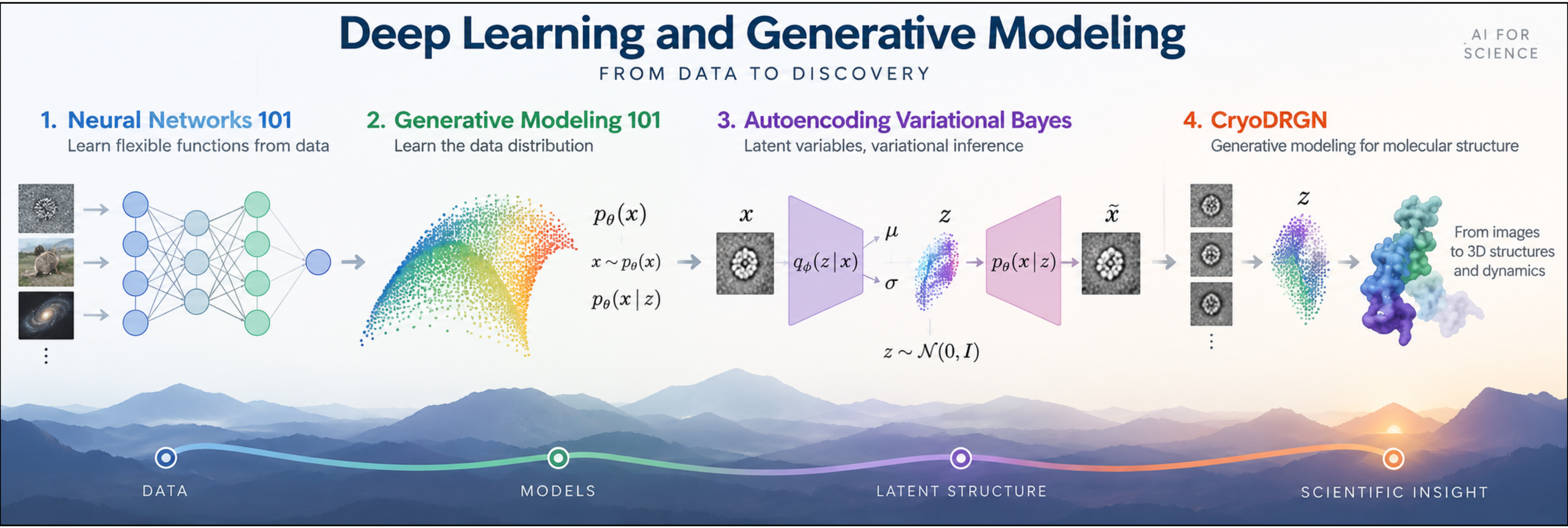
Deep Learning and Generative Modeling 101



Ellen Zhong
Department of Computer Science
Princeton University

Fall 2026
COS 597Y Week 3

PRINCETON UNIVERSITY

Course Logistics

- Everyone should be on slack by now!
- Tentative presentation schedule shared. Let us know if you don’t see your name listed.
  - Once guest lectures are confirmed, I will share a google sheet with (suggested) papers to finalize the schedule

Next Week

- Guest lecture by Zeming Lin (EvolutionaryScale / Biohub)
- Class will be on Zoom (link will be posted in Slack).
- Readings:
  - Attention is all you need
  - Language modeling materializes a world model of protein biology (biorXiv Jun 26)
- Optional Readings:
  - Scaling Laws for Neural Language Models
  - Training Compute-Optimal Large Language Models

Outline for today's lecture

- Neural Networks 101
- Generative Modeling 101
- Autoencoding Variational Bayes (Jack & Sol)
- CryoDRGN (Philipp & Ariana)

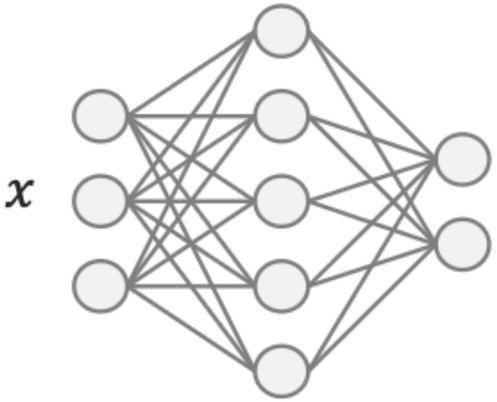
- Post-lecture feedback (anonymous to the presenters, due at 11:59pm):
  - What worked well?
  - What could be improved?
  - What is one question, idea, or takeaway you had?
- Feedback is a gift and giving good feedback is a skill!
- Good feedback is specific, actionable, and constructive.

This lecture

- Review a few key concepts in neural network training
- Develop a taxonomy of generative models
- Curates and adapts material from several excellent teaching resources, including:
  - ETH Zürich, AI in the Sciences and Engineering (Fall 2024)
  - MIT 6.7960, Deep Learning (Fall 2024)

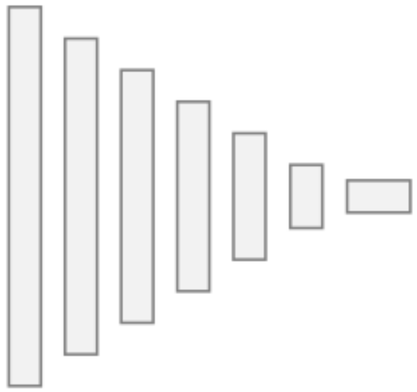
Deep learning

Neural networks are simply **flexible functions** fit to data



$$\hat{y} = NN(x;\theta) = W_2\sigma(W_1 x + b_1)$$

With enough parameters, neural networks can approximate any* arbitrarily complex function
= **universal approximation**



$$\hat{y} = P(\text{dog} \mid x) = 1$$

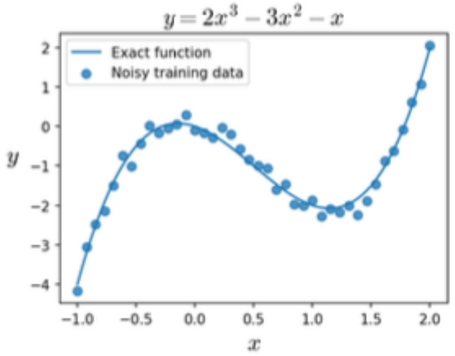
x = array of RGB values

Convolutional neural network

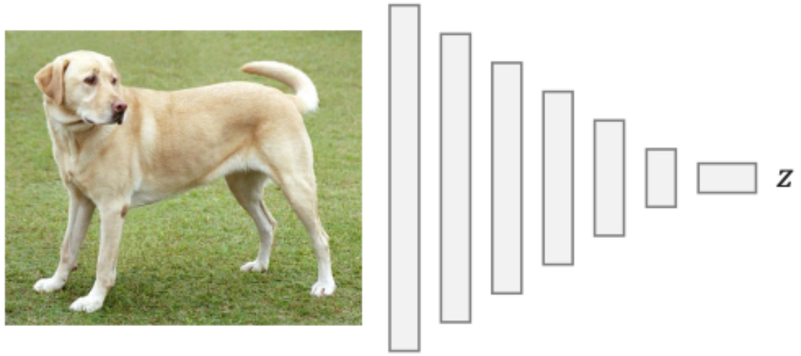
Slide credit: AI in the Sciences and Engineering, ETH Zürich (F24)

Popular deep learning tasks

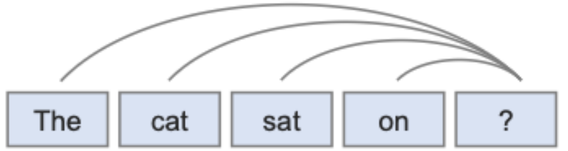
Supervised learning – **regression**



Unsupervised learning – **feature learning**



Unsupervised learning – **autoregression**



Supervised learning – **classification**

$P(\text{dog} \mid x) = 1$

Unsupervised learning – **generative modelling**

but in all cases, the neural network is still a function fit to data!

Slide credit: AI in the Sciences and Engineering, ETH Zürich (F24)

Training

Other names for this:
- *"Statistical inference"*
- *Learning, amortized inference*

Inference

Other names for this:
- *Prediction*
- *Thinking, reasoning, cognition*

Pre-training

Given data, learn a model or representation

Example methods:
- Generative modeling
- Representation learning

Post-training

Given a model and new data, update the model

Example methods:
- Finetuning
- RLHF

Given a model and data during deployment, *update the model or its behavior*

Example methods:
- Prompting
- In-Context Learning
- Test-Time Training
- Continual learning
- Feedback control

Search

Given a model and a query, find the best answer to the query

Example methods:
- Best-of-N
- Beam search
- MCTS
- Chain-of-Thought

"Reinforcement learning", STaR, self-instruct, self-play, …

Slide credit: Phillip Isola, MIT 6.7960 (F24)

The Bitter Lesson

Rich Sutton

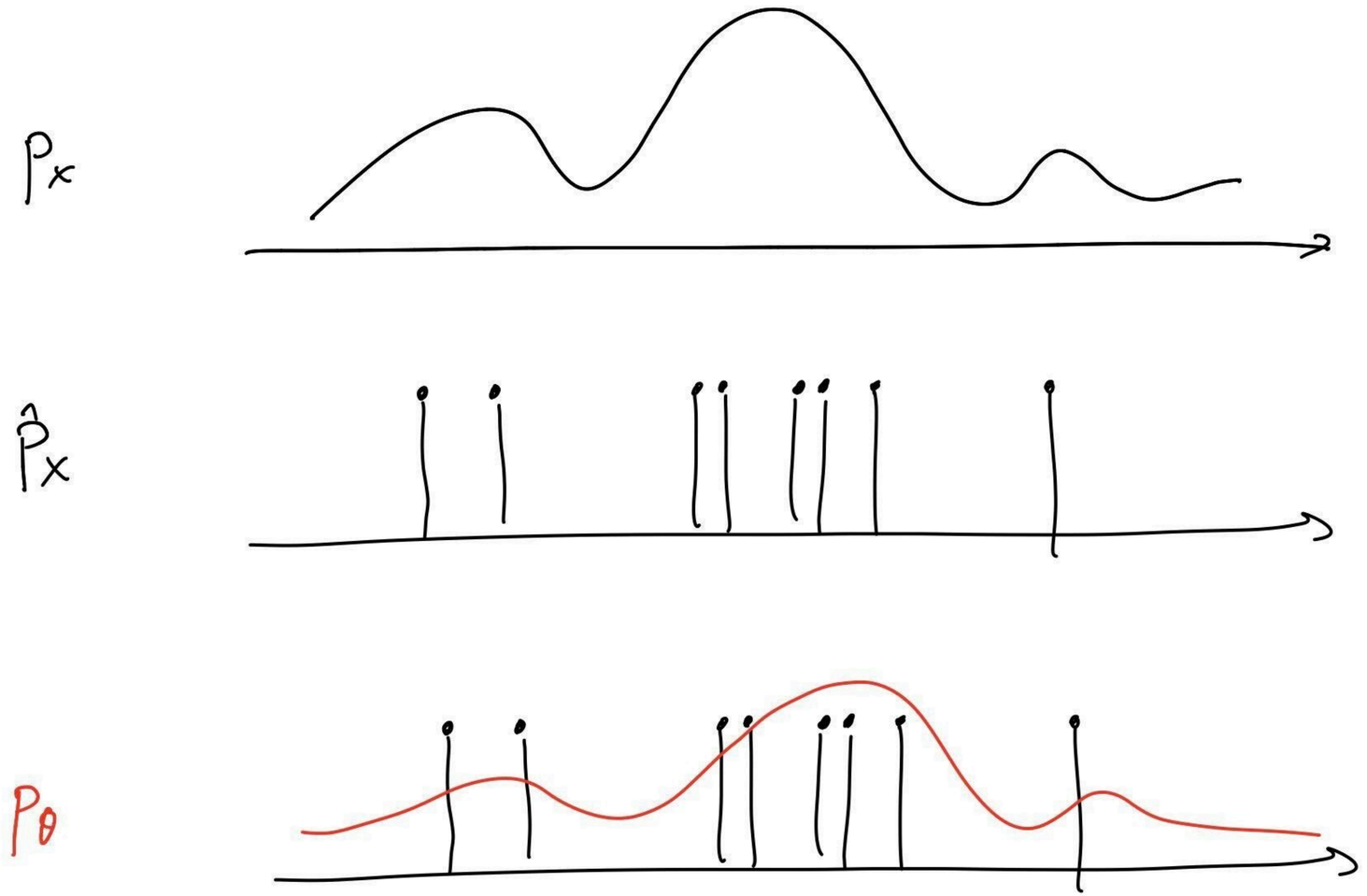
March 13, 2019

One thing that should be learned from the bitter lesson is the great power of general purpose methods, of methods that continue to scale with increased computation even as the available computation becomes very great. The two methods that seem to scale arbitrarily in this way are *search* and *learning*.

Challenges of NN function fitting

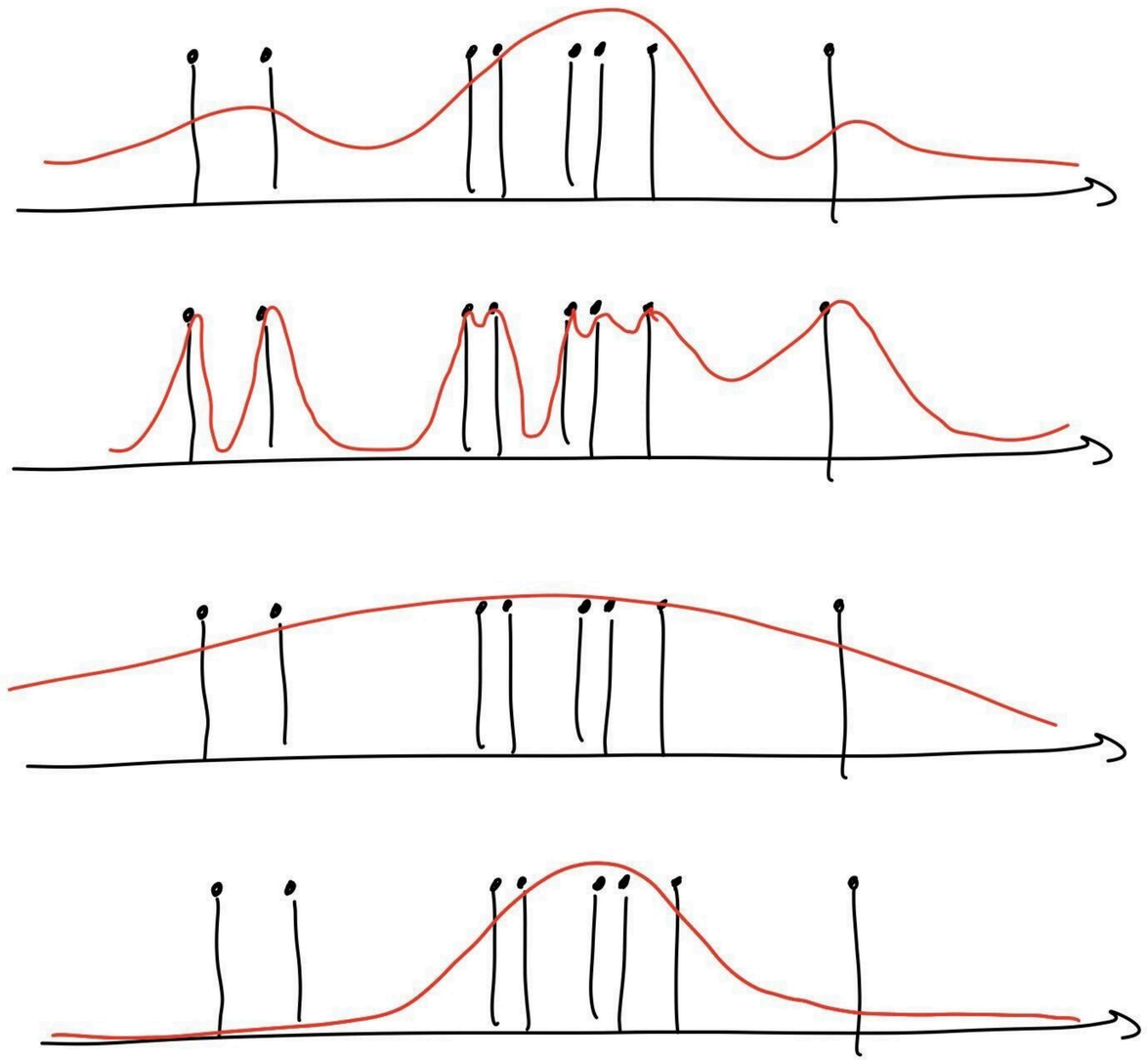
- Three sources of error when training neural networks
  - Approximation error
  - Estimation error
  - Optimization error

Density Estimation



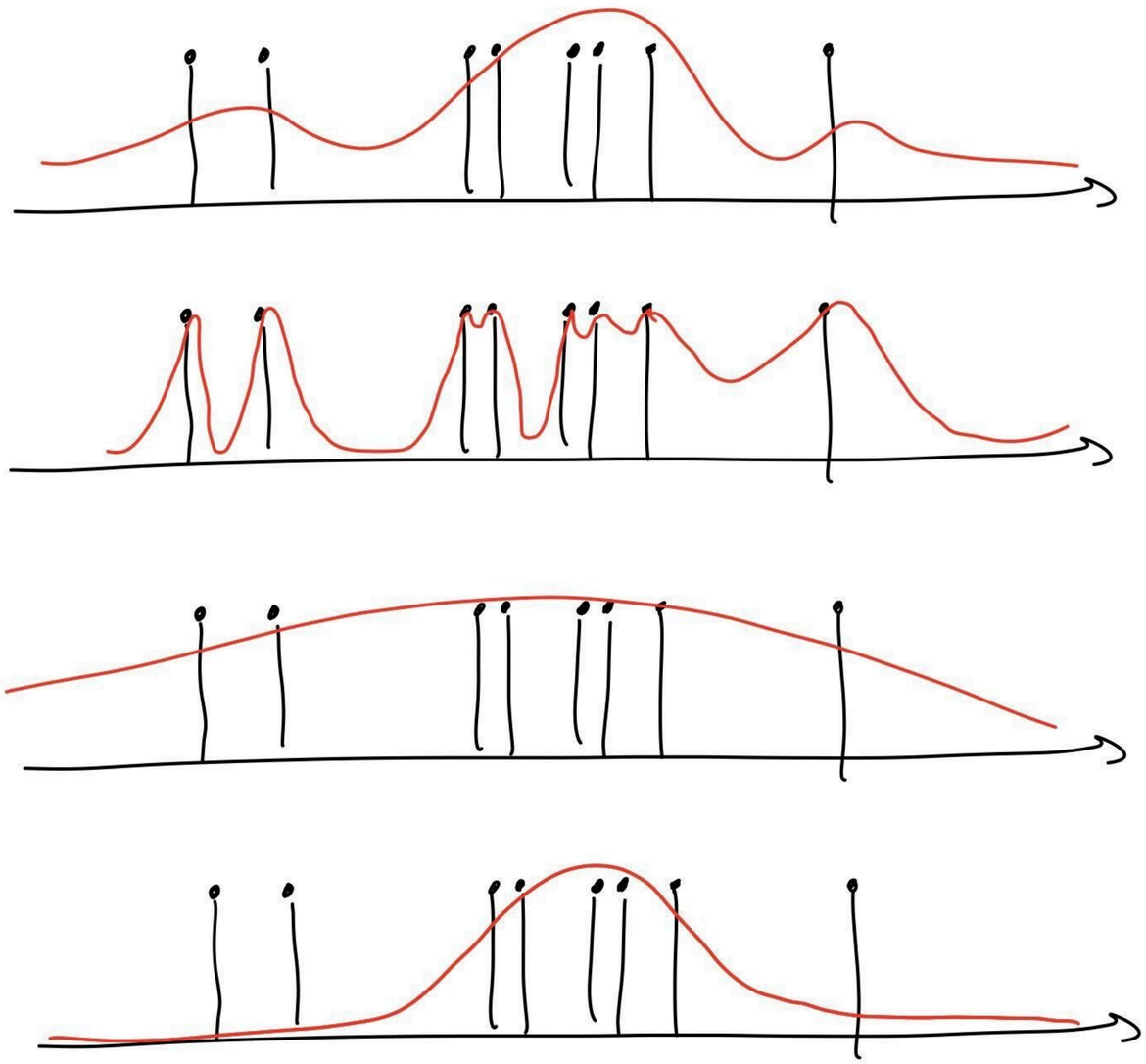
Slide credit: Mark Goldstein

Density Estimation: failure modes



Slide credit: Mark Goldstein

Density Estimation: failure modes

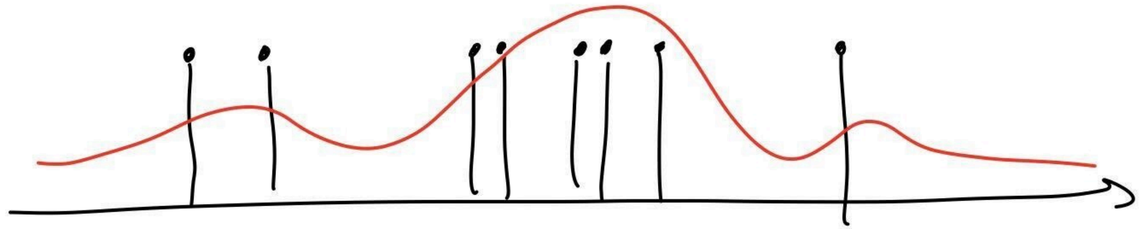


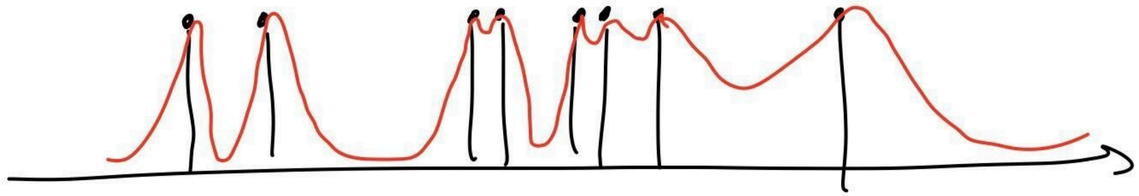
Overfitting / memorization

Undercutting / oversmoothing

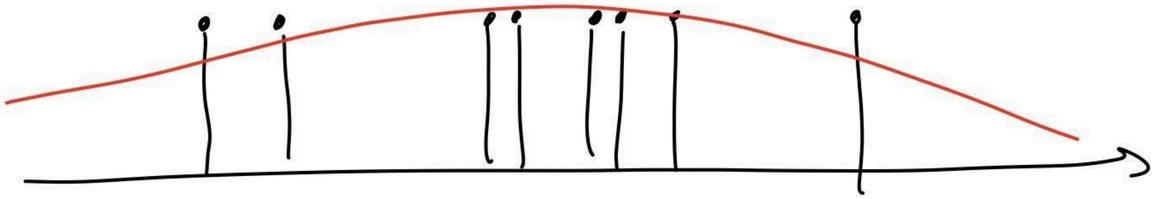
Mode collapse

Density Estimation: failure modes

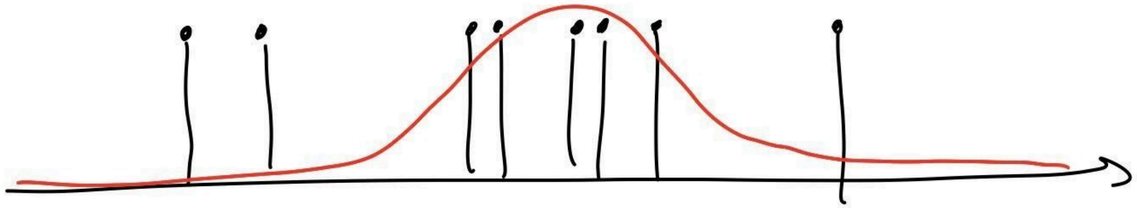




Overfitting / memorization

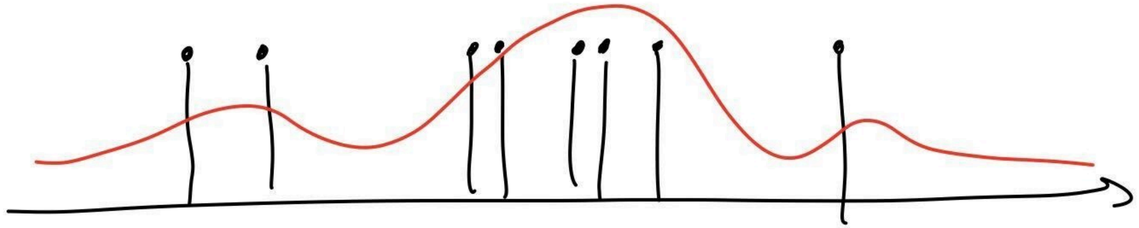


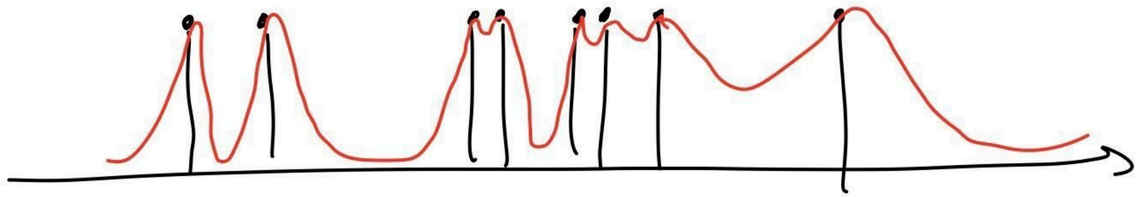
Undercutting / oversmoothing



Mode collapse

Density Estimation: failure modes



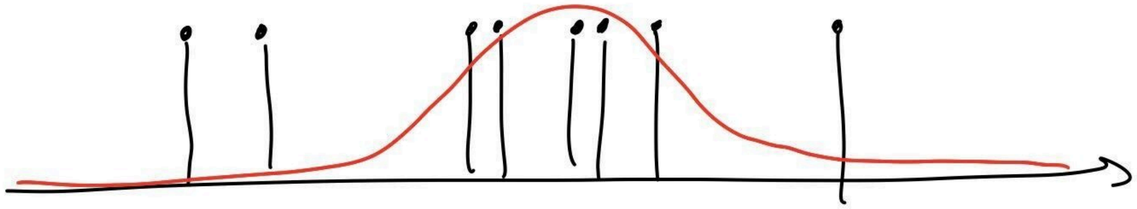


Overfitting / memorization

Have protein-ligand cofolding methods moved beyond memorisation?

Peter Škrinjar, Jérôme Eberhardt, Gerardo Tauriello, Torsten Schwede, Janani Durairaj

doi: https://doi.org/10.1101/2025.02.03.636309



Mode collapse

Density Estimation: failure modes

Have protein-ligand cofolding methods moved beyond memorisation?

Peter Škrinjar, Jérôme Eberhardt, Gerardo Tauriello, Torsten Schwede, Janani Durairaj

doi: https://doi.org/10.1101/2025.02.03.636309

Deep learning has driven major breakthroughs in protein structure prediction, however the next critical advance is accurately predicting how proteins interact with small molecule ligands, to enable real-world applications such as drug discovery. Recent cofolding methods aim to address this challenge, but evaluating their performance has been inconclusive due to the lack of relevant bench-marking datasets. Here we present a comprehensive evaluation of four leading all-atom cofolding methods using our newly introduced benchmark dataset Runs N' Poses, which comprises 2,600 high-resolution protein-ligand systems released after the training cutoff used by these methods. We demonstrate that current cofolding approaches largely memorise ligand poses from their training data, hindering their use for *de novo* drug design. With this assessment and benchmark dataset, we aim to accelerate progress in the field by allowing for a more realistic assessment of the current state-of-the-art deep learning methods for predicting protein-ligand interactions.

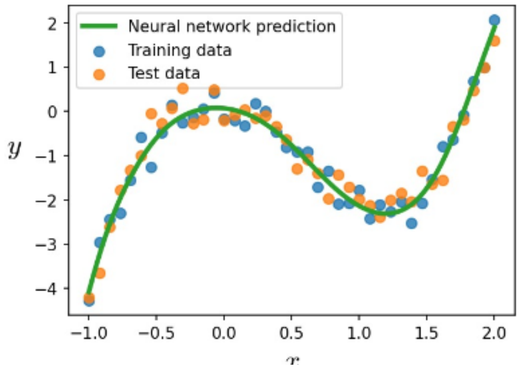
Understanding error sources

Training loss:

$$L(\theta) = \frac{1}{N}\sum_{i}^{N}(NN(x_i;\theta) - y_i)^2$$
$$D = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

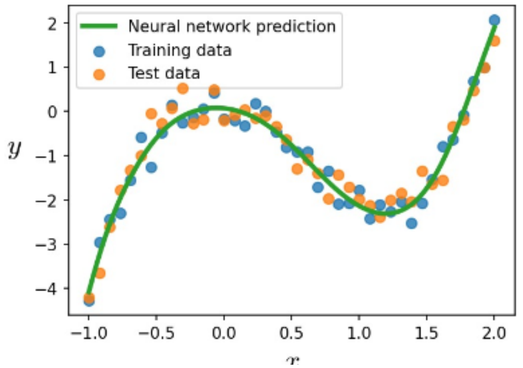
This is also known as the **empirical loss**, and can be more generally written as:

$$L(\theta) = \frac{1}{N}\sum_{i}^{N} l(NN(x_i;\theta), y_i), \qquad x, y \sim p(x, y)$$



Slide credit: AI in the Sciences and Engineering, ETH Zürich (F24)

Understanding error sources



Training loss:

$$L(\theta) = \frac{1}{N}\sum_{i}^{N}(NN(x_i;\theta) - y_i)^2$$
$$D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$$

This is also known as the **empirical loss**, and can be more generally written as:

$$L(\theta) = \frac{1}{N}\sum_{i}^{N} l(NN(x_i;\theta), y_i), \qquad x, y \sim p(x, y)$$

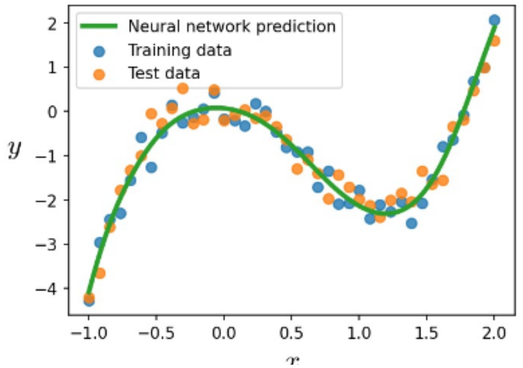
But what we really want to minimise is the **expected loss**:

$$\mathcal{L}(\theta) = \iint l(NN(x;\theta), y)\, p(x, y) dx dy$$
$$= E_{(x,y)\sim p}[l(x, y;\theta)]$$

Why is this impossible in practice?

Slide credit: AI in the Sciences and Engineering, ETH Zürich (F24)

Understanding error sources



Training loss:

$$L(\theta) = \frac{1}{N}\sum_{i}^{N}(NN(x_i;\theta) - y_i)^2$$
$$D = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

This is also known as the **empirical loss**, and can be more generally written as:

$$L(\theta) = \frac{1}{N}\sum_{i}^{N} l(NN(x_i;\theta), y_i), \qquad x, y \sim p(x, y)$$

But what we really want to minimise is the **expected loss**:

$$\mathcal{L}(\theta) = \iint l(NN(x;\theta), y)\, p(x, y) dx dy$$
$$= E_{(x,y)\sim p}[l(x, y;\theta)]$$

Why is this impossible in practice?
Because, usually, we only have access to a finite number of samples from $p(x, y)$ (our training dataset, D)

Slide credit: AI in the Sciences and Engineering, ETH Zürich (F24)

Understanding error sources

Training loss:

$$L(\theta) = \frac{1}{N}\sum_{i}^{N}(NN(x_i;\theta) - y_i)^2$$
$$D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$$

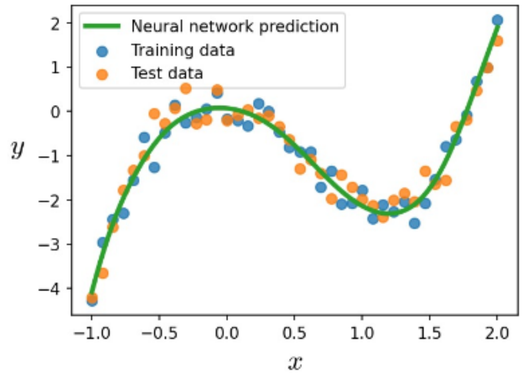
This is also known as the **empirical loss**, and can be more generally written as:

$$L(\theta) = \frac{1}{N}\sum_{i}^{N} l(NN(x_i;\theta), y_i), \qquad x, y \sim p(x, y)$$

But what we really want to minimise is the **expected loss**:

$$\mathcal{L}(\theta) = \iint l(NN(x;\theta), y)\, p(x, y) dx dy$$
$$= E_{(x,y)\sim p}[l(x, y;\theta)]$$

Why is this impossible in practice?
Because, usually, we only have access to a finite number of samples from $p(x, y)$ (our training dataset, D)



This leads to

1) **Estimation error** (finite amount of training data) (aka generalisation error)

$$\mathcal{E}_{\text{est}} = \mathcal{L}(NN) - \mathcal{L}(NN^*)$$

NN which minimises empirical loss

NN which minimises expected loss

Slide credit: AI in the Sciences and Engineering, ETH Zürich (F24)

Understanding error sources

Training loss:

$$L(\theta) = \frac{1}{N}\sum_{i}^{N}(NN(x_i;\theta) - y_i)^2$$
$$D = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

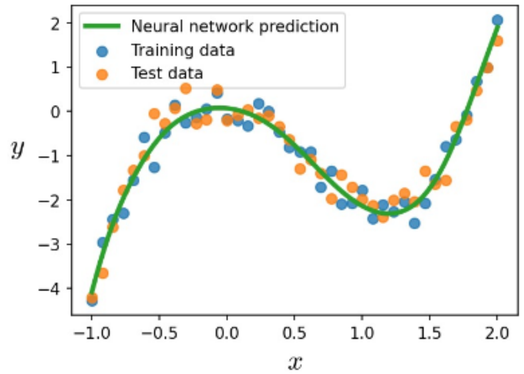
This is also known as the **empirical loss**, and can be more generally written as:

$$L(\theta) = \frac{1}{N}\sum_{i}^{N} l(NN(x_i;\theta), y_i), \qquad x, y \sim p(x,y)$$

But what we really want to minimise is the **expected loss**:

$$\mathcal{L}(\theta) = \iint l(NN(x;\theta), y)\, p(x,y)dxdy$$
$$= E_{(x,y)\sim p}[l(x,y;\theta)]$$

Why is this impossible in practice?
Because, usually, we only have access to a finite number of samples from $p(x,y)$ (our training dataset, D)



This leads to

1) **Estimation error** (finite amount of training data) (aka generalisation error)

$$\mathcal{E}_{\text{est}} = \mathcal{L}(NN) - \mathcal{L}(NN^*)$$

NN which minimises empirical loss — NN which minimises expected loss

We also have

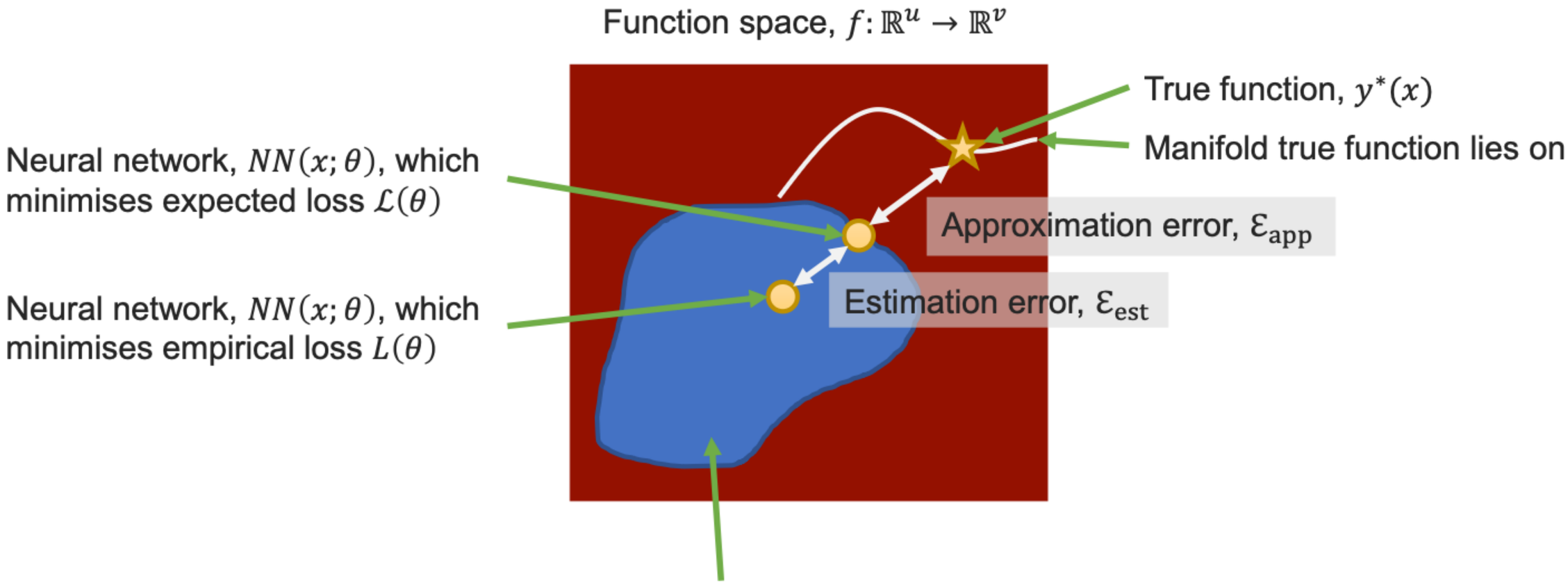
2) **Approximation error** (limited expressivity of neural network)

$$\mathcal{E}_{\text{app}} = \mathcal{L}(NN^*) - \mathcal{L}(y^*)$$

NN which minimises expected loss — True function which minimises expected loss

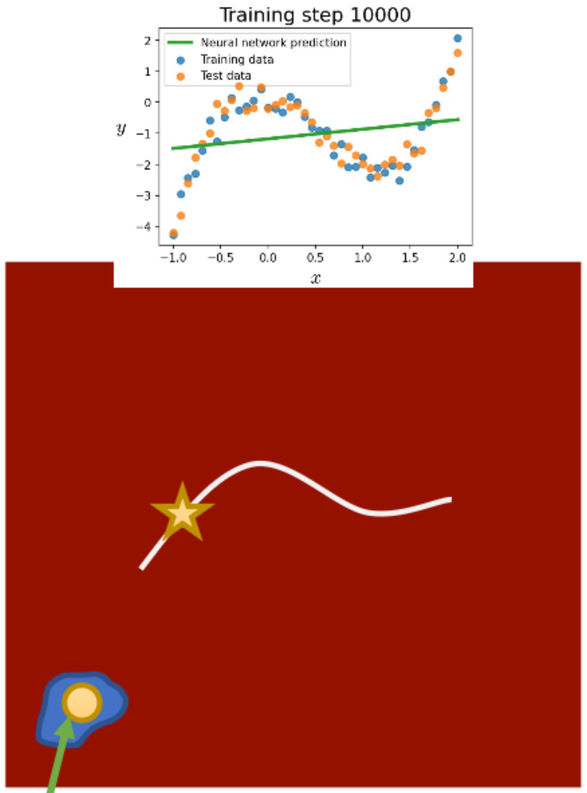
Slide credit: AI in the Sciences and Engineering, ETH Zürich (F24)

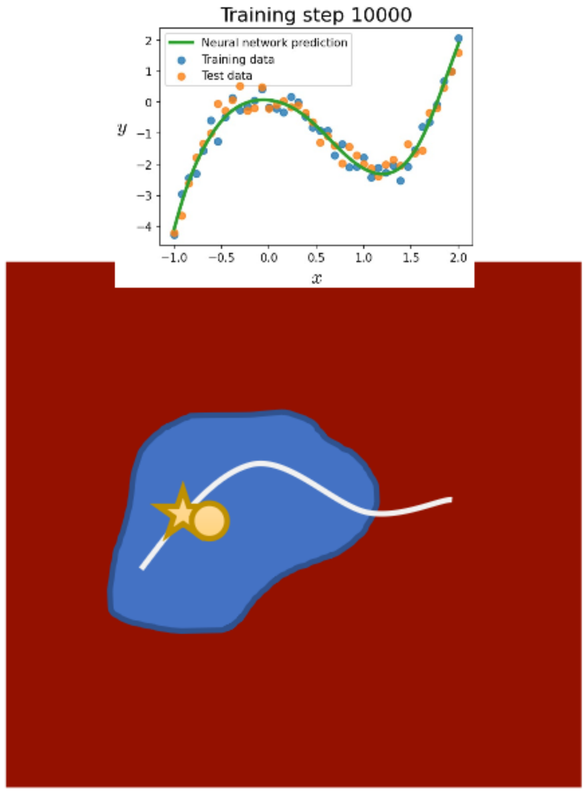
Function space

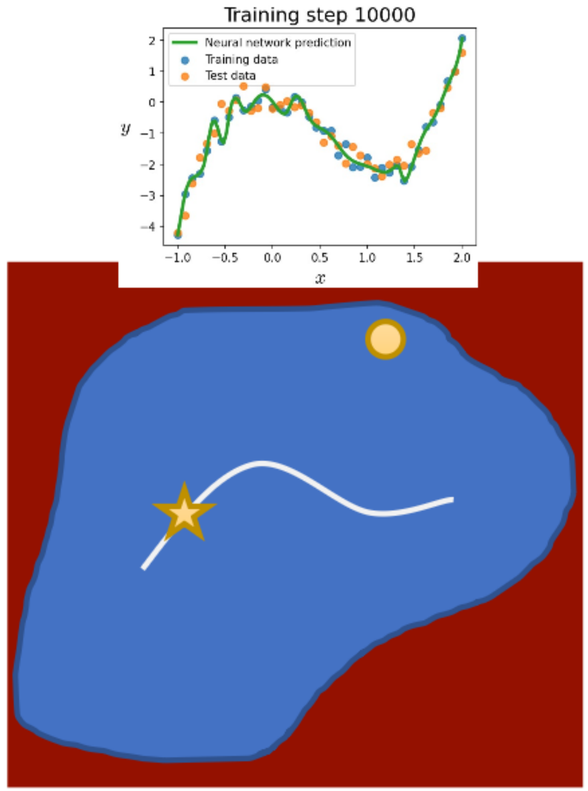


Slide credit: AI in the Sciences and Engineering, ETH Zürich (F24)

Function space







Underfit

Ideal model

Overfit

Neural network, $NN(x;\theta)$, which minimises empirical loss $L(\theta)$

Slide credit: AI in the Sciences and Engineering, ETH Zürich (F24)

Improving model performance

If you have high bias (or underfitting)
- Increase model **complexity** (e.g. number of free parameters)
- Or modify model architecture (shift location of hypothesis space)

If you have high variance (or overfitting)
- Increase the amount of training **data**
- Or **regularise** the neural network in some other way

Slide credit: AI in the Sciences and Engineering, ETH Zürich (F24)

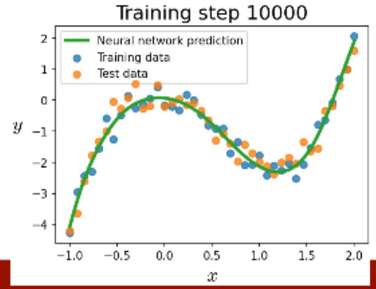
Regularisation

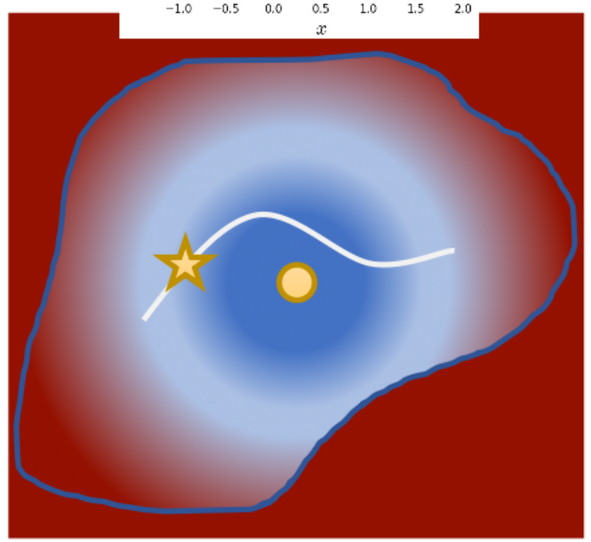
Regularisation = **restrict** the space of possible functions a neural network can learn / **prefer** certain regions of the function space / impart a **prior** on the learning algorithm

Ways to regularise neural networks:

- Reduce model complexity
- Modify model architecture

- Increase amount of (or augment) training data
- Weight regularization / pruning
- Additional loss terms
- Dropout
- Early stopping

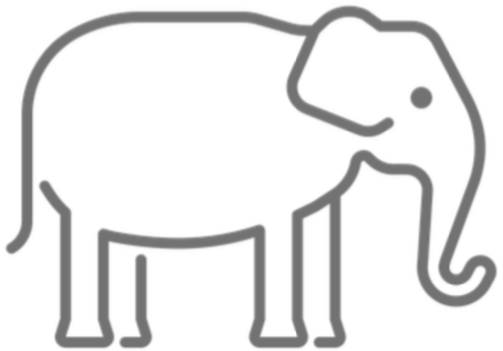




Regularised

Slide credit: AI in the Sciences and Engineering, ETH Zürich (F24)

Another source of error…



Up until now, we have assumed that the learned neural network parameters **globally** minimise the empirical loss, i.e.

$$\theta^* = \min_\theta L(\theta)$$

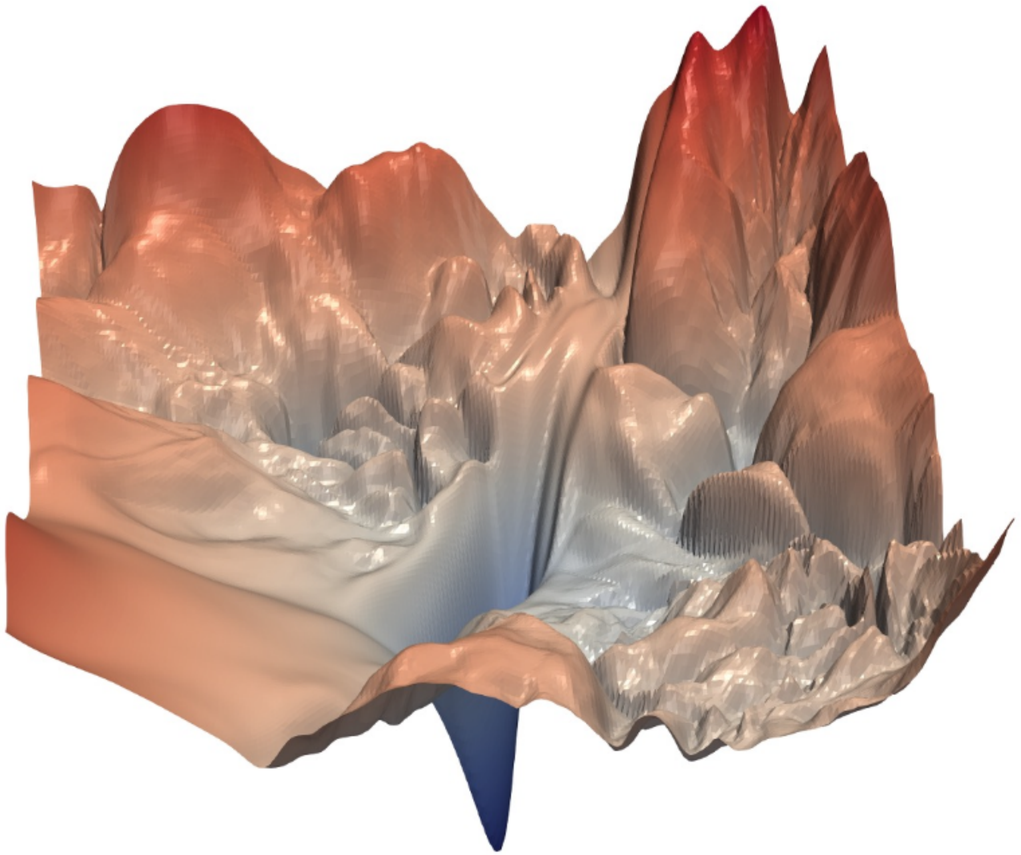
⚠ But, in practice, finding the global minima is **extremely** hard

So, we must also consider the effect of **optimisation error** on the performance of neural networks

⚠ => Even though a large enough neural network can represent any arbitrary function (universal approximation theorem), there is **no guarantee** finding this network is tractable!

Slide credit: AI in the Sciences and Engineering, ETH Zürich (F24)

Loss landscape



Li et al, Visualizing the Loss Landscape of Neural Nets, NeurIPS (2018)

2D visualization of the loss function

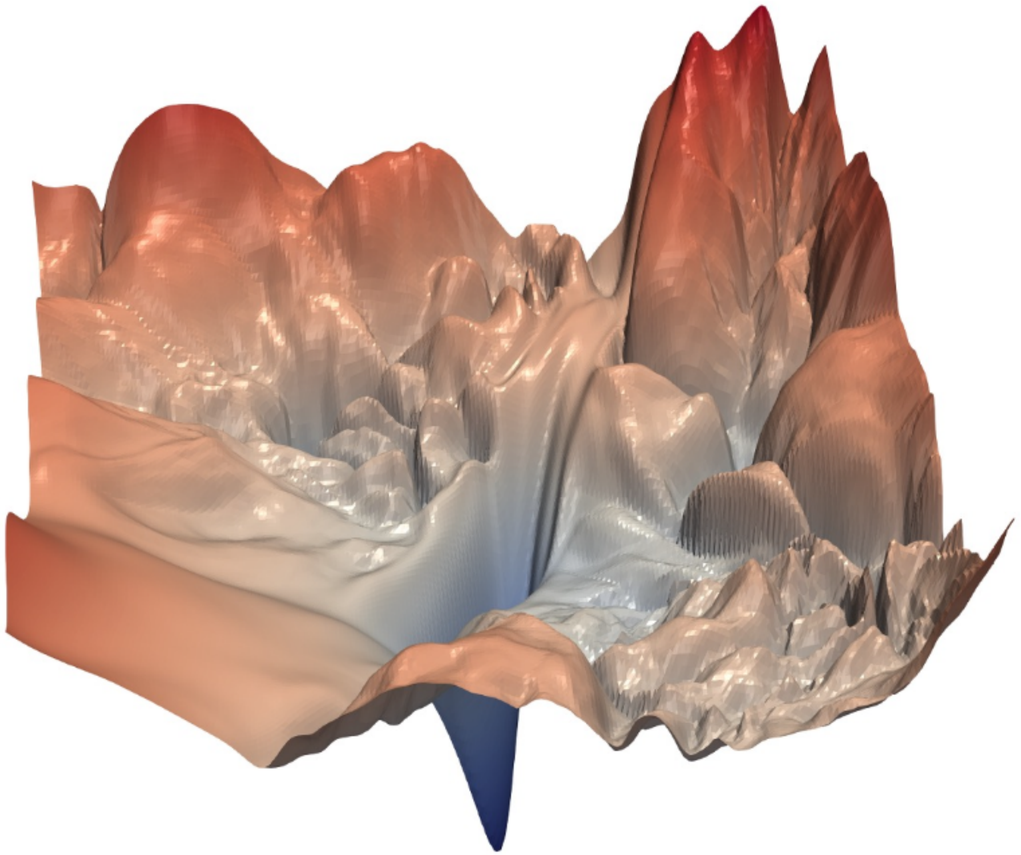
$$l(\alpha, \beta) = L(\theta^* + \alpha u + \beta v)$$

Where u and v are 2 randomly sampled directional vectors in the parameter space

Loss surface for ResNet-56 without skip connections trained on CIFAR-10 (natural images)

⚠ Loss function is very high dimensional and contains many **local minima**

Loss landscape



Li et al, Visualizing the Loss Landscape of Neural Nets, NeurIPS (2018)

2D visualization of the loss function

$$l(\alpha, \beta) = L(\theta^* + \alpha u + \beta v)$$

Where u and v are 2 randomly sampled directional vectors in the parameter space

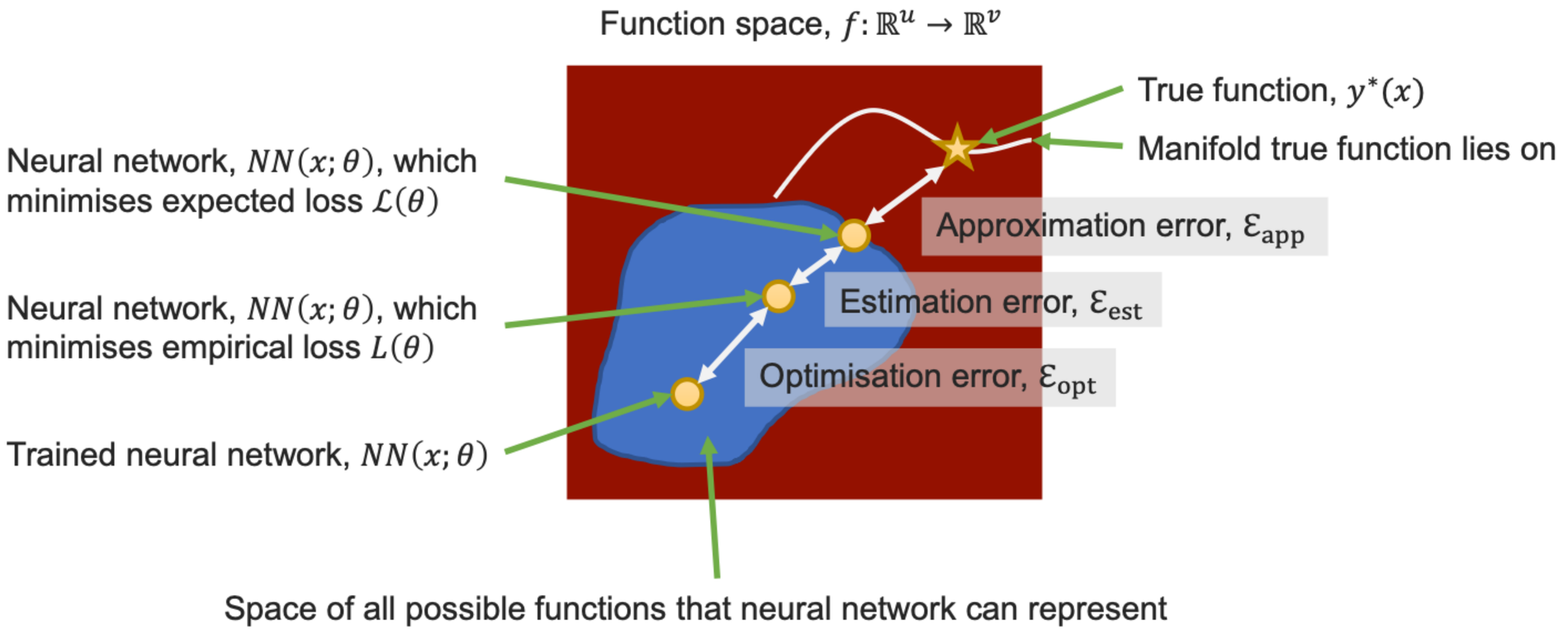
Loss surface for ResNet-56 without skip connections trained on CIFAR-10 (natural images)

⚠ Loss function is very high dimensional and contains many **local minima**

Better initialization
- Vanishing gradients
- Batch normalization
- Resnets

Stochastic optimizers
- SGD
- Adam

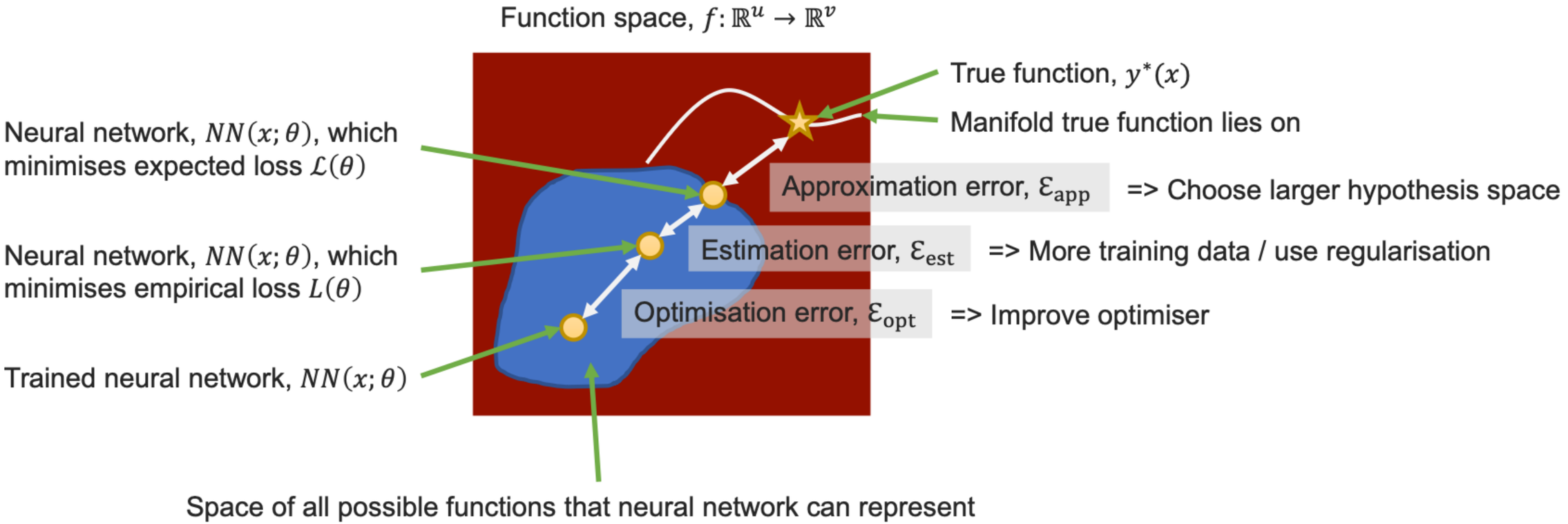
All error sources



$$\mathcal{E} = \mathcal{E}_{app} + \mathcal{E}_{est} + \mathcal{E}_{opt}$$

Slide credit: AI in the Sciences and Engineering, ETH Zürich (F24)

All error sources

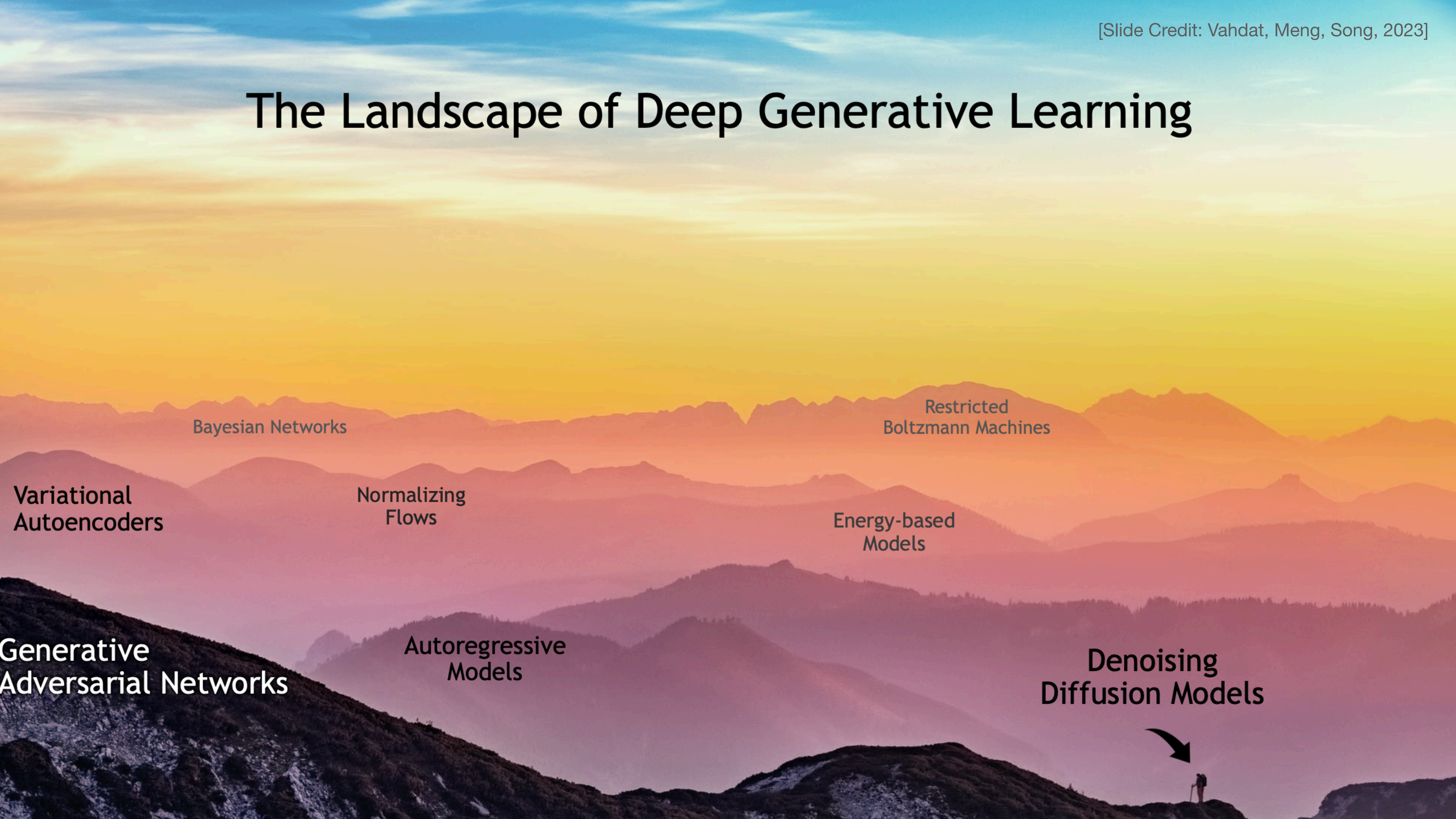


$$\varepsilon = \varepsilon_{app} + \varepsilon_{est} + \varepsilon_{opt}$$

Slide credit: AI in the Sciences and Engineering, ETH Zürich (F24)

[Slide Credit: Vahdat, Meng, Song, 2023]

The Landscape of Deep Generative Learning



What is a generative model?

1. An algorithm that generates data

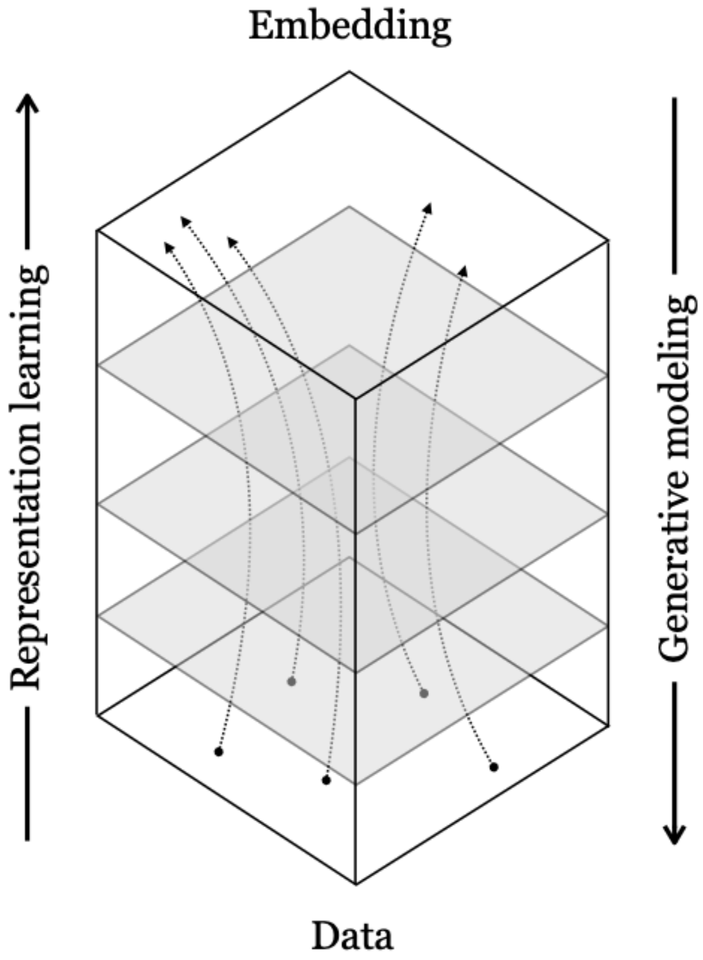
2. A statistical model of the joint distribution of some data, $p(x, y, \ldots)$

Slide credit: Phillip Isola, MIT 6.7960 (F24)

Generative modeling vs. representation learning

Representation learning:
mapping data to abstract representations (analysis)

Generative modeling:
mapping abstract representations to data (*synthesis*)

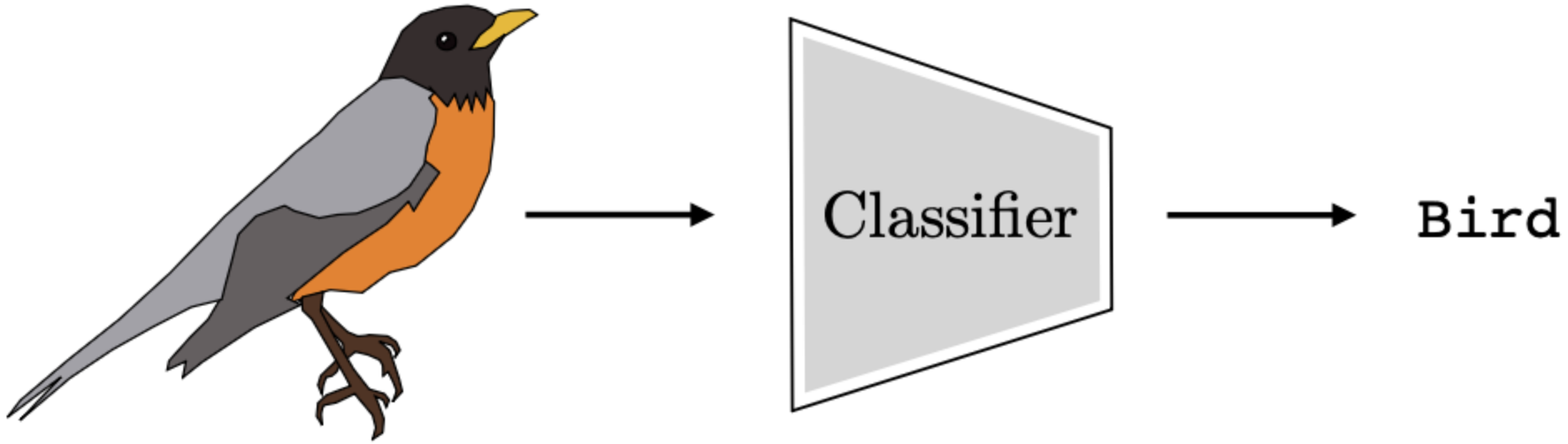


Slide credit: Phillip Isola, MIT 6.7960 (F24)

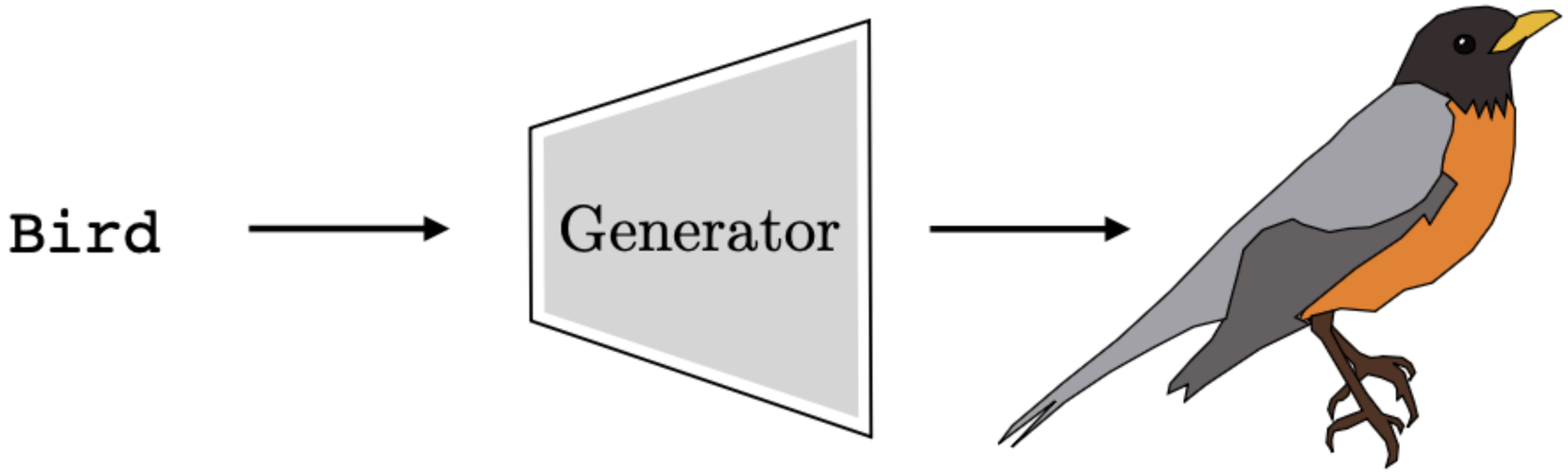
Math background

- So far we have mostly thought of neural nets as mappings $f_\theta : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ is some space of possible outputs.
  - e.g., image classifier into d classes: $f_\theta : \mathbb{R}^{N \times M \times C} \to \{1,...,d\}$
- Now, we will instead consider neural nets as mappings $f_\theta : \mathcal{X} \to \mathcal{P}(\mathcal{Y})$, where $\mathcal{P}(\mathcal{Y})$ is the space of probability distributions over $\mathcal{Y}$.
  - e.g., softmax regression to model $P(\text{class} \mid X = \mathbf{x})$: $f_\theta : \mathbb{R}^{N \times M \times C} \to \Delta^{d-1}$
  - *Main perspective: the outputs of our neural nets are, implicitly or explicitly, distributions*

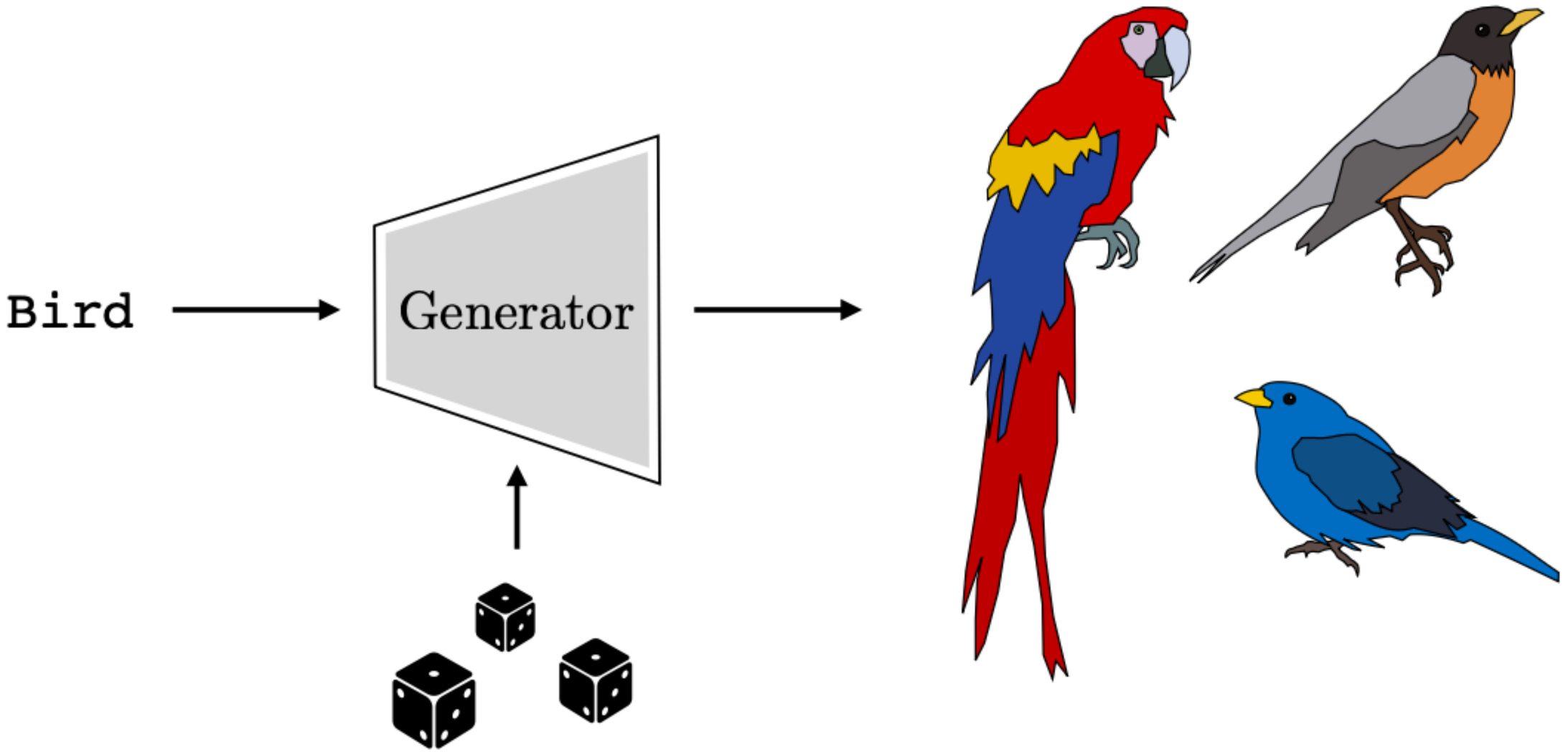
Slide credit: Phillip Isola, MIT 6.7960 (F24)



Slide credit: Phillip Isola, MIT 6.7960 (F24)



Slide credit: Phillip Isola, MIT 6.7960 (F24)



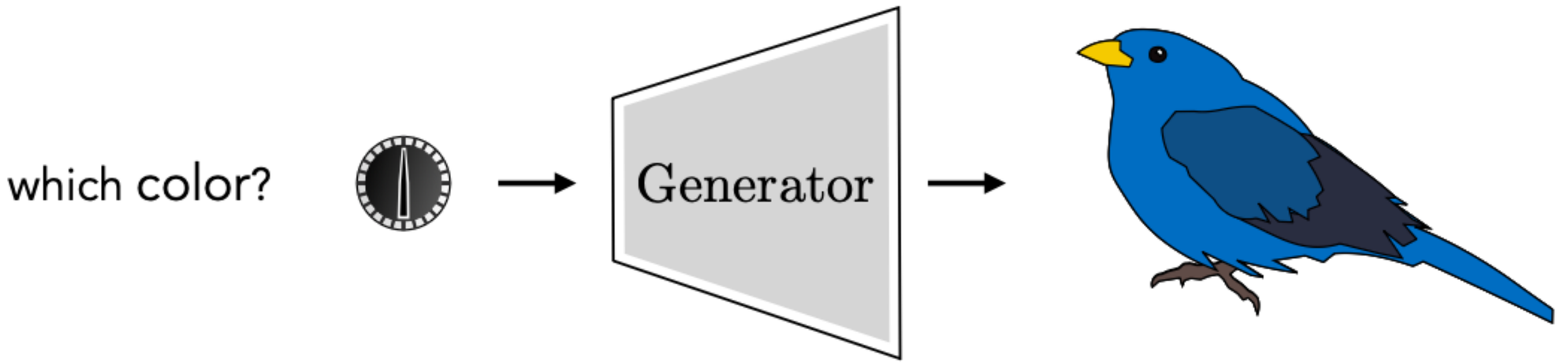
Slide credit: Phillip Isola, MIT 6.7960 (F24)

which color?

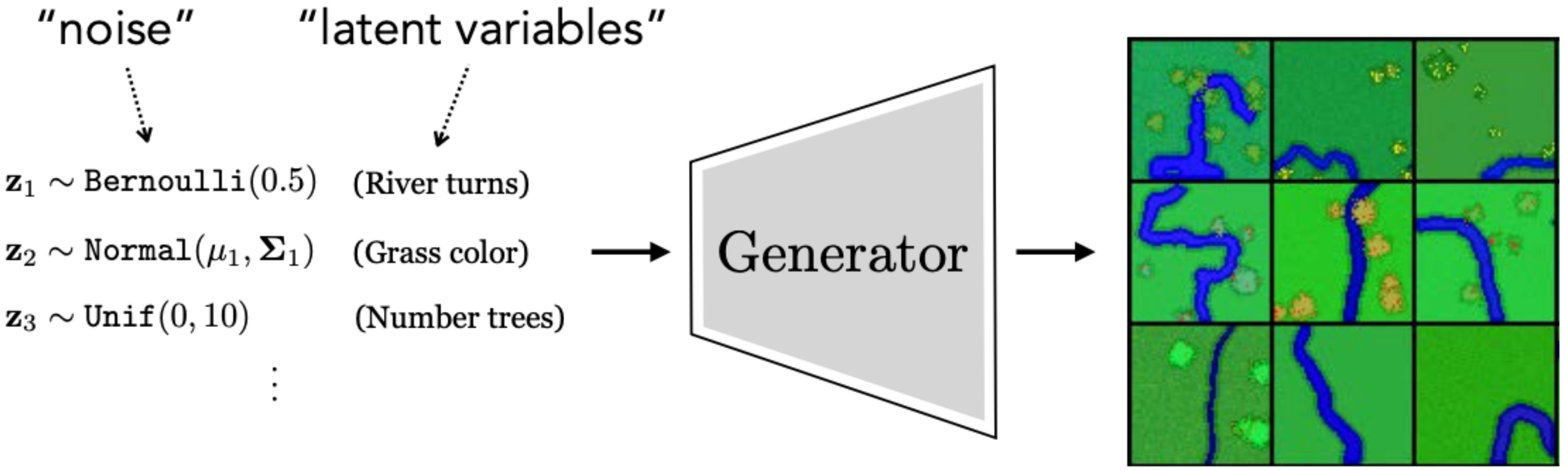
what angle?

what size?

Slide credit: Phillip Isola, MIT 6.7960 (F24)



Slide credit: Phillip Isola, MIT 6.7960 (F24)



Concept #1: **noise is latent variables**

Slide credit: Phillip Isola, MIT 6.7960 (F24)

Learning data generators

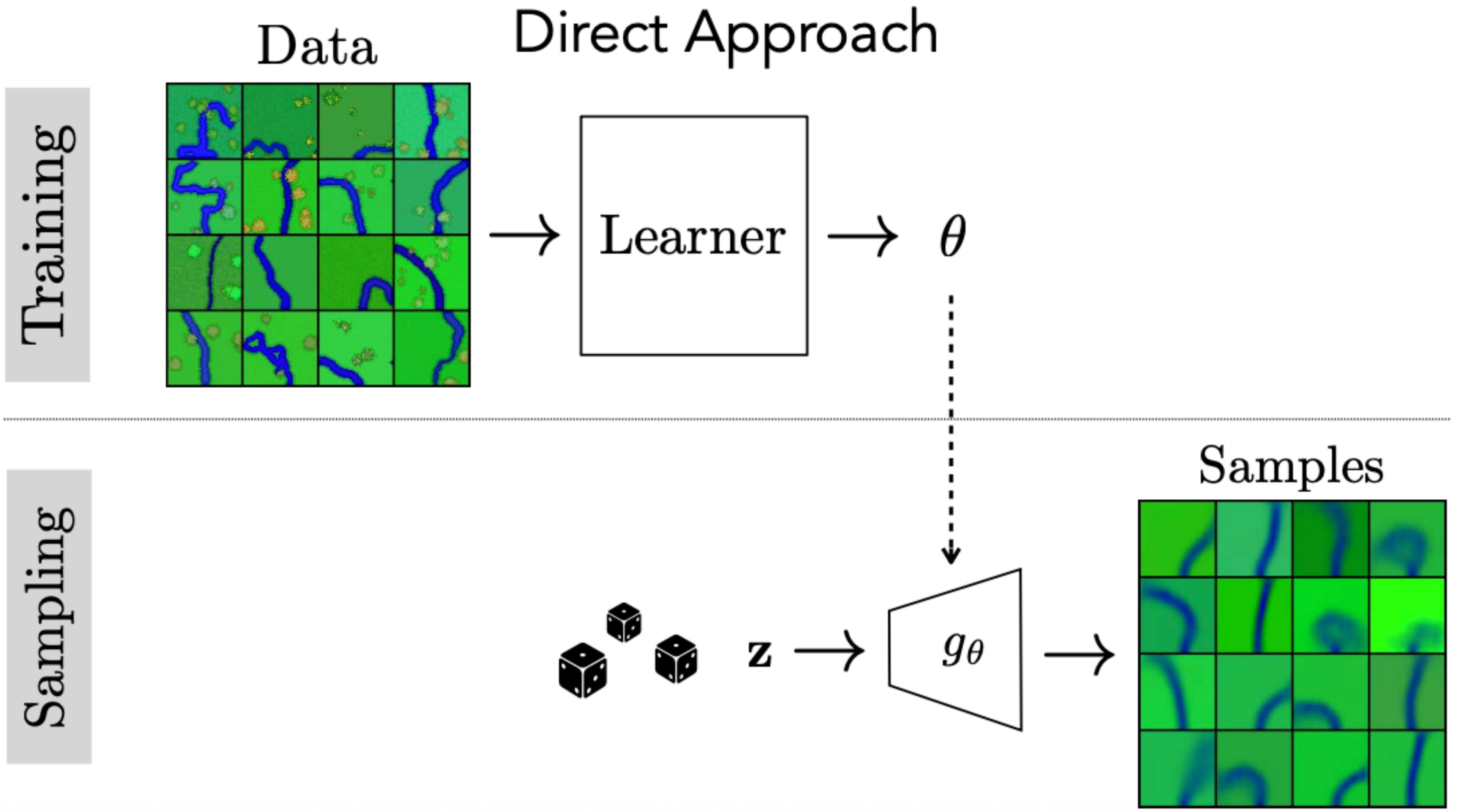
Two approaches:

confusingly, sometimes called an "implicit generative model"

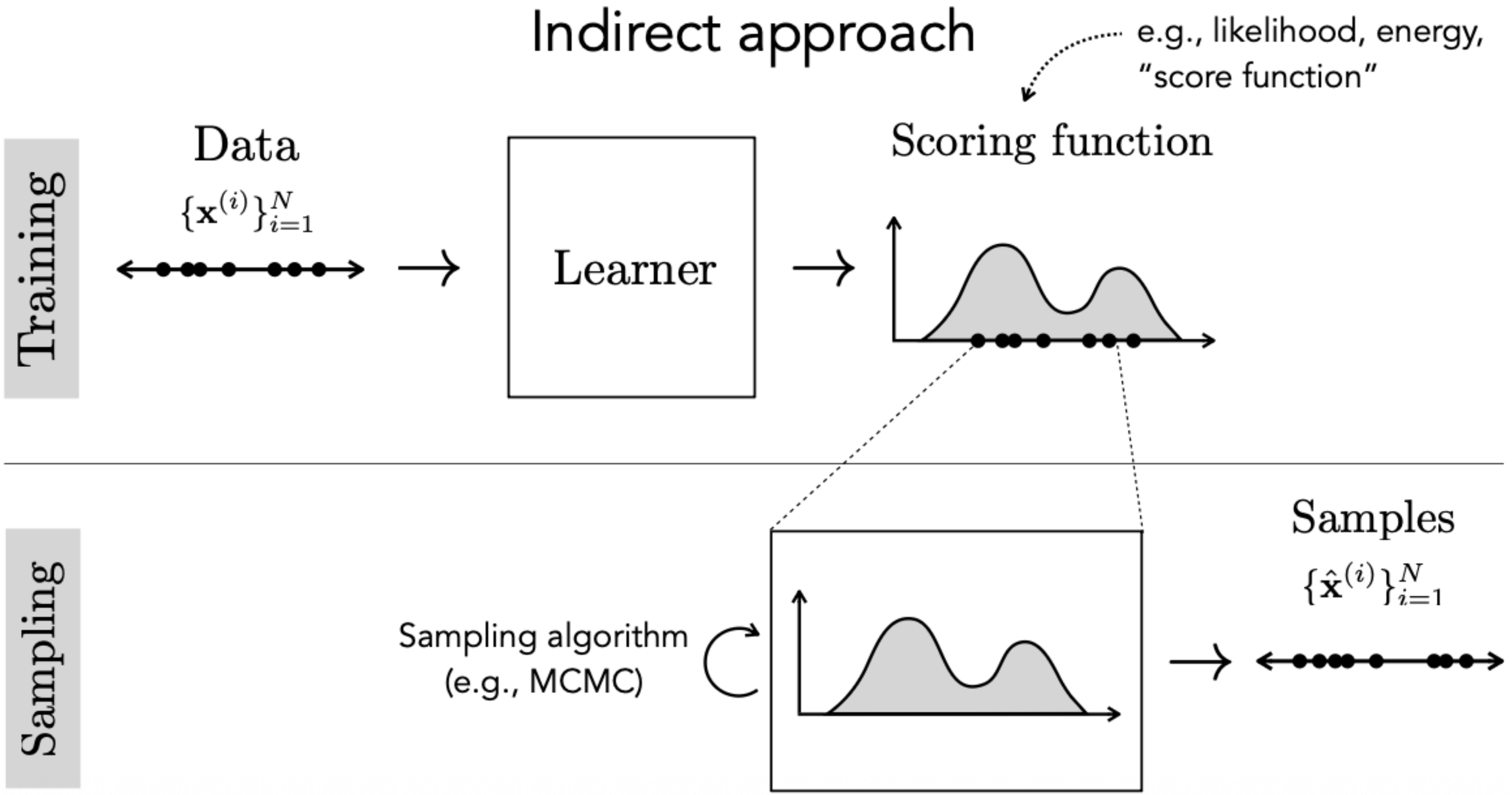
1. **Direct approach**: learn a function that generates data directly $G : \mathcal{Z} \to \mathcal{X}$

2. **Indirect approach**: learn a function that scores data; generate data by finding points that score highly under this function $E : \mathcal{X} \to \mathbb{R}$

Slide credit: Phillip Isola, MIT 6.7960 (F24)



Slide credit: Phillip Isola, MIT 6.7960 (F24)



Slide credit: Phillip Isola, MIT 6.7960 (F24)

What's the goal of generative modeling?

Make synthetic data that "looks like" real data.

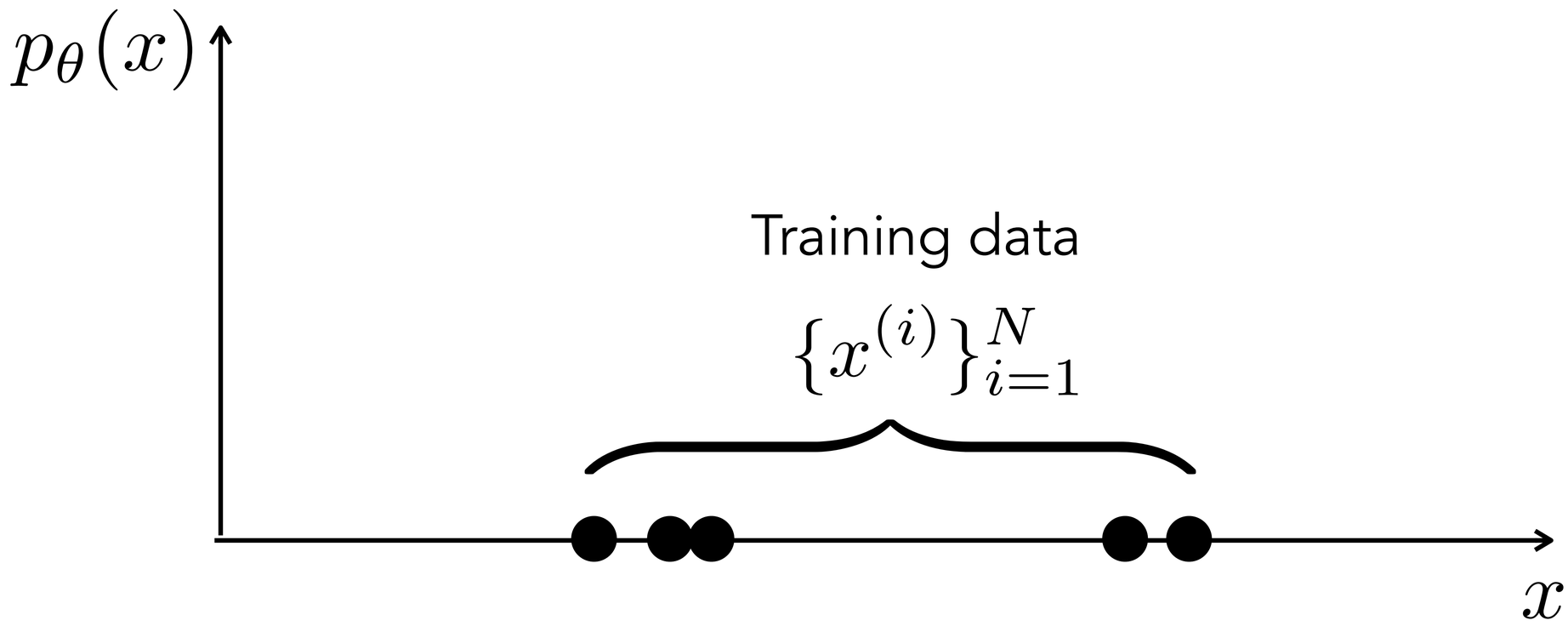
How to measure "looks like"?

The main answer in deep generative models is: "has high probability under a density model fit to real data."

Slide credit: Phillip Isola, MIT 6.7960 (F24)

Density models

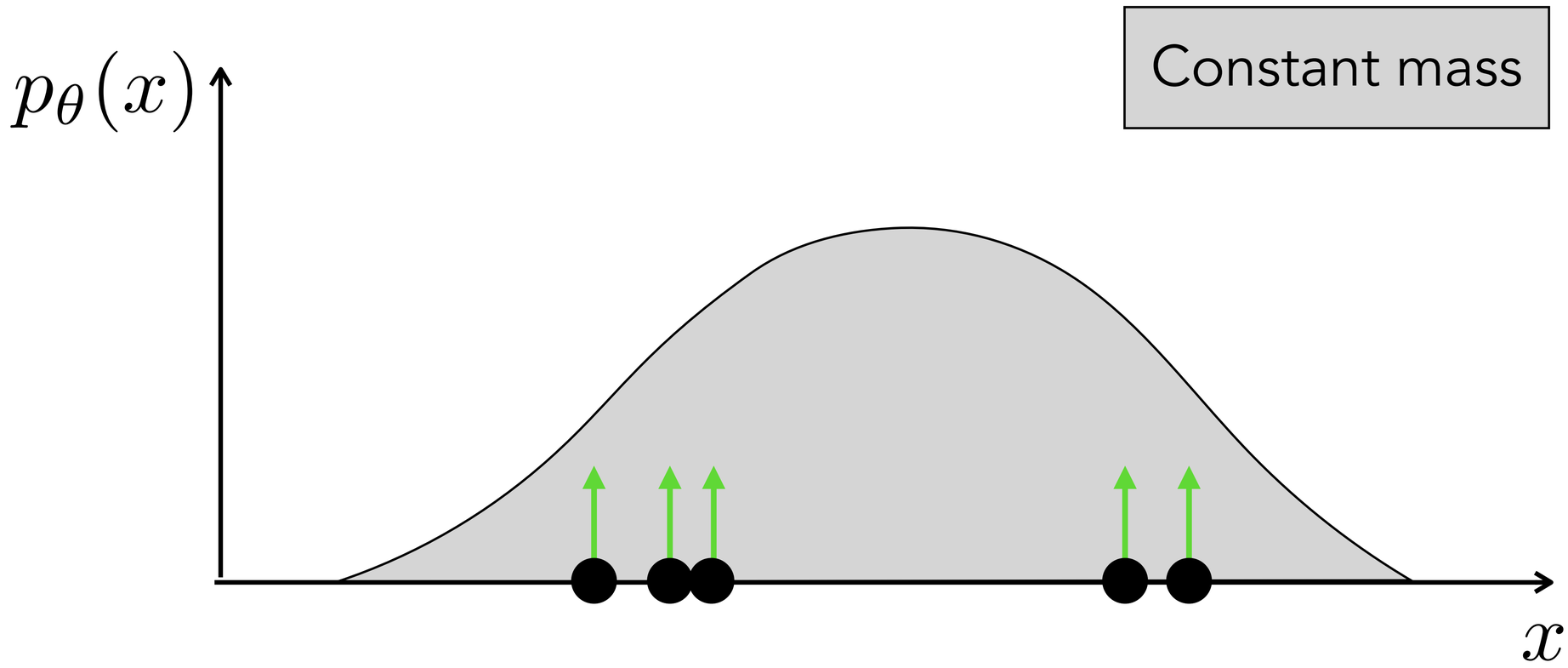
$$p_\theta : \mathcal{X} \to [0, \infty)$$



Slide credit: Phillip Isola, MIT 6.7960 (F24)

Density models

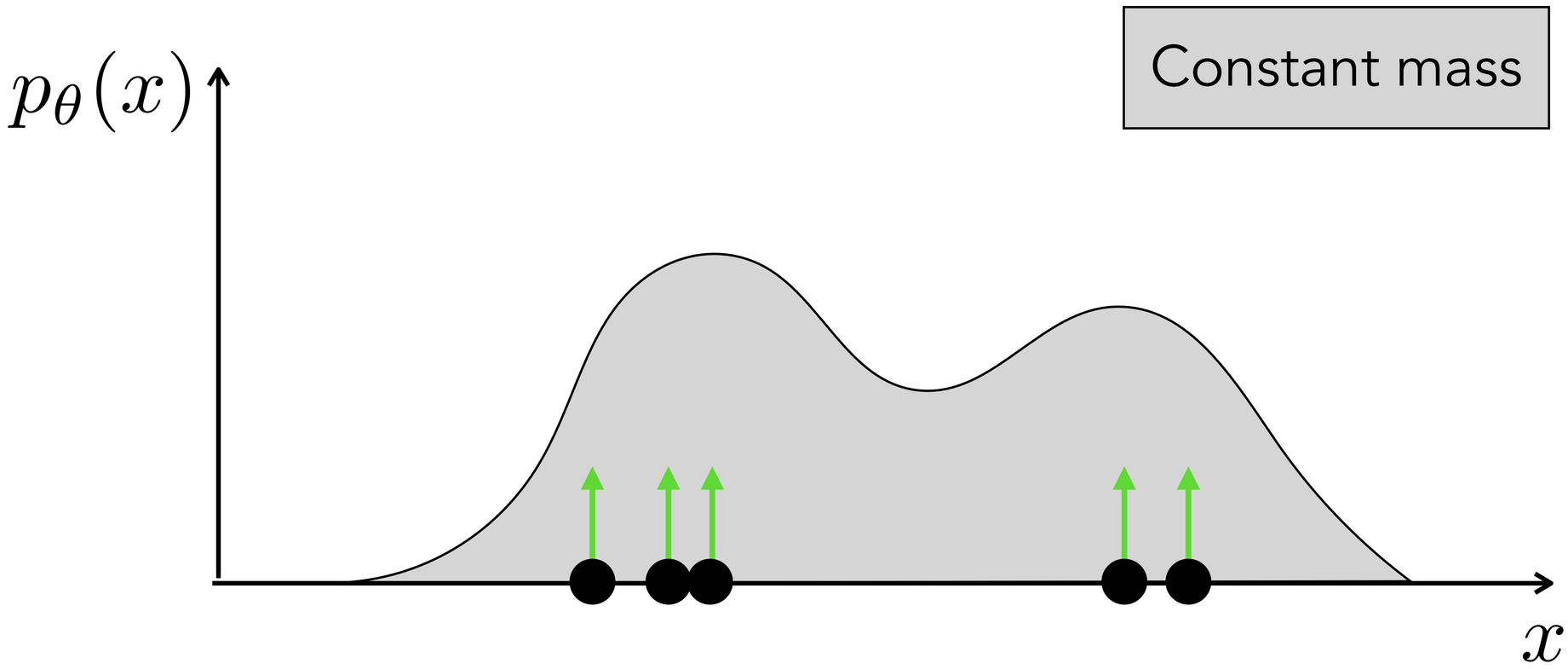
$$p_\theta : \mathcal{X} \to [0, \infty)$$



Slide credit: Phillip Isola, MIT 6.7960 (F24)

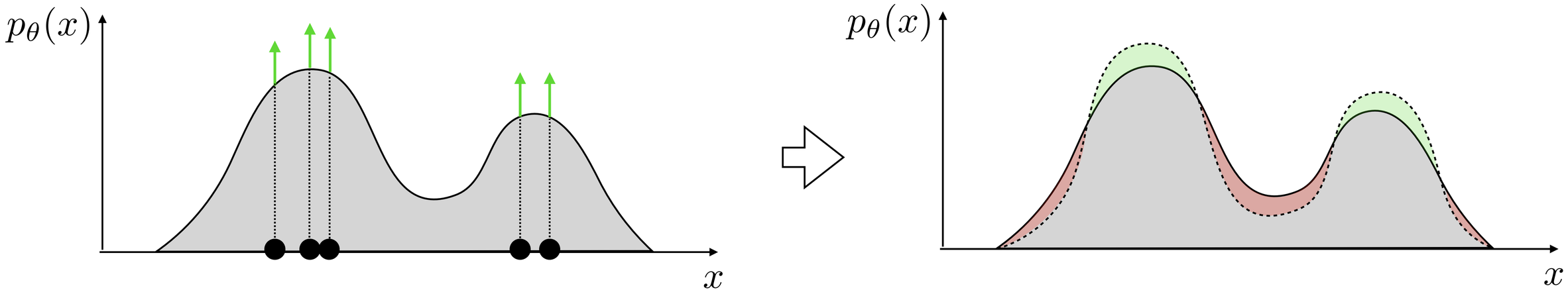
Density models

$$p_\theta : \mathcal{X} \to [0, \infty)$$



Slide credit: Phillip Isola, MIT 6.7960 (F24)

Density models



$$p_\theta^* = \arg\min_{p_\theta} \mathrm{KL}(p_{\texttt{data}}, p_\theta)$$

$$= \arg\min_{p_\theta} \mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}\left[-\log \frac{p_\theta(\mathbf{x})}{p_{\texttt{data}}(\mathbf{x})}\right]$$

$$= \arg\max_{p_\theta} \mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}\left[\log p_\theta(\mathbf{x})\right] - \mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}\left[\log p_{\texttt{data}}(\mathbf{x})\right]$$

$$= \arg\max_{p_\theta} \mathbb{E}_{\mathbf{x}\sim p_{\texttt{data}}}\left[\log p_\theta(\mathbf{x})\right] \quad \triangleleft \quad \text{dropped second term since no dependence on } p_\theta$$

$$\approx \arg\max_{p_\theta} \frac{1}{N}\sum_{i=1}^{N} \log p_\theta(\mathbf{x}^{(i)})$$

max likelihood

Slide credit: Phillip Isola, MIT 6.7960 (F24)

What's the goal of generative modeling?

The goal is not to replicate the training data but to make ***new*** data that is ***realistic*** (captures the essential properties of real data)

One way to quantify this is: likelihood of the ***test data*** under the model. (A model that memorizes the training data is overfit in exactly the same sense as a classifier can be overfit.)

$$\{x^{(i)}_{\texttt{test}}\}_{i=1}^{N}, \quad x^{(i)}_{\texttt{test}} \sim p_{\texttt{data}}$$

$$\text{generalization error} = \sum_i \log p_\theta(x^{(i)}_{\texttt{test}})$$

Slide credit: Phillip Isola, MIT 6.7960 (F24)

Comparing popular model types

Method	Latents?	Density/Energy?	Generator?
Energy-based models	✗	✓(energy only)	✗
Gaussian	✗	✓	✗
Autoregressive models	✗	✓	✓(slow)
Diffusion models	✓(high-dimensional)	✗	✓(slow)
GANs	✓	✗	✓
VAEs	✓	✗	✓

* rough categorization of vanilla versions of each model

Slide credit: Phillip Isola, MIT 6.7960 (F24)

Why Generative Modeling for Science?

- Let’s discuss!