Generative Models

Instructor: John Thickstun

Discussion Board: Available on Canvas

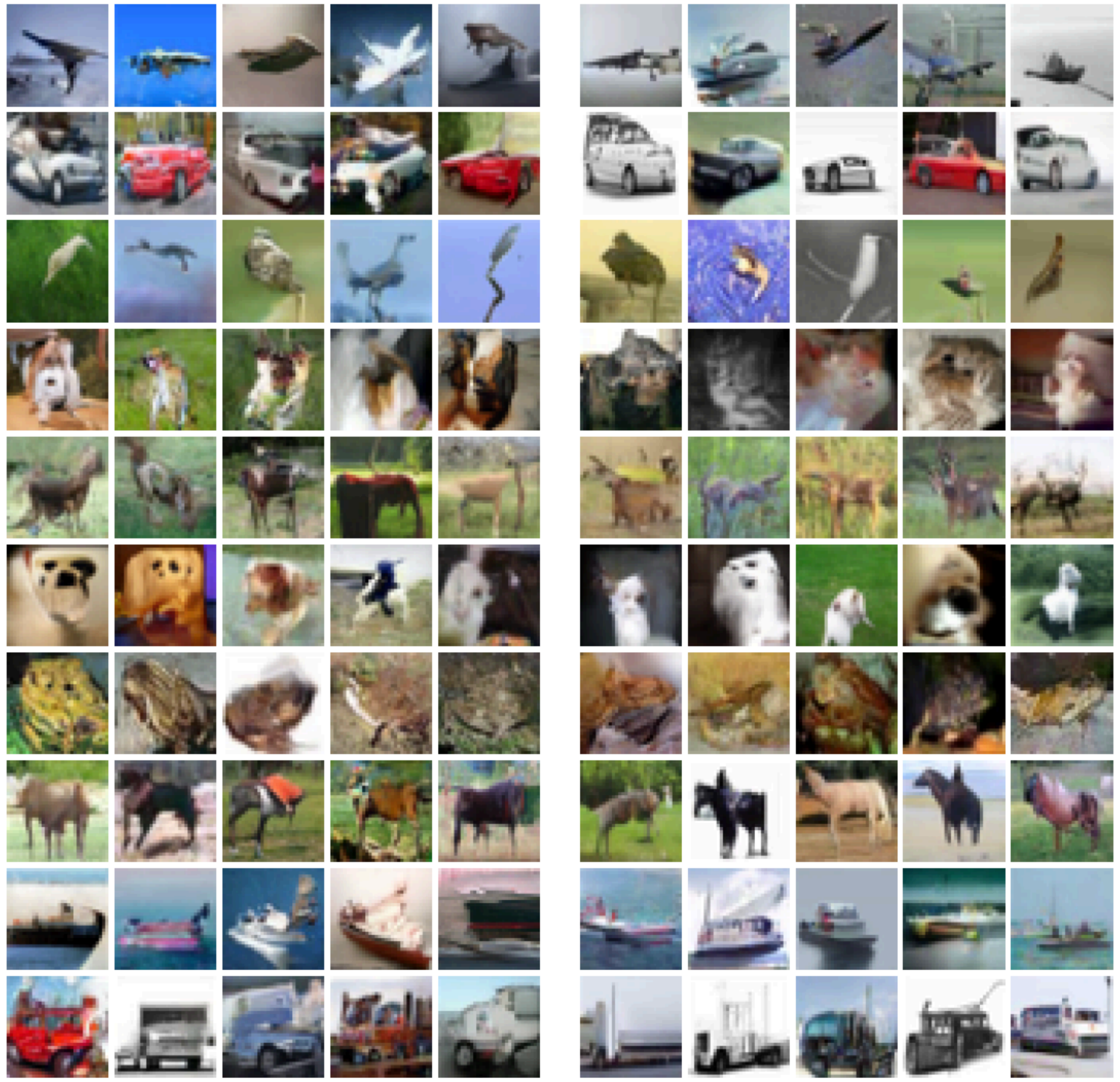
Zoom Link: Available on Canvas (or email the instructor)

Instructor Contact: thickstn@cs.washington.edu

Course Webpage: https://courses.cs.washington.edu/courses/cse599i/20au/

Course Overview

- **Autoregressive Models**
- Variational Autoencoders
- Generative Adversarial Nets
- Generative Flow
- Energy-Based Models



Parmar et. al. (ICML, 2018)

Course Overview

- Autoregressive Models
- **Variational Autoencoders**
- Generative Adversarial Nets
- Generative Flow
- Energy-Based Models

Vahdat and Kautz (Preprint 2020)

Course Overview

- Autoregressive Models
- Variational Autoencoders
- **Generative Adversarial Nets**
- Generative Flow
- Energy-Based Models

Brock et. al. (ICLR, 2019)

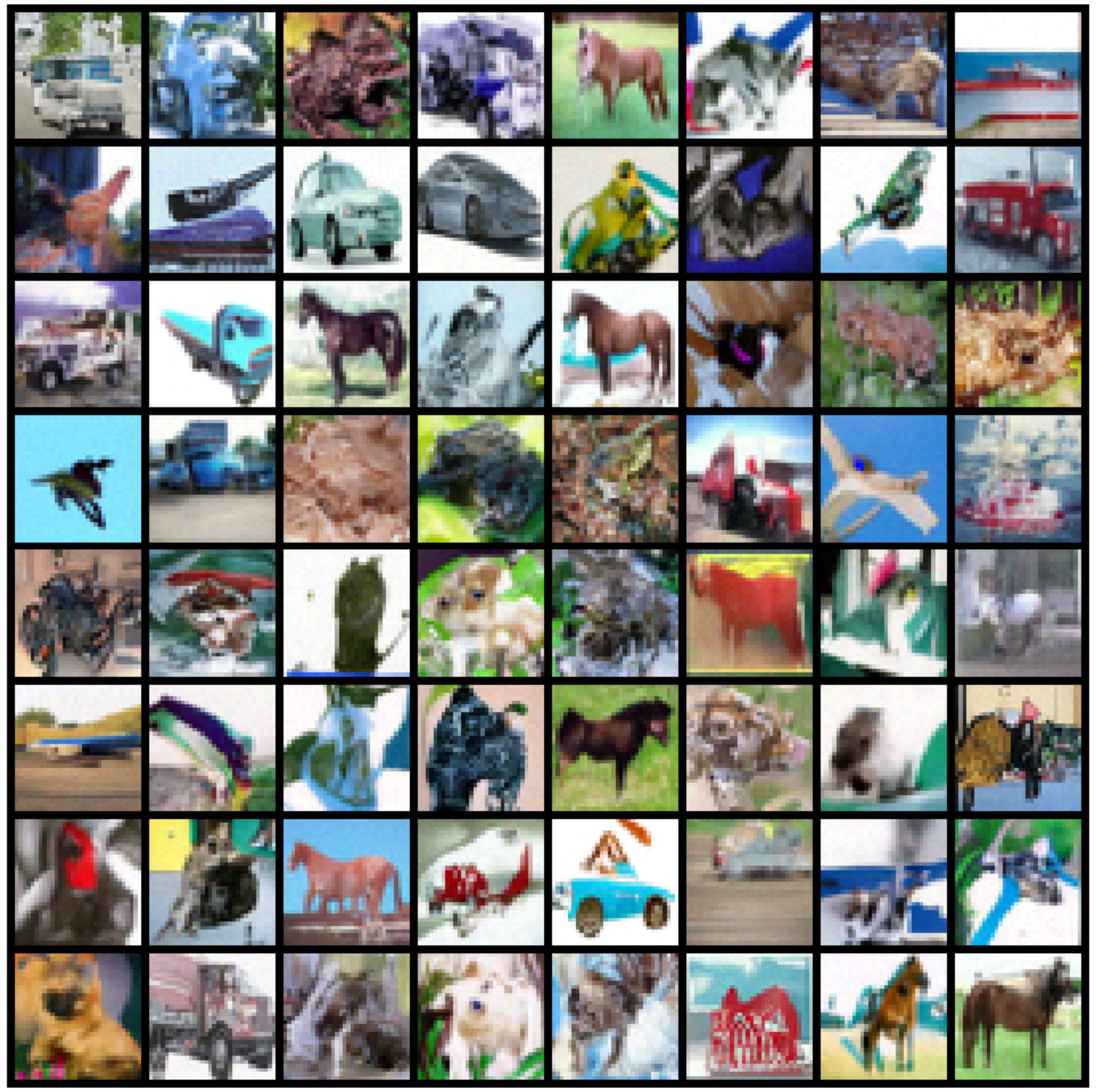
Course Overview

- Autoregressive Models
- Variational Autoencoders
- Generative Adversarial Nets
- **Generative Flow**
- Energy-Based Models

Ma et. al. (Neurips, 2019)

Course Overview

- Autoregressive Models
- Variational Autoencoders
- Generative Adversarial Nets
- Generative Flow
- **Energy-Based Models**



Song and Ermon (ICML, 2018)

Computing Resources

Hyperparamter	MNIST 28×28	CIFAR-10 32×32	ImageNet 32×32	CelebA HQ 64×64	CelebA HQ 256×256	FFHQ 256×256
# epochs	400	400	40	500	400	200
batch size per GPU	200	32	8	16	4	4
# normalizing flows	0	2	2	2	4	4
# latent variable scales	2	1	1	3	5	5
# groups in each scale	5, 10	30	28	5, 10, 20	4, 4, 4, 8, 16	4, 4, 4, 8, 16
spatial dims of z in each scale	4^2, 8^2	16^2	16^2	8^2, 16^2, 32^2	8^2, 16^2, 32^2, 64^2, 128^2	8^2, 16^2, 32^2, 64^2, 128^2
# channel in z	20	20	20	20	20	20
# initial channels in enc.	32	128	192	64	32	32
# residual cells per group	1	2	2	2	2	2
λ	0.01	0.1	0.01	0.01	0.01	0.1
GPU type	16-GB V100	16-GB V100	16-GB V100	16-GB V100	32-GB V100	32-GB V100
# GPUs	2	8	32	8	24*	24*
total train time (h)	24	100	200	150	180	220

NVAE: Vahdat and Kautz (Preprint 2020)

Class Computing Resources

- https://colab.research.google.com/
- 1 GPU (probably a K80) or TPU per person
- Homeworks are designed to run in Colab with Python and PyTorch
- Homework 0:
    - ➡ Familiarize yourself with Google Colab
    - ➡ Familiarize yourself with PyTorch

Expectations

- Generative models are an active area of research.
- Lectures will give a high-level sketch of ideas.
- Lecture notes (website) will give a more complete treatment, with references.
- I'll try to present a clear picture of what's going on mathematically.
- We'll also try to understand how these ideas are put into practice at scale.
- There will be a lot of content in this class:
    - If you want to focus more on either the theory or empirical side, that's fine.
    - If certain topics interest you more than others: also fine.

Course Project

- Anything related to generative models, theoretical or applied.
- Partner with up to 4 people.
- Consider what computing resources you might need and plan ahead.
- Examples of possible projects:
  - An application of generative models to your own research.
  - Reproduction of empirical results reported in a recent paper.
  - Exposition or extension of a technical theoretical result in a recent paper.
  - Application of generative modeling techniques to a novel dataset.

5-Minute Break

Generative Modeling

- Given finite samples $x_1, \ldots, x_n \sim p$, and unlimited samples $z \sim q$.
- Learn a function $g_\theta : \mathcal{Z} \to \mathcal{X}$, which induces a distribution p_θ on $\mathcal{X}$.
- E.g. if $q(z) = \mathrm{Uniform}(0, 1)$ and $g_\theta(z) = \theta z$ then $p_\theta(x) = \mathrm{Uniform}(0, \theta)$.
- Learn the parameters so that $p_\theta \approx p$.
- Generate samples $x = g_\theta(z) \sim p_\theta$.

Sampling from a Gaussian

- Given samples $x_1, \ldots, x_n \sim \mathcal{N}(\mu, \sigma^2)$, and unlimited samples $z \sim \mathcal{N}(0,1)$.
- Estimate the mean and variance: $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2$.
- Define $g(z) = \hat{\mu} + \hat{\sigma} z$.
- Generate samples $x = g(z) \sim \mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$.

Pushforward Distributions

Definition 1. *Given a probability space $(\mathcal{Z}, q)$, a (measurable) function $g : \mathcal{Z} \to \mathcal{X}$ induces a* ***pushforward*** *distribution on $\mathcal{X}$ defined, for any (measurable) set $A \subset \mathcal{X}$ by*

$$\Pr(A) = \int_{g^{-1}(A)} q(z)\, dz.$$

- If $\mathcal{Z}$ and $\mathcal{X}$ are both discrete spaces: $\Pr(A) = \sum_{z\,:\,g(z)\in A} q(z).$
- Suppose $z \sim \mathrm{Uniform}(0,1)$ and $g(z) = \mathbf{1}_{z<p}$, then $g(z) \sim \mathrm{Bernoulli}(p)$.

Inverse Transform Sampling

- I have samples from $\mathrm{Uniform}(0,1)$. I want samples with CDF F.
- Define a generator (the inverse CDF) $g(z) = \inf\{x : F(x) \geq z\}$.
- If $z \sim \mathrm{Uniform}(0,1)$, then $g(z)$ is distributed according to F.
- Convert samples from the uniform distribution to an arbitrary distribution!

Finite Generative Modeling

- Given finite samples $x_1, \ldots, x_n \sim p$ from a finite space $\mathcal{X}$.
- Estimate the probability mass $\hat{\pi}_x$ of each element of $\mathcal{X}$.
- Sample from the distribution $\hat{p}(x) = \hat{\pi}_x$ using Inverse Transform Sampling.
- What's a good estimate $\hat{p}(x)$ of the probability mass function $p(x)$?

Maximum Likelihood Estimation

- What's a good estimate $\hat{p}(x)$ of the probability mass function $p(x)$?

- What about count statistics (the MLE)?

$$\hat{\pi}_x = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{x_i=x}.$$

- What happens if "$d > n$"?

- I.e. if the size of $\mathcal{X}$ is larger than the number of observed samples.

Continuous Modeling

- Given finite samples $x_1, \ldots, x_n \sim p$ from a continuous distribution $p(x)$.
- Estimate the probability of each element of $\mathcal{X}$?
- Using the MLE?

$$\hat{p}(x) = \begin{cases} \frac{1}{n} & \text{if } x \in \{x_1, \ldots, x_n\}, \\ 0 & \text{otherwise.} \end{cases}$$

Parametric Estimation

- For example, learning a Gaussian.
- Restricting to a parametric family of functions regularizes the problem.
- But we want to learn expressive distributions…
- Parameterize with an expressive family: neural nets!

Maximize the Likelihood?

- Use a neural network to parameterize $g_\theta : \mathcal{Z} \to \mathcal{X}$.
- Optimize θ so the pushforward distribution $p_\theta(x)$ approximates $p(x)$.
- How do we optimize? Maximize the likelihood?

$$\sup_\theta \mathop{\mathbb{E}}_{x \sim p} \log p_\theta(x) \approx \sup_\theta \frac{1}{n} \sum_{i=1}^{n} \log p_\theta(x_i).$$

- How do we compute $p_\theta(x)$?

Calculating the Likelihood?

Definition 1. *Given a probability space $(\mathcal{Z}, q)$, a (measurable) function $g : \mathcal{Z} \to \mathcal{X}$ induces a* ***pushforward*** *distribution on $\mathcal{X}$ defined, for any (measurable) set $A \subset \mathcal{X}$ by*

$$\Pr(A) = \int_{g^{-1}(A)} q(z)\, dz.$$

- How do we compute $p_\theta(x)$ defined by pushforward $g_\theta : \mathcal{Z} \to \mathcal{X}$?

$$\Pr(A) = \Pr(g^{-1}(A)) = \int_{g^{-1}(A)} q(z)\, dz = \int_A q(g^{-1}(x))|\nabla_x g^{-1}(x)|\, dx.$$

- So the density is given by $p_\theta(x) = q(g_\theta^{-1}(x))|\nabla_x g_\theta^{-1}(x)|$.
- Uh oh.

Density Estimation

- Approximating the function $p(x)$ is called the density estimation problem.
- Suppose we (somehow) optimize a generator g_θ such that $p_\theta \approx p$.
- If we can compute $g_\theta^{-1}(x)$ and $\nabla_x g_\theta^{-1}(x)$ then we have a density estimator.
- In many cases, generation (sampling) is easier than density estimation.