CS 760: Machine Learning
Unsupervised Learning III: Generative Models

Josiah Hanna

University of Wisconsin-Madison

November 2, 2023

Learning Outcomes

- **At the end of today's lecture, you will be able to:**
  - Explain and apply PCA
  - Explain the goal of generative modeling / density estimation.
  - Identify a few methods for generative modeling.

Outline

- **Finish PCA (Use slides from last time)**
- **Intro to Generative Models**
  - histograms,
- **Flow-based Models**
  - Transformations, training, sampling
- **Generative Adversarial Networks (GANs)**
  - Generators, discriminators, training, examples

Generative Models

- Goal: learn an underlying process for (unlabeled) data.

- Recall generative vs discriminative modeling from Naive Bayes lecture.

Applications: Generate Images

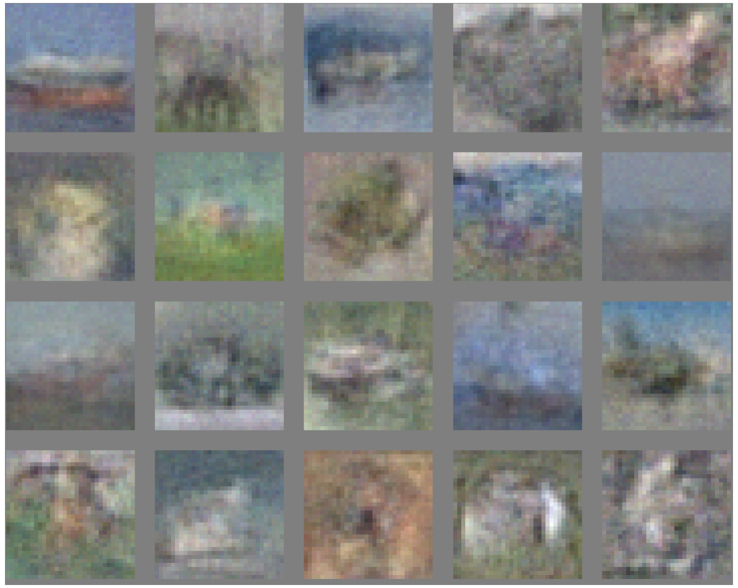
- Old idea---tremendous growth
- Historical evolution:

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

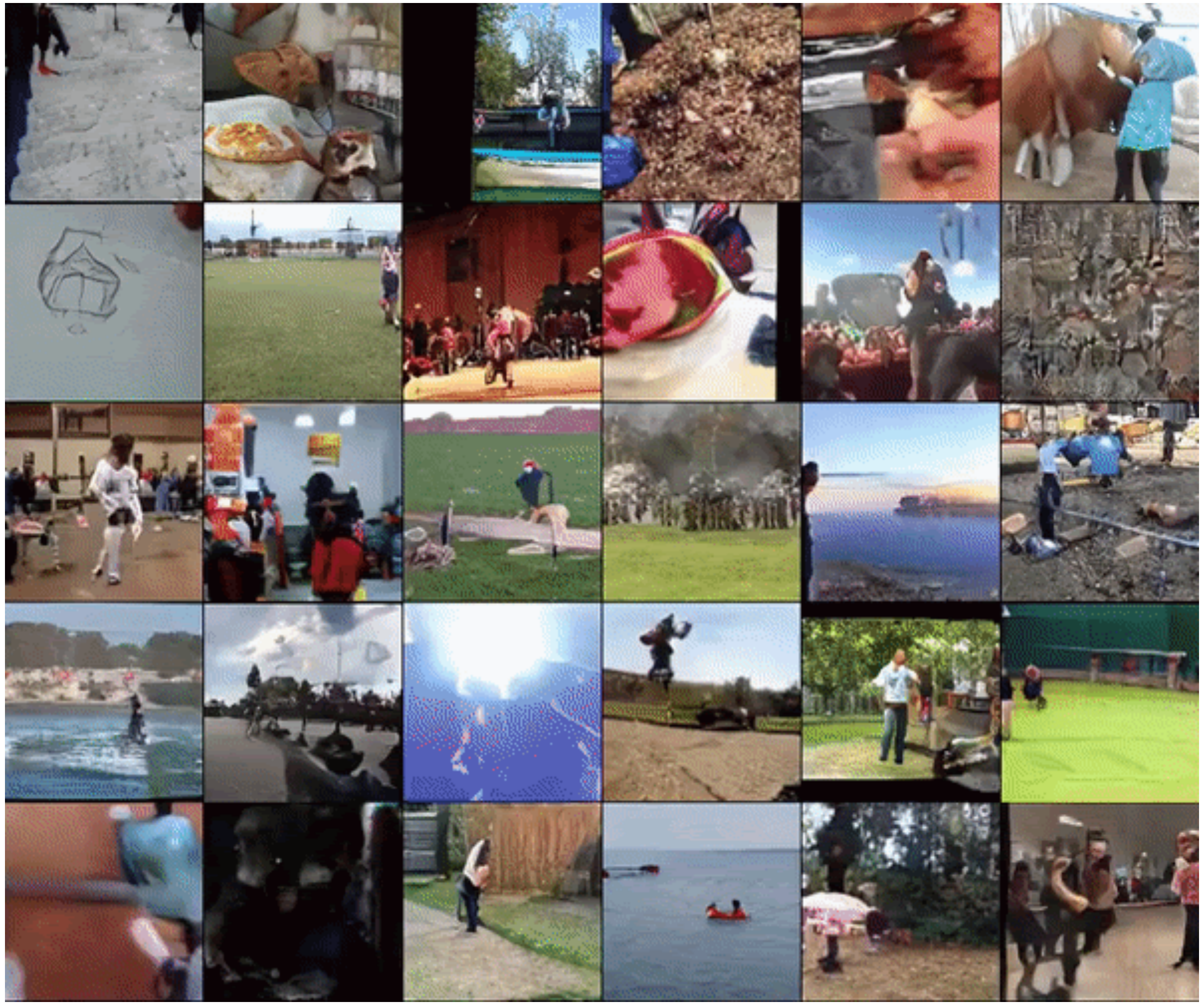
Applications: Generate Images/Video

- 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
  - Similar to nonlinear dimensionality reduction

- Obtain **good representations**
  - Then can fine-tune for particular tasks
  - Unlabeled data is cheap, labeled data is not.

Goal: Learn a Distribution

- Want to estimate p_{data} from samples

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

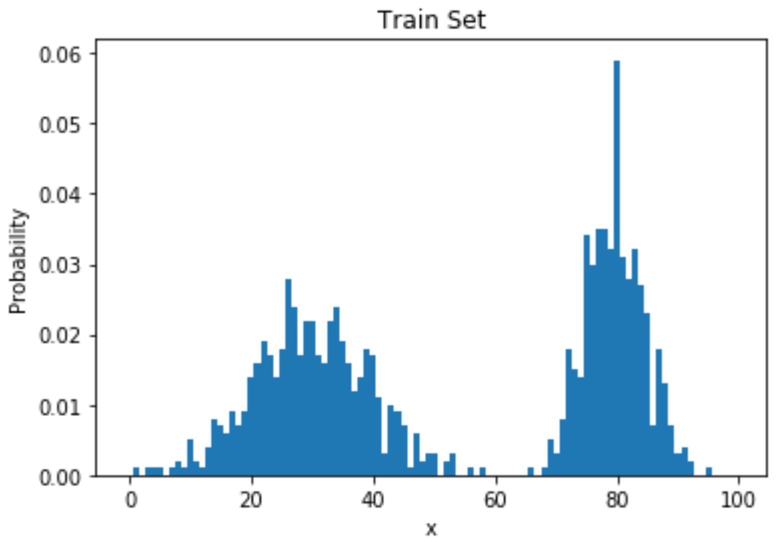
- Desired abilities:
  - **Inference**: compute p(x) for some x
  - **Sampling**: obtain a sample from p(x)

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**: build a histogram:
- Bin data space into k groups.
  - Estimate p_1, p_2, …, p_k
- Train this model:
  - Count times bin i appears in dataset



Histograms: Inference & Samples

- **Inference**: check our estimate of p_i
- **Sampling**: straightforward, select bin i with probability p_i, then select uniformly from bin i.

- But …
  - inefficient in high dimensions

Parametrizing Distributions

- Don't store each probability, store $p_\theta(x)$

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

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

Parametrizing Distributions

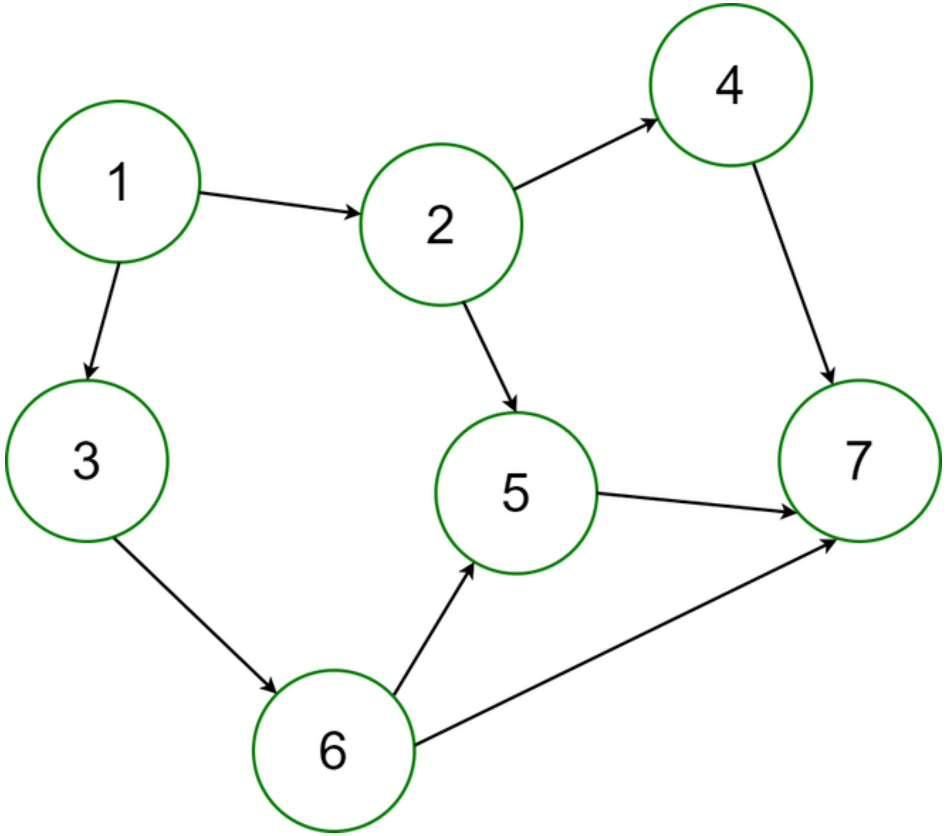
- One approach: likelihood-based
  - We know how to train with **maximum likelihood**

  - Then, train with SGD

  - Just need to make some choices for $p_\theta(x)$
    - For example, recall Gaussian mixture models.
    - But many types of data have more complex underlying distributions.

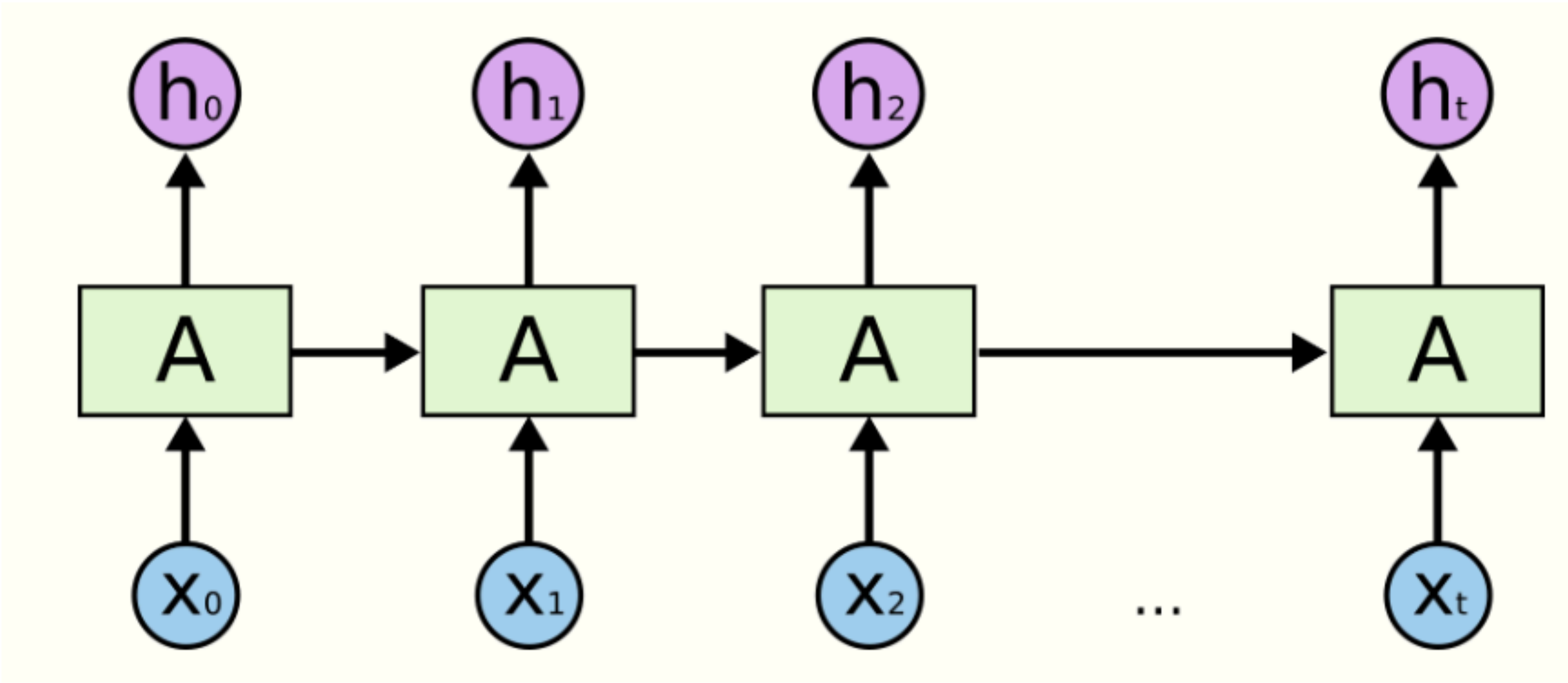
Parametrizing Distributions: Bayes Nets

- Coming up next week.



Parametrizing Distributions: Autoregressive models

- E.g., recurrent neural networks, transformers.



Flow Models

- One way to specify $p_\theta(x)$

- Use a latent variable z with a “simple” (e.g Gaussian) distribution.

- Then use a “complex” transformation, $x = f_\theta(z)$.

Flow Models

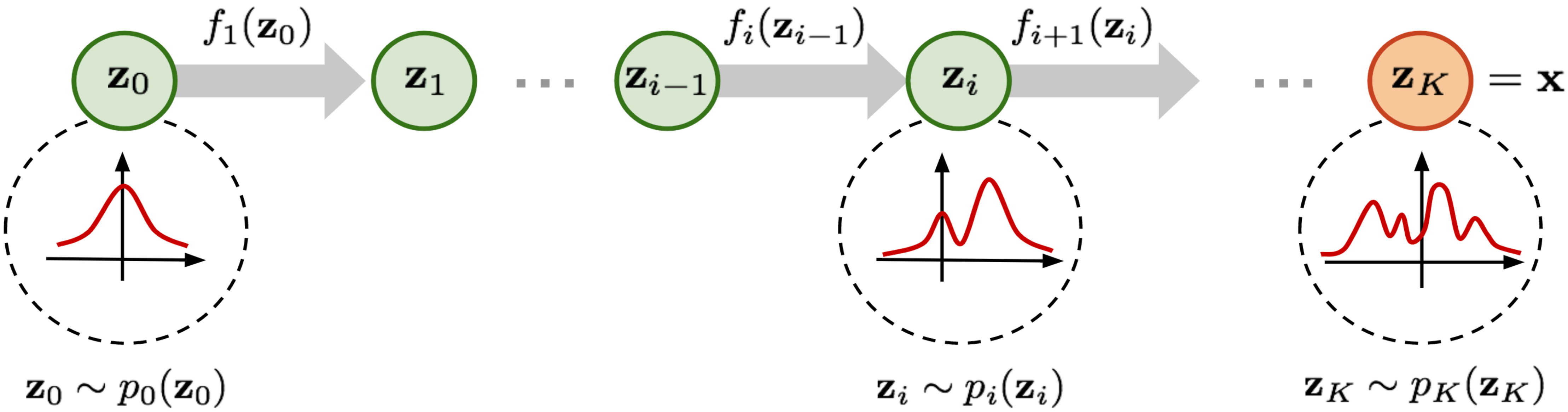
- We will need to compute the inverse transformation and take its derivative as well.
- So compose of multiple "simple" transformations

$$x = f_{\theta_k}(f_{\theta_{k-1}}(\dots f_{\theta_1}(z)))$$

$$z = f_{\theta_1}^{-1}(f_{\theta_2}^{-1}(\dots f_{\theta_k}^{-1}(x)))$$

Flow Models

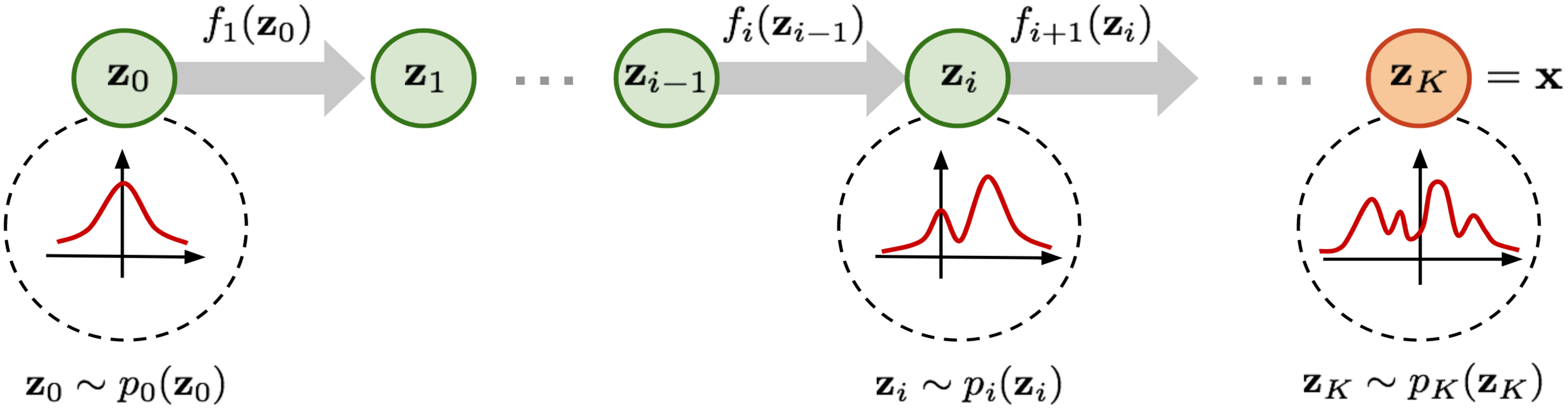
- Transform a simple distribution to a complex one via a chain of invertible transformations (the "flow")



Lilian Weng

Flow Models: 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 flow to get a sample x



Lilian Weng

Flow Models: How to train?

- Relationship between $p_x(x)$ and $p_z(z)$ (densities of x and z), given that $x = f_\theta(z)$?

$$p_x(x) = p_z(f_\theta^{-1}(x)) \left| \frac{\partial f_\theta^{-1}(x)}{\partial x} \right|$$

Determinant of Jacobian matrix

Flow Models: Training

$$\max_\theta \sum_i \underbrace{\log\Big(p_x(x^{(i)};\theta)\Big)}_{\text{Maximum Likelihood}} = \max_\theta \left(\sum_i \log\Big(p_z(f_\theta^{-1}(x^{(i)}))\Big) + \log\left|\frac{\partial f_\theta^{-1}(x^{(i)})}{\partial x}\right| \right)$$

Latent variable version

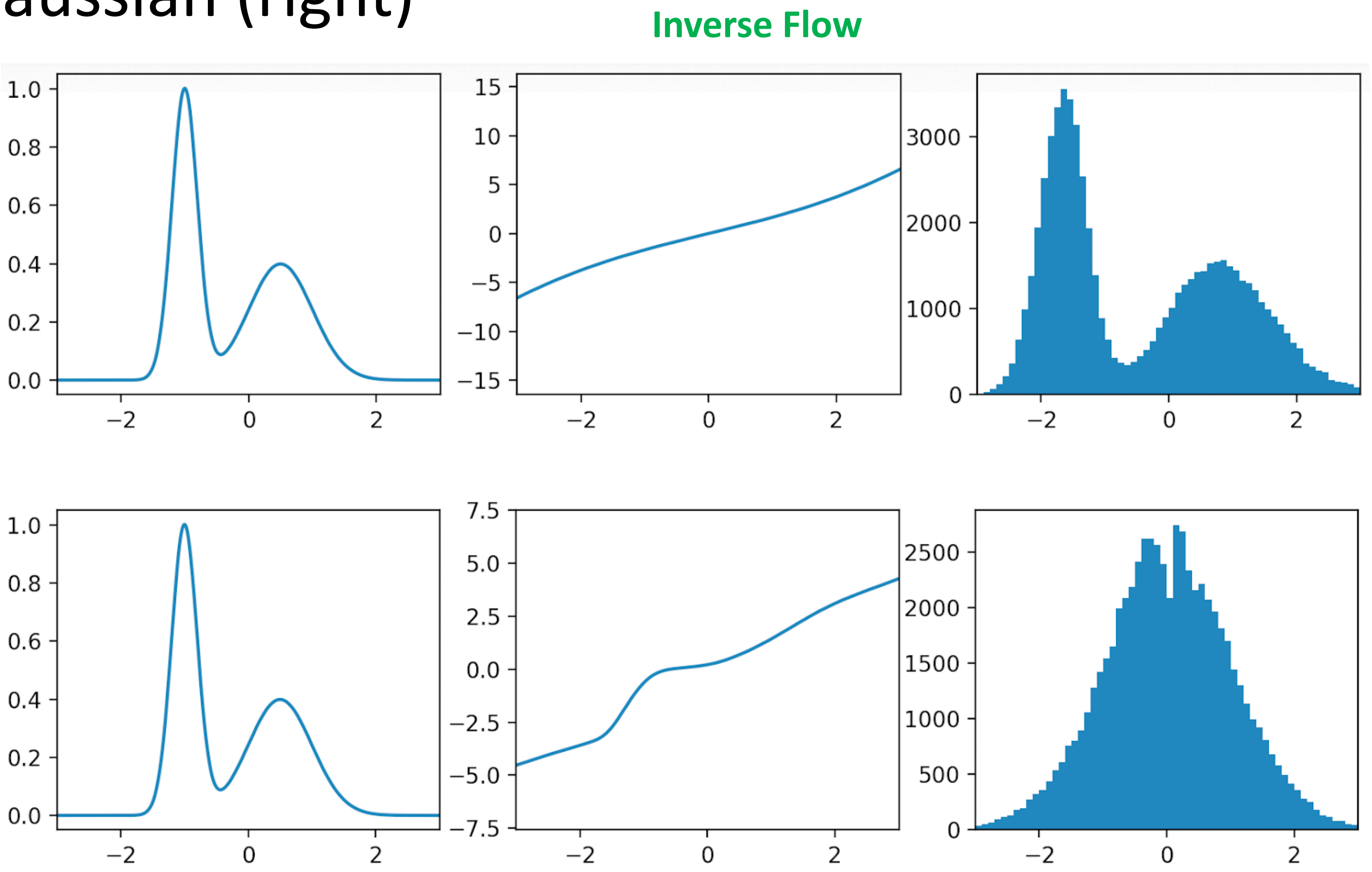
Determinant of Jacobian matrix

Flows: Example

- Inverse flow to a Gaussian (right)

- **Before** training:

- **After** 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:
- Desirable properties:
  - Invertible
  - Differentiable

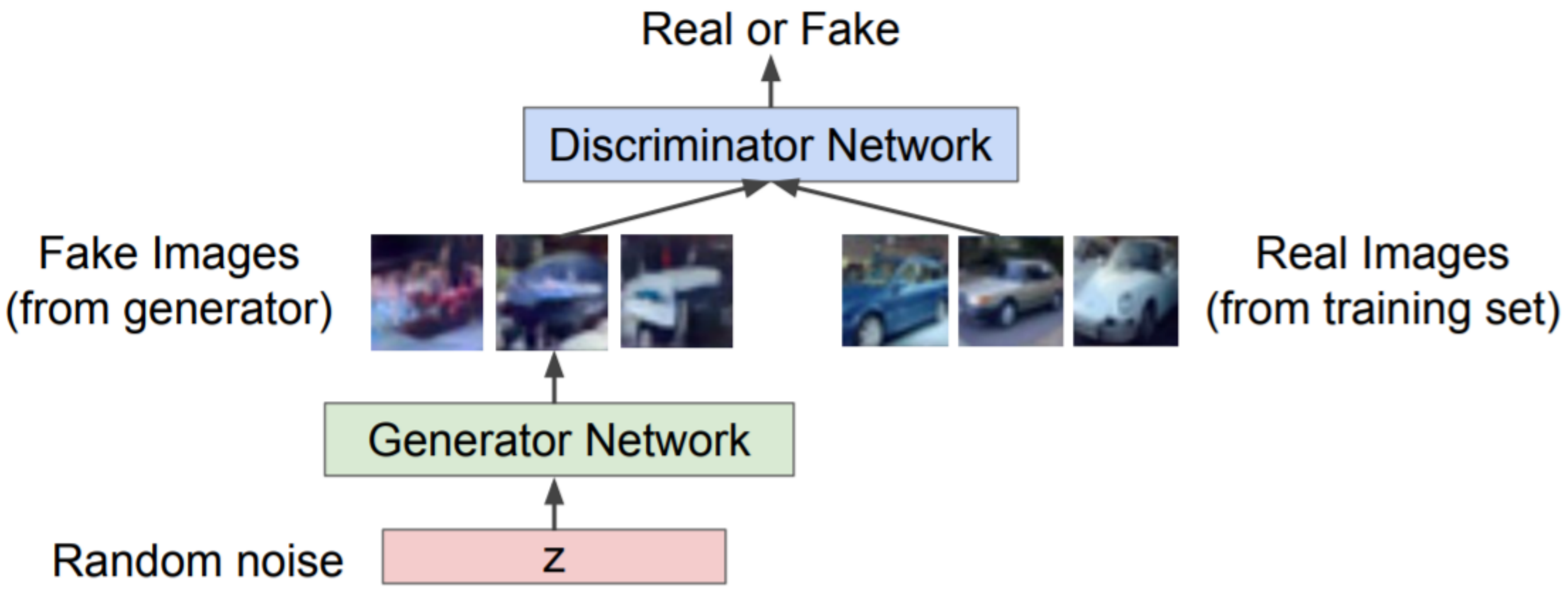
GANs: Generative Adversarial Networks

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

- GANs do this.
  - Think of art forgery
  - 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.

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

Real data, want to classify 1

Fake data, want to classify 0

GAN Training: 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

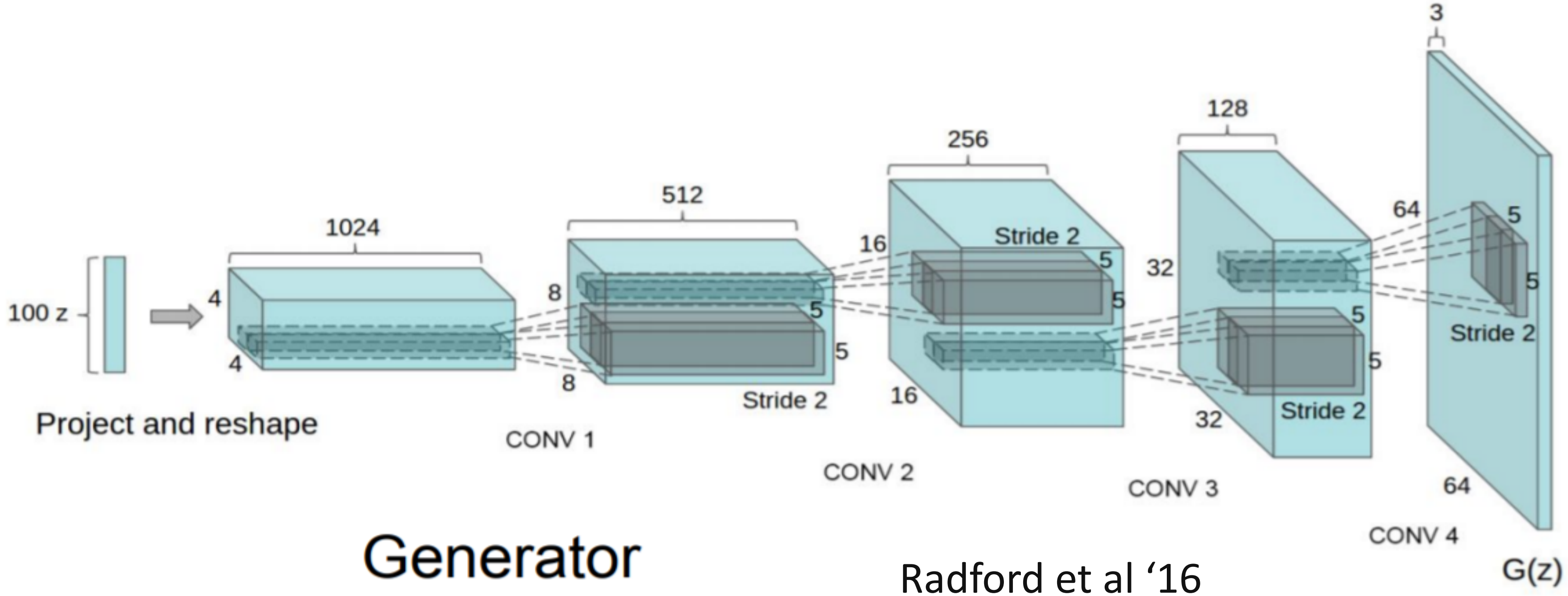
$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

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

- Can still be challenging.

GAN Architectures

- **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)
  - Similar to a reversed 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