

*Kavli-Villum Summer School on Gravitational Waves*

# Machine Learning for Gravitational Waves

*Corfu – September 29, 2023*

Stephen Green



---

# Outline

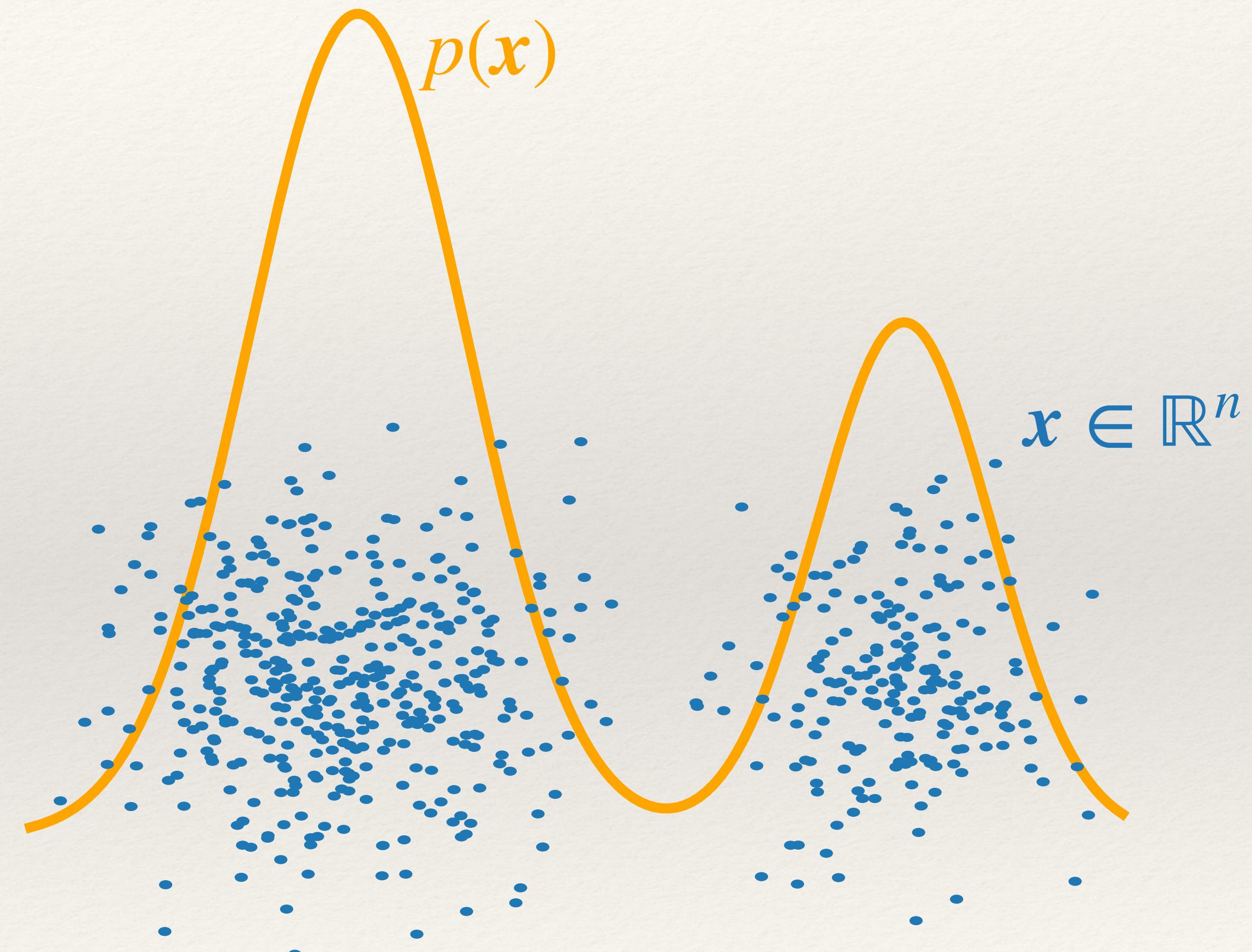
---

- ❖ Intro to machine learning and deep learning
- ❖ Density estimation with normalising flows
- ❖ Gravitational wave inference

# Introduction to machine learning

Machine learning uses computers to learn probability distributions from data

- ❖ Given the learned distribution,
  1. Draw new samples similar to existing,  
 $x \sim p(x)$
  2. Evaluate the density at new sample points,  
 $p(x)$
- ❖ Also works for conditional distributions  $p(y | x)$



# Introduction to machine learning

- ❖ Modern machine learning techniques (such as deep learning) allow for high-dimensional data and extremely complicated distributions.
- ❖ Examples: Large language models   $p(\text{next word} \mid \text{previous words})$ 
  - ❖ Sample repeatedly to generate long, coherent texts



What is the most interesting LISA source, supermassive black hole binaries or extreme mass ratio inspirals?



Extreme Mass Ratio Inspirals (EMRIs) are generally considered more interesting as they allow for precise testing of General Relativity and offer detailed information about the fields around supermassive black holes, potentially leading to groundbreaking discoveries in physics and astrophysics.

# Machine learning

- ❖ Image generation

$$p(\text{image} \mid \text{description})$$

*“Physics students doing calculations on a chalkboard outside in Corfu, with a beach in the background and a gravitational wave on the board.”*



# Gravitational wave examples

❖ **Classification:** Learn a distribution over a discrete space

$$p(y | \boldsymbol{x}) \quad \boldsymbol{x} \in \mathbb{R}^n, y \in \{1, \dots, k\}$$

- Detection: Is there a signal? Yes or no?
- Glitch classification: Assign observed glitches to classes.

❖ **Regression:** Learn a distribution over continuous variables

$$p(y | \boldsymbol{x}) \quad \boldsymbol{x} \in \mathbb{R}^n, y \in \mathbb{R}^m$$

- Waveform modeling: Predict a signal given the source parameters
- Parameter estimation: Predict the source parameters given the data
- **Unsupervised learning:** Anomaly detection — is data  $\boldsymbol{x}$  consistent with other examples?

supervised learning  
labeled training set

# Machine learning recipe

---

1. Build a dataset of training examples.

- ▶  $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$  pairs

2. Define a parametrized probabilistic model for the data.

- ▶  $p_{\text{model}}(\mathbf{y} | \mathbf{x}; \boldsymbol{\phi})$

3. Choose a measure of performance for the model on the data.

- ▶ “loss function” — typically maximum likelihood

4. Fit the model to the data according to the performance metric.

- ▶ Optimize  $\boldsymbol{\phi}$ .

# Maximum likelihood estimation

Loss function is determined by **maximising the likelihood of the training data under the model.**

❖ **Unsupervised learning.** Assume we have

1.  $N$  independent samples  $\mathbf{x}^{(i)} \sim p_{\text{data}}(\mathbf{x})$
2. Parametrized model  $p_{\text{model}}(\mathbf{x}; \boldsymbol{\phi})$

$$\begin{aligned}\boldsymbol{\phi}_{\text{ML}} &= \arg \max_{\boldsymbol{\phi}} p_{\text{model}}(\mathbf{X}; \boldsymbol{\phi}) \\ &= \arg \max_{\boldsymbol{\phi}} \prod_{i=1}^N p_{\text{model}}(\mathbf{x}^{(i)}; \boldsymbol{\phi}) \\ &= \arg \max_{\boldsymbol{\phi}} \sum_{i=1}^N \log p_{\text{model}}(\mathbf{x}^{(i)}; \boldsymbol{\phi}) \\ &= \arg \max_{\boldsymbol{\phi}} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \log p_{\text{model}}(\mathbf{x}; \boldsymbol{\phi})\end{aligned}$$

negative log probability loss

# Conditional distributions

---

- ❖ **Supervised learning:** Estimate a conditional probability  $p_{\text{model}}(\mathbf{y} \mid \mathbf{x}; \boldsymbol{\phi})$
- ❖ Generalise the maximum likelihood estimator:

$$\begin{aligned}\boldsymbol{\phi}_{\text{ML}} &= \arg \max_{\boldsymbol{\phi}} \sum_{i=1}^N \log p_{\text{model}}(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}; \boldsymbol{\phi}) \\ &= \arg \max_{\boldsymbol{\phi}} \mathbb{E}_{p_{\text{data}}(\mathbf{x}, \mathbf{y})} \log p_{\text{model}}(\mathbf{y} \mid \mathbf{x}; \boldsymbol{\phi})\end{aligned}$$

# Example: Linear regression

- ❖ Suppose we have labelled data  $(\mathbf{x}^{(i)}, y^{(i)})$ .
- ❖ Let  $p_{\text{model}}(y | \mathbf{x}; \boldsymbol{\phi}) = \mathcal{N}(\mu(\mathbf{x}), \sigma^2)(y)$  where  $\mu(\mathbf{x}) = \boldsymbol{\phi} \cdot \mathbf{x}$ ;  $\sigma$  fixed.
- ❖ Loss function follows automatically,

$$\begin{aligned} J(\boldsymbol{\phi}) &= -\sum_{i=1}^N \log p(y^{(i)} | \mathbf{x}^{(i)}; \boldsymbol{\phi}) \\ &= \frac{N}{2} \log 2\pi\sigma^2 + \sum_{i=1}^N \frac{(y^{(i)} - \mu(\mathbf{x}^{(i)}))^2}{2\sigma^2} \end{aligned}$$

$\propto$  mean squared error

- ❖ Can solve exactly  $\nabla_{\boldsymbol{\phi}} J = 0 \implies \boldsymbol{\phi}_{\text{ML}} = (X^\top X)^{-1} X^\top \mathbf{y}$

# More general regression

- ❖ More generally  $\mu(\mathbf{x})$  does not have to be linear. We can increase the **representational capacity** of the model by using more complicated functions.

- E.g., polynomial  $\mu(\mathbf{x}) = b + \sum_{i=1}^k w_i x^i$  (can still solve in closed form)
- E.g. nonparametric regression

nearest neighbour: For any  $\mathbf{x}$ , find the nearest  $\mathbf{x}^{(i)}$  in the training set and return  $y^{(i)}$ .

- E.g., neural network
- ❖ Not all models can be optimised in closed form.

# Stochastic gradient descent

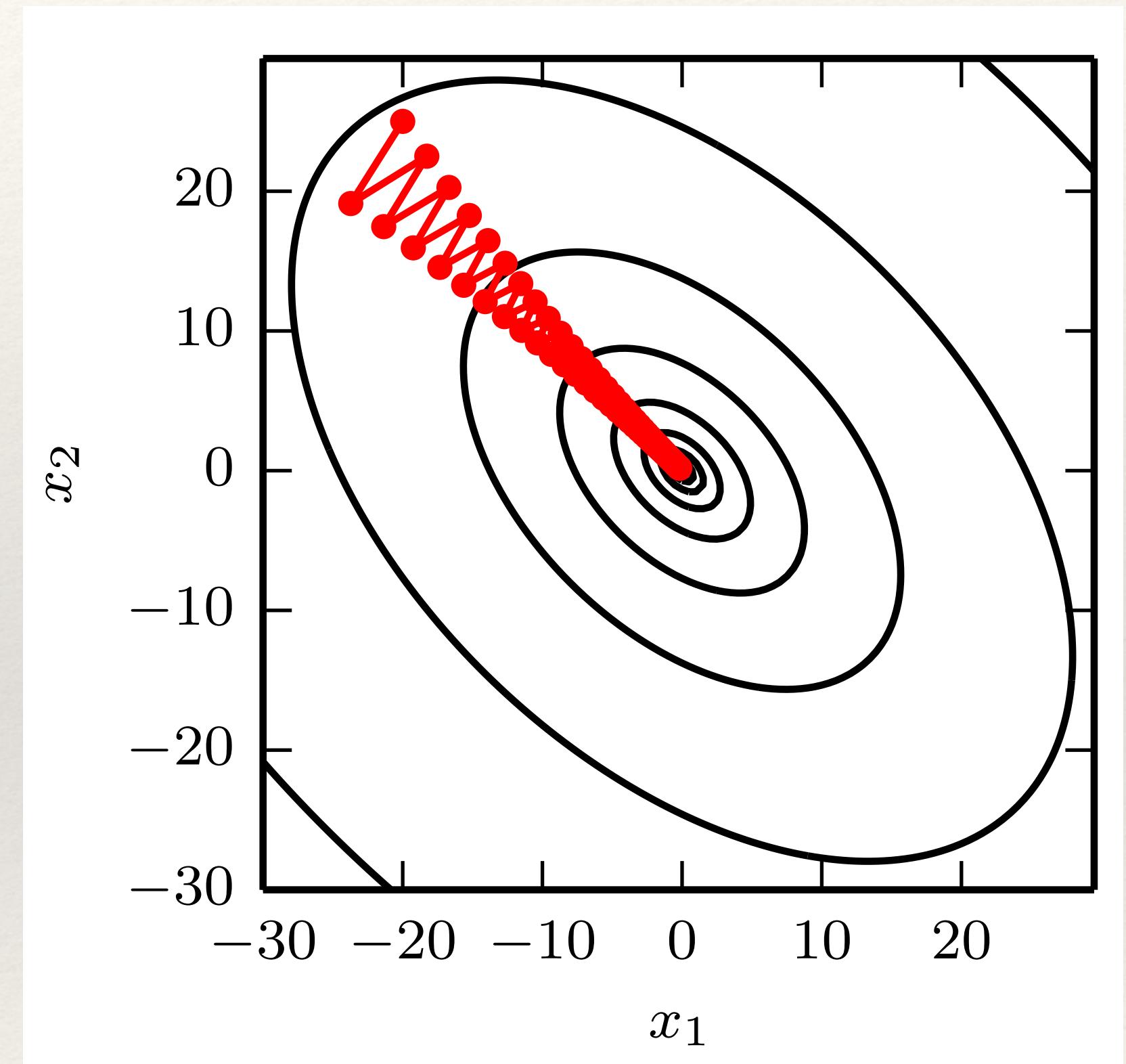
- ❖ In the case where a closed-form minimum is not available, **gradient descent** can be used to **optimize** the loss, i.e., to tune  $\phi$  to approach the minimum.

- ❖ Starting from a point  $\phi_0$  we can move to a new point by following the gradient

$$\phi_1 = \phi_0 - \epsilon \nabla_{\phi} J|_{\phi_0}$$

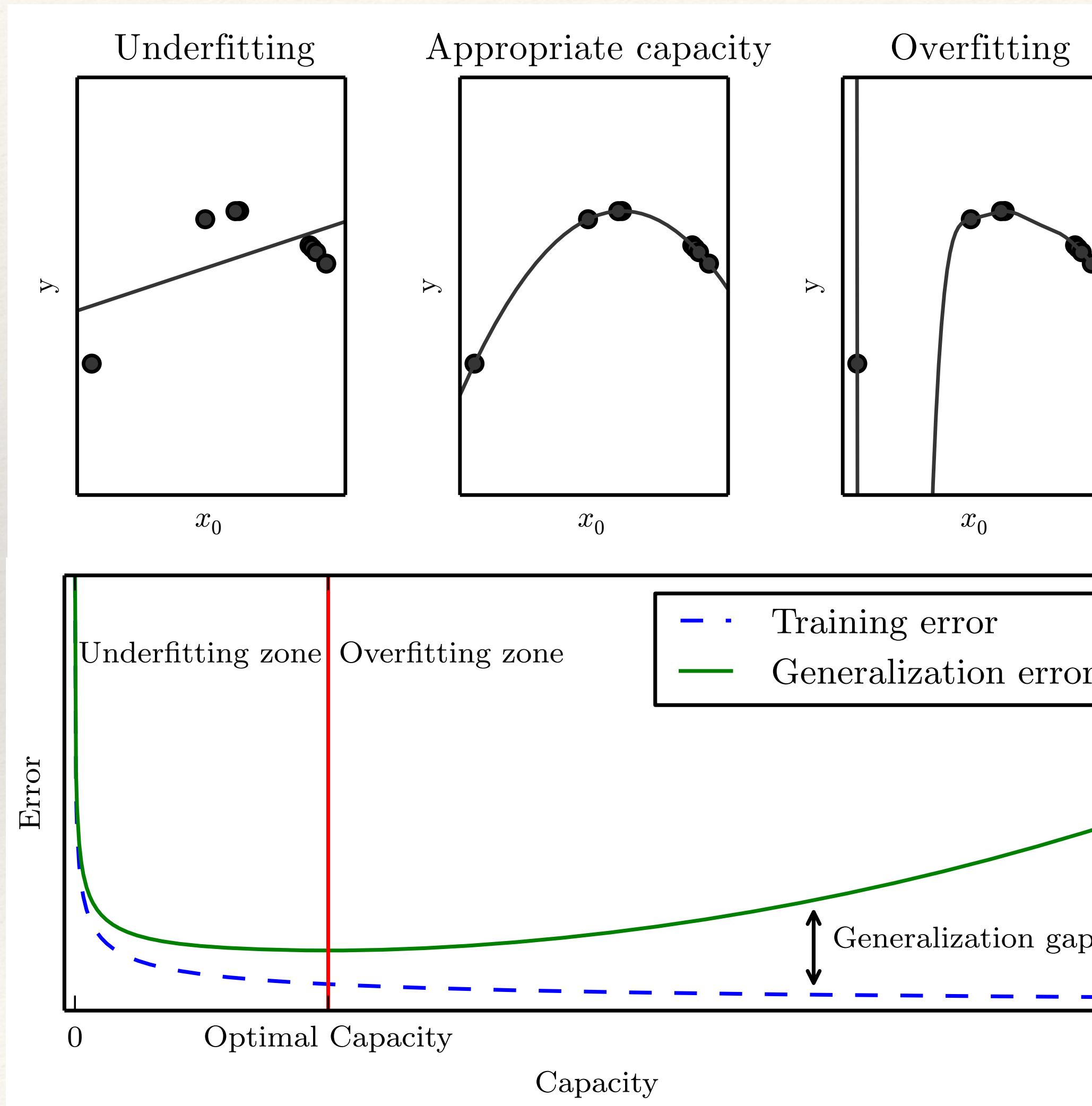
“Learning rate”

- ❖ For ML loss, gradient reduces to the sum of per-example gradients, so break into **minibatches** — **stochastic GD**.
- ❖ Two advantages: (1) faster to compute each update, and (2) introduces stochasticity, which helps avoid local minima.



Goodfellow et al (2016)

# Risk of overfitting



- ❖ High capacity models run the risk of overfitting. The algorithm must perform well not just on data used for training, but must also **generalize** to new data.
- ❖ Capacity should be chosen to minimize **generalization error**.
- ❖ A **larger training set** will allow for better generalization.

Goodfellow et al (2016)

# Summary so far...

---

- ❖ A machine learning algorithm requires the following:
  1. **dataset** — for supervised learning  $\{\mathbf{x}^{(i)}, y^{(i)}\}$  pairs
  2. **model** — E.g., linear regression  $p_{\text{model}}(y | \mathbf{x}) = \mathcal{N}(\boldsymbol{\phi}^\top \mathbf{x}, 1)(y)$
  3. **loss function** — E.g.,  $J(\boldsymbol{\phi}) = -\mathbb{E}_{p_{\text{data}}(\mathbf{x})} \log p_{\text{model}}(\mathbf{x})$
  4. **optimization algorithm** — E.g., stochastic gradient descent

# Deep learning

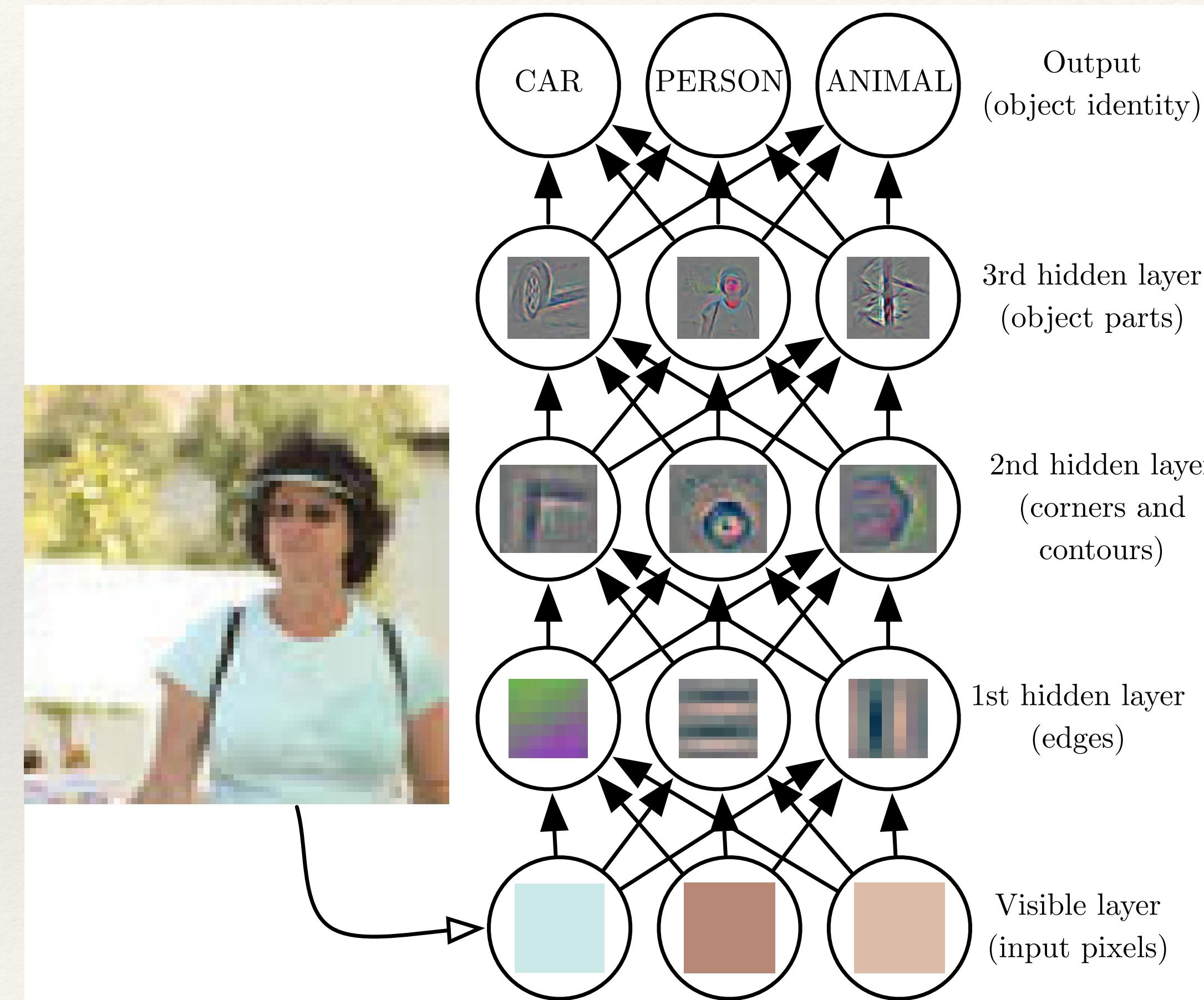
---

# Introduction to deep learning

---

- ❖ In many cases, to use a machine learning algorithm, one must first reduce the raw data to a small number of high-level features  $x$ , which are provided as input to the algorithm.
- ❖ This **representation** is often specified by hand, but it can also be learned from lower-level features, or raw data.
- ❖ Deep learning seeks to learn higher level representations in terms of lower level ones by composing functions.

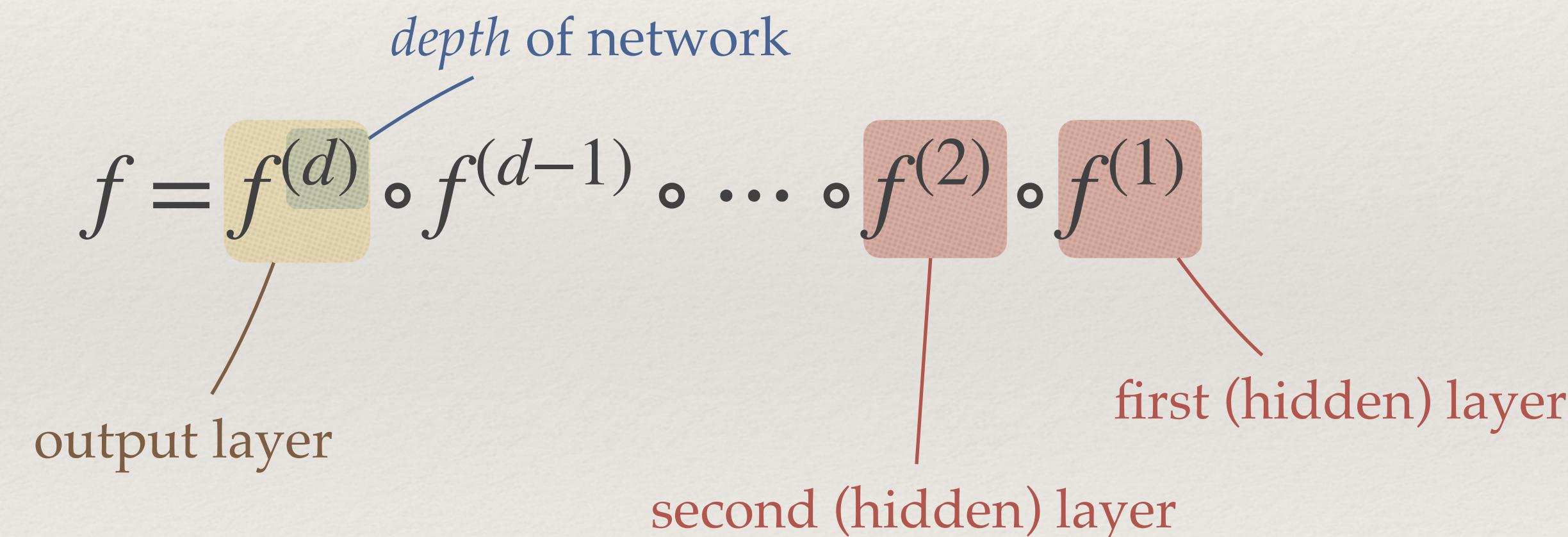
# Introduction to deep learning



Goodfellow *et al* (2016)

# Feedforward neural networks

- ❖ Feedforward neural networks or multilayer perceptrons (MLPs) are the classic deep learning model.
- ❖ Defines a mapping  $y = f(x; \phi)$  as a composition of simpler mappings:

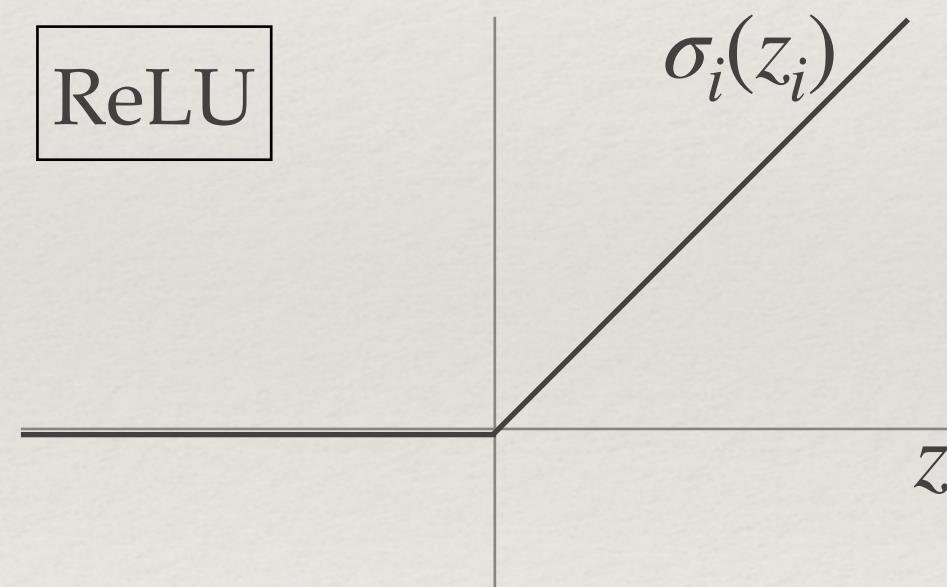


- ❖ “Feed-forward” because there is no feedback of later layers on earlier ones.

# Feedforward neural networks

$$f = f^{(d)} \circ f^{(d-1)} \circ \dots \circ f^{(2)} \circ f^{(1)}$$

- ❖ Each layer is of the form



$$f^{(j)}(\mathbf{h}) = \sigma_j \left( \mathbf{W}_j^\top \mathbf{h} + \mathbf{b}_j \right)$$

activation function  
• often nonlinear

weight matrix  
bias vector

linear mapping  $\mathbf{z}_j = \mathbf{W}_j^\top \mathbf{h} + \mathbf{b}_j$

The diagram shows the mathematical expression for a layer's output. The term  $\sigma_j$  is highlighted in red and connected by a red arrow to the text "activation function". The terms  $\mathbf{W}_j^\top \mathbf{h}$  and  $\mathbf{b}_j$  are highlighted in blue and connected by a blue arrow to the text "linear mapping". Above the equation, "weight matrix" points to  $\mathbf{W}_j^\top$  and "bias vector" points to  $\mathbf{b}_j$ .

- ❖ Weights and biases are the parameters defining the model  $\boldsymbol{\phi} \equiv \{\mathbf{W}_j, \mathbf{b}_j\}_{j=1}^d$ . These are tuned during training.

# Feedforward neural networks

---

$$f = f^{(d)} \circ f^{(d-1)} \circ \dots \circ f^{(2)} \circ f^{(1)}$$

$$f^{(j)}(\mathbf{h}) = \sigma_j \left( \mathbf{W}_j^\top \mathbf{h} + \mathbf{b}_j \right)$$

- ❖ The MLP is therefore defined by
  - **depth** (number of layers)
  - **widths** (dimensions of hidden layers)
  - **choice of activation functions**
- ❖ Training uses stochastic gradient descent with gradients calculated using **back-propagation** (the chain rule).

# Output layer

- ❖ The activation function for the output layer is determined by the nature of the output and the distribution we are modeling.
- ❖ Example: For **regression**, typically take the output to be the **mean** of a Gaussian distribution

$$p_{\text{model}}(y | x) = \mathcal{N}(f(x), I)(y)$$

- Since the mean is unconstrained, use a **linear** output layer

$$\sigma_{\text{linear}}(z) = z$$

- Maximum likelihood gives the mean squared error loss (as before).

# Back-propagation

---

- ❖ To train the network using some form of gradient descent, it is necessary to be able to **efficiently compute gradients** with respect to all of the network parameters (weights and biases).
- ❖ This is accomplished using a form of automatic differentiation call **back-propagation**.
- ❖ Relies on **compositional nature** of neural networks

$$f = f^{(d)} \circ f^{(d-1)} \circ \dots \circ f^{(2)} \circ f^{(1)}$$

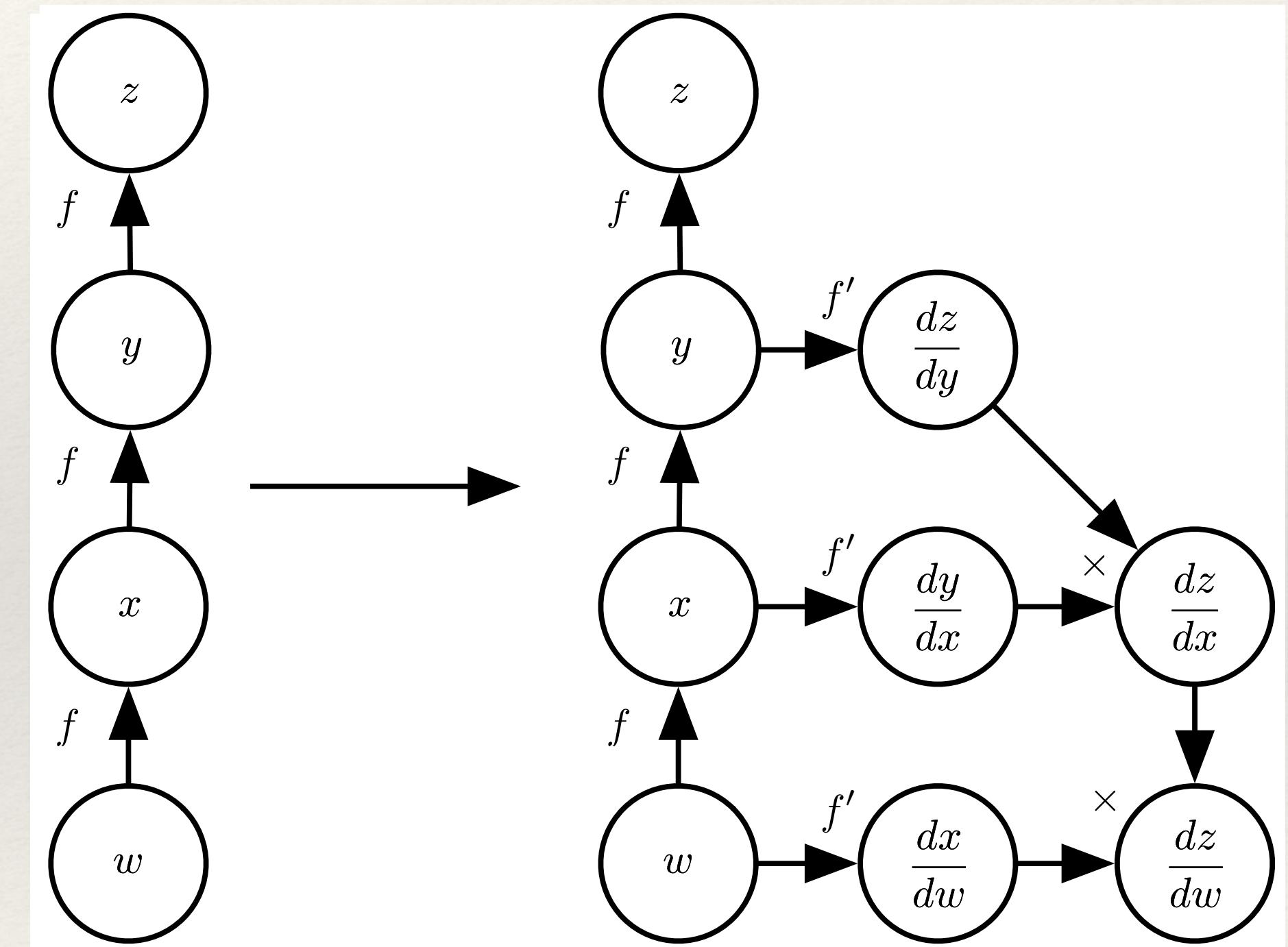
plus the **chain rule of calculus** and differentiability of all operations.

# Back-propagation

- ❖ It is important to organize the calculation in an **efficient** way, and not carry out the same calculation multiple times.

$$\begin{aligned}\frac{\partial z}{\partial w} &= \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w} \\ &= f'(y)f'(x)f'(z) \\ &= f'(f(f(w)))f'(f(w))f'(w)\end{aligned}$$

Efficient implementations in  
every deep-learning library  
(PyTorch, TensorFlow, JAX, ...)



Goodfellow et al (2016)

# Beyond point estimates

- ❖ Neural networks can parametrize **more complicated distributions**, including **uncertainty estimates** (as needed for GW parameter estimation).
- ❖ Example: Normal distribution where we also fit for the **covariance matrix**  $p_{\text{model}}(y | x) = \mathcal{N}(\mu(x), \Sigma(x))$ 
  - For diagonal covariance, better to use the precision matrix, and enforce positivity using, e.g., softplus activation.
- ❖ Example: **Mixture density networks**  $p_{\text{model}}(y | x) = \sum_k p_k(x) \mathcal{N}(\mu_k(x), \Sigma_k(x))$ 
  - ❖ Sum of normal distributions weighted by categorical (multinouilli) output
- ❖ Example: **Normalizing flow** can model much more complicated distributions.
- ❖ In all cases, maximum likelihood loss gives an appropriate loss function.

# Architectural choices

---

- ❖ We have described the most basic of neural network architectures, the fully-connected feed-forward network.
- ❖ **Possible modifications:**
  - Sparse connectivity (sparse weight matrices)
  - Shared weights (e.g., convolutional networks)
  - Connections between non-adjacent layers (residual networks)
  - Recursive or recurrent connections
- ❖ Usage will depend on characteristics of the data, amount of computer resources, symmetry properties, etc.

# Simulation-based inference for GWs

# GW parameter estimation

- ❖ Goal is to obtain the posterior distribution  $p(\theta | d)$ 
  - $\theta$  — source parameters (masses, spins, sky position and orientation)
  - $d$  — observed strain data
- ❖ For stationary Gaussian noise, the likelihood is tractable, so MCMC can be used to sample from the posterior.
- ❖ ML approaches enable:
  1. Speed improvements (amortized inference);
  2. Inference with real noise.

$$p(d | \theta) \propto \exp\left(-\frac{1}{2}(d - h(\theta))^T (d - h(\theta))_{S_n}\right)$$

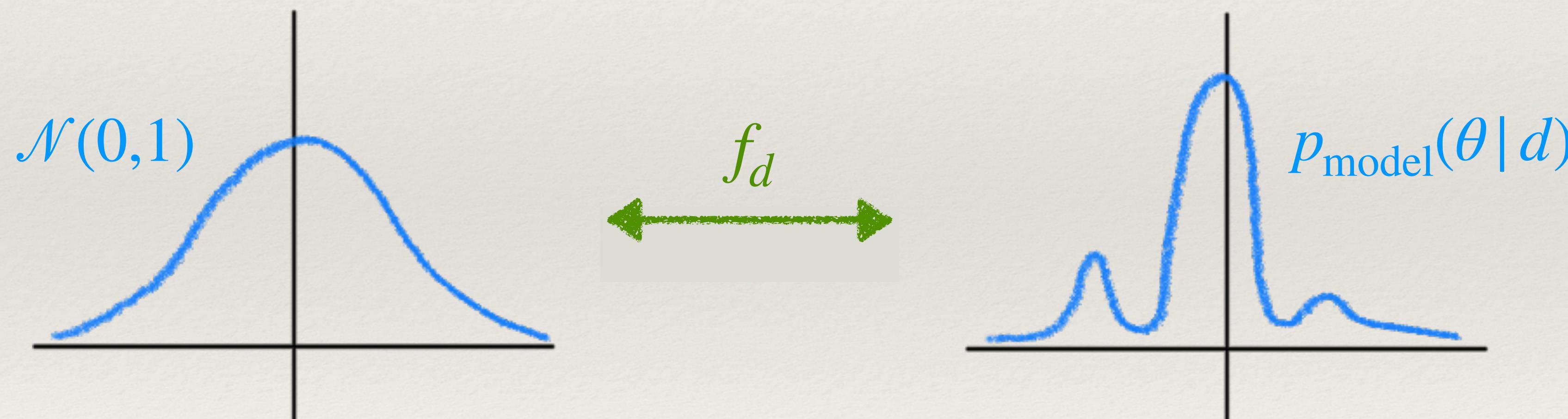
# GW parameter estimation

## ❖ Approaches

- ❖ Neural posterior estimation (NPE) — **learn  $p(\theta | d)$** 
  - ❖ Complicated distribution requires flexible density estimator, e.g., **normalising flows.**
- ❖ Neural likelihood estimation (NLE) — **learn  $p(d | \theta)$** 
  - ❖ Not typically used due to high dimensionality of  $d$ .
- ❖ Neural ratio estimation (NRE)
  - ❖ Train a **classifier** to **distinguish samples from  $p(\theta, d)$  and  $p(\theta)p(d)$** . This gives likelihood : evidence ratio.

# Normalizing flows

- ❖ For the GW posterior we require very flexible model distributions.
- ❖ A **normalizing flow** is an architecture well suited for this task. This represents a complex distribution in terms of a mapping from a much simpler one



$$p_{\text{model}}(\theta | d) = \mathcal{N}(0,1)^D(f_d^{-1}(\theta)) \left| \det J_{f_d}^{-1} \right|$$

# Normalizing flow

$$p_{\text{model}}(\theta | d) = \mathcal{N}(0, 1)^D(f_d^{-1}(\theta)) \left| \det J_{f_d}^{-1} \right|$$

- ❖ Requirements:
  1. Invertible
  2. Simple Jacobian determinant
- ❖ Parametrize  $f_d$  using neural network.

Fast to evaluate and sample from  $p_{\text{model}}(\theta | d)$

needed to evaluate loss

# Normalizing flow

- ❖ Requirements:

1. Invertible
2. Simple Jacobian determinant



- ❖ Use a sequence of “coupling transforms”:

$$c_{d,i}(u) = \begin{cases} u_i & \text{if } i \leq D/2 \\ c_i(u_i; u_{1:\frac{D}{2}}, d) & \text{if } i > D/2 \end{cases}$$

Hold fixed half of the components

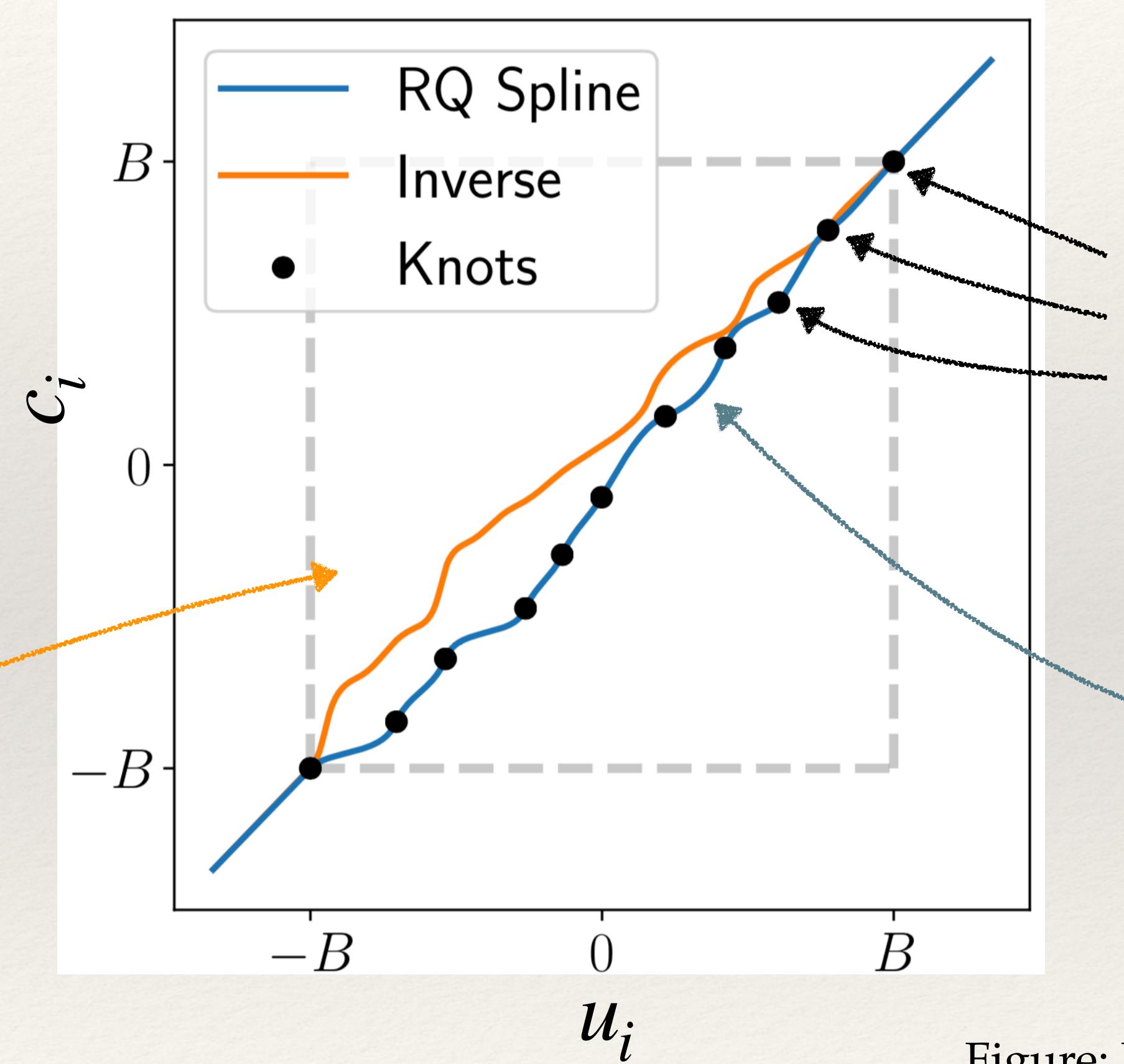
Transform remaining components element-wise, conditional on other half and  $s$ .

- ❖  $c_i$  should be **differentiable** and have **analytic inverse** with respect to  $u_i$ .

# Normalizing flow

- ❖ Spline flow  
(Durkan et al, 2019):

analytic inverse



**knots and derivatives**  
output of neural  
network;  
input  $(u_{1:\frac{D}{2}}, d)$

rational-quadratic  
spline interpolation

Figure: Durkan *et al* (2019)

# Training

- ❖ Train using  $(\theta^{(i)}, d^{(i)})$  pairs
    1. Draw parameters from prior:  $\theta^{(i)} \sim p(\theta)$
    2. Simulate data:  $d^{(i)} = h(\theta^{(i)}) + n^{(i)}$

waveform model      noise realisation, e.g.,  $n^{(i)} \sim p_{S_n}(n)$

- ❖ Free to choose any data representation (e.g., frequency domain, time domain, etc.). This will impact how well the network learns.
  - ❖ **Key point:** Even though we learn the posterior, our training data consist only of simulated data (not posterior samples).

# Simulation-based inference

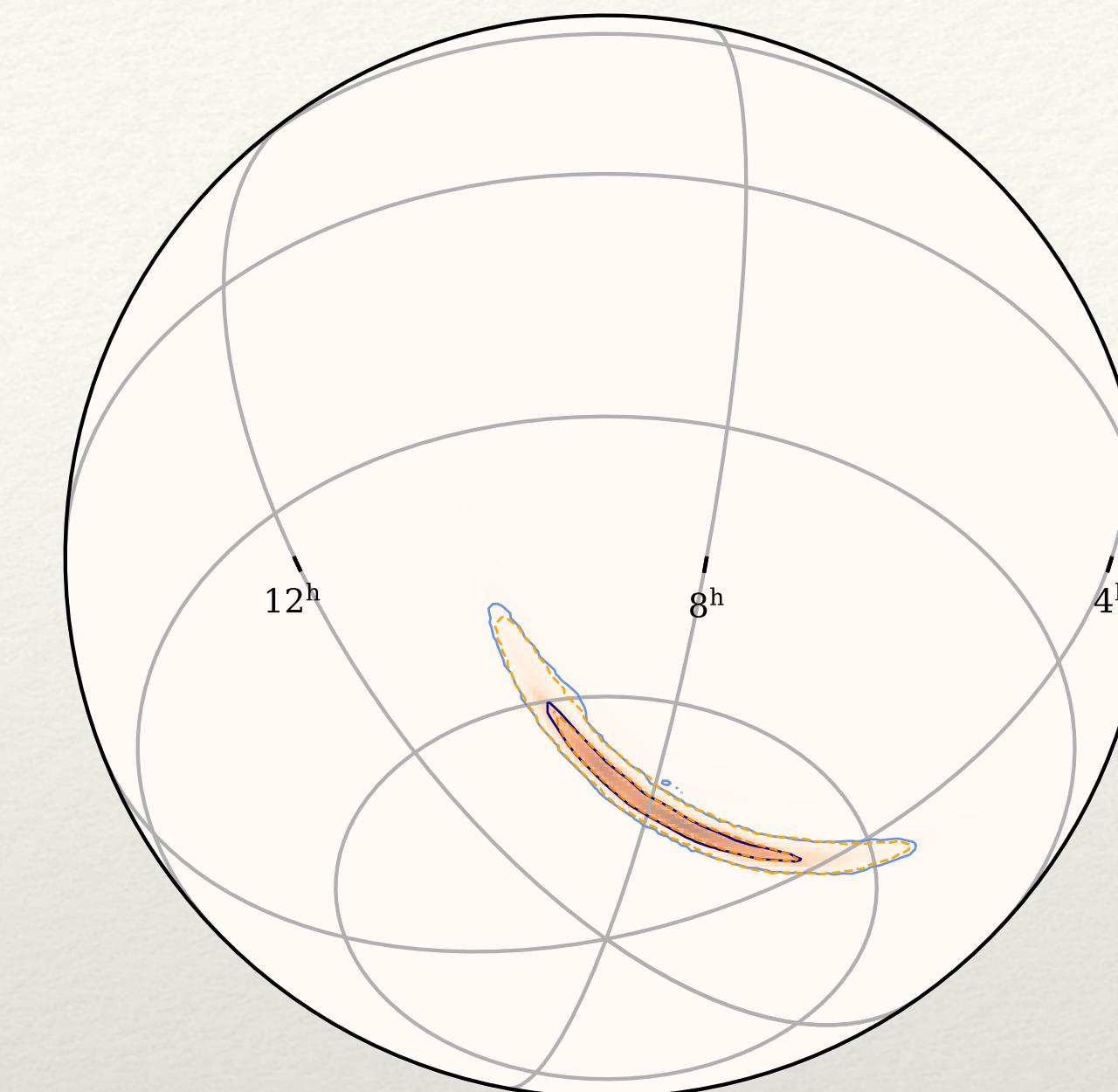
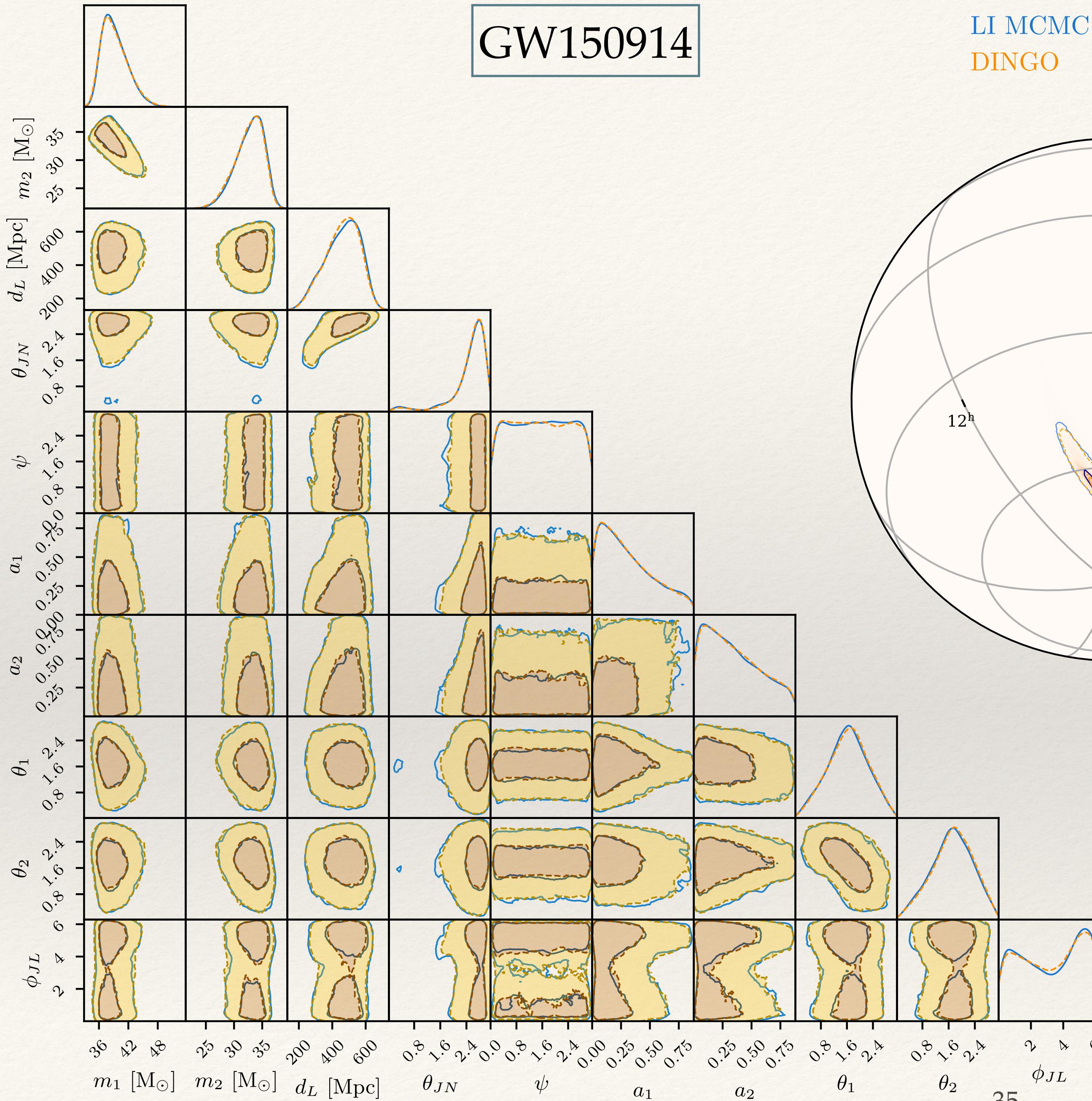
# Simulation-based inference

- ❖ Loss obtained by **minimising the Kullbeck-Liebler (KL) divergence** between the model and the true posterior,

$$\begin{aligned} L &= \mathbb{E}_{p_{\text{data}}(d)} D_{\text{KL}} [p_{\text{data}}(\theta|d) \| p_{\text{model}}(\theta|d)] \\ &= \int dd p_{\text{data}}(d) \int d\theta p_{\text{data}}(\theta|d) \log \frac{p_{\text{data}}(\theta|d)}{p_{\text{model}}(\theta|d)} \quad \curvearrowright \text{Bayes' theorem} \\ &\simeq \int d\theta p_{\text{data}}(\theta) \int dd p_{\text{data}}(d|\theta) [-\log p_{\text{model}}(\theta|d)] \\ &\approx \sum_{\substack{\theta^{(i)} \sim p_{\text{data}}(\theta) \\ d^{(i)} \sim p_{\text{data}}(d|\theta^{(i)})}} -\log p_{\text{model}}(\theta^{(i)}|d^{(i)}) \end{aligned}$$

# GW150914

LI MCMC  
DINGO



- ❖ This generates the **posterior**, matching MCMC
- ❖ 50,000 samples in  $\sim 20$  s

---

# Conclusions

---

- ❖ To specify a machine learning algorithm, require (1) **training data**, (2) **a model**, (3) a **loss function**, and (4) an **optimization algorithm**.
- ❖ For GW parameter estimation,
  1. **Training data:** Parameters  $\theta^{(i)}$  and simulated data sets  $d^{(i)}$
  2. **Model:** Normalising flow
- ❖ **Simulation-based inference is especially useful when likelihoods are intractable.**
- ❖ Next: **Tutorial!**

# EXTRA SLIDES

---

# Data augmentation

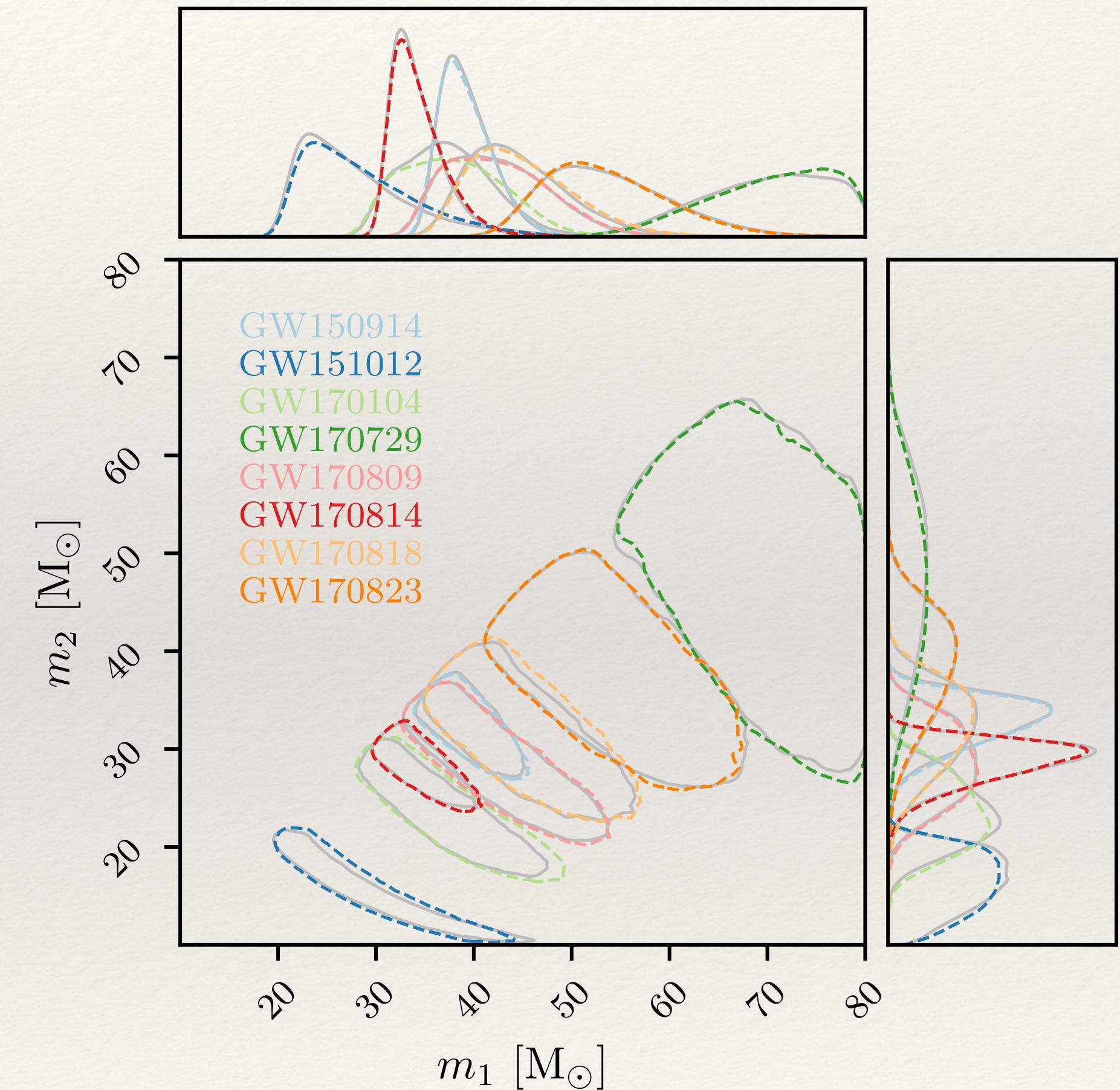
---

- ❖ Large networks (involving  $O(10^8)$  free parameters) are required for accurate inference. This runs the risk of overfitting unless the training dataset is very large.
- ❖ For LVK inference, we use  $5 \times 10^6$  training waveforms. However, we augment these with inexpensive transformations during training:
  - ❖ Each epoch, draw new noise realisations and extrinsic parameters.
  - ❖ If waveform models were fast enough (e.g., galactic binaries for LISA using Michael's code) could generate them on the fly as well.

# Adapting to changing detectors

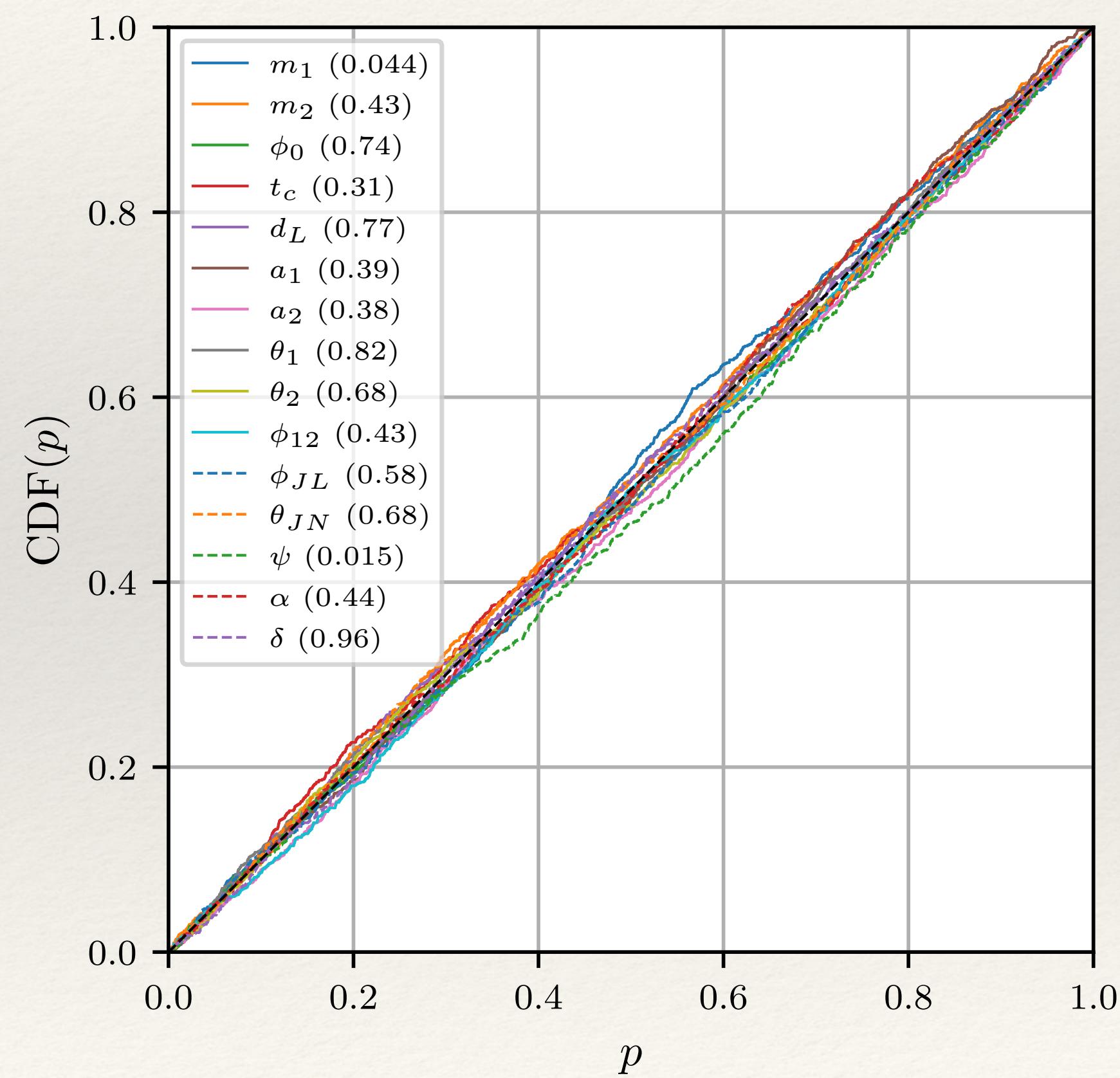
- ❖ LVK detector noise is mostly stationary Gaussian, but it does vary from event to event.
  - ❖ For classical methods, estimate the PSD and use it in the noise-weighted inner product.
- ❖ For SBI, **augment the training data with PSD fluctuations**. This can involve an empirical PSD distribution, e.g., example PSDs estimated throughout an observing run.
- ❖ **Include as additional context for the model,**

$$p_{\text{model}}(\theta | d, S_n)$$



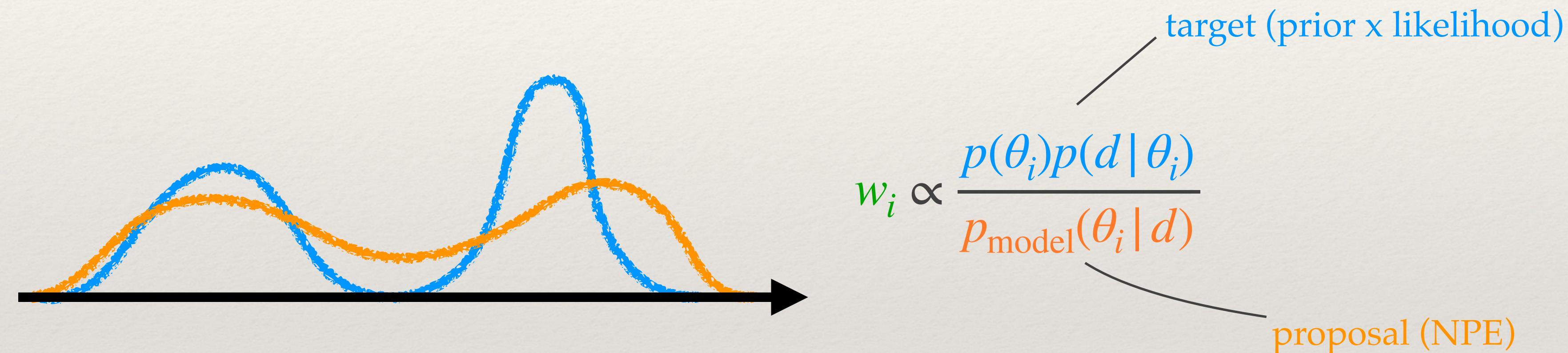
# Validating results

- ❖ Standard GW parameter estimation tests can be used to validate results
  - ❖ P-P plots
    - ❖ Simulated data
    - ❖ “within-distribution”
  - ❖ Comparisons against other samplers
    - ❖ Real data
    - ❖ “out-of-distribution”



# Validating results

- ❖ Since we have access to the GW likelihood and the NPE density, we can use **importance sampling** to compare.



- ❖ **Effective number of samples**  $n_{\text{eff}} = \frac{\left( \sum_i w_i \right)^2}{\sum_i w_i^2}$  as measure of performance.

- ❖ **Evidence**  $p(d) \approx \frac{1}{n} \sum_{i=1}^n w_i$

# Importance sampling

