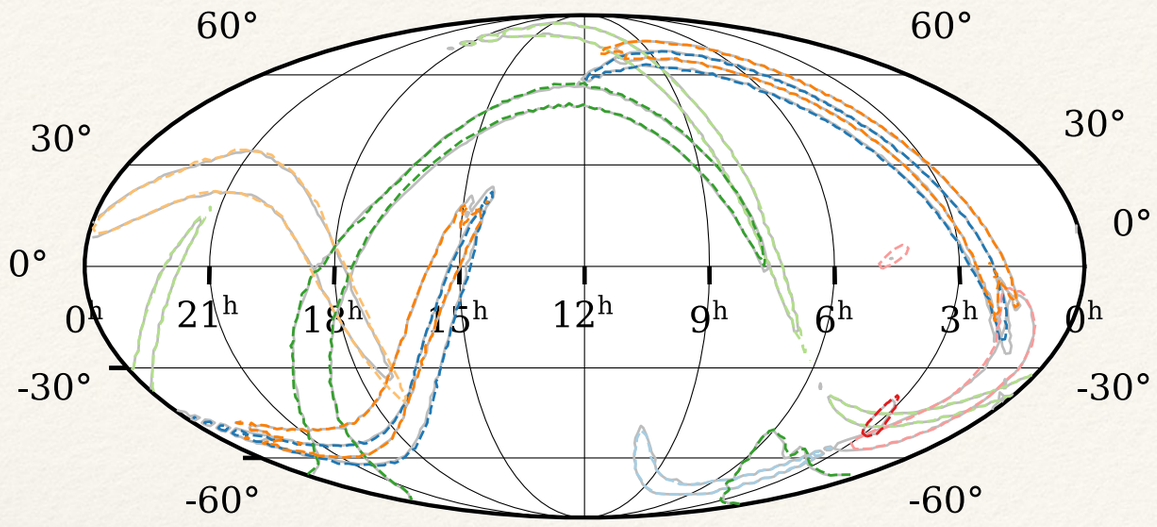



*LISA Data Analysis: from classical methods to machine learning*

*November 22, 2022*

# Machine Learning for GW Parameter Estimation

**Stephen Green**
(University of Nottingham)

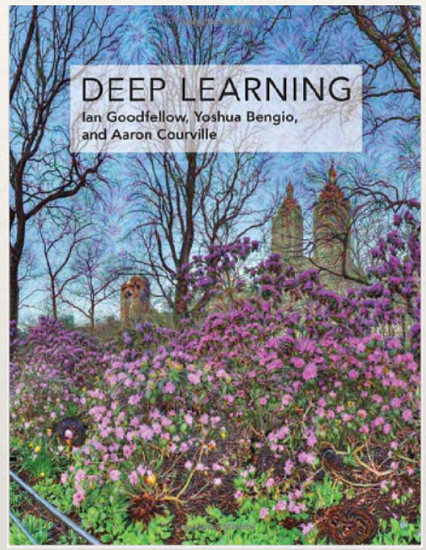
with **Natalia Korsakova**
(APC)

# Outline

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

# References

- **Textbook: "Deep Learning" by Goodfellow, Bengio, and Courville**
  - Free online at https://www.deeplearningbook.org
  - Lecture covers parts of Chapters 5, 6, 9, 20




- **PyTorch**
  - Machine learning library used for tutorial
  - Other tutorials at https://pytorch.org
- **Paper references**
  - GW parameter estimation with deep learning: 2008.03312, 2106.12594, 2210.05686.

# Machine learning

# Introduction to machine learning

- **Machine learning uses computers to learn patterns from data.**
  - Typically used to solve problems that are hard to program in conventional ways. Instead, train by example.

- Typically we have a dataset $\{\boldsymbol{x}^{(i)}\}$ consisting of many data points $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$. The data points may or may not have associated labels $\boldsymbol{y}^{(i)} \in \mathbb{R}^m$.
  - **Unsupervised:** learn $p(\boldsymbol{x})$
    - Examples: density estimation, sampling
  - **Supervised:** learn $p(\boldsymbol{y} \mid \boldsymbol{x})$
    - Examples: regression, classification

# Supervised learning for GWs

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

$$p(y \mid \boldsymbol{x}) \qquad \boldsymbol{x} \in \mathbb{R}^n,\ y \in \{1, \ldots, k\}$$

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

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

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

  - Waveform modeling: Predict a signal given the source parameters
  - Parameter estimation: Predict the source parameters given the data

# Machine learning recipe

1. Build a dataset of training examples.
    - $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ pairs
2. Define a parametrised probabilistic model for the data.
    - $p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$
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.
    - choose $\boldsymbol{\theta}$.

# Maximum likelihood estimation

- Given a model, the loss function is determined by **maximising the likelihood of the training data under the model.**
- **Unsupervised learning.** Assume we have
  1. $N$ independent samples $\boldsymbol{x}^{(i)} \sim p_{\text{data}}(\boldsymbol{x})$
  2. Parametrised model $p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$
- **Maximum likelihood estimate**

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

negative log probability loss

# Conditional distributions

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

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

# Example: Linear regression

- Suppose we have labelled data $(\boldsymbol{x}^{(i)}, y^{(i)})$.
- Let $p_{\text{model}}(y \mid \boldsymbol{x}) = \mathcal{N}\left(\mu(\boldsymbol{x}), \sigma^2\right)(y)$ where $\mu(\boldsymbol{x}) = \boldsymbol{\theta} \cdot \boldsymbol{x}$; $\sigma$ fixed.
- Using the PDF $p(y|\boldsymbol{x};\boldsymbol{\theta}) = \dfrac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\dfrac{(y-\mu(\boldsymbol{x}))^2}{2\sigma^2}\right)$

  we obtain the loss function

$$J(\theta) = -\sum_{i=1}^{N} \log p(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta})$$

$$= \frac{N}{2}\log 2\pi\sigma^2 + \sum_{i=1}^{N} \frac{(y^{(i)} - \mu(\boldsymbol{x}^{(i)}))^2}{2\sigma^2}$$

$\propto$ mean squared error

- Can solve exactly $\nabla_{\boldsymbol{\theta}} J = 0 \quad \Longrightarrow \quad \boldsymbol{\theta}_{\text{ML}} = \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y}$

# More general regression

- More generally $\mu(\boldsymbol{x})$ does not have to be linear. We can increase the **representational capacity** of the model by using more complicated functions.
  - E.g., polynomial $\mu(x) = b + \sum_{i=1}^{k} w_i x^i$ (can still solve in closed form)
  - E.g. nonparametric regression

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

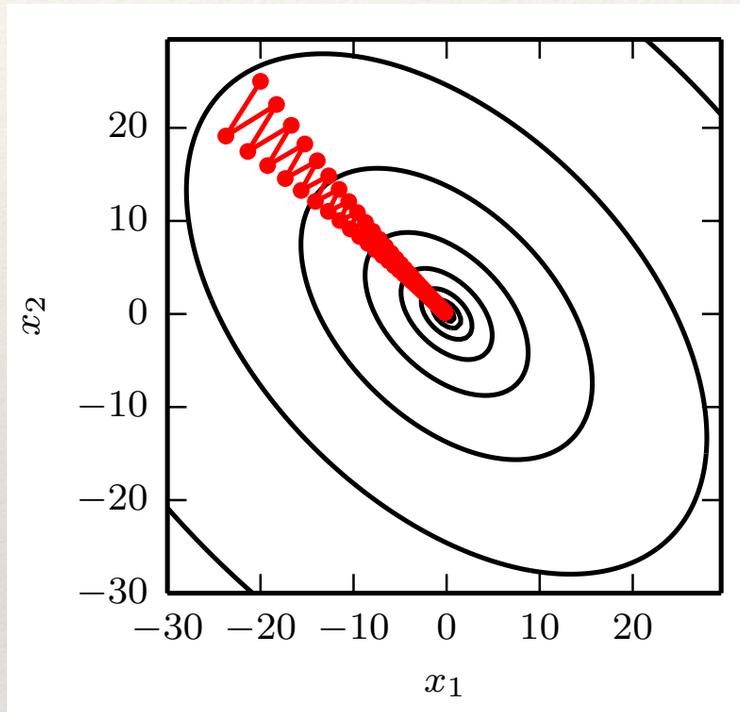
  - E.g., neural network
- Not all models can be optimised in closed form.

# 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 $\boldsymbol{\theta}$ to approach the minimum.
- Starting from a point $\boldsymbol{\theta}_0$ we can move to a new point by following the gradient

$$\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 - \epsilon \nabla_{\boldsymbol{\theta}} J|_{\boldsymbol{\theta}_0}$$

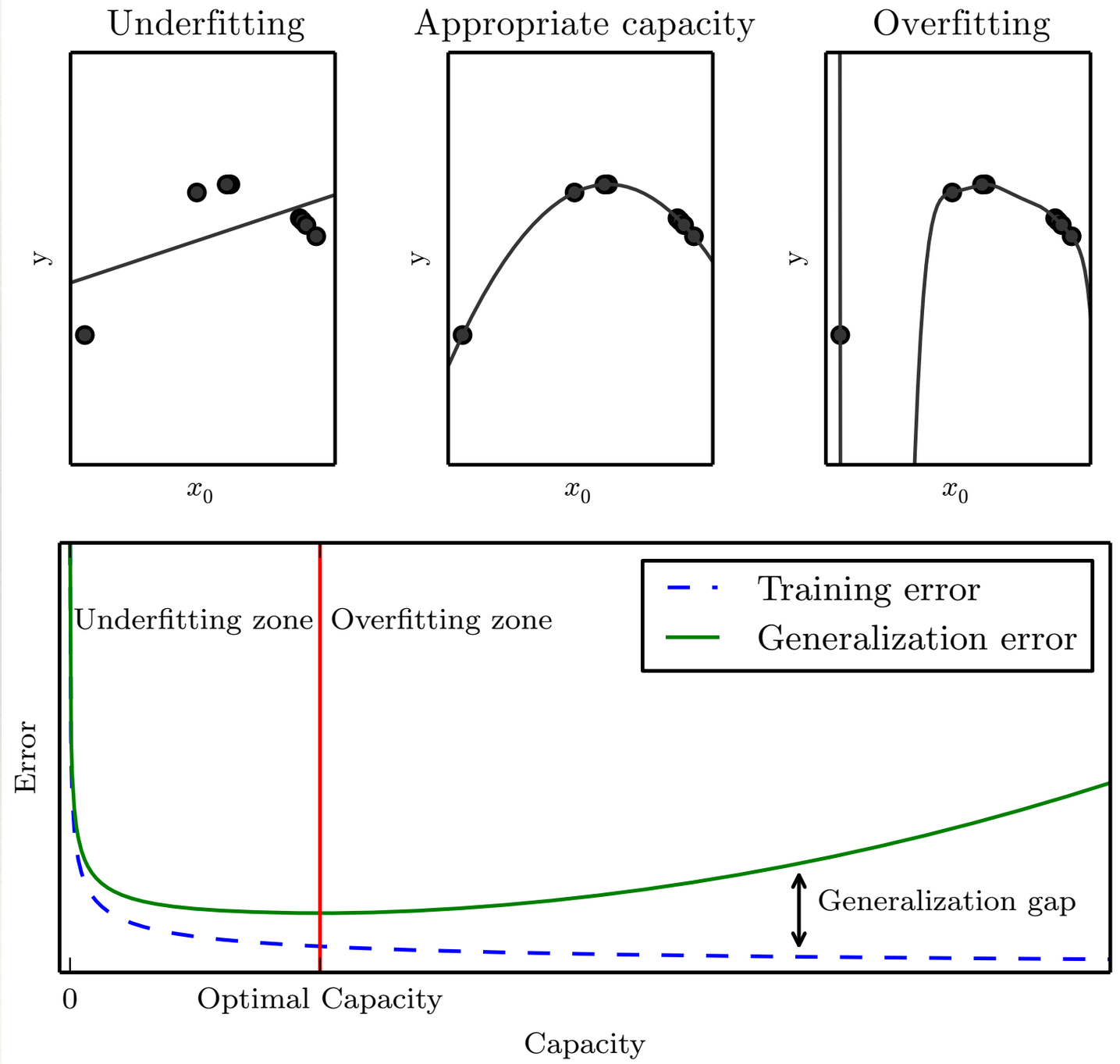
"Learning rate"




Goodfellow et al (2016)

- 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.

# 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:

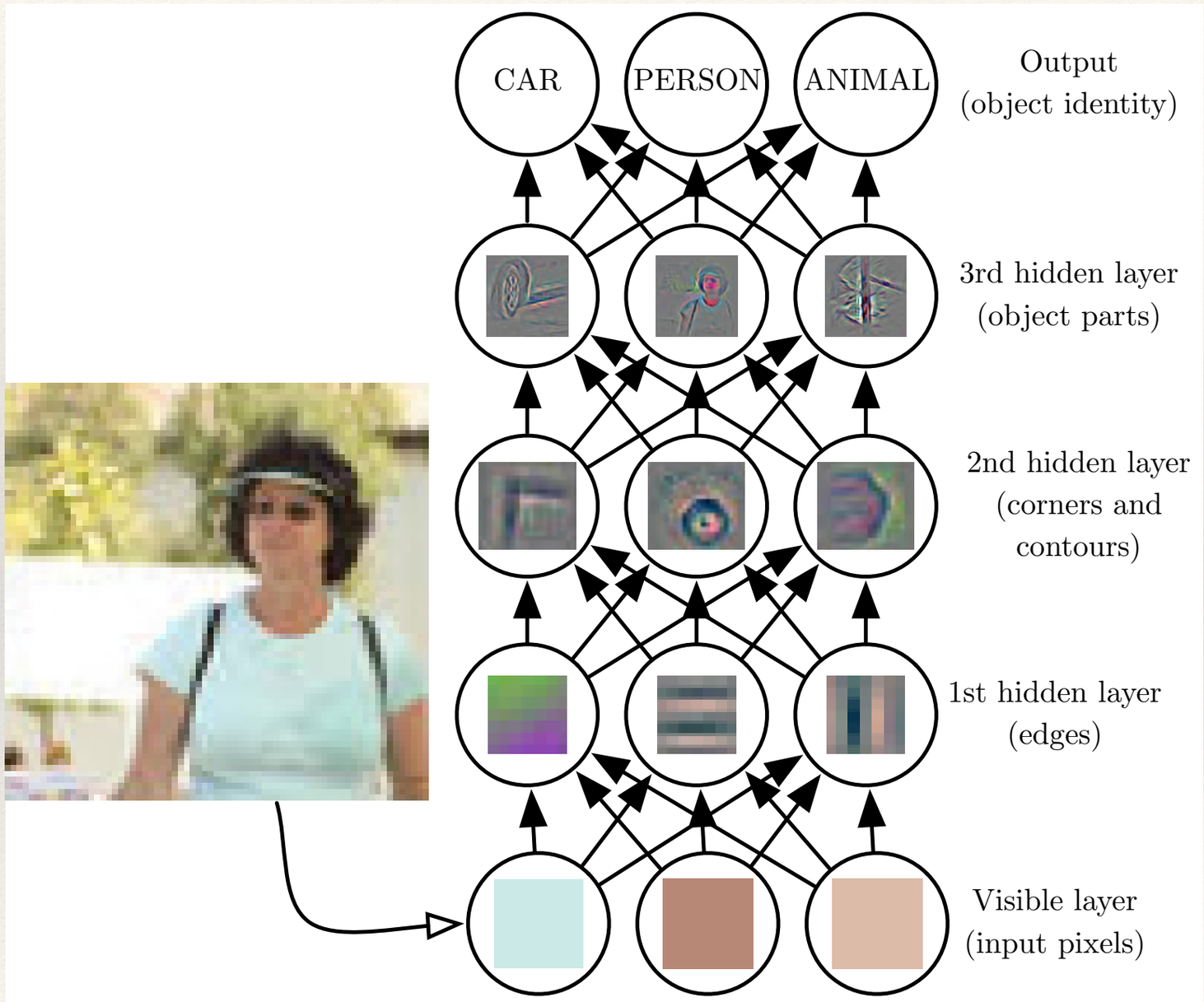
1. **dataset** — for supervised learning $\{\boldsymbol{x}^{(i)}, y^{(i)}\}$ pairs
2. **model** — E.g., linear regression $p_{\text{model}}(y \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{\theta}^\top \boldsymbol{x}, 1)(y)$
3. **loss function** — E.g., $J(\theta) = -\mathbb{E}_{p_{\text{data}}(\boldsymbol{x})} \log p_{\text{model}}(\boldsymbol{x})$
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 $\boldsymbol{x}$, 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.

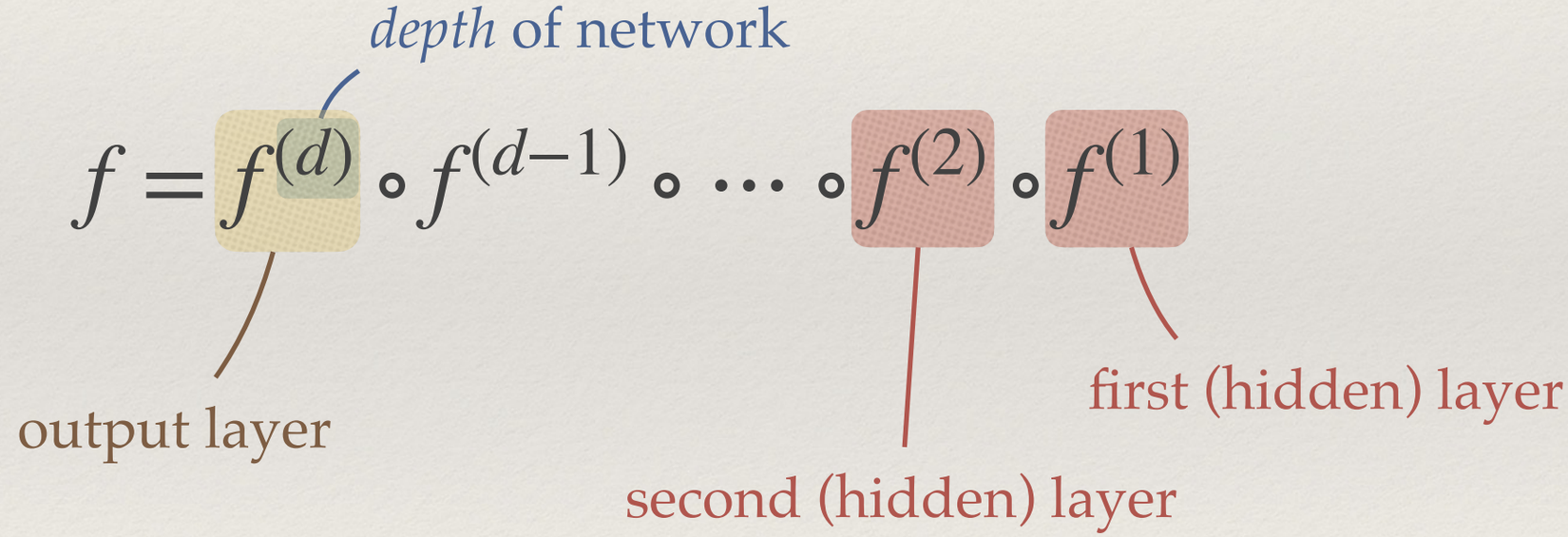
# Introduction to deep learning




Goodfellow *et al* (2016)

# Feedforward neural networks

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

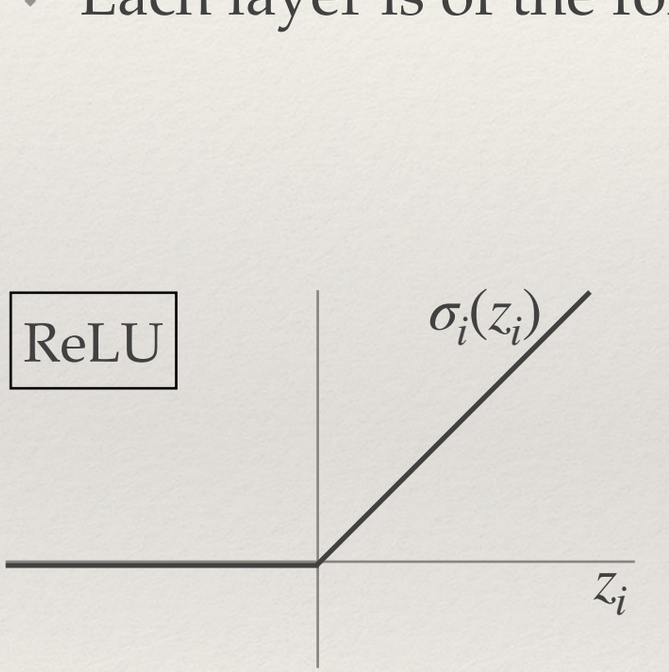


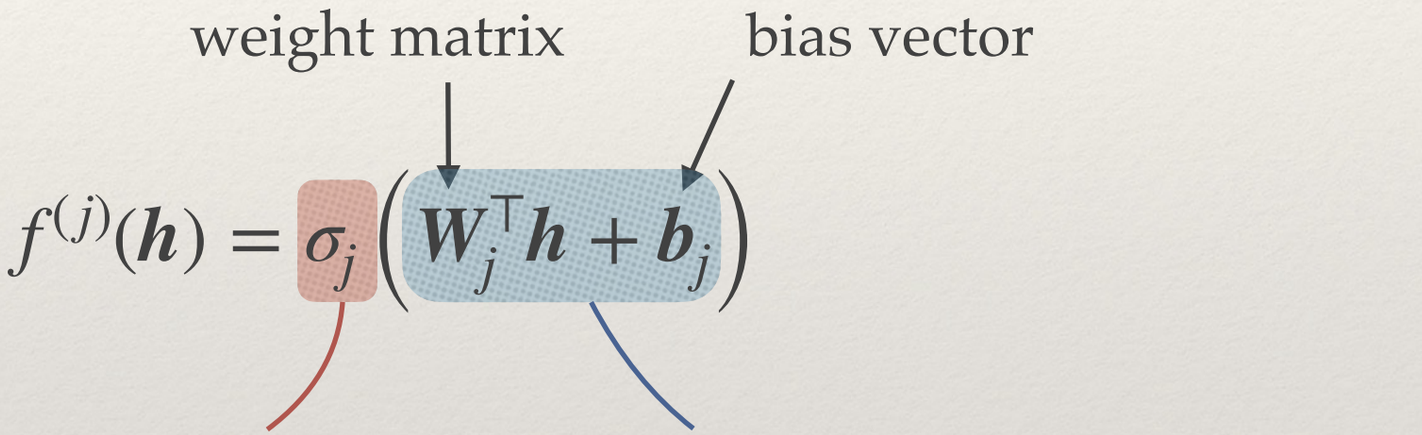

- "Feed-forward" because there is no feedback of later layers on earlier ones.

# Feedforward neural networks

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

- Each layer is of the form




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

# Feedforward neural networks

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

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

- 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

$$p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}\left(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{I}\right)(\boldsymbol{y})$$

- 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

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

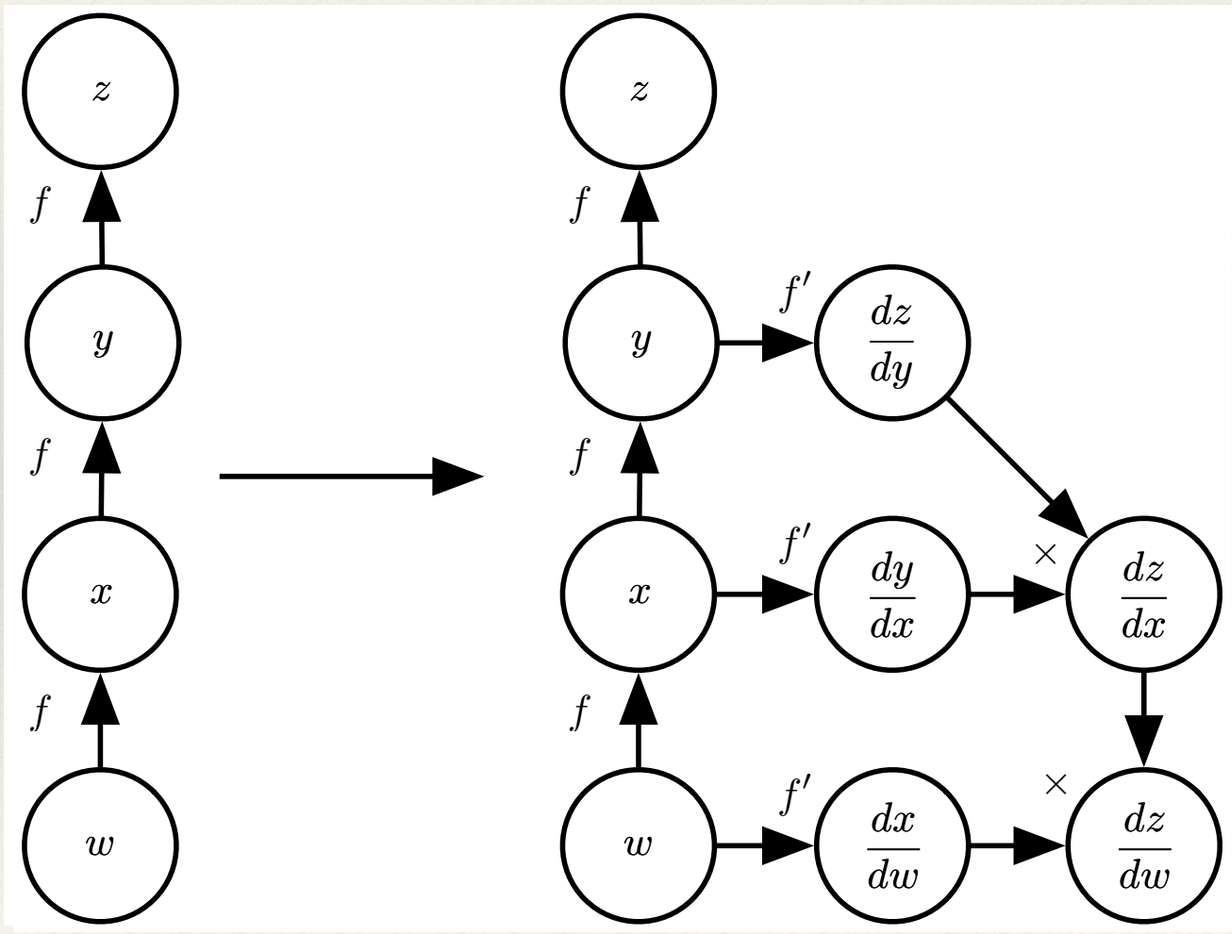
plus the **chain rule of calculus** and differentiability of all operations.

# 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 framework (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**

$$p_{\text{model}}(\boldsymbol{y}\,|\,\boldsymbol{x}) = \mathcal{N}\left(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\Sigma}(\boldsymbol{x})\right)$$

  - For diagonal covariance, better to use the precision matrix, and enforce positivity using, e.g., softplus activation.
- Example: **Mixture density networks**
  - Sum of normal distributions weighted by categorical (multinouilli) output

$$p_{\text{model}}(\boldsymbol{y}\,|\,\boldsymbol{x}) = \sum_k p_k(\boldsymbol{x})\mathcal{N}\left(\boldsymbol{\mu}_k(\boldsymbol{x}), \boldsymbol{\Sigma}_k(\boldsymbol{x})\right)$$

- 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.

# Gravitational-wave parameter estimation

# 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, and given by

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

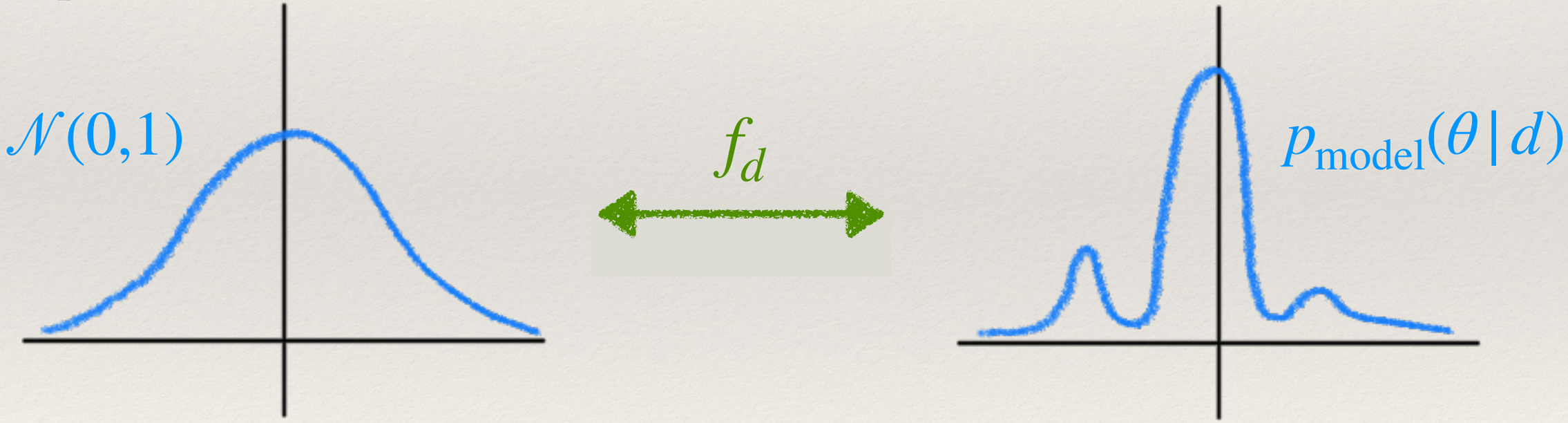
  so MCMC can be used to sample from the posterior.
- ML approaches enable:
  1. Speed improvements (amortized inference);
  2. Inference with real noise.

# 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.
- **Approaches to speed up classical methods**
  - **Normalising flows** to speed up dynesty sampler (nessai).

# Normalising flows

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




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

# Normalising flow

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

- **Requirements:**
  1. Invertible
  2. Simple Jacobian determinant
- Parametrise $f_d$ using **neural network.**

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

needed to evaluate loss

# Normalizing flow

- **Requirements:**

  1. Invertible ✓
  2. Simple Jacobian determinant ✓ $\det J_{f_d} = \prod_{i=\frac{D}{2}+1}^{D} c_i' \left(u_i; u_{1:\frac{D}{2}}, d\right)$

- Use a sequence of "coupling transforms":

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

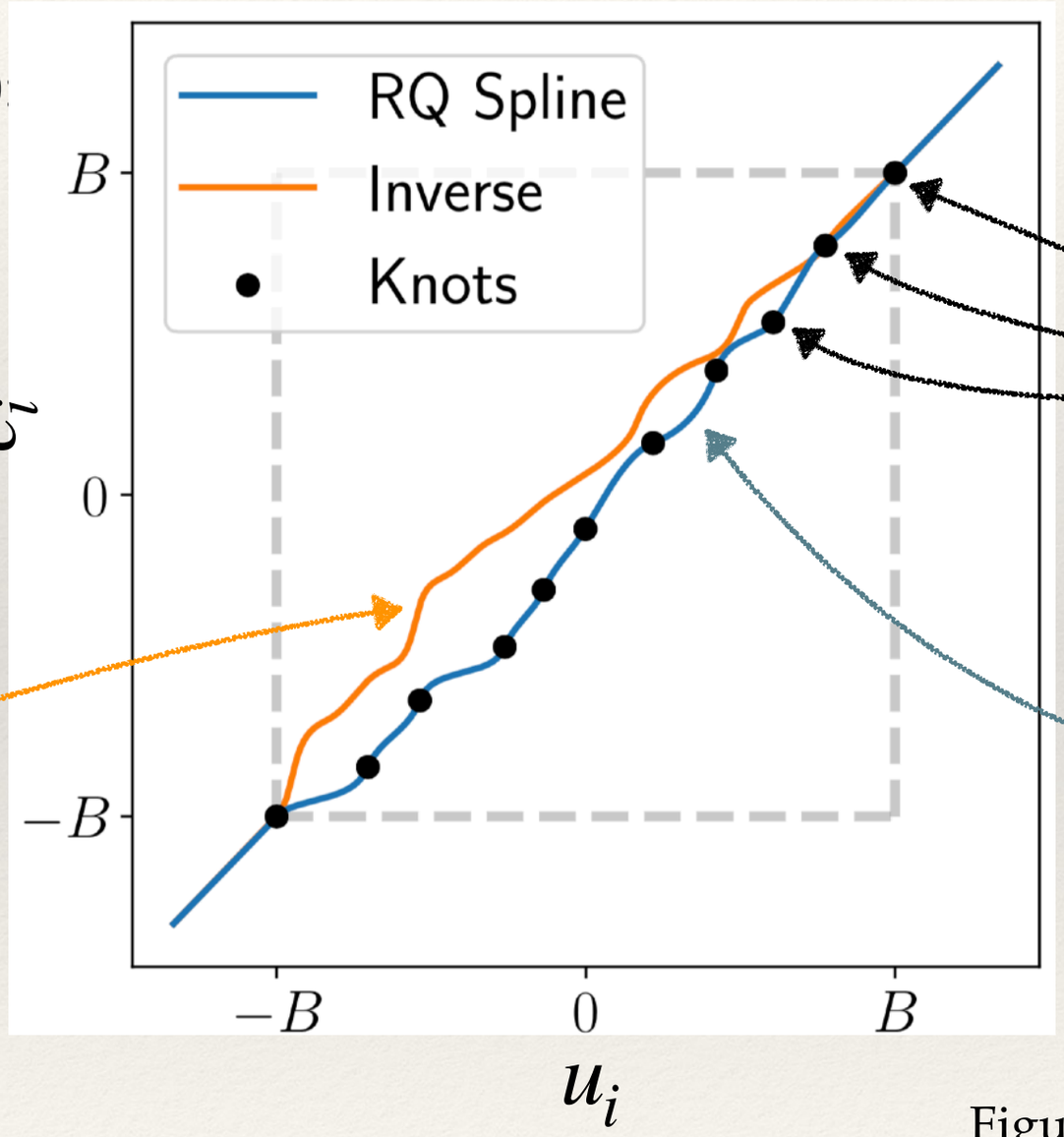
Hold fixed half of the components

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

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

# Normalising flow

- **Spline flow**
  (Durkan et al, 2019)




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

analytic inverse

rational-quadratic spline interpolation

Figure: Durkan *et al* (2019)

# Normalising flow

- Sequence of flows can give very complicated distribution

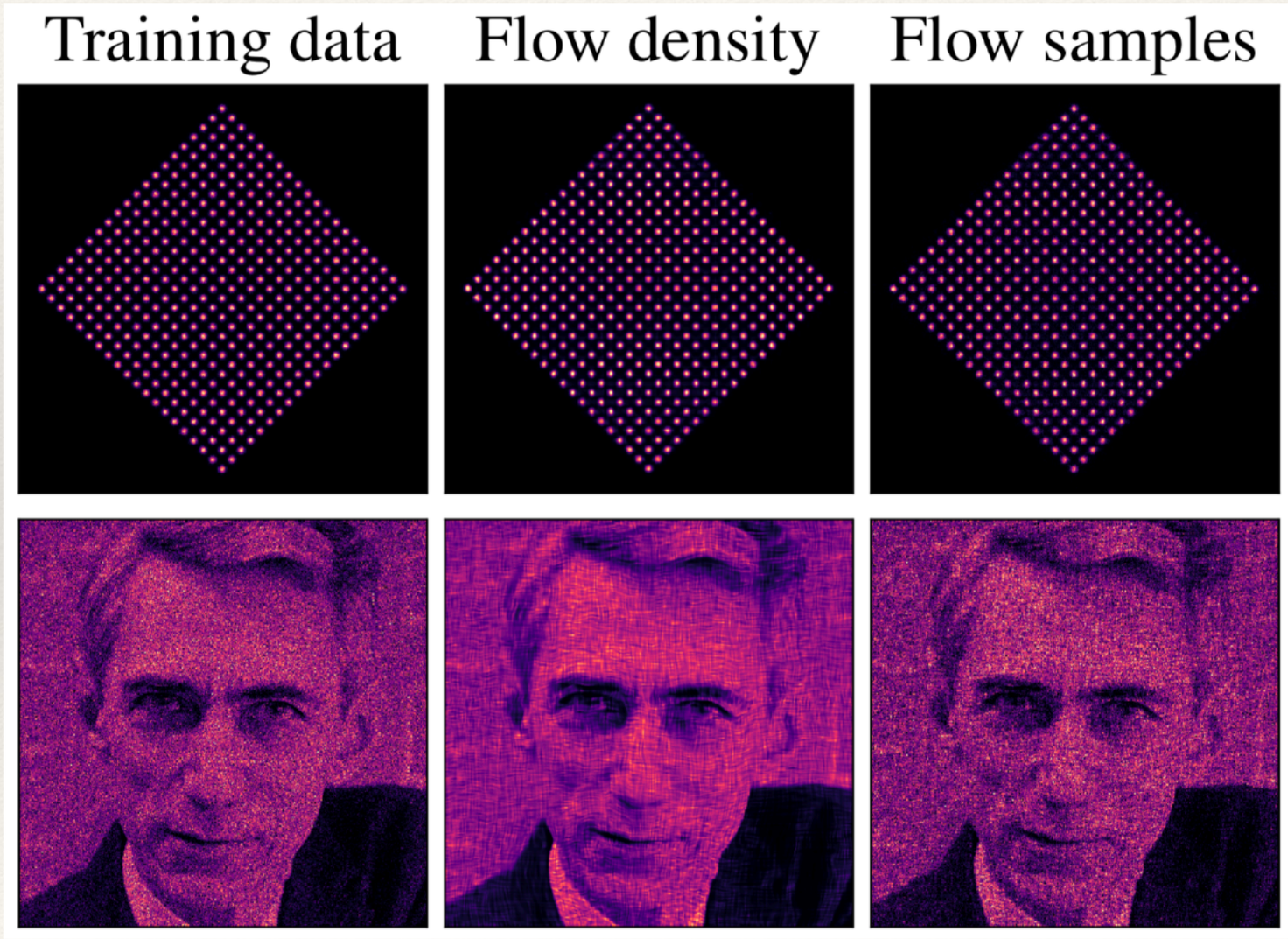



Image: Durkan *et al* (2019)

# 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., $n^{(i)} \sim p_{S_\mathrm{n}}(n)$

- We are free to choose any data representation (e.g., frequency domain, time domain, PCA). This will impact how well the network learns.
- For NPE, minimise cross-entropy loss $J = \mathbb{E}_{p_\mathrm{data}(\theta,d)} \left[ -\log p_\mathrm{model}(\theta \,|\, d) \right]$
- **Key point:** Even though we learn the posterior, our training data consist only of simulated data (not posterior samples).

# Simulation-based inference

- Maximum likelihood estimation loss can also be derived by **minimising the Kullbeck-Liebler (KL) divergence** between the model and the true posterior,

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

Bayes' theorem

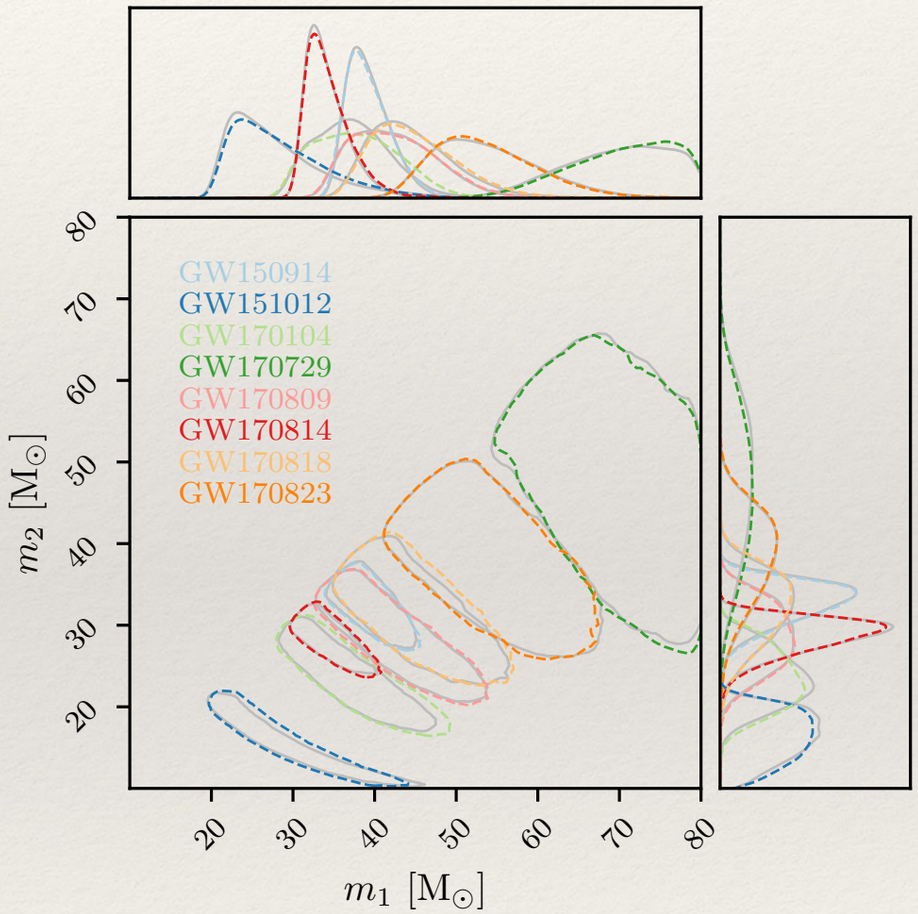
# Data augmentation

- 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

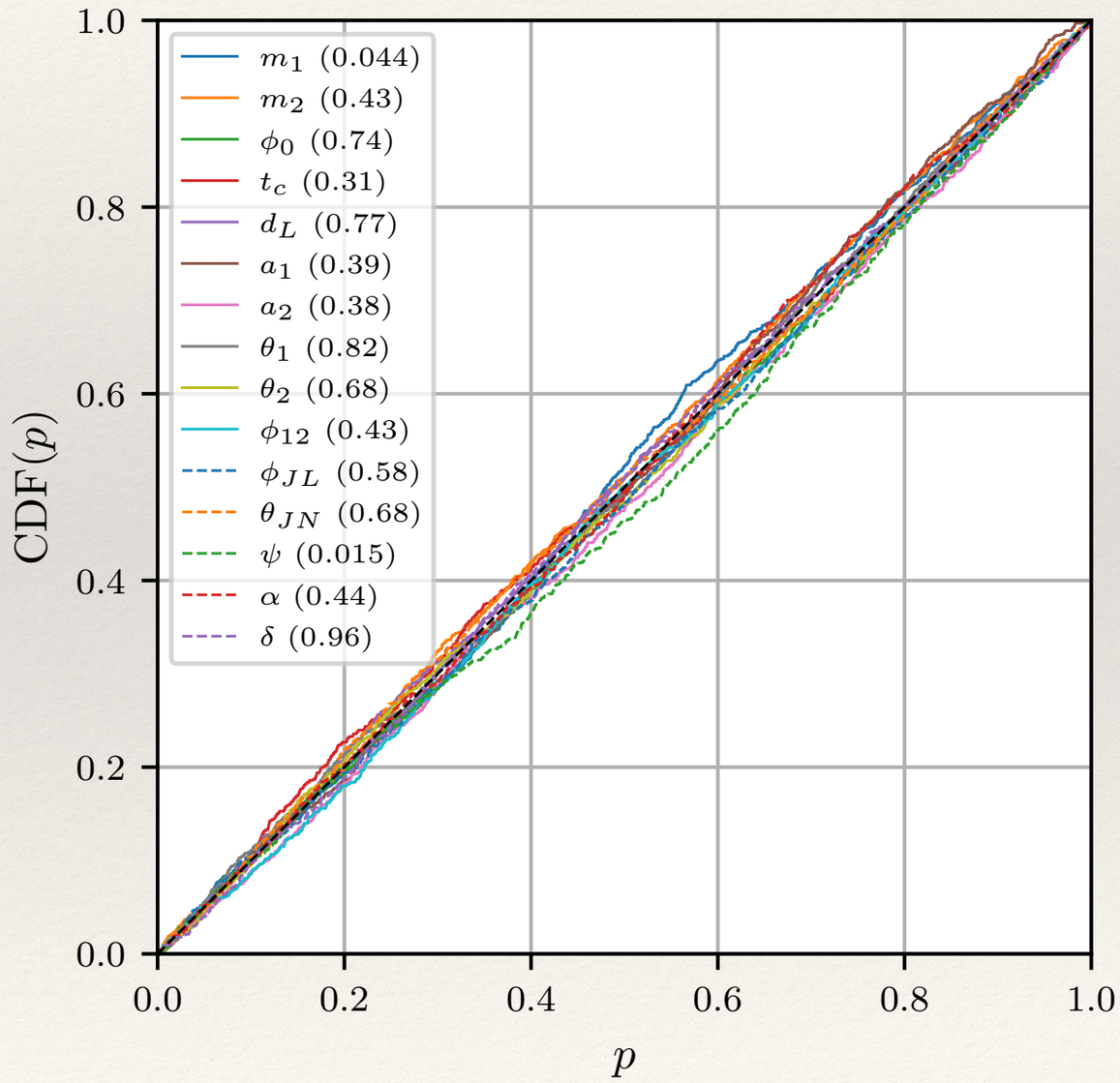
- LVK detector noise is mostly stationary Gaussian, but it does vary from event to event.
  - For classical methods, estimate the PSD and use it in the noise-weighted inner product.
- For SBI, augment the training data with PSD fluctuations. This can involve an empirical PSD distribution, e.g., example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

$$p_{\text{model}}(\theta \mid d, S_{\text{n}})$$



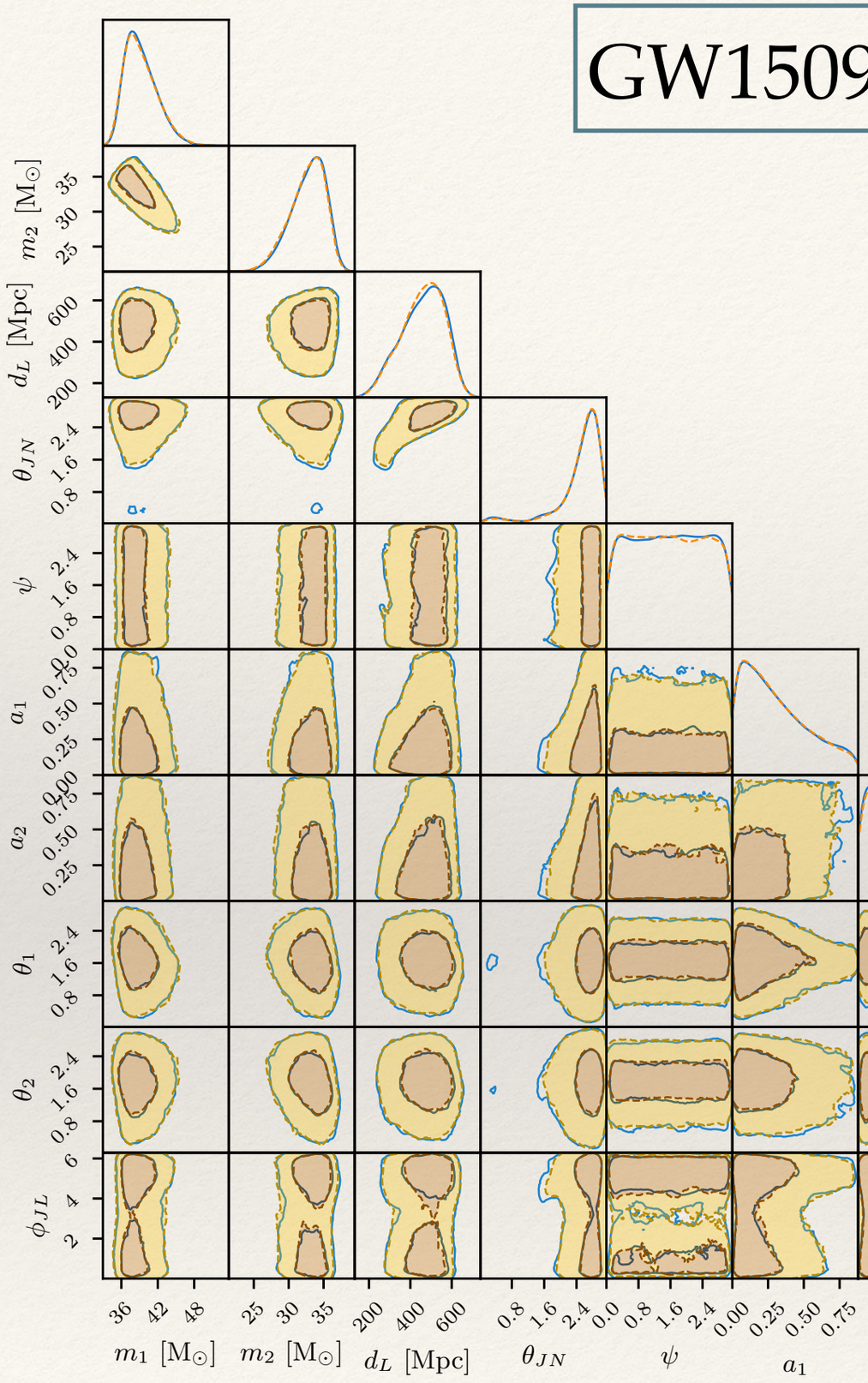
# Validating results

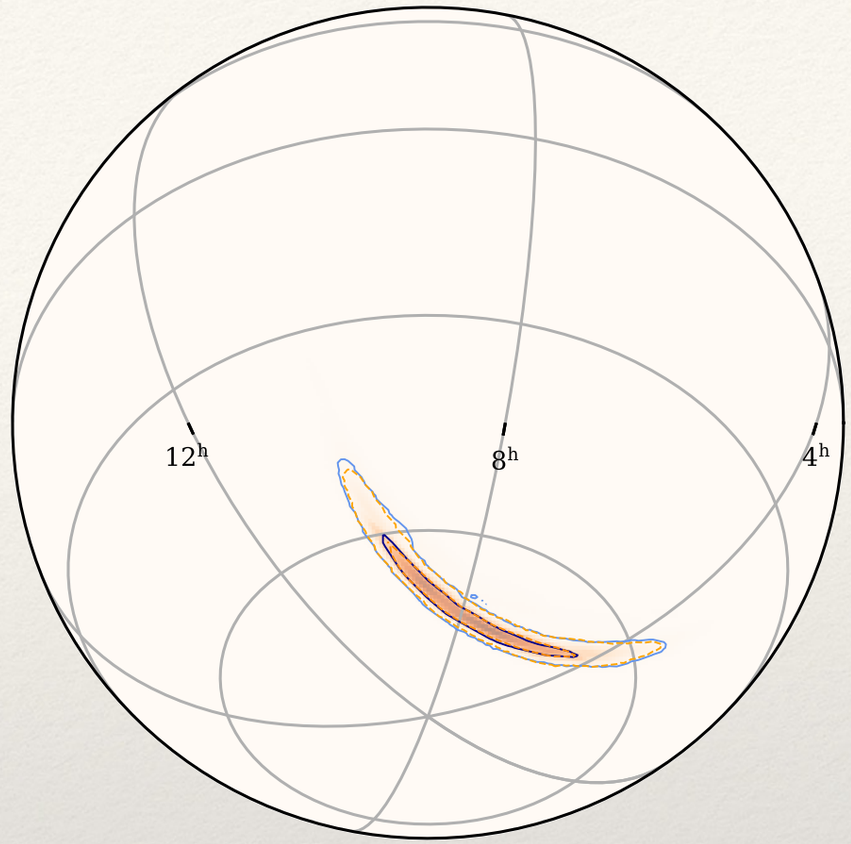
- Standard GW parameter estimation tests can be used to validate results
  - **P-P plots**
    - Simulated data
      - "within-distribution"
  - **Comparisons against other samplers**
    - Real data
    - "out-of-distribution"



# GW150914

LI MCMC
DINGO

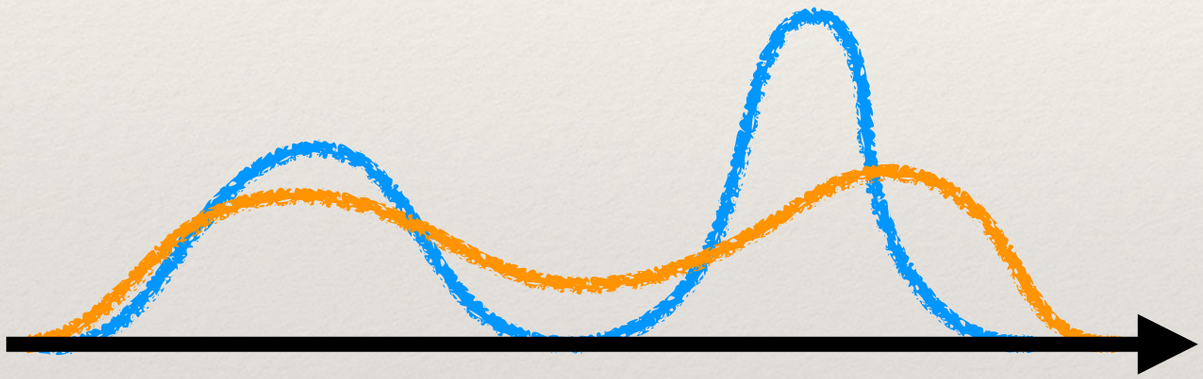



- This **generates the posterior**, matching MCMC
- 50,000 samples in ~ 20 s

# Validating results

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



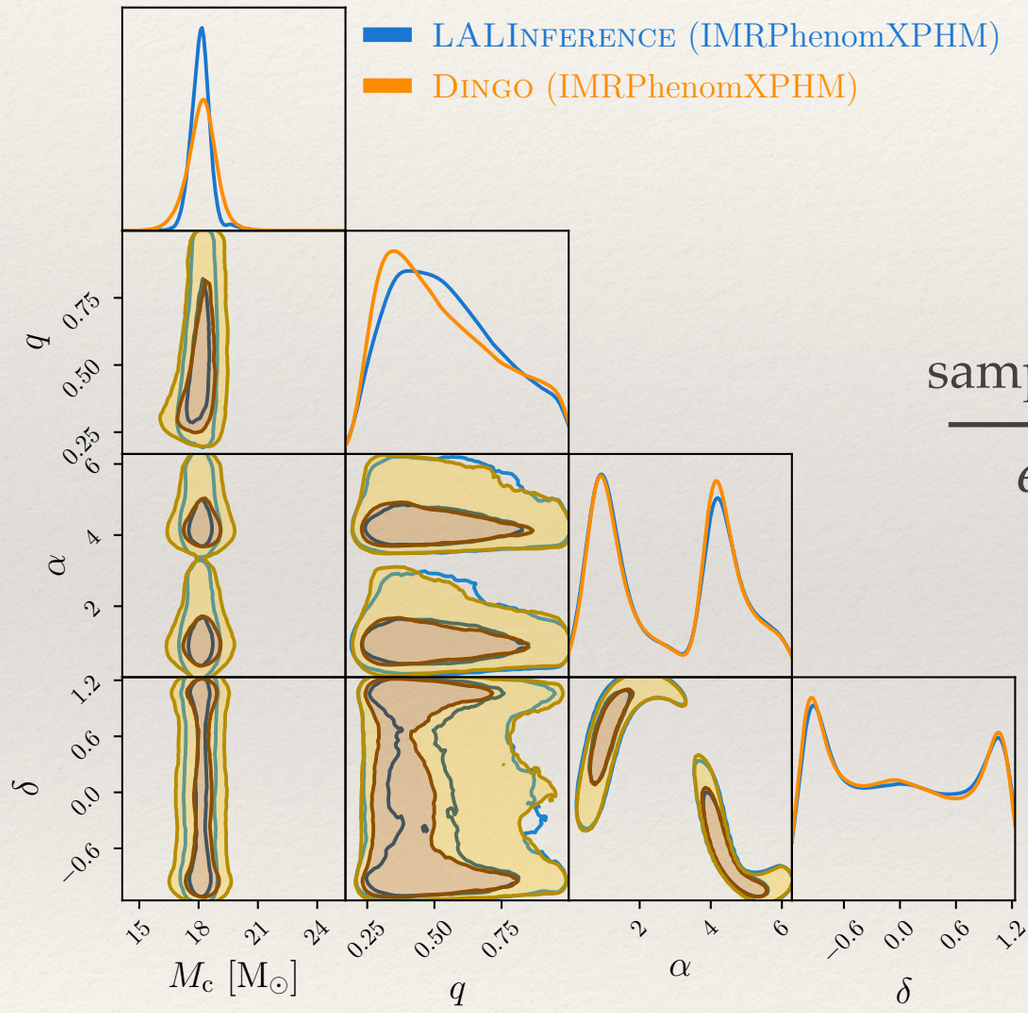
target (prior x likelihood)

$$w_i \propto \frac{p(\theta_i)p(d\,|\,\theta_i)}{p_{\text{model}}(\theta_i\,|\,d)}$$

proposal (NPE)

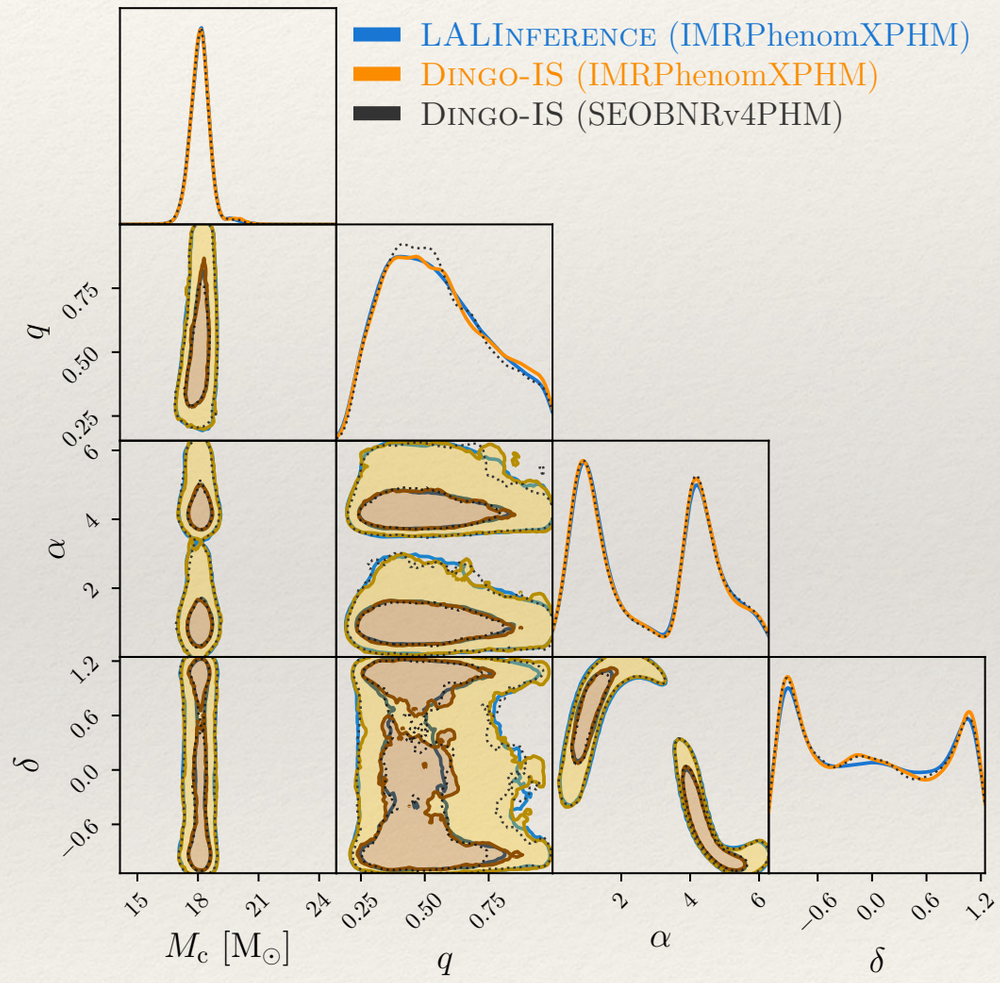
- **Effective number of samples** $n_{\text{eff}} = \dfrac{\left(\sum_i w_i\right)^2}{\sum_i w_i^2}$ as measure of performance.
- **Evidence** $p(d) \approx \dfrac{1}{n}\sum_{i=1}^{n} w_i$

# Importance sampling




sampling efficiency

$\epsilon = 12.5\,\%$



# LISA challenges

- **Overlapping events**
  - Build into simulator + expand parameter space to include multiple events. Challenge is dimensionality of new parameter space.
- **Realistic noise**
  - Naturally treat effects like data gaps or nonstationary noise without having to evaluate likelihoods with complicated off-diagonal covariance, or artificial techniques like in-painting.
  - No need to infer glitch parameters. Instead, automatically marginalise them.
- **High signal to noise**

# 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. Can naturally treat LISA challenges such as overlapping events and non-stationary Gaussian noise.
- Tomorrow: **Tutorial!**

Thank you!