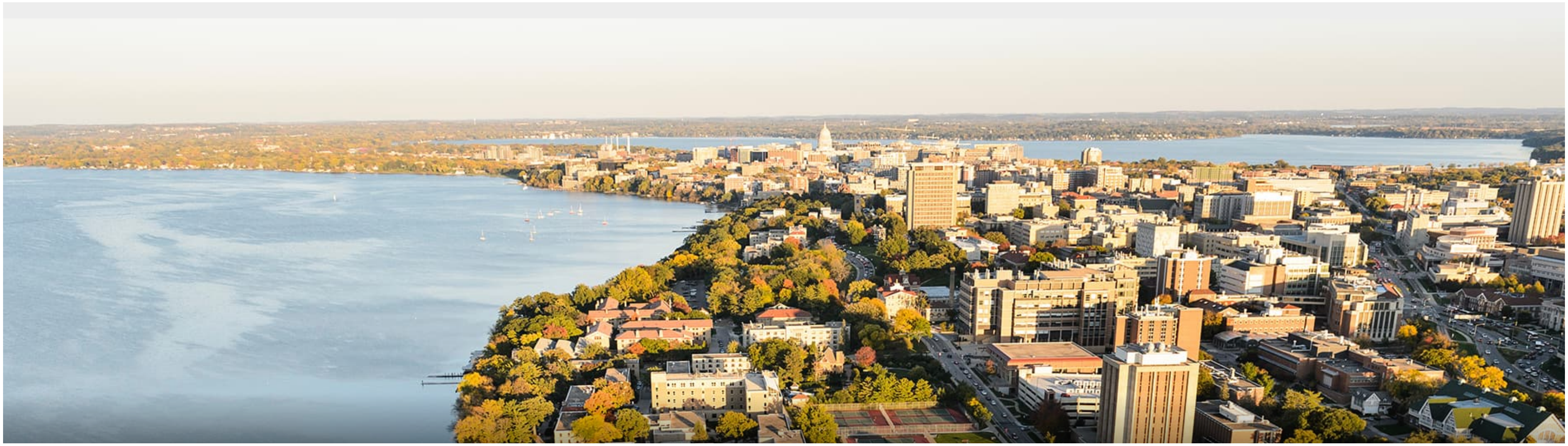


CS 760: Machine Learning
Generative Models

Ilias Diakonikolas

University of Wisconsin-Madison

Nov. 1, 2022

Announcements

- **Logistics**:
  - Congrats on the getting through the midterm!

- Class roadmap:

Nov. 1	Generative Models
Nov. 3	Kernels + SVMs
Nov. 8	Graphical Models I
Nov. 10	Graphical Models II

Outline

- **Intro to Generative Models**
  - Applications, histograms, autoregressive models
- **Flow-based Models**
  - Transformations, training, sampling
- **Generative Adversarial Networks (GANs)**
  - Generators, discriminators, training, examples

Generative Models

- Goal: capture our data distribution.
  - Recall our **discriminative** vs. **generative** discussion
  - Generative models exist in supervised & unsupervised settings
  - Today: focus is on **unsupervised**

neurohive

Applications: Generate Images

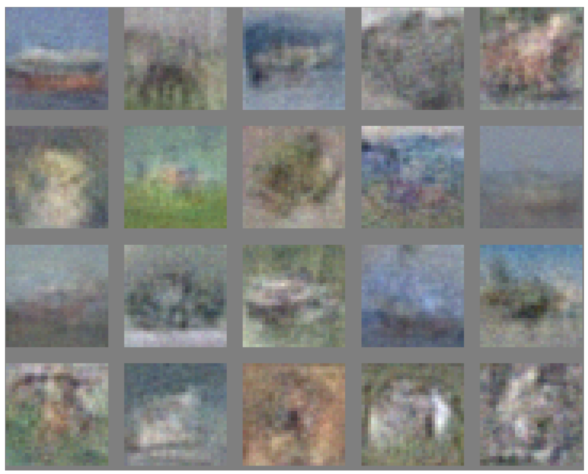
- Old idea---tremendous growth
- Historical evolution:

2006: Hinton et al

2013: Kingma & Welling

Applications: Generate Images

- More recently, GAN models: 2014
  - Goodfellow et al



Applications: Generate Images

- More recently, GAN models
  - StyleGAN, Karras, Laine, Aila, 2018

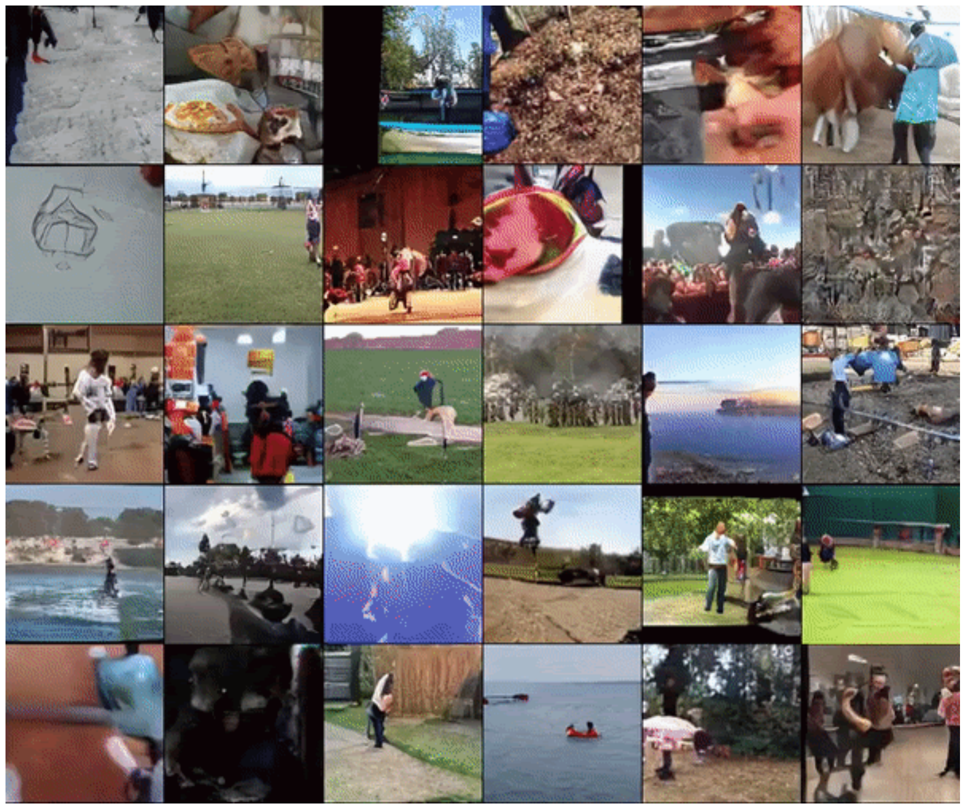
Applications: Generate Images/Video

- GANs can also generate video
  - Plus transfer:

CycleGAN: Zhu, Park, Isola & Efros, 2017

Applications: Generate Video

- GANs can also generate video (DVD-GAN, Clark et al)



Additional Applications

- **Compress** data
  - Can often do better than fixed methods like JPEG
- Generate **additional training** data
  - Use for training a model
- Obtain **good representations**
  - Then can fine-tune for particular tasks

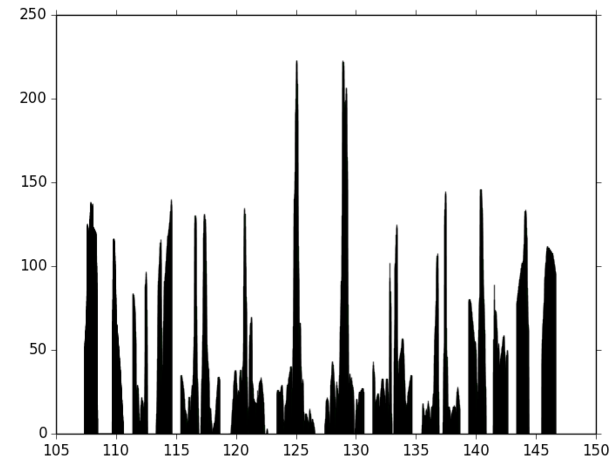
Goal: Learn a Distribution

- Want to estimate p_{data} from samples

$$x^{(1)}, x^{(2)}, \ldots, x^{(n)} \sim p_{\text{data}}(x)$$

- Useful abilities to have:
  - **Inference**: compute p(x) for some x
  - **Sampling**: obtain a sample from p(x)

- As always need efficiency for this too…

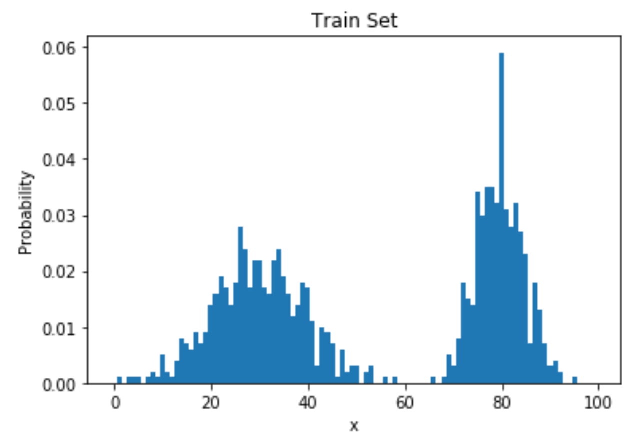


Goal: Learn a Distribution

- Want to estimate p_{data} from samples

$$x^{(1)}, x^{(2)}, \ldots, x^{(n)} \sim p_{\text{data}}(x)$$

- **One way**: if discrete valued-variables, build a histogram:
- Say in {1, …, k}.
  - Estimate p_1, p_2, …, p_k
- Train this model:
  - Count times #i appears in dataset



Histograms: Inference & Samples

- **Inference**: check our estimate of p_i
- **Sampling**:
  - Produce the cumulative distribution $F_i = p_1 + \ldots + p_i$
  - Get a random value uniformly in [0,1]
  - Get smallest value i so that $u \le F_i$

- Easy, but…
  - Too many values to compute (recall this from Naïve Bayes)
  - MNIST: 28x28 means 2^{784} probabilities

Parametrizing Distributions

- Don't store each probability, store $p_\theta(x)$
  - We saw the conditional version of this for Naïve Bayes

- One approach: likelihood-based
  - Good: we know how to train with **maximum likelihood**

$$\arg\min_\theta -\frac{1}{n}\sum_{i=1}^{n}\log p_\theta(x^{(i)})$$

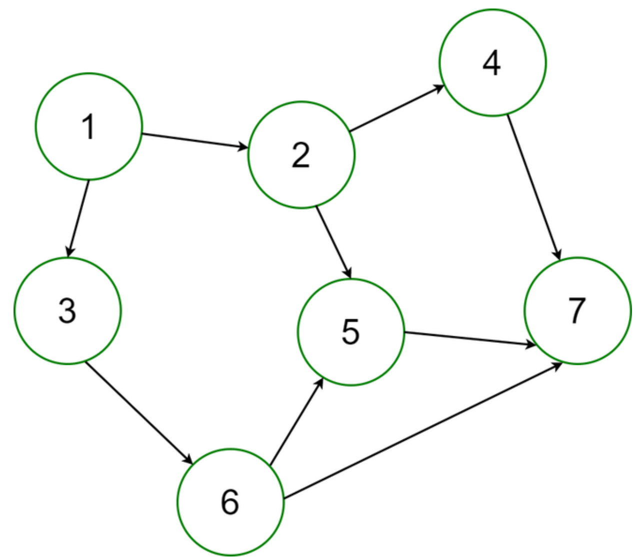
  - Recall that we can think of this as minimizing KL divergence

Parametrizing Distributions

- One approach: likelihood-based
  - Good: we know how to train with **maximum likelihood**
  - Then, train with SGD
  - We've been doing this all along for supervised learning… just need to make some choices for $p_\theta(x)$

Parametrizing Distributions: Bayes Nets

- Bayes nets: a useful tool
- A Bayes net: a DAG that represents a probability distribution
  - DAG: directed acyclic graph
  - Say graph is G = (V, E), and for node v, pa(v) denotes its parents:
  - **Example**: pa(7) = ?



Parametrizing Distributions: Bayes Nets

- Bayes nets: a useful tool
- A Bayes net: a DAG that represents a probability distribution
  - DAG: directed acyclic graph
  - Say graph is G = (V, E), and for node v, pa(v) denotes its parents:
  - Helps represent distribution in a compact way:

$$p(x_1, \ldots, x_d) = \prod_{v \in V} p(x_v | x_{\mathrm{pa}(v)})$$

Parametrizing Distributions: Bayes Nets

- A Bayes net: a DAG that represents a probability distribution
  - Say graph is G = (V, E), and for node v, pa(v) denotes its parents:
  - Helps represent distribution in a compact way:

$$p(x_1, \ldots, x_d) = \prod_{v \in V} p(x_v | x_{\mathrm{pa}(v)})$$

  - Compare to standard factorization: chain rule

$$p(x_1, \ldots, x_d) = \prod_{v \in V} p(x_v | x_1, x_2, \ldots, x_{v-1})$$

  - If G sparse, conditional probability terms are much smaller.

Autoregressive Models

- Use a Bayes net for the features

$$\log p_\theta(x_1, \ldots, x_d) = \sum_{i=1}^{d} \log p_\theta(x_i | \mathrm{pa}(x_i))$$

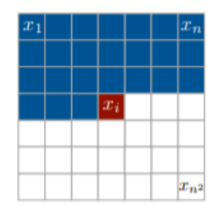
- Then we can directly plug these into our MLE estimation
- Some practical questions:
  - To help generalization, share parameters (we did this for CNNs, RNNs).
  - In fact can directly use RNNs.

Autoregressive Models: RNNs

- Can use the Bayes net idea to just model a sequence
- Apply to dxd images:

$$p(x) = \prod_{i=1}^{d^2} p(x_{i,R}|\underbrace{p(x_{1,R}, \dots, x_{i-1,R})}_{\text{Red Channel Pixels}} p(x_{i,B}|\underbrace{p(x_{1,B}, \dots, x_{i-1,B})}_{\text{Blue Channel Pixels}} p(x_{i,G}|\underbrace{p(x_{1,G}, \dots, x_{i-1,G})}_{\text{Green Channel Pixels}}$$

- Each pixel depends on the previous pixels
- Same function/parameters used for each



van den Oord et al ‘16

PixelRNN: Samples

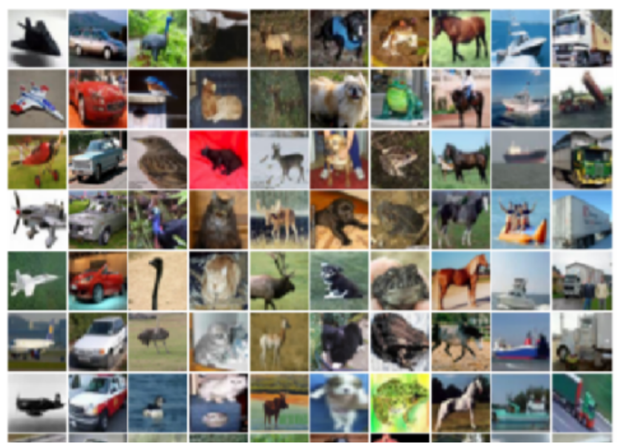
- Trained on ImageNet

- Use for **completion**:
  - Left: covered
  - Right: original
  - Middle: completed

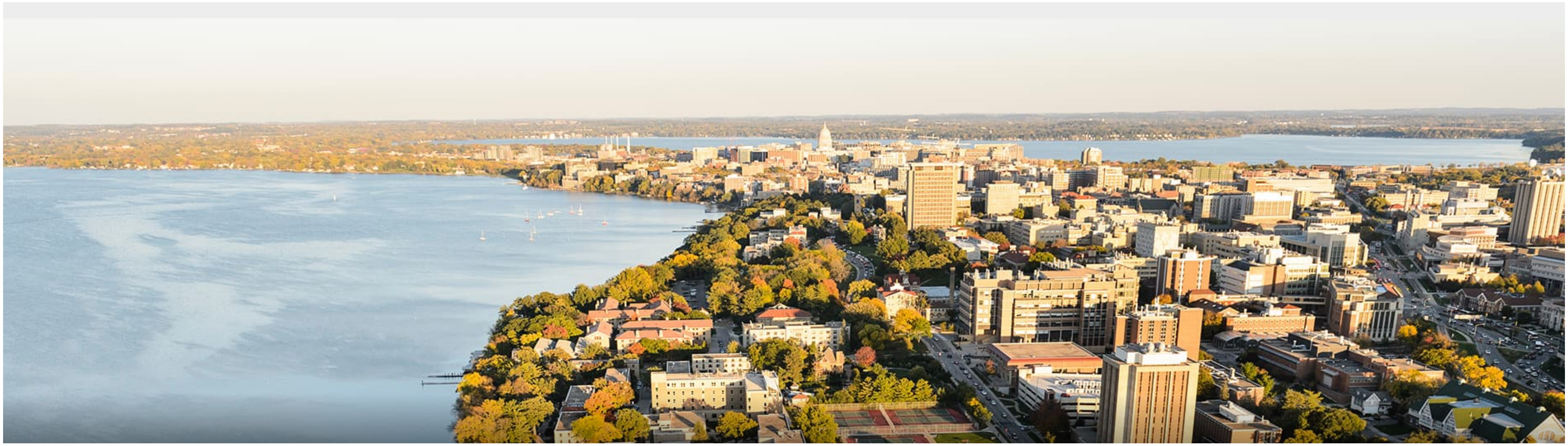
van den Oord et al '16

PixelRNN: Samples

- Upside: can evaluate p(x) pretty easily, samples are good
- Downside: sequential generation (need all the previous pixels) might be slow
  - Many variants: combine with CNNs, architectural tricks



pixelCNN++, Salimans et al ‘17

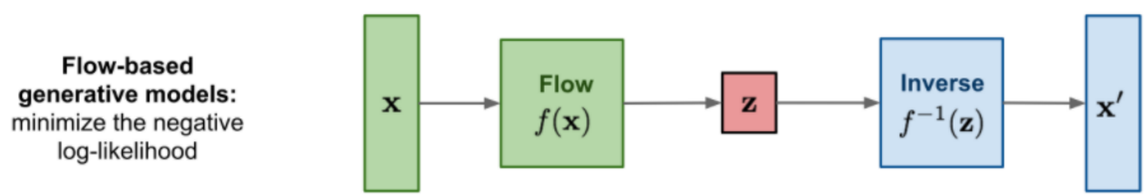


Break & Quiz

Flow Models

- Still want to fit $p_\theta(x)$
- Some goals:
  - Good fit for the data
  - Computing a probability: the actual value of $p_\theta(x)$ for some x
  - Ability to sample
  - Also: a **latent representation**

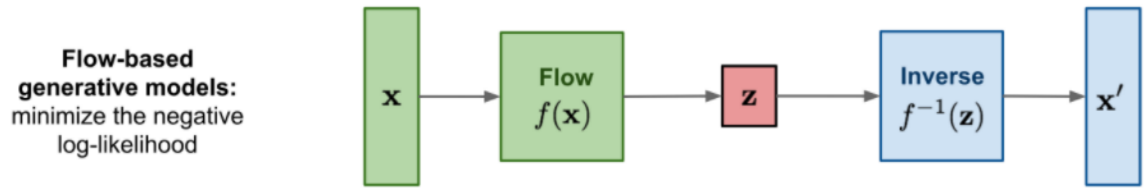
- Won't model $p_\theta(x)$ directly… instead we'll get some latent variable z

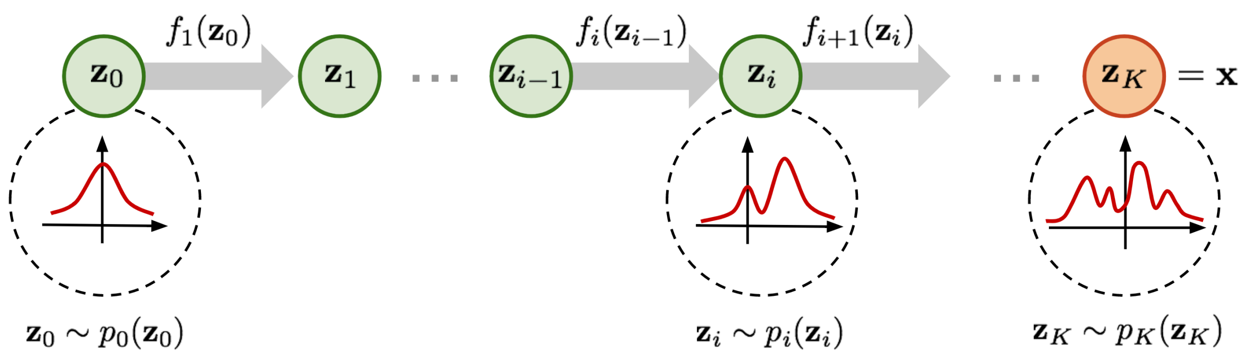


Lilian Weng

Flow Models

- **Key idea**: transform a simple distribution to complex
  - Use a chain of transformations (the “flow”)

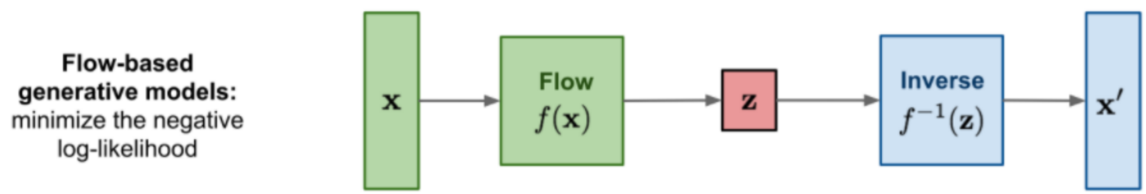




Lilian Weng

Flow Models

- **Key idea**: transform a simple distribution to complex
  - Use a chain of invertible transformations (the "flow")



  - How to sample?
    - Sample from Z (the latent variable)---has a simple distribution that lets us do it: Gaussian, uniform, etc.
    - Then run the sample z through the inverse flow to get a sample x
  - How to train? Let's see…

Flow Models: Density Relationships

- **Key idea**: transform a simple distribution to complex
  - Use a chain of transformations (the “flow”)

- How does each transformation affect the density p?

Latent variable

Transformation

$$z = f_\theta(x)$$

$$p_\theta(x)\,dx = p(z)\,dz$$

Determinant of
Jacobian matrix

$$p_\theta(x) = p(f_\theta(x)) \left| \frac{\partial f_\theta(x)}{\partial x} \right|$$

Flow Models: Training

- **Key idea**: transform a simple distribution to complex
  - Use a chain of transformations (the "flow")

- How does training change?
  - **Idea**: might be easier to optimize p_Z

$$\max_\theta \underbrace{\sum_i \log}_{\text{Maximum Likelihood}} p_\theta(x^{(i)}) = \max_\theta \sum_i \log p_Z(f_\theta(x^{(i)})) + \log \left| \frac{\partial f_\theta}{\partial x}(x^{(i)}) \right|$$

Latent variable version

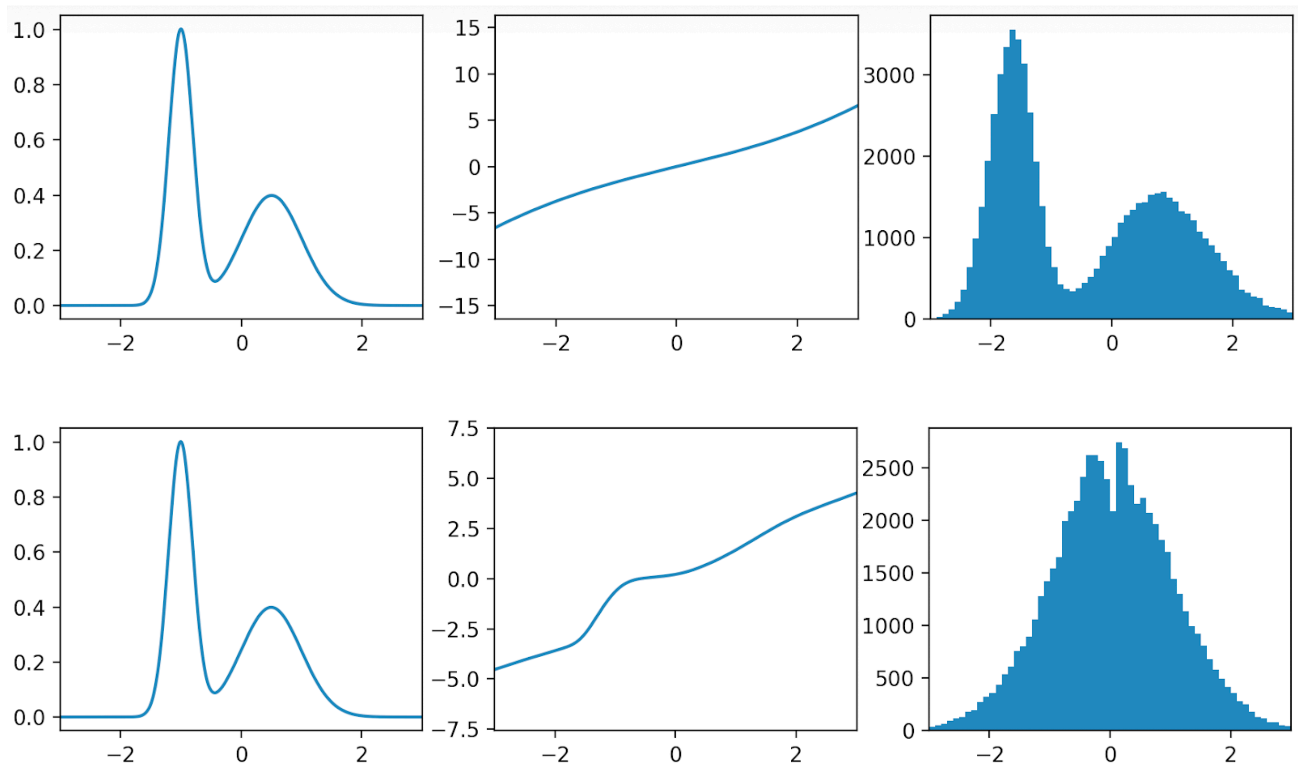
Determinant of Jacobian matrix

Can extend to many chained transformations…

Flows: Example

- Flow to a Gaussian (right)
- **Before** training:
- **After** training:

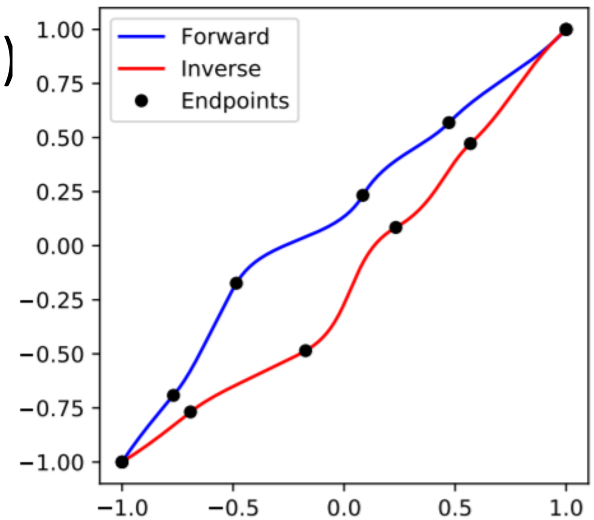
Flow



UC Berkeley: Deep Unsupervised Training

Flows: Transformations

- What kind of f transformations should we use?
- Many choices:
  - Affine: $f(x) = A^{-1}(x - b)$
  - Elementwise: $f(x_1, \ldots, x_d) = (f(x_1), \ldots, f(x_d))$
  - Splines:



(a) Forward and inverse transformer

Papamakarios et al’ 21

- Desirable properties:
  - Invertible
  - Differentiable (forward and inverse)

Break & Quiz

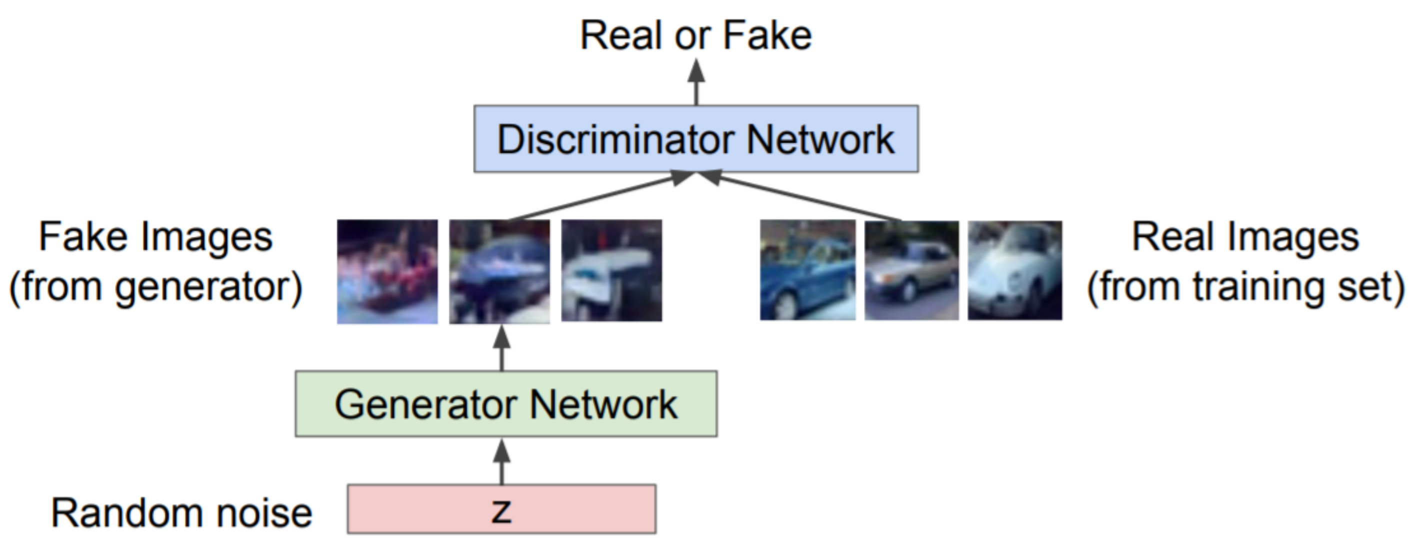
GANs: Generative Adversarial Networks

- So far, we've been modeling the density…
  - What if we just want to get high-quality samples?

- GANs do this. Based on a clever idea:
  - Art forgery: very common through history
  - Left: original
  - Right: forged version
  - Two-player game. **Forger** wants to pass off the forgery as an original; **investigator** wants to distinguish forgery from original

GANs: Basic Setup

- Let’s set up networks that implement this idea:
  - Discriminator network: like the **investigator**
  - Generator network: like the **forger**



Stanford CS231n / Emily Denton

GAN Training: Discriminator

- How to train these networks? Two sets of parameters to learn: θ_d (discriminator) and θ_g (generator)

- Let's fix the generator. What should the discriminator do?
  - Distinguish fake and real data: binary classification.
  - Use the cross entropy loss, we get

$$\max_{\theta_d} \mathbb{E}_{x\sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Real data, want to classify 1

Fake data, want to classify 0

GAN Training: Generator & Discriminator

- How to train these networks? Two sets of parameters to learn: θ_d (discriminator) and θ_g (generator)

- This makes the discriminator better, but also want to make the generator more capable of fooling it:
  - Minimax game! Train jointly.

$$\min_{\theta_g} \max_{\theta_d} \mathbb{E}_{x \sim p_{\text{data}}} \underbrace{\log D_{\theta_d}(x)}_{\text{Real data, want to classify 1}} + \mathbb{E}_{z \sim p(z)} \underbrace{\log(1 - D_{\theta_d}(G_{\theta_g}(z)))}_{\text{Fake data, want to classify 0}}$$

GAN Training: Alternating Training

- So we have an optimization goal:

$$\min_{\theta_g} \max_{\theta_d} \mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

- Alternate training:
  - **Gradient ascent**: fix generator, make the discriminator better:

$$\max_{\theta_d} \mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

  - **Gradient descent**: fix discriminator, make the generator better

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

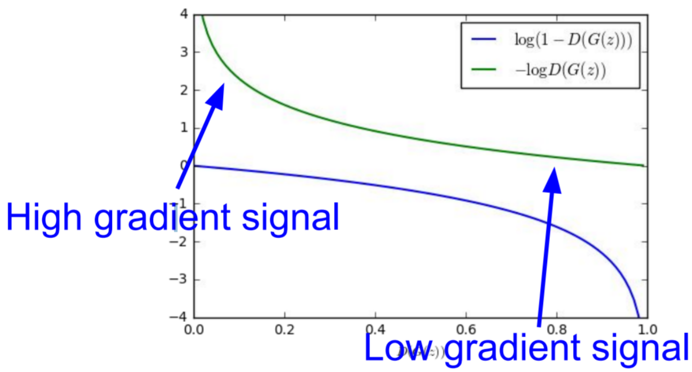
GAN Training: Issues

- Training often not stable
- Many tricks to help with this:
  - Replace the generator training with

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

  - Better gradient shape
  - Choose number of alt. steps carefully

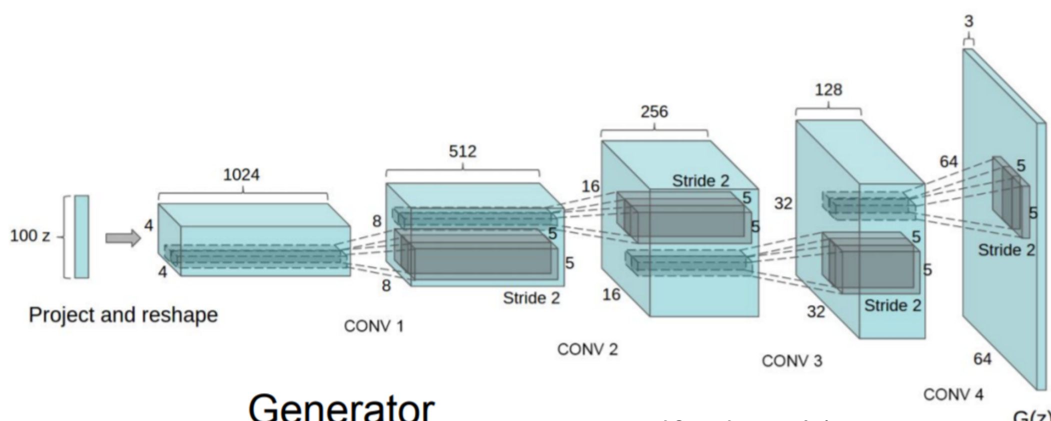
- Can still be challenging.



Stanford CS231n

GAN Architectures

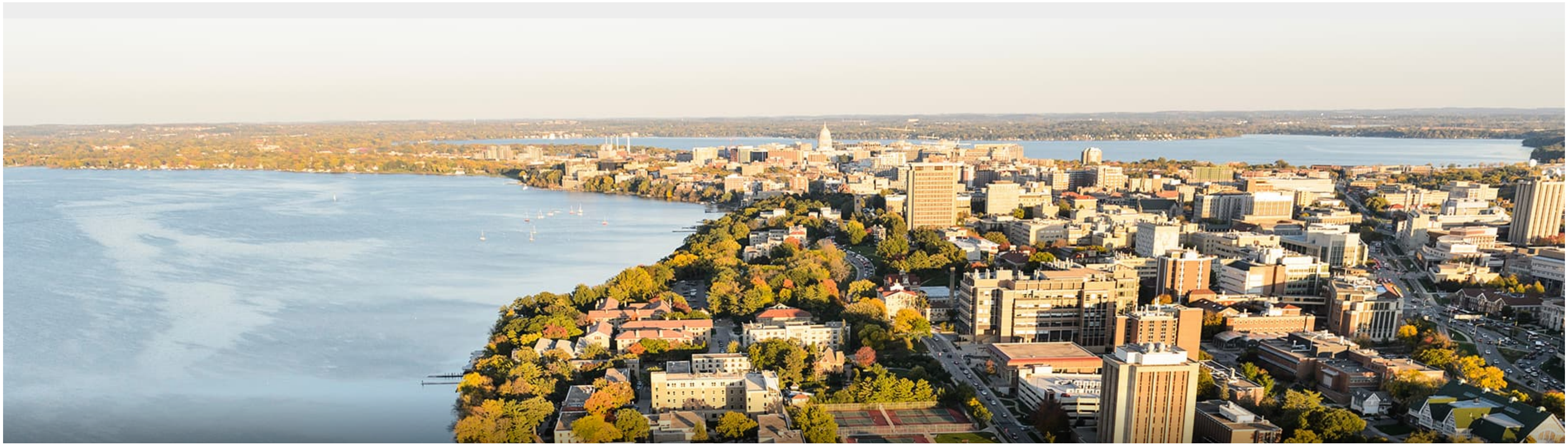
- So far we haven't commented on what the networks are
- **Discriminator**: image classification, use a **CNN**
- What should **generator** look like
  - Input: noise vector z. Output: an image (ie, volume 3 x width x height)
  - Can just reverse our CNN pattern…



Radford et al '16

GANs: Example

- From Radford's paper, with 5 epochs of training:



Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, Pedro Domingos, Jerry Zhu, Yingyu Liang, Volodymyr Kuleshov, Fei-Fei Li, Justin Johnson, Serena Yeung, Pieter Abbeel, Peter Chen, Jonathan Ho, Aravind Srinivas, Fred Sala