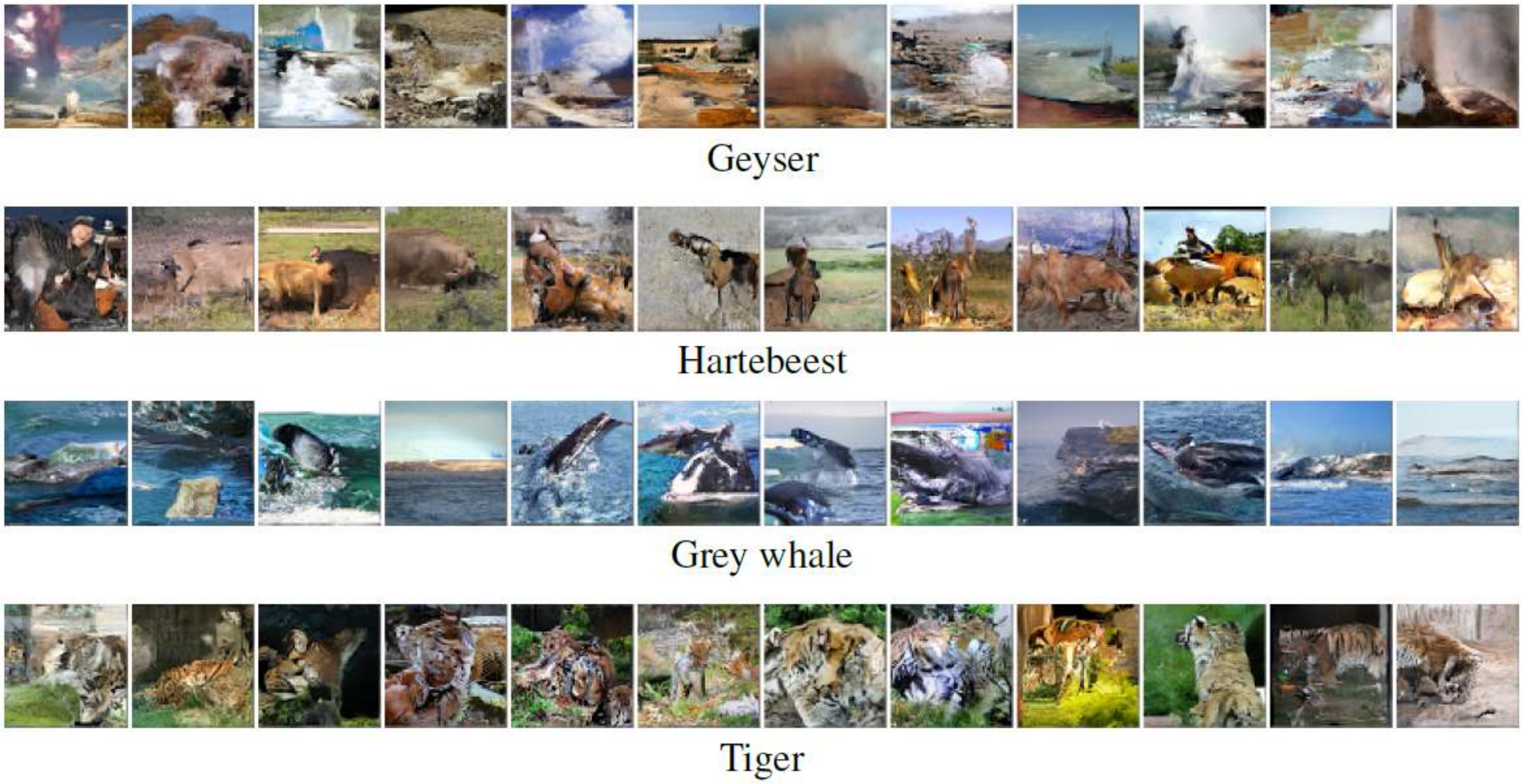
- **Idea:** Represent conditional distribution as a convolution layer!
- Considers larger context (receptive field)
- Practically can be implemented by applying a mask, zeroing out “future” pixels
- Faster training but still slow generation
  - Limited to smaller images

Oord et al., Conditional Image Generation with PixelCNN Decoders

Pixel CNN

Oord et al., Conditional Image Generation with PixelCNN Decoders

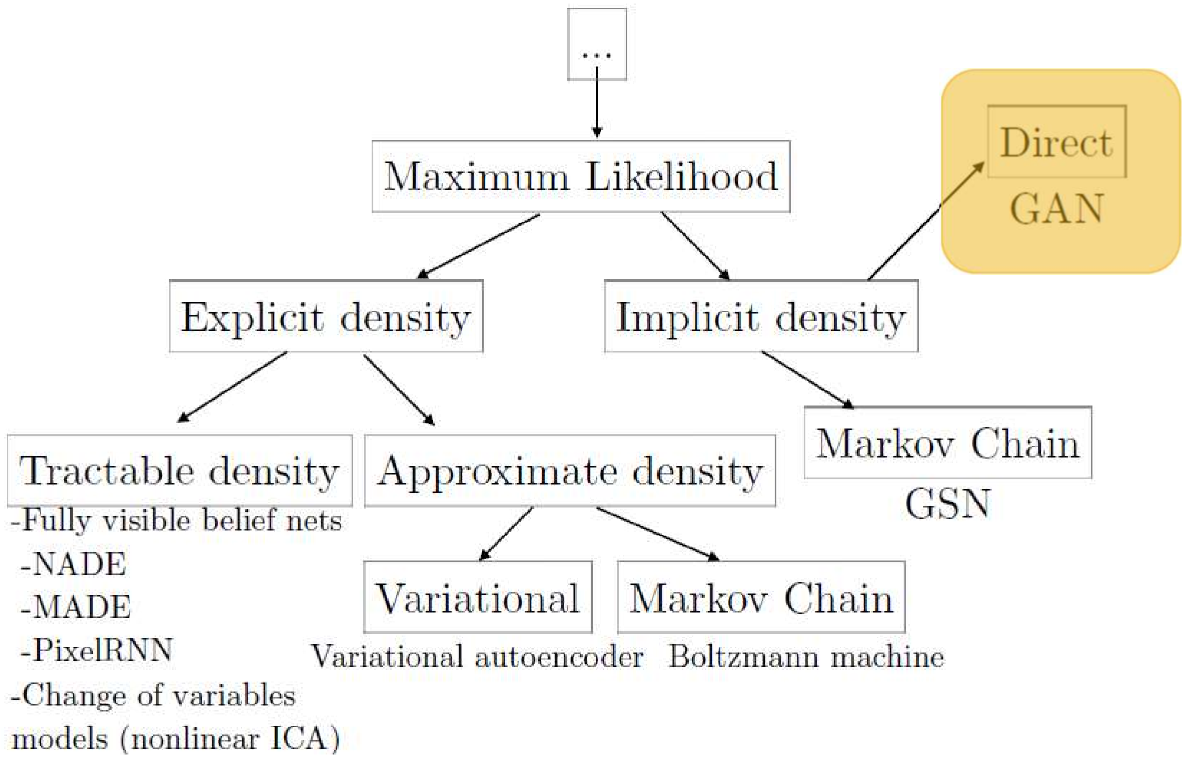
Example Results: Image Completion (PixelRNN)



Oord et al., Conditional Image Generation with PixelCNN Decoders

Example Images (PixelCNN)

Generative Adversarial Networks (GANs)



Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks

Generative Models

- *Implicit* generative models do not actually learn an explicit model for $\boldsymbol{p}(\boldsymbol{x})$

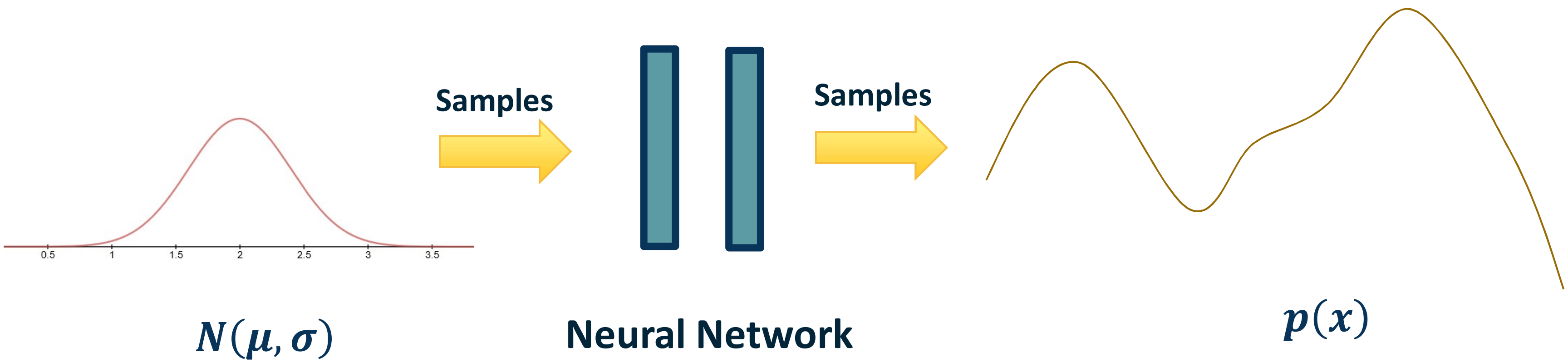
- Instead, learn to *generate samples* from $\boldsymbol{p}(\boldsymbol{x})$
  - Learn good feature representations
  - Perform data augmentation
  - Learn world models (a simulator!) for reinforcement learning

- How?
  - **Learn to sample** from a neural network output
  - **Adversarial training** that uses one network's predictions to train the other (dynamic loss function!)
  - **Lots of tricks** to make the optimization more stable

Implicit Models

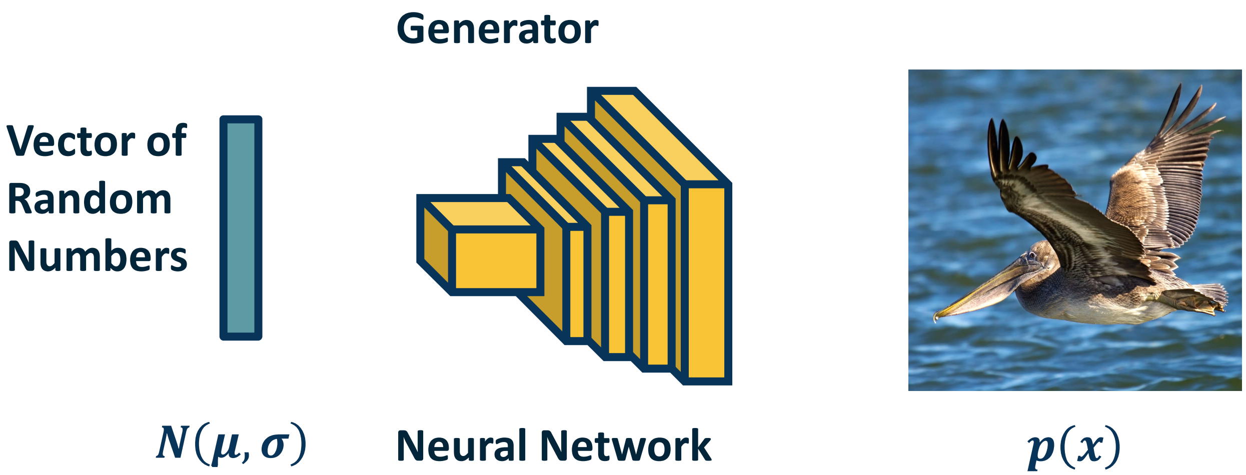
Georgia Tech

- We would like to *sample* from $\boldsymbol{p}(\boldsymbol{x})$ using a neural network
- **Idea:**
  - Sample from a simple distribution (Gaussian)
  - Transform the sample to $\boldsymbol{p}(\boldsymbol{x})$



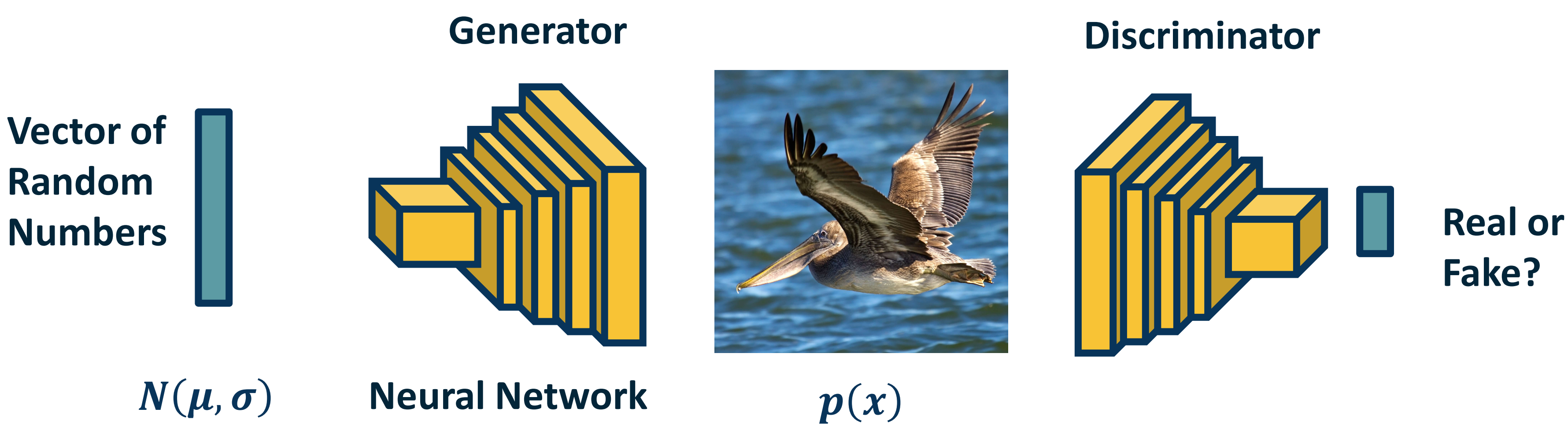
Learning to Sample

- Input can be a vector with (independent) Gaussian random numbers
- We can use a CNN to generate images!



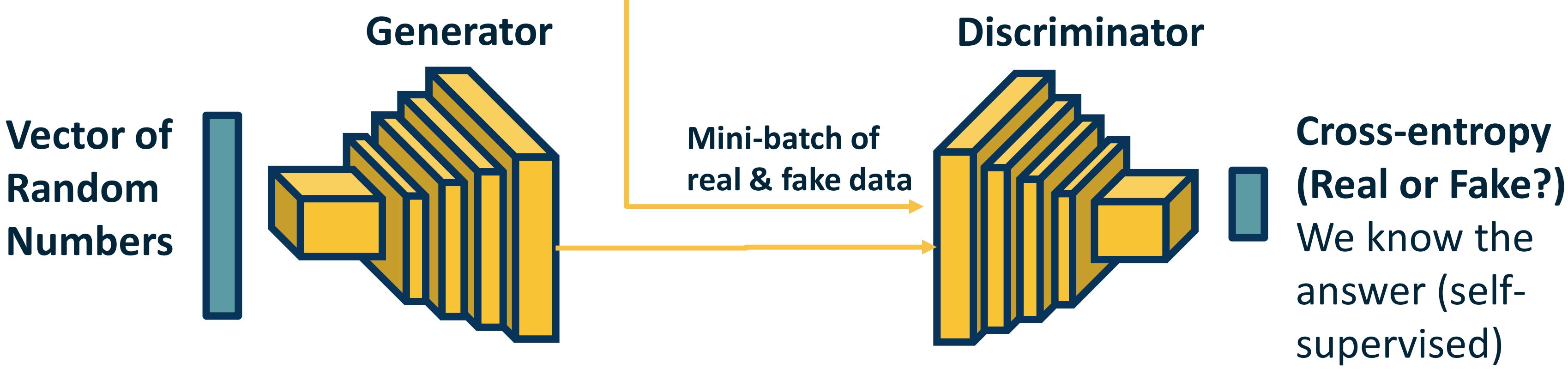
Generating Images

- **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



Adversarial Networks

- **Generator:** Update weights to improve realism of generated images
- **Discriminator:** Update weights to better discriminate



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

- 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

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks

Mini-max Two Player 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:

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

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks

Mini-max Two Player Game

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

- 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 $\mathrm{D}(\mathrm{G}(\mathrm{z}))$ is pushed down to 0)
  - In other words, discriminator wants to classify real images as real (1) and fake images as fake (0)

Discriminator Perspective

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

- 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 $\mathrm{D}(\mathrm{G}(\mathrm{z}))$ 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

Generator Perspective

- 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:

Sample from fake

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \underbrace{\mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}_{\text{How well discriminator does (0 for fake)}}$$

Generator *minimizes*

How well discriminator does (0 for fake)

  - 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

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks

- 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:

Sample from real **Sample from fake**

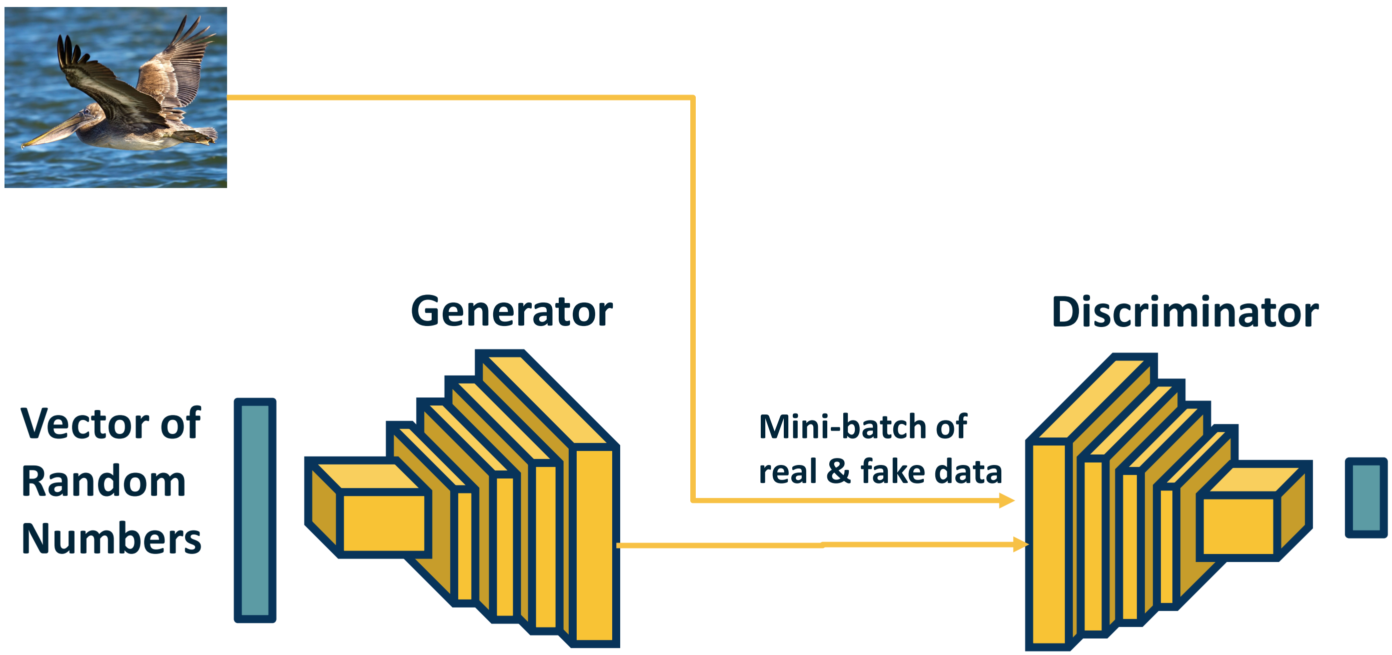
$$\min_G \max_D V(D,G) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]}_{\text{How well discriminator does (1 for real)}} + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}_{\text{How well discriminator does (0 for fake)}}$$

Discriminator *maximizes*

  - 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

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks

Mini-max Two Player Game



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

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

Generator Loss

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

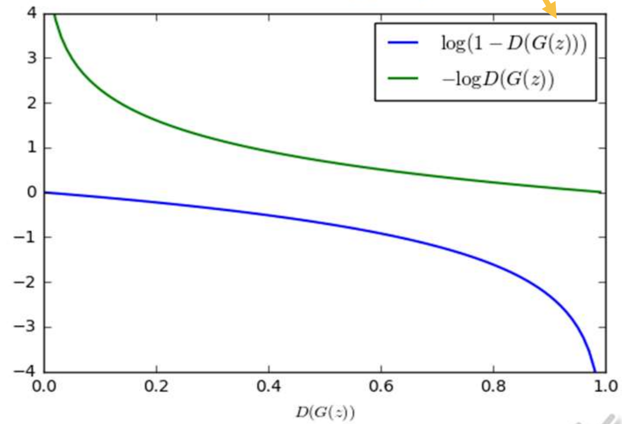
Discriminator Loss

Generative Adversarial Networks (GANs)

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

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

  - 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***:

$$\max_{\theta_g} \mathbb{E}_{z\sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Plot from CS231n, Fei-Fei Li, Justin Johnson, Serena Yeung

Converting to Max-Max Game

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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

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

  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

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

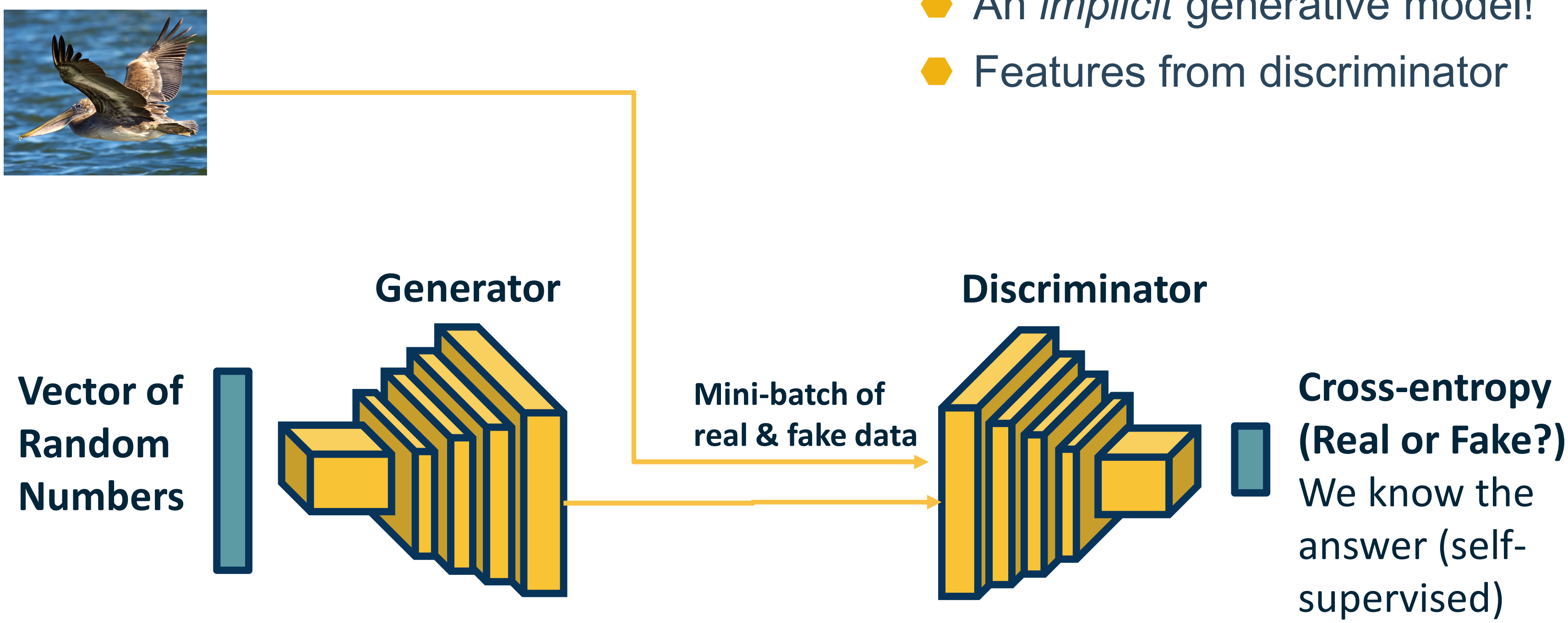
end for

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

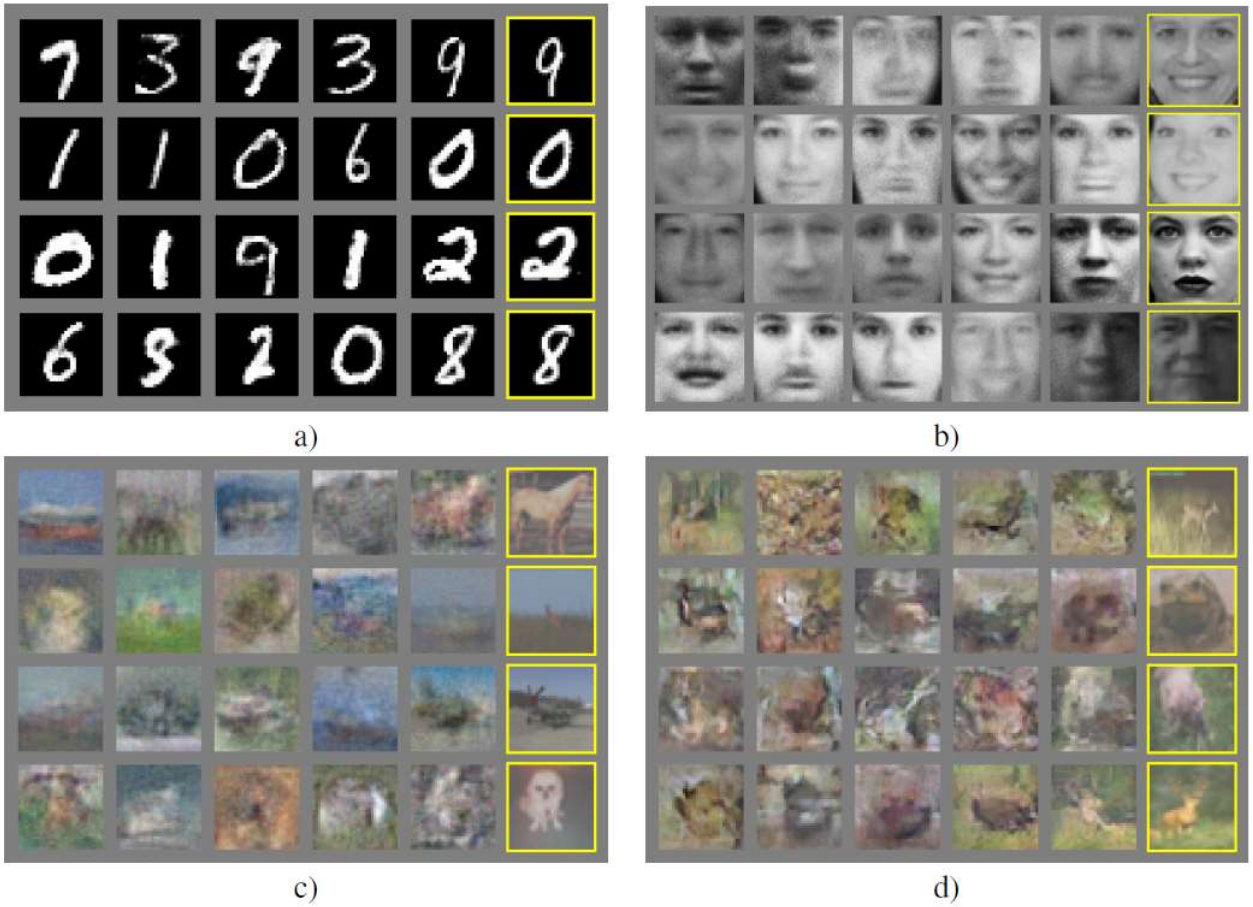
Goodfellow, NeurIPS 2016 Generative Adversarial Nets

Final Algorithm

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



Generative Adversarial Networks (GANs)



- Low-resolution images but look decent!
- Last column are nearest neighbor matches in dataset

Early Results

Georgia Tech

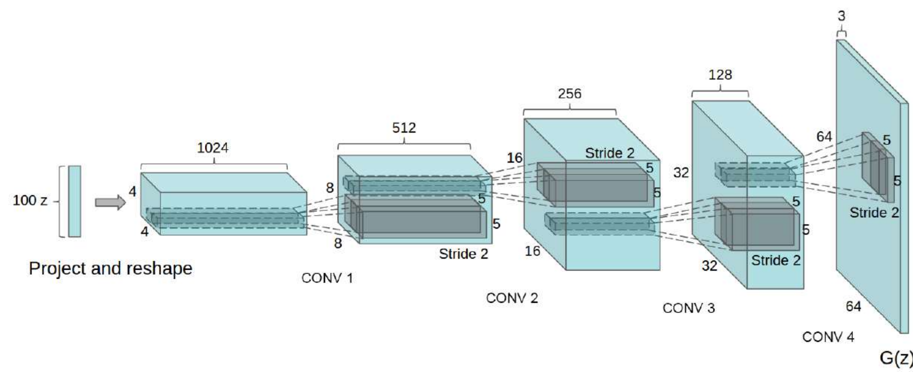
- GANs are very difficult to train due to the mini-max objective
- Advancements include:
  - More stable architectures
  - Regularization methods to improve optimization
  - Progressive growing/training and scaling

Goodfellow, NeurIPS 2016 Generative Adversarial Nets

Difficulty in Training

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.



Radford et al., Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks

DCGAN

- 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!

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

Kodali et al., On Convergence and Stability of GANs (also known as How to Train your DRAGAN)

Regularization

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

Brock et al., Large Scale GAN Training for High Fidelity Natural Image Synthesis

Example Generated Images - BigGAN

(a) 128×128 (b) 256×256 (c) 512×512 (d)

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).

Brock et al., Large Scale GAN Training for High Fidelity Natural Image Synthesis

Failure Examples - BigGAN

https://www.youtube.com/watch?v=PCBTZh41Ris

Video Generation

Georgia Tech

- 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!

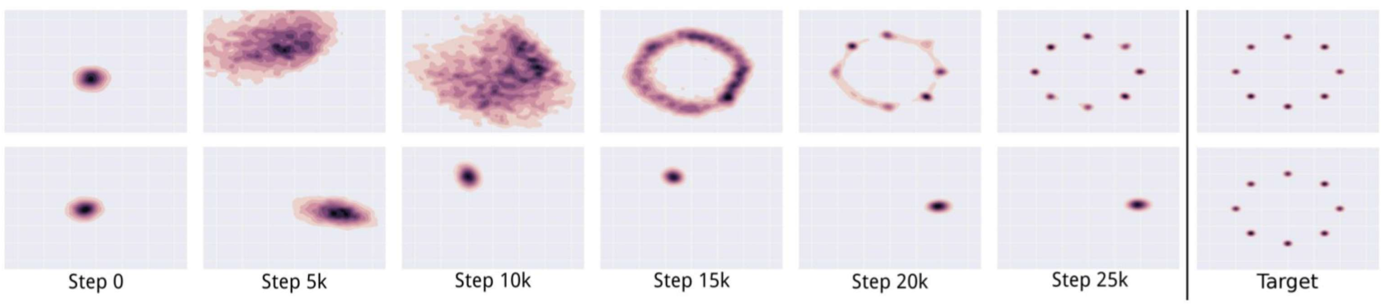
Cons:
- Trickier / more unstable to train
- Can't solve inference queries such as p(x), p(z|x)

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

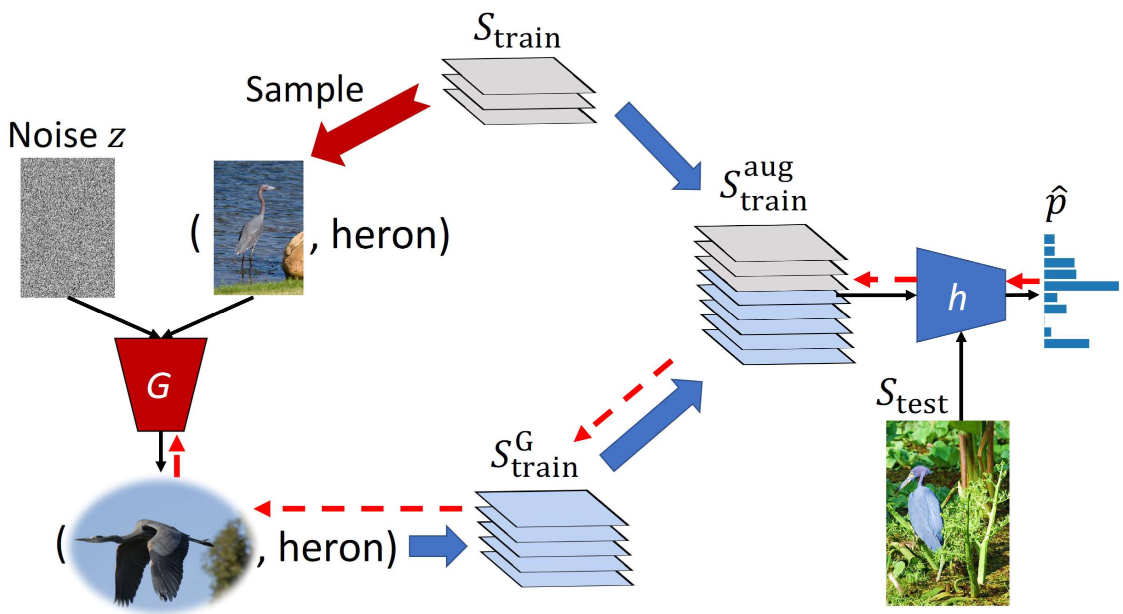
Mode Collapse

- 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

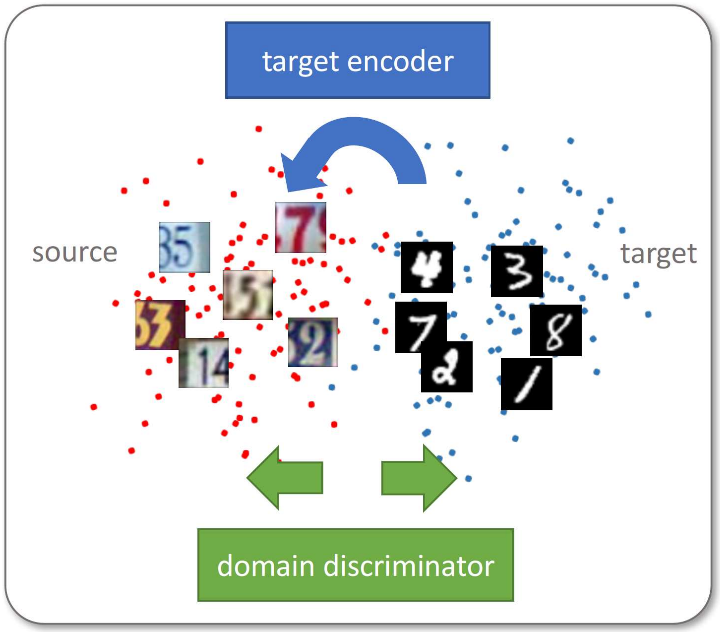


Low-Shot Learning from Imaginary Data, Yu-Xiong Wang, Ross Girshick, Martial Hebert, Bharath Hariharan

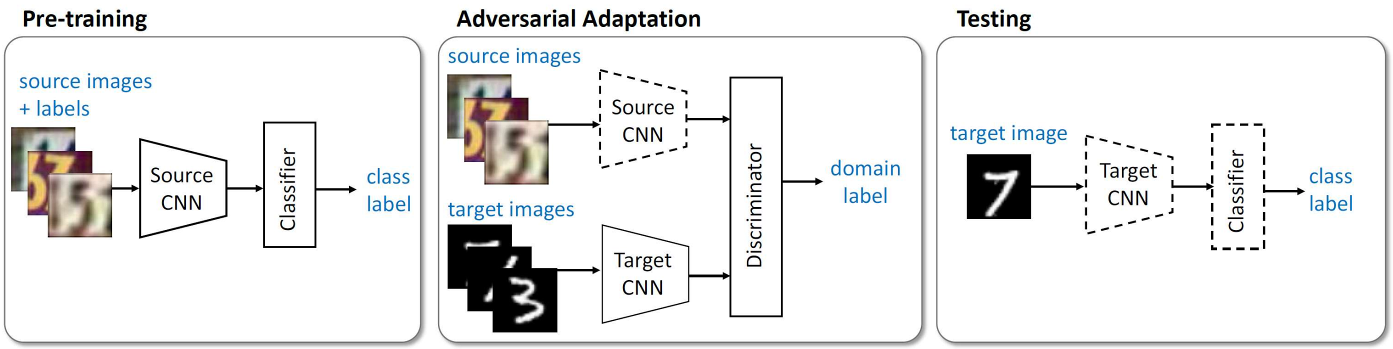
Georgia Tech

Application: Domain Adaptation

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



Approach

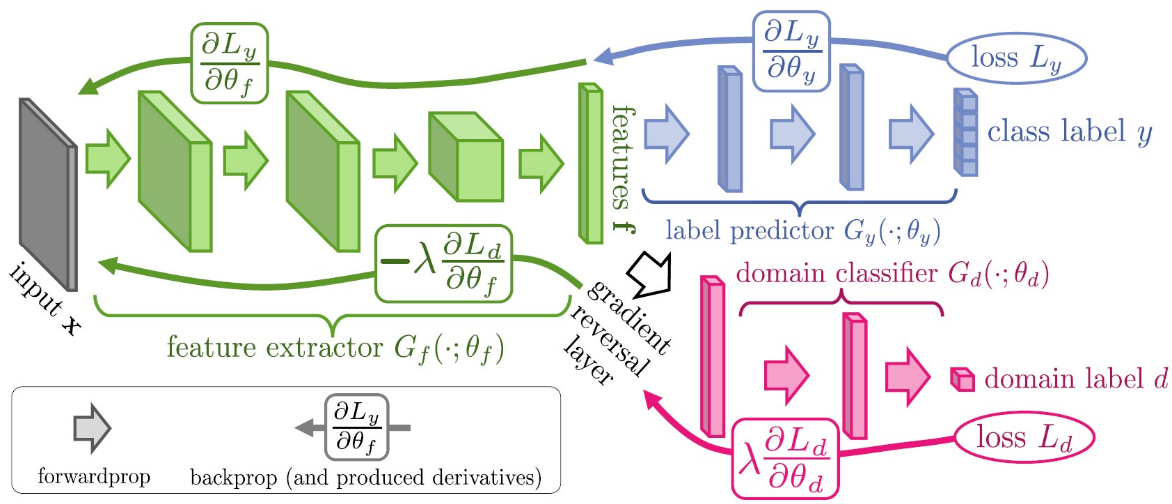


Method	MNIST → USPS	USPS → MNIST	SVHN → MNIST
Source only	0.752 ± 0.016	0.571 ± 0.017	0.601 ± 0.011
Gradient reversal	0.771 ± 0.018	0.730 ± 0.020	0.739 [16]
Domain confusion	0.791 ± 0.005	0.665 ± 0.033	0.681 ± 0.003
CoGAN	0.912 ± 0.008	0.891 ± 0.008	did not converge
ADDA (Ours)	0.894 ± 0.002	0.901 ± 0.008	0.760 ± 0.018

Table 2: Experimental results on unsupervised adaptation among MNIST, USPS, and SVHN.

Aside: Other ways to Align

digital SLR camera | low-cost camera, flash



[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

Summary