Machine Learning:

A new toolbox for Theoretical Physics

Juan Rojo

VU Amsterdam & Theory group, Nikhef

D-ITP Advanced Topics in Theoretical Physics

25/11/2019

Recap from Lecture 1

Setting up the problem

problems in **Supervised Machine Learning** are defined by the following ingredients:

(1) Input dataset: $\mathscr{D} = (\boldsymbol{X}, \boldsymbol{Y})$

(2) Model: $f\left(\boldsymbol{X}, \boldsymbol{\theta}\right)$

(3) Cost function: $C\left(\boldsymbol{Y}; f(\boldsymbol{X}; \boldsymbol{\theta}\right)$

The cost function measures how well the model (for a specific choice of its parameters) is able to describe the input dataset

Fitting the model means determining the values of its parameters which **minimise the cost function**

$$\left.\frac{\partial C\left(\boldsymbol{Y}; f(\boldsymbol{X}; \boldsymbol{\theta}\right)}{\partial \theta_i}\right|_{\theta=\theta_{\text{opt}}} = 0$$

Model fitting

The simplest possible model is a polynomial: **polynomial regression**

$$f_\alpha(x;\boldsymbol{\theta}_\alpha) = \sum_{j=0}^{N_\alpha} \theta_{\alpha,j} x^j$$

for example the model class that contains all possible cubic polynomials is

$$f_3(x;\boldsymbol{\theta}_3) = \sum_{j=0}^{3} \theta_{3,j} x^j$$

a more complex model does not necessarily imply a more predictive one: the appropriate amount of complexity depends on the **features of the data sample** (*e.g.* size, variability)

in polynomial regression the model parameter are given by **least-squares method**

$$\hat{\theta} = \arg\ \min_\theta \left\{ \sum_{i=1}^{n} \left(y_i - f_\alpha(x_i;\boldsymbol{\theta}_\alpha) \right)^2 \right\}$$

minimise sum of residuals squared

Why Machine Learning is difficult

- **Fitting existing data** is conceptually different from **making predictions about new data**

- Increasing the model complexity can improve the description of the training data but **reduce the predictive power** of the model due to overfitting, unless a suitable regularisation strategy is implemented

- For complex and/or small datasets, simple models can be better at prediction than complex models. The ``**right" amount of complexity** cannot in general be determined from first principles

- It is difficult to **generalise** beyond the situations encountered in the training set: the model cannot learn what it has not seen

- Many problems that are **approachable in principle** can become **unfeasible in practice**, e.g. due to computational limitations, lack of convergence, instability ……

Deep Learning successes boosted by hardware developments

Today's lecture

- The bias/variance tradeoff
- (Deep) Neural Networks
- NN training: Backpropagation in Gradient Descent
- Regularisation of Neural Networks (cross-validation)
- Case Study I: the proton structure from neural nets
- Supervised Learning for Classification: logistic regression
- Case Study II: Higgs pair production at the LHC

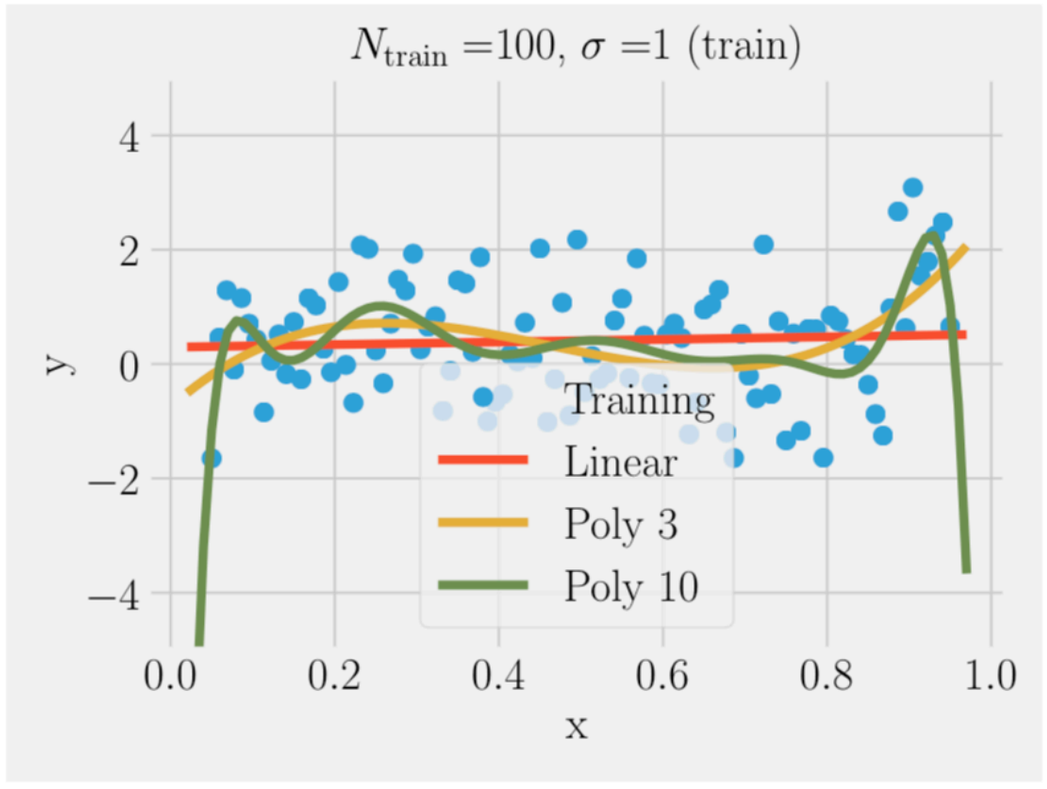
Tutorial 2:

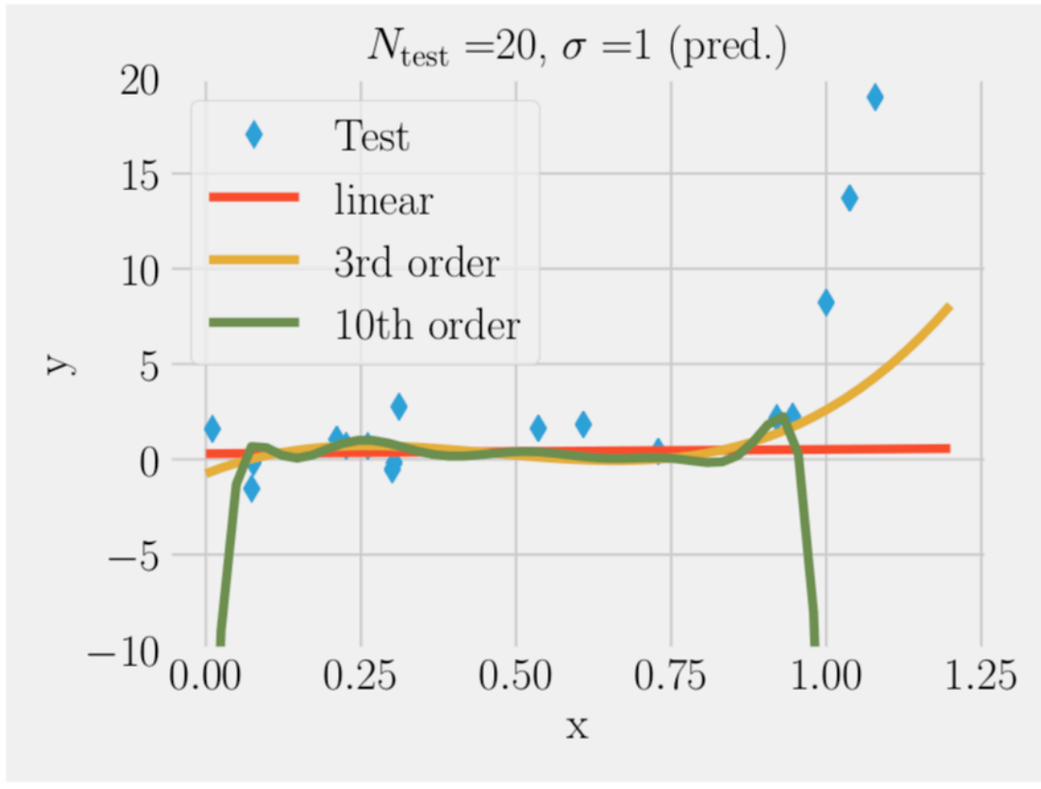
(a) Deep Neural Network Training

(b) Classification of supersymmetric particles

The bias-variance tradeoff

The bias-variance trade-off





increasing the model complexity can improve the description of the training data but **reduce the predictive power** of the model due to overfitting

The bias-variance trade-off

Our starting point is the **underlying law** ***y=f(x)*** which we aim to learn from a dataset (x_i,y_i), $i=1,\ldots,N$. We will also need an **hypothesis set** ***H*** containing all functions that we consider to be good candidates for the underlying law

The goal of **Statistical Learning Theory** is to determine a function from the hypothesis set *H* that approximates *f(x)* as best as possible, ideally in a strict mathematical limit

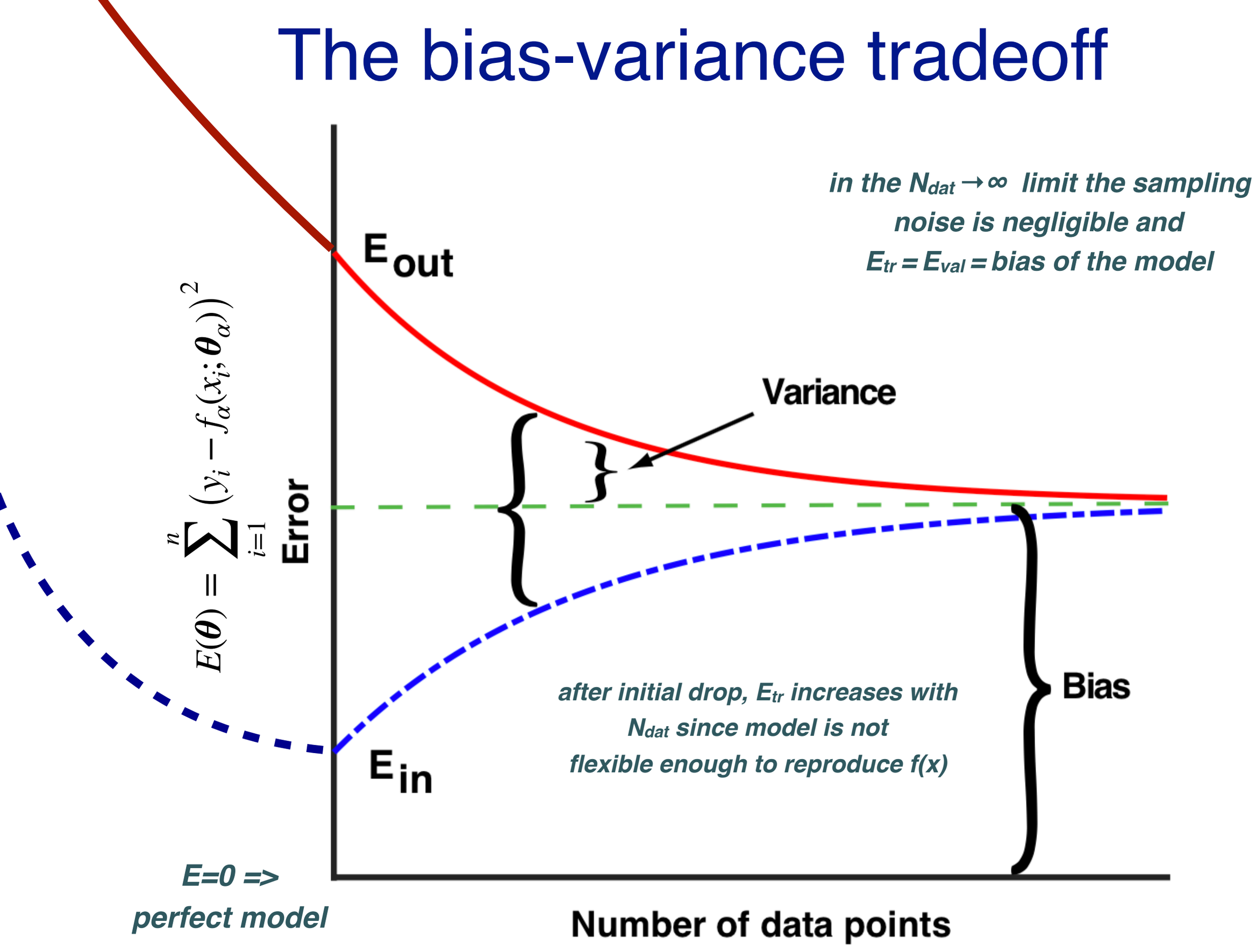
Here we will provide an **intuitive picture** of how Statistical Learning works

Consider the following situation:

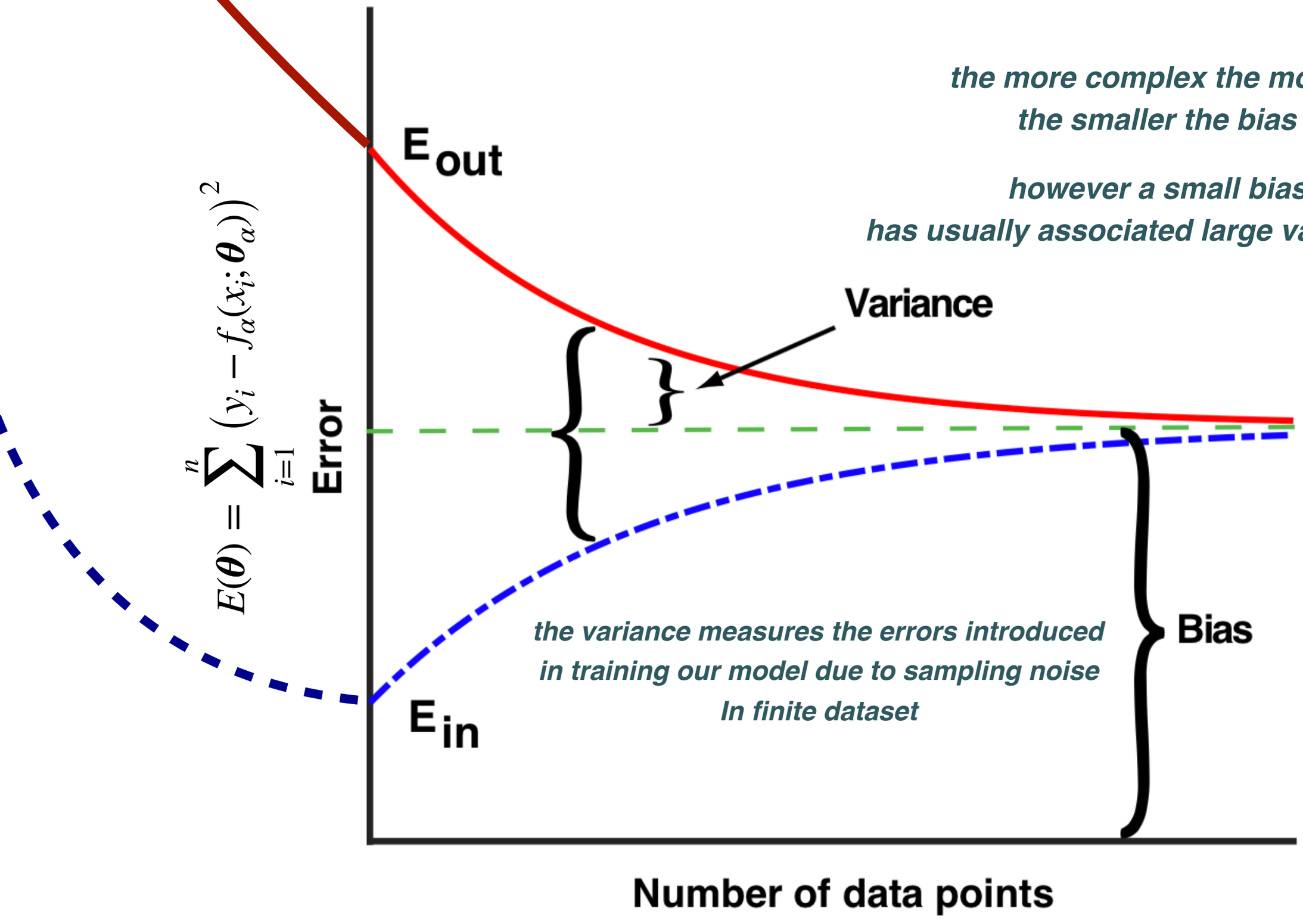
- The **underlying law** is so complex that we cannot aim to exactly reproduce *f(x)*
- We want to study the dependence of E_{tr} and E_{val} with the number of data points

This setting allows us to present one of the most important concepts in the theory of Machine Learning: **the bias-variance tradeoff**

The bias-variance tradeoff



The bias-variance tradeoff



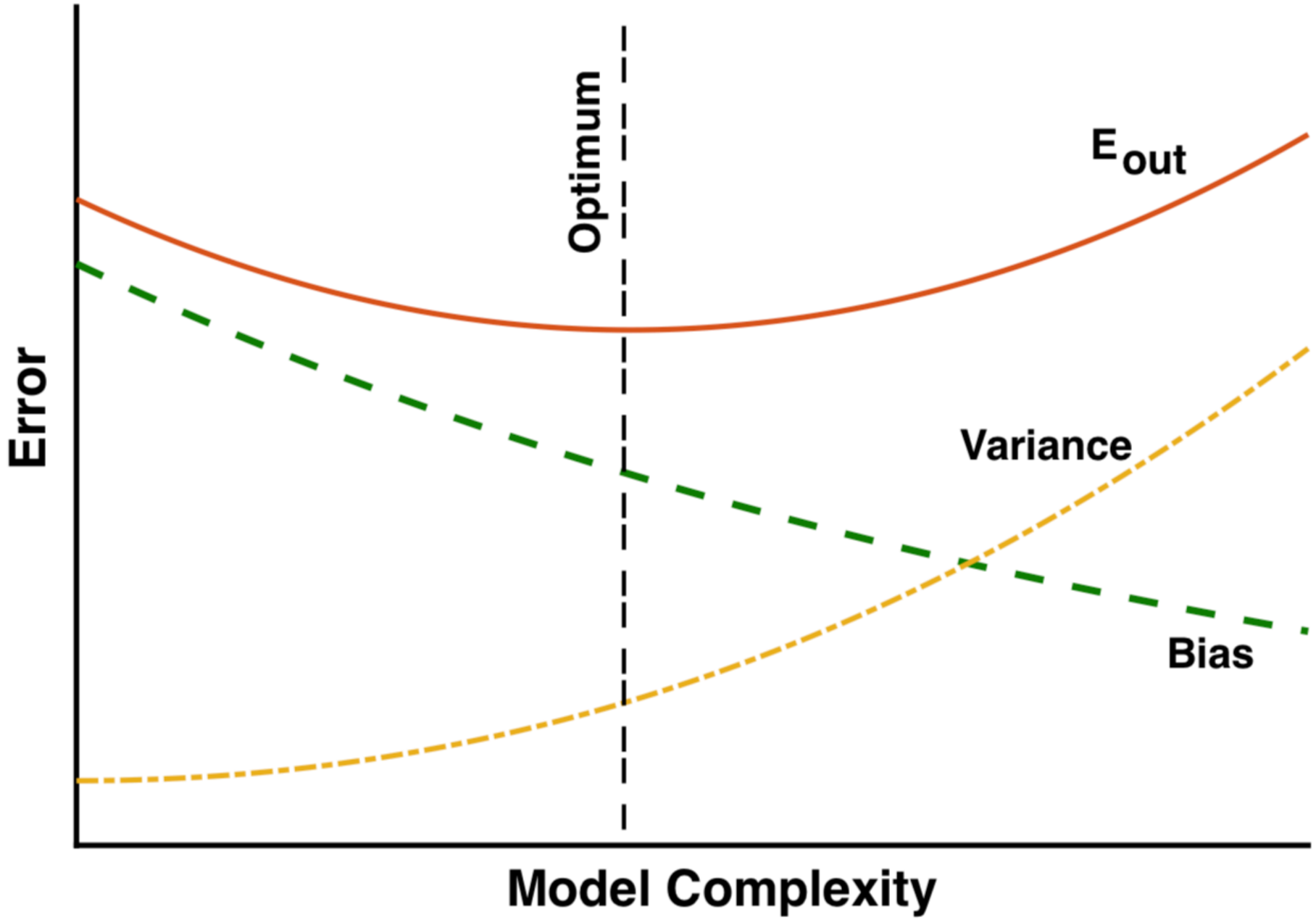
the more complex the model,
the smaller the bias

however a small bias
has usually associated large variance ….

the variance measures the errors introduced
in training our model due to sampling noise
In finite dataset

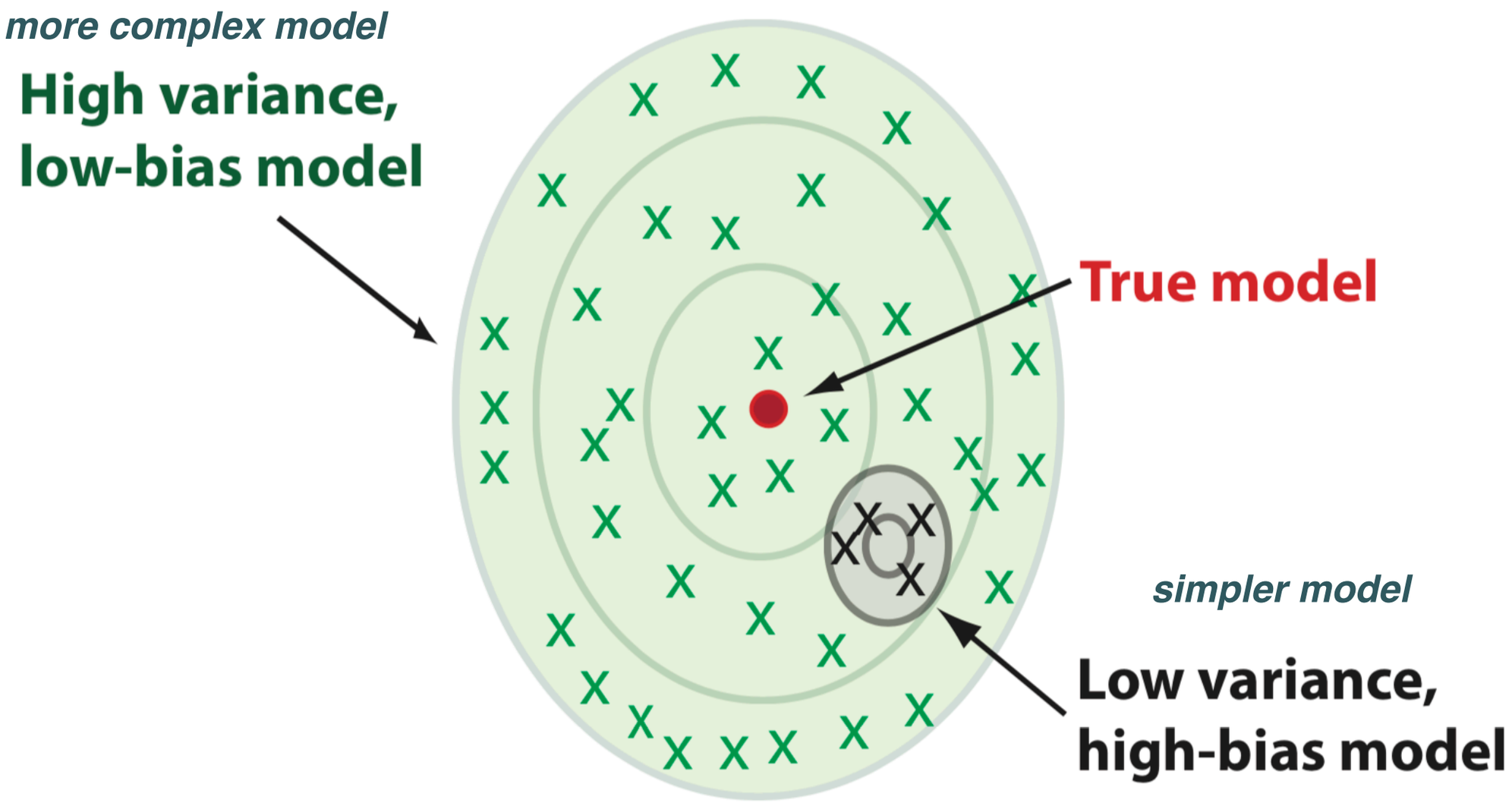
The bias-variance tradeoff

Optimal model performance (measured by minimising the generalisation error E_{val}) is typically achieved at intermediate levels of model complexity: the **bias-variance tradeoff**



The bias-variance tradeoff

Optimal model performance (measured by minimising the generalisation error E_{val}) is typically achieved at intermediate levels of model complexity: the **bias-variance tradeoff**



The bias-variance tradeoff

we can provide further insight on the bias-variance tradeoff with a simple scenario

assume a **model** extracted from a **training set** $\longrightarrow$ $\hat{y}(\boldsymbol{x}_{\rm tr})$

and that the true underlying law is given by

$$Y = y(\boldsymbol{X}) + \epsilon\,, \qquad y(\boldsymbol{x}_{\rm tr}) = E(Y, \boldsymbol{X} = \boldsymbol{x}_{\rm tr})$$

zero-mean, fixed variance noise

take a data point **outside the training set, *(x₀,y₀)***, and compute **its variance** in our model

$$E[\left(Y(x_0) - \hat{y}(x_0)\right)^2] = \left(y_0^2 - 2y_0 E[\hat{y}(x_0)] + E[\hat{y}^2(x_0)] + E[\epsilon^2]\right)$$

underlying law *model*

where we set to zero terms linear in the stochastic noise

The bias-variance tradeoff

we can provide further insight on the bias-variance tradeoff with a simple scenario

assume a **model** extracted from a **training set** $\longrightarrow$ $\hat{y}(\boldsymbol{x}_{\rm tr})$

and that the true underlying law is given by

$$Y = y(\boldsymbol{X}) + \epsilon\,, \qquad y(\boldsymbol{x}_{\rm tr}) = E(Y, \boldsymbol{X} = \boldsymbol{x}_{\rm tr})$$

zero-mean, fixed variance noise

take a data point **outside the training set, *(x_0,y_0)***, and compute **its variance** in our model

$$E[\left(Y(x_0) - \hat{y}(x_0)\right)^2] = \left(\text{Bias}[\hat{y}(x_0)]\right)^2 + \text{Var}[\hat{y}(x_0))] + \text{Var}(\epsilon)$$

$$\text{Bias}[\hat{y}(x_0))] = E[\hat{y}(x_0)] - y_0$$

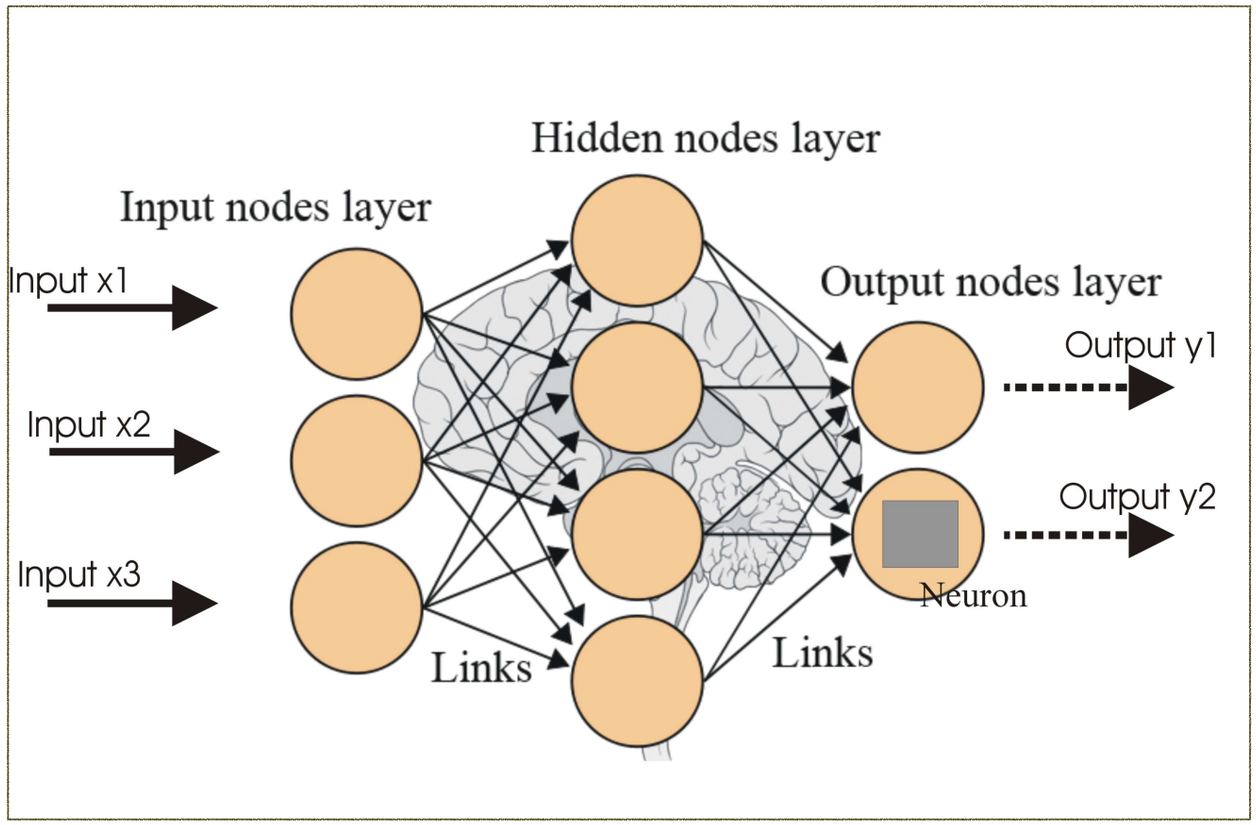
$$\text{Var}[\hat{y}(x_0))] = E[\hat{y}^2(x_0)] - E[\hat{y}(x_0)]^2$$

the **model generalisation power** decreases both with its bias and its variance

(Deep) Neural Networks

Artificial Neural Networks

Inspired by **biological brain models**, **Artificial Neural Networks** (ANNs) are mathematical algorithms designed to excel where domains as their evolution-driven counterparts outperforms traditional algorithms in tasks such as **pattern recognition, forecasting, classification**, ...



in ML context, ANN provide a flexible, powerful non-linear model for many problems

Neural Networks

Neural Nets can be defined as **neural-inspired nonlinear models** for supervised learning

- The basic unit of a NN is the **neuron,** a transformation of a set of *d* input features into a scalar output

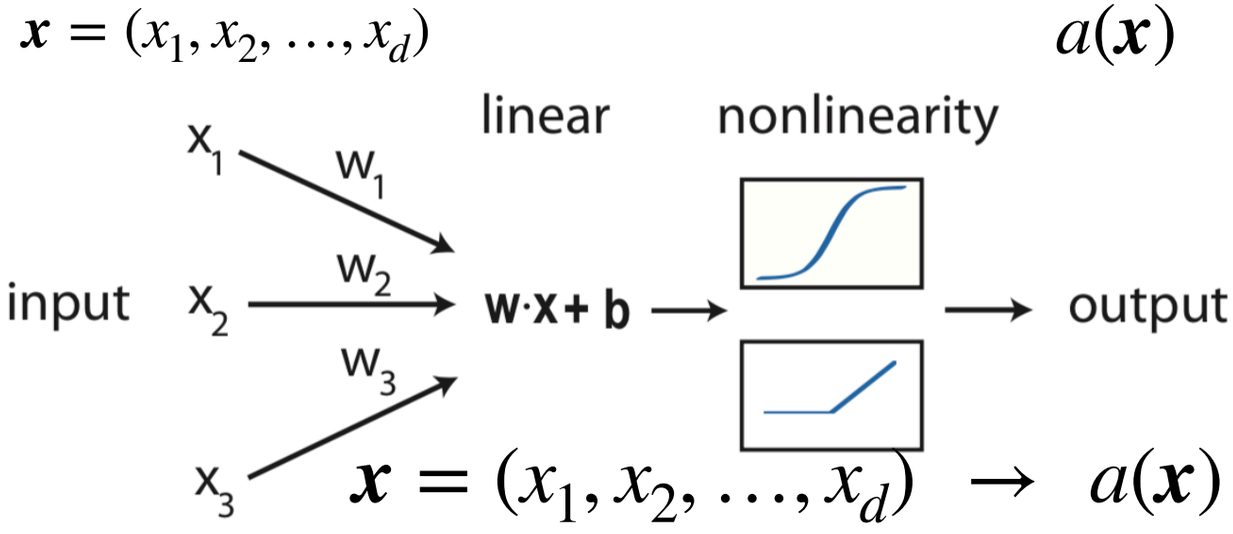
$$\boldsymbol{x} = (x_1, x_2, \ldots, x_d) \rightarrow a(\boldsymbol{x})$$

mapping

neuron inputs

neuron output (activation state)

biology analog: input are electric pulses, output the activation state of the neuron



Neural Networks

Neural Nets can be defined as **neural-inspired nonlinear models** for supervised learning

- The basic unit of a NN is the **neuron,** a transformation of a set of *d* input features into a scalar output

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_d) \rightarrow a(\boldsymbol{x})$$

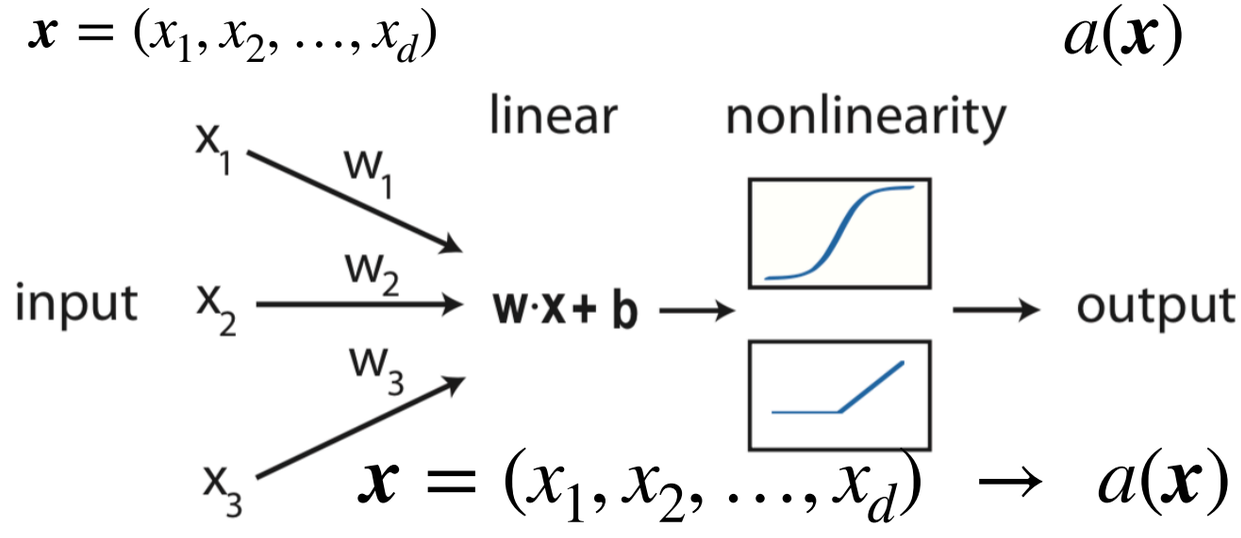
mapping

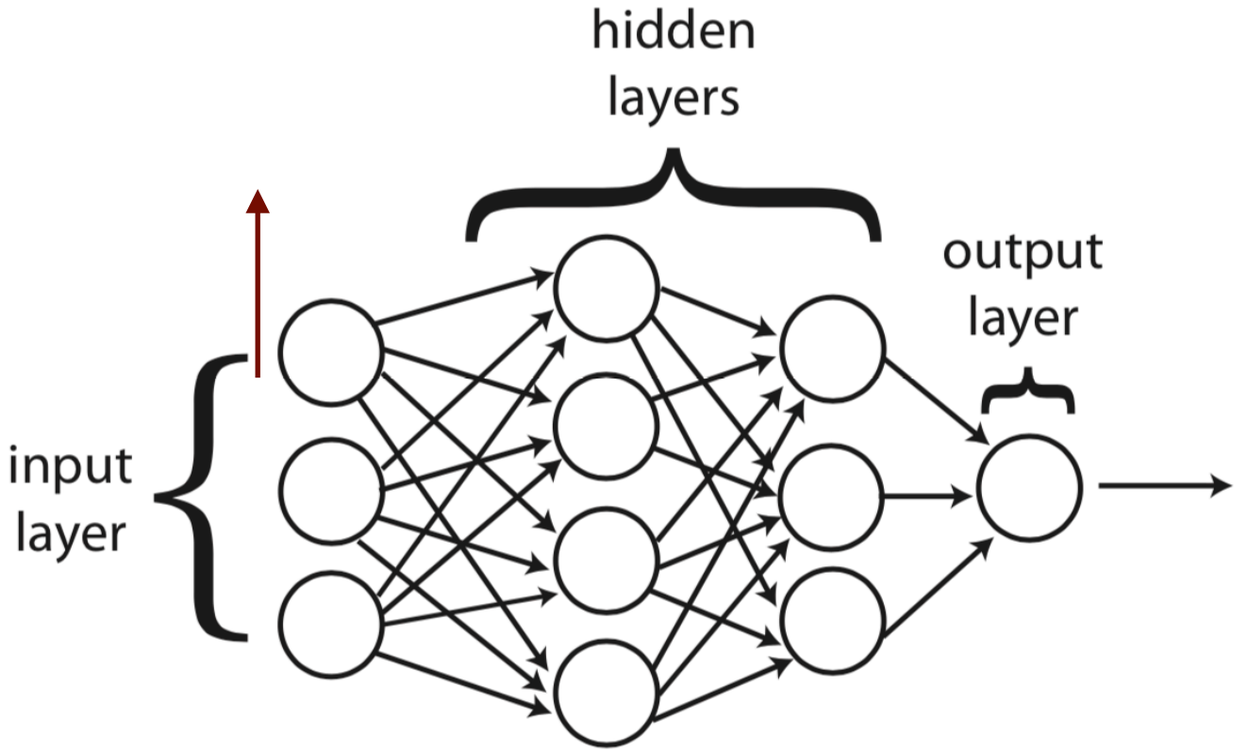
neuron inputs

neuron output (activation state)

- These neutrons are arranged in **layers**, which in turn are stacked on each other. The intermediate layers are called **hidden layers**

- Here we will focus on **feed-forward NNs,** where the output of the neurons of the previous layer becomes the input of the neurons in the subsequent layer





Neural Networks

Neural Nets can be defined as **neural-inspired nonlinear models** for supervised learning

the neural network output has two components

a **linear operation**, weighting the various inputs

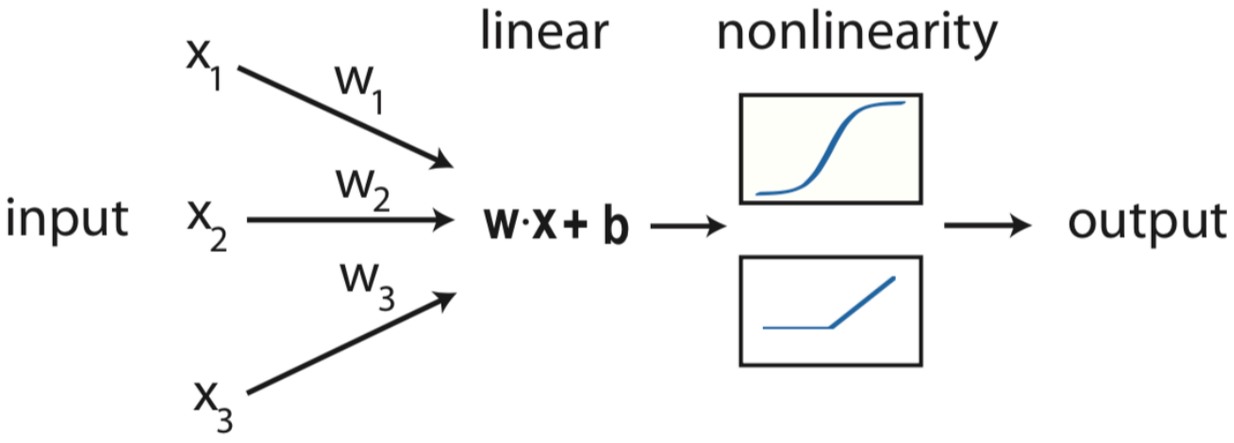
$$z^{(i)} = \boldsymbol{x}^T \cdot \boldsymbol{\theta}^{(i)} + \boldsymbol{\theta}_0^{(i)}$$

neuron inputs — *i-th neuron parameters* — *i-th neuron bias*

a **non-linear transformation**

$$a^{(i)}(\boldsymbol{x}) = \sigma_i(z^{(i)})$$

i-th neuron activation state — *i-th neuron activation function (non-linear)*

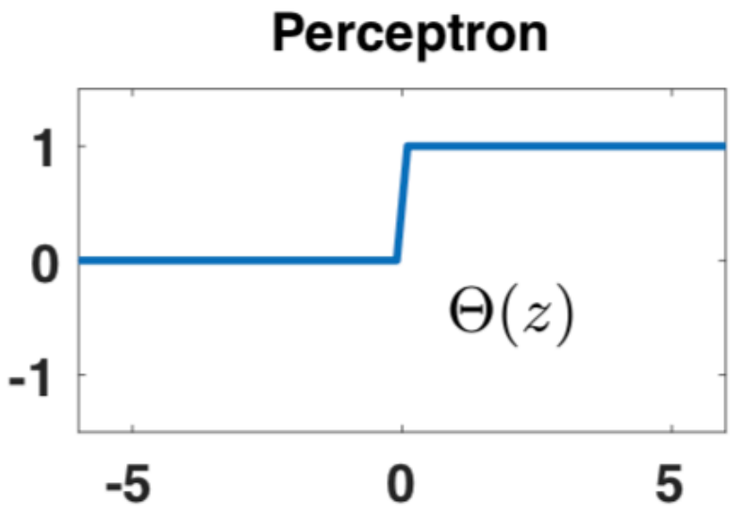


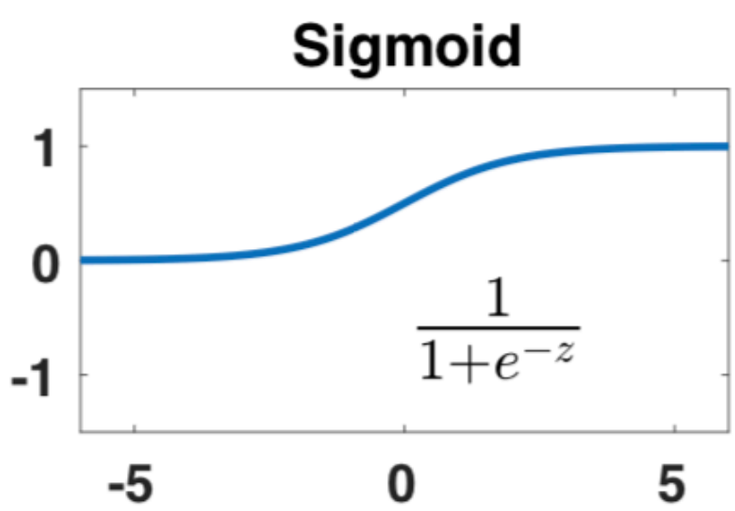
the parameters of NNs are often called ``weights" and ``thresholds"

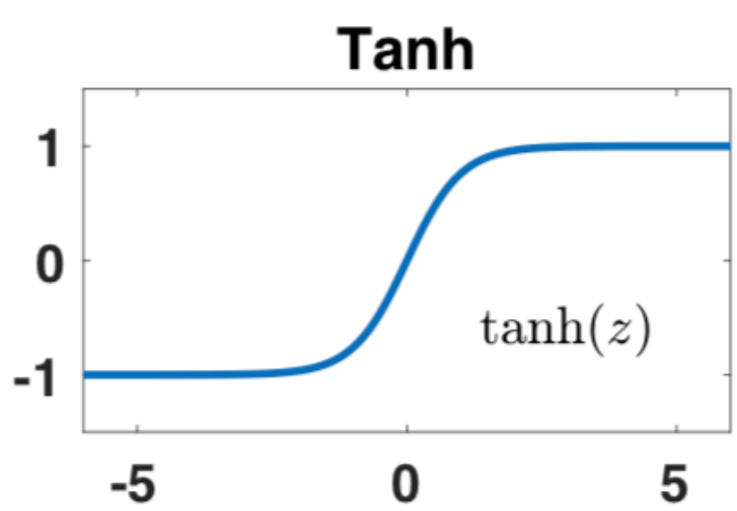
the choice of non-linear activation function affects the **computational and training properties** of the neural nets, since they modify the output gradients required for GD training

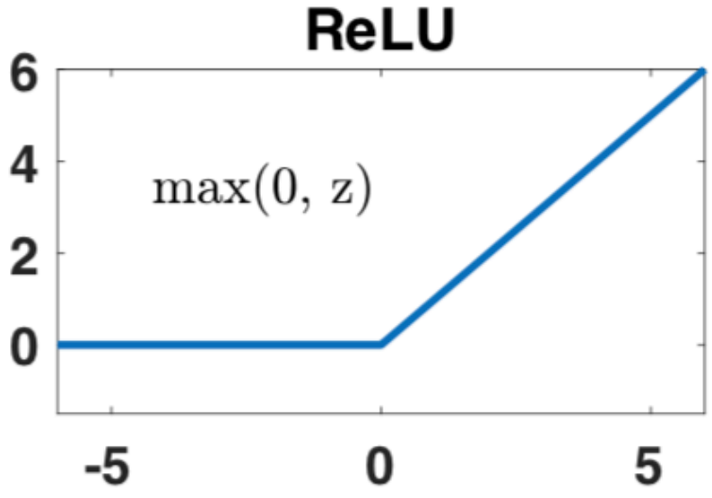
NN nonlinearly only via activation functions!

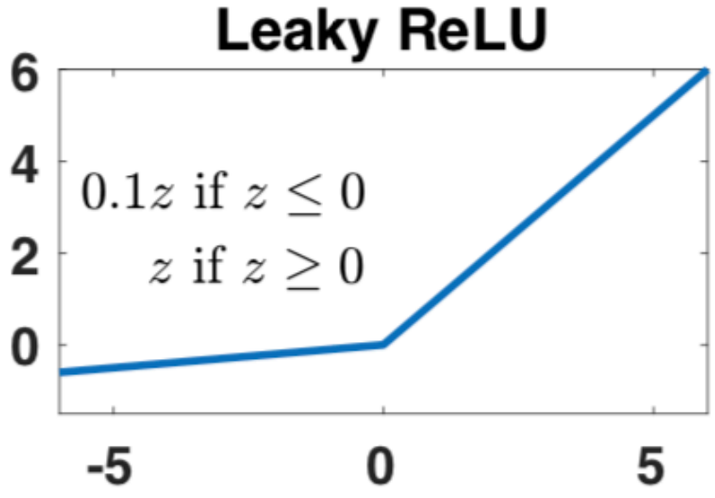
Activation functions

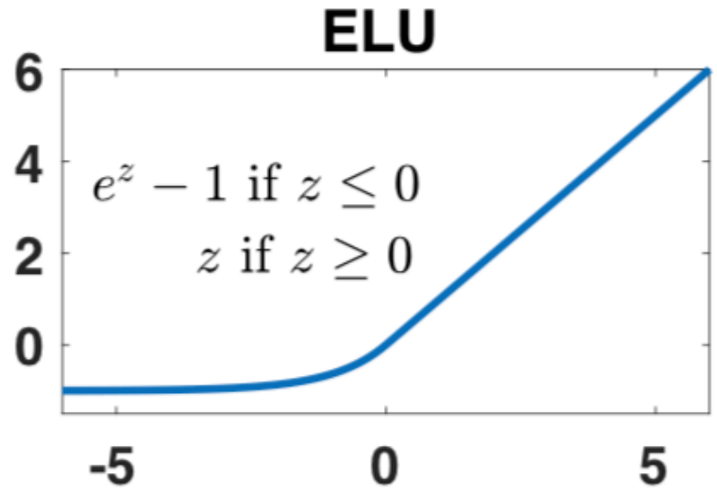






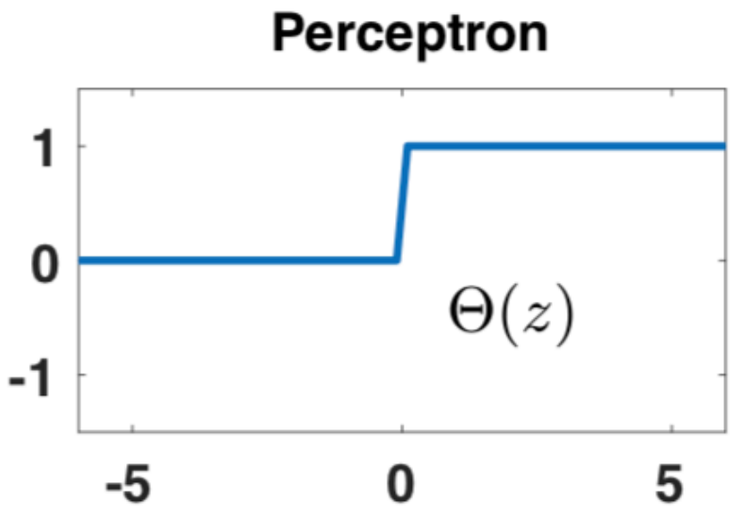


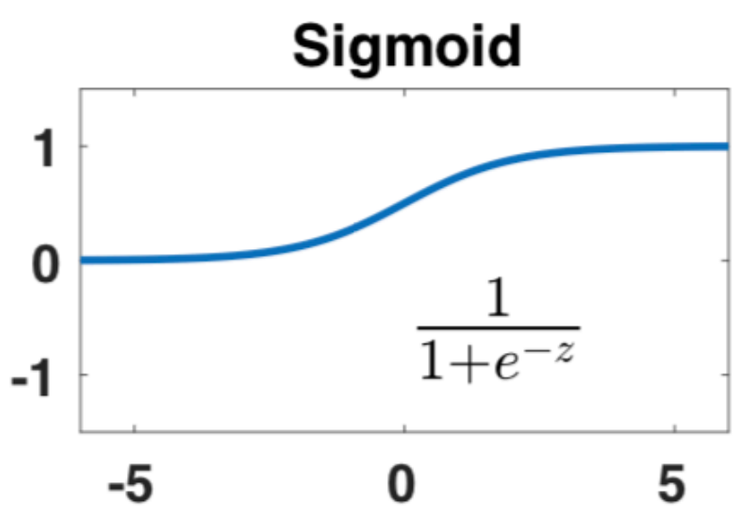


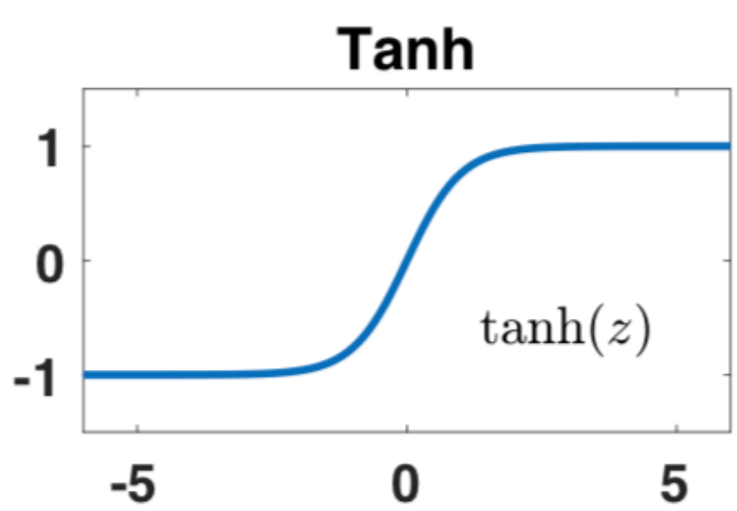


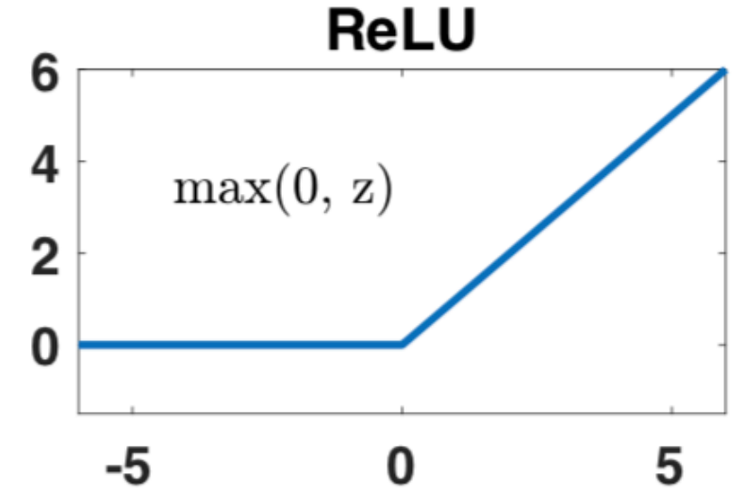
what is the main difference between these various activation functions?

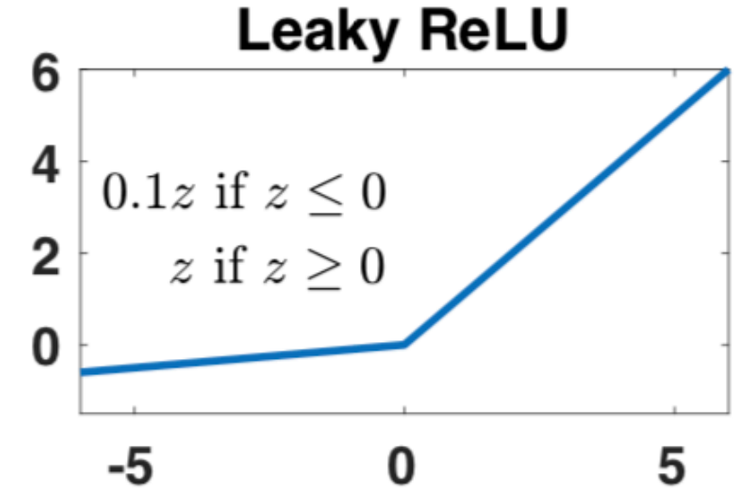
Activation functions

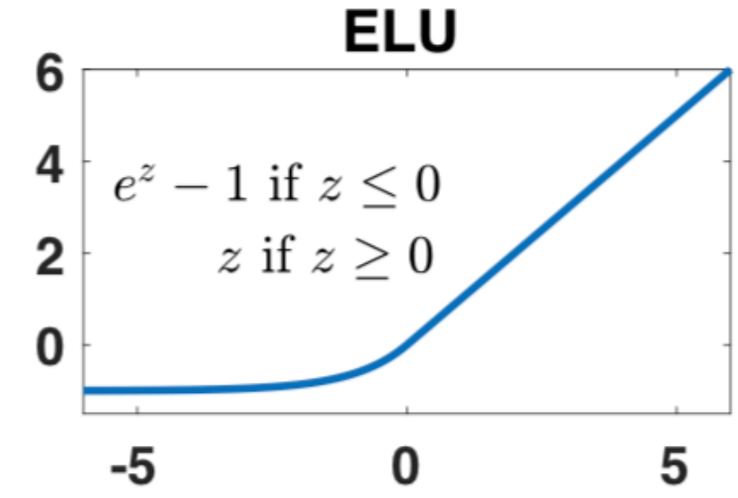












- Activation functions can be classified between those that **saturate at large inputs** (*e.g.* sigmoid) and those that **do not saturate at large inputs** (*e.g.* Rectified Linear Units ReLU)

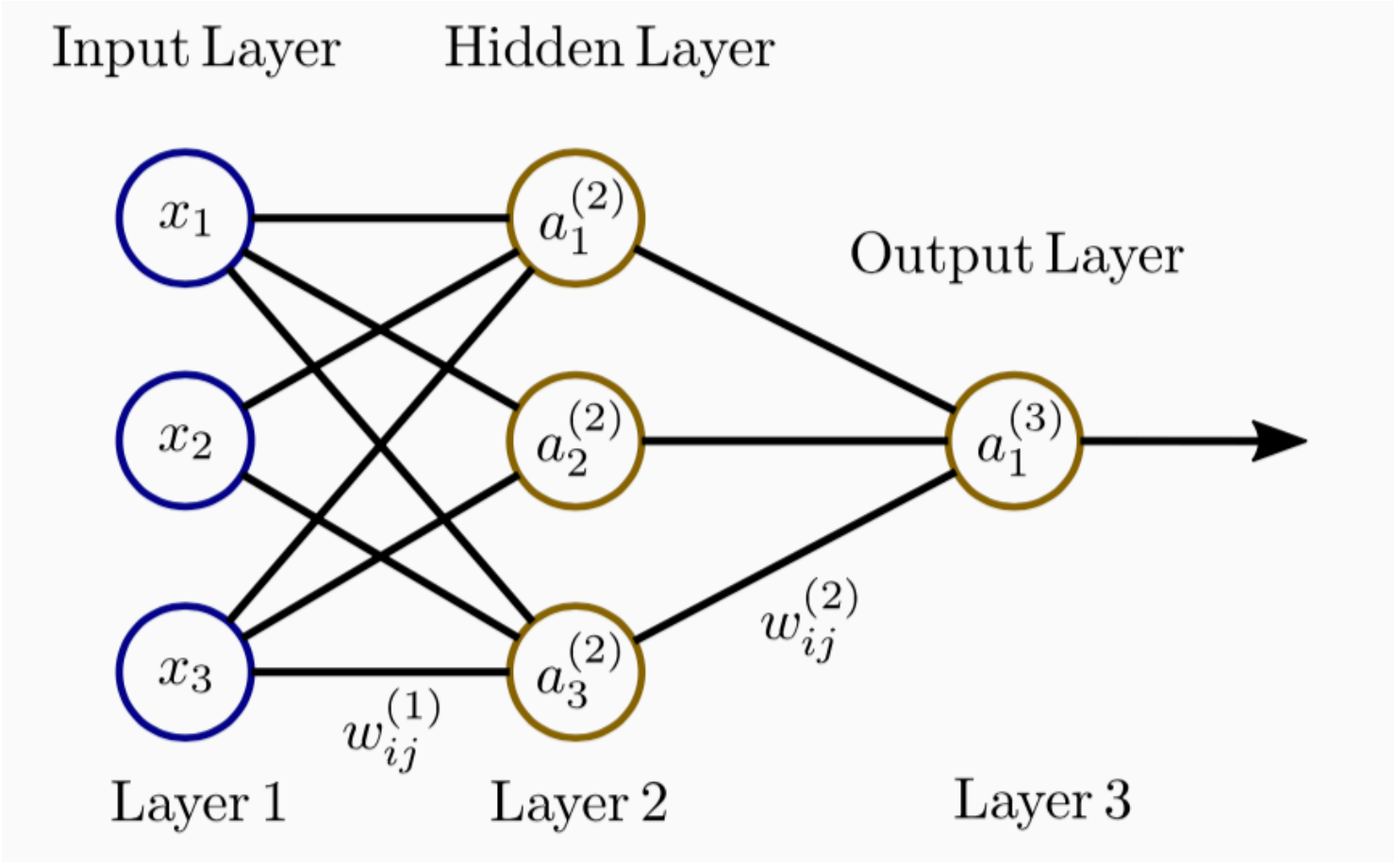
- The choice of non-linearities has important implications for **GD NN training methods:**

sigmoid $\longrightarrow \left.\frac{d\sigma}{dz}\right|_{z\gg 1} = 0$ ***ReLU*** $\longrightarrow \left.\frac{d\sigma}{dz}\right|_{z\gg 1} \neq 0$

vanishing gradients are problematic for deep networks: loss of sensitivity

A simple network

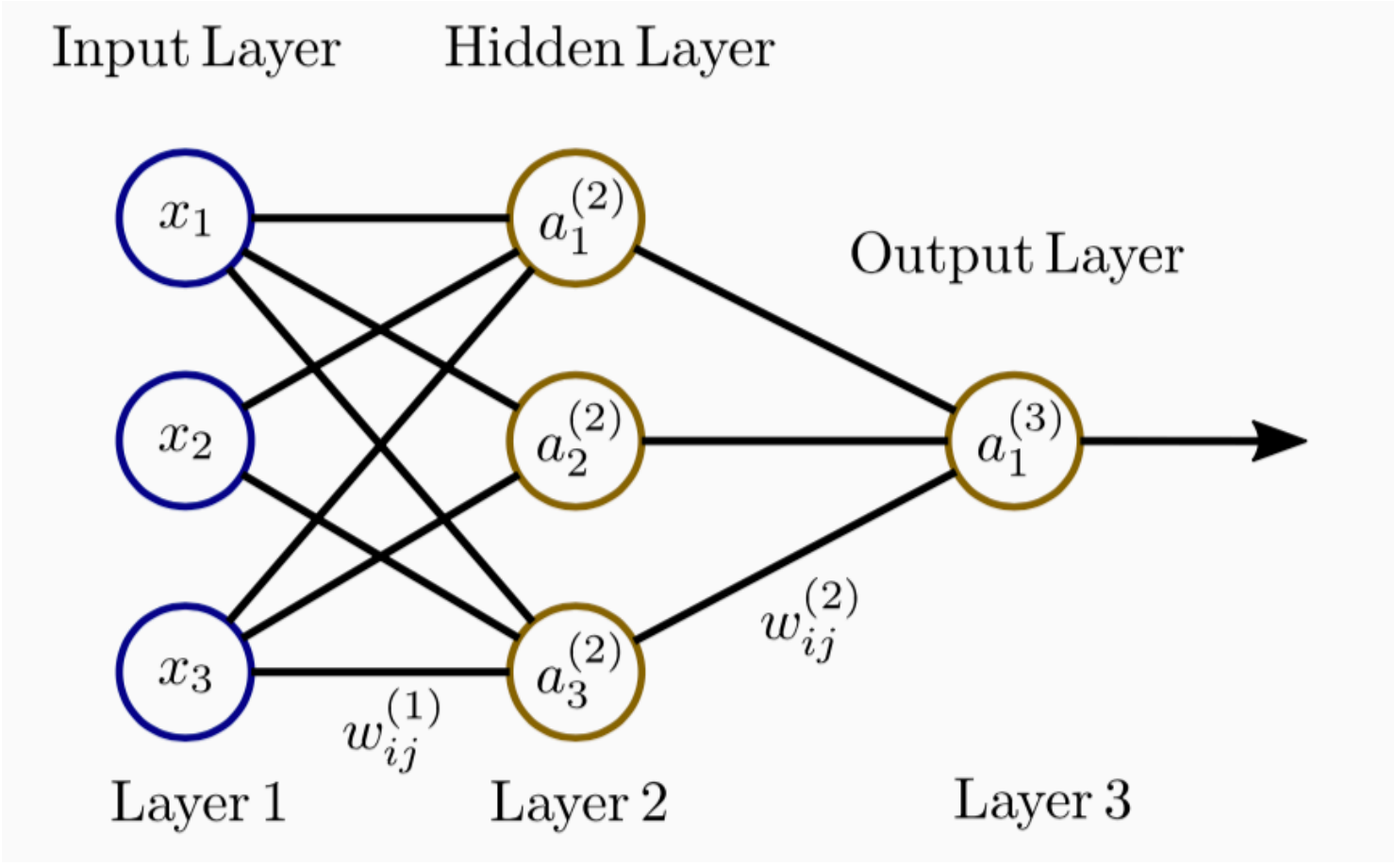
to realise that NNs are nothing mysterious but just a **complex non-linear mapping** between inputs and outputs, consider the explicit calculation of the output of a simple network



can we evaluate analytically $a_1^{(3)}$ as function of x_1, x_2, x_3?

A simple network

to realise that NNs are nothing mysterious but just a **complex non-linear mapping** between inputs and outputs, consider the explicit calculation of the output of a simple network



using the three inputs (Layer 1), compute activation states of neurons in Layer 2

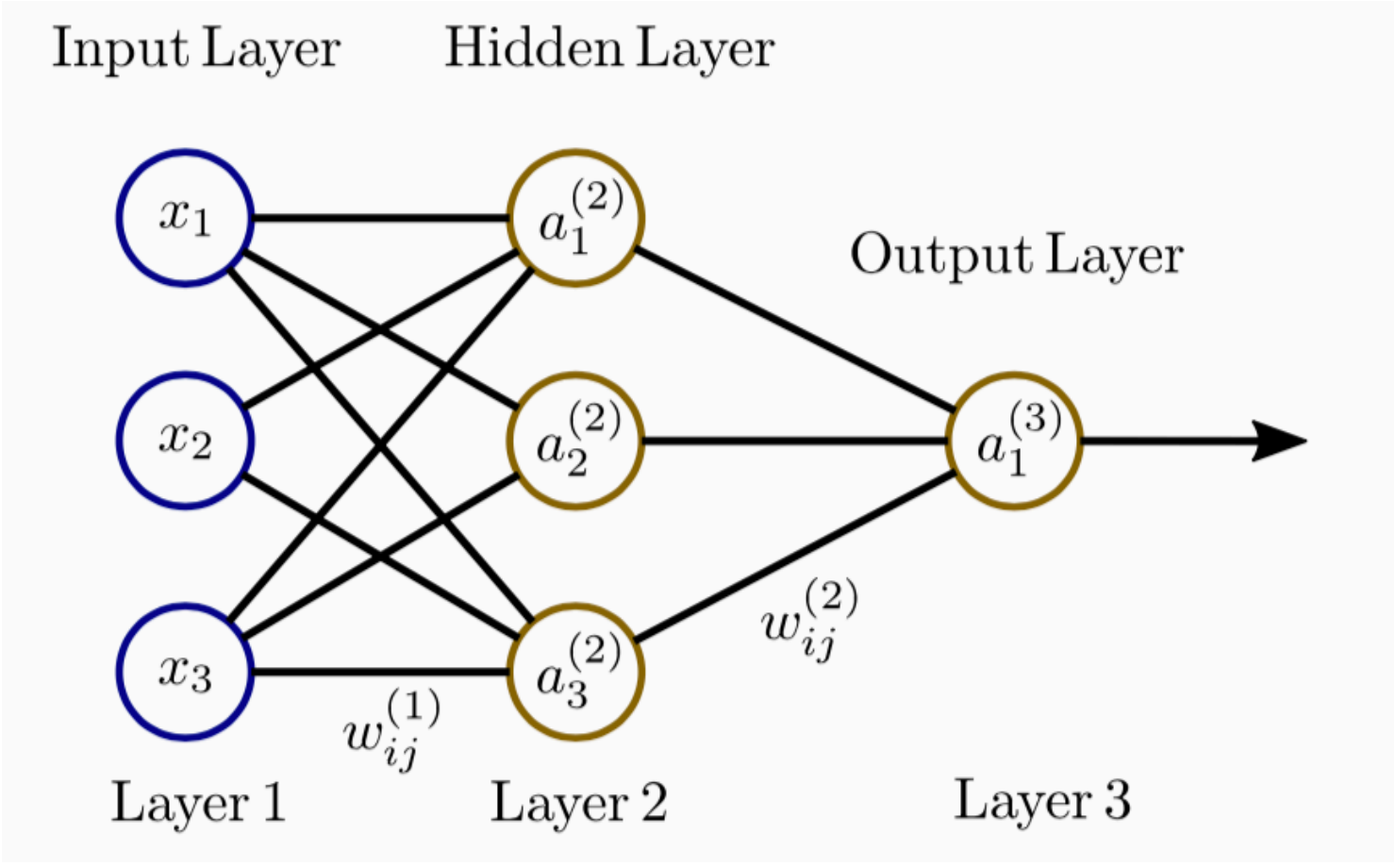
$$a_1^{(2)} = \sigma\left(x_1\theta_{11}^{(1)} + x_2\theta_{12}^{(1)} + x_3\theta_{13}^{(1)} + \theta_{10}^{(1)}\right)$$

$$a_2^{(2)} = \sigma\left(x_1\theta_{21}^{(1)} + x_2\theta_{22}^{(1)} + x_3\theta_{23}^{(1)} + \theta_{20}^{(1)}\right)$$

$$a_2^{(3)} = \sigma\left(x_1\theta_{31}^{(1)} + x_2\theta_{32}^{(1)} + x_3\theta_{33}^{(1)} + \theta_{30}^{(1)}\right)$$

A simple network

to realise that NNs are nothing mysterious but just a **complex non-linear mapping** between inputs and outputs, consider the explicit calculation of the output of a simple network



using the activation states of neurons in Layer 2, compute activation state of output neuron

$$a_1^{(3)} = \sigma\left(a_1^{(2)}\theta_{11}^{(2)} + a_2^{(2)}\theta_{12}^{(2)} + a_3^{(2)}\theta_{13}^{(2)} + \theta_{10}^{(2)}\right)$$

NN output is **analytical function** of the inputs and the model parameters

A simple network

to realise that NNs are nothing mysterious but just a **complex non-linear mapping** between inputs and outputs, consider the explicit calculation of the output of a simple network

$$a_1^{(3)} = \sigma\Bigg(\sigma\left(x_1\theta_{11}^{(1)} + x_2\theta_{12}^{(1)} + x_3\theta_{13}^{(1)} + \theta_{10}^{(1)}\right)\theta_{11}^{(2)} +$$

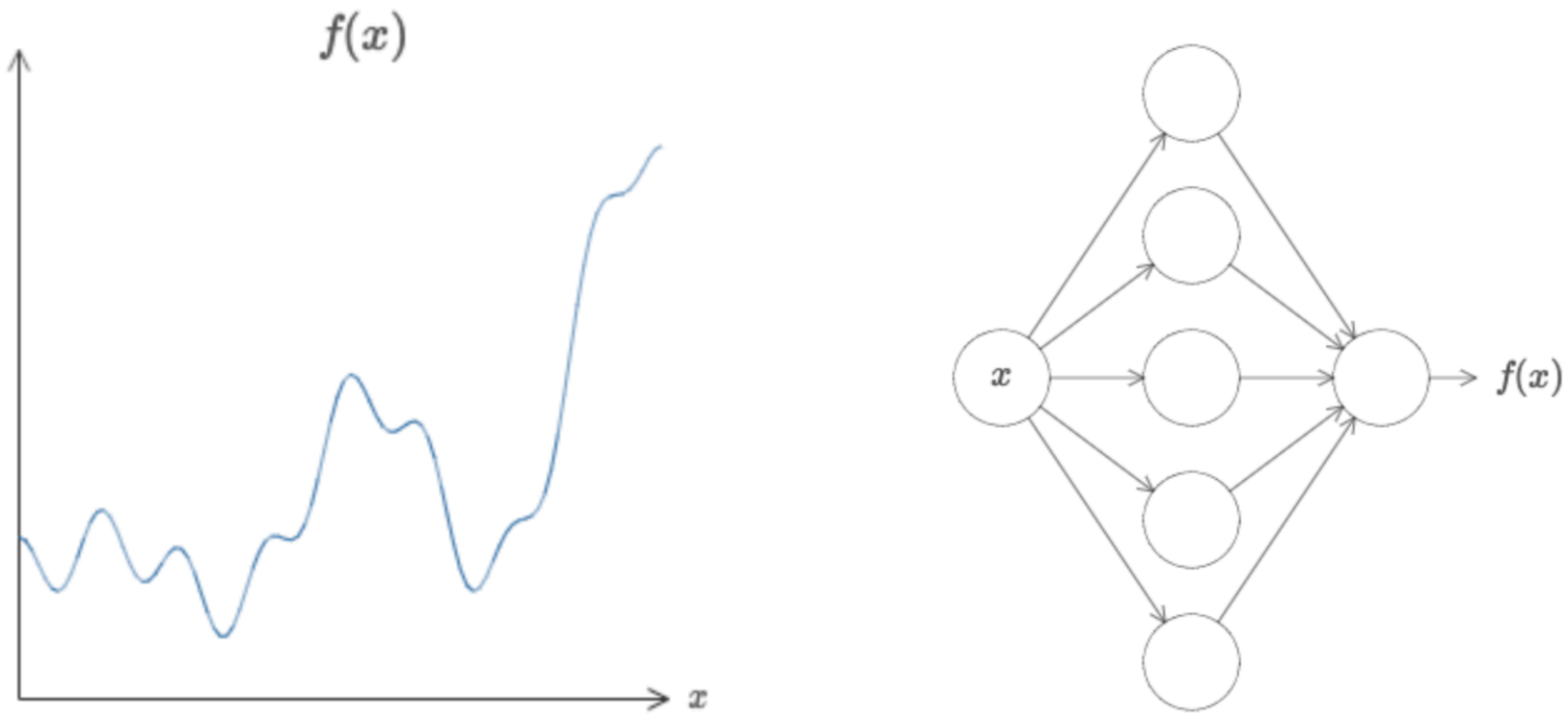
$$\sigma\left(x_1\theta_{21}^{(1)} + x_2\theta_{22}^{(1)} + x_3\theta_{23}^{(1)} + \theta_{20}^{(1)}\right)\theta_{12}^{(2)} +$$

$$\sigma\left(x_1\theta_{31}^{(1)} + x_2\theta_{32}^{(1)} + x_3\theta_{33}^{(1)} + \theta_{30}^{(1)}\right)\theta_{13}^{(2)} + \theta_{10}^{(2)}\Bigg)$$

substitute the activation function *e.g.* for the sigmoid and you have the analytic output of a simple NN

The Universal Approximation Theorem

theorem: a neural network with a single hidden layer and enough neurones can **approximate any continuous, multi-input/multi-output function** with arbitrary accuracy



neural networks exhibit *universality properties:* no matter what function we want to compute, we know (theorem!) that there is a neural network which can carry out this task

See M. Nielsen, *Neural Networks and Deep Learning*: http://neuralnetworksanddeeplearning.com/chap4.html

Deep Networks

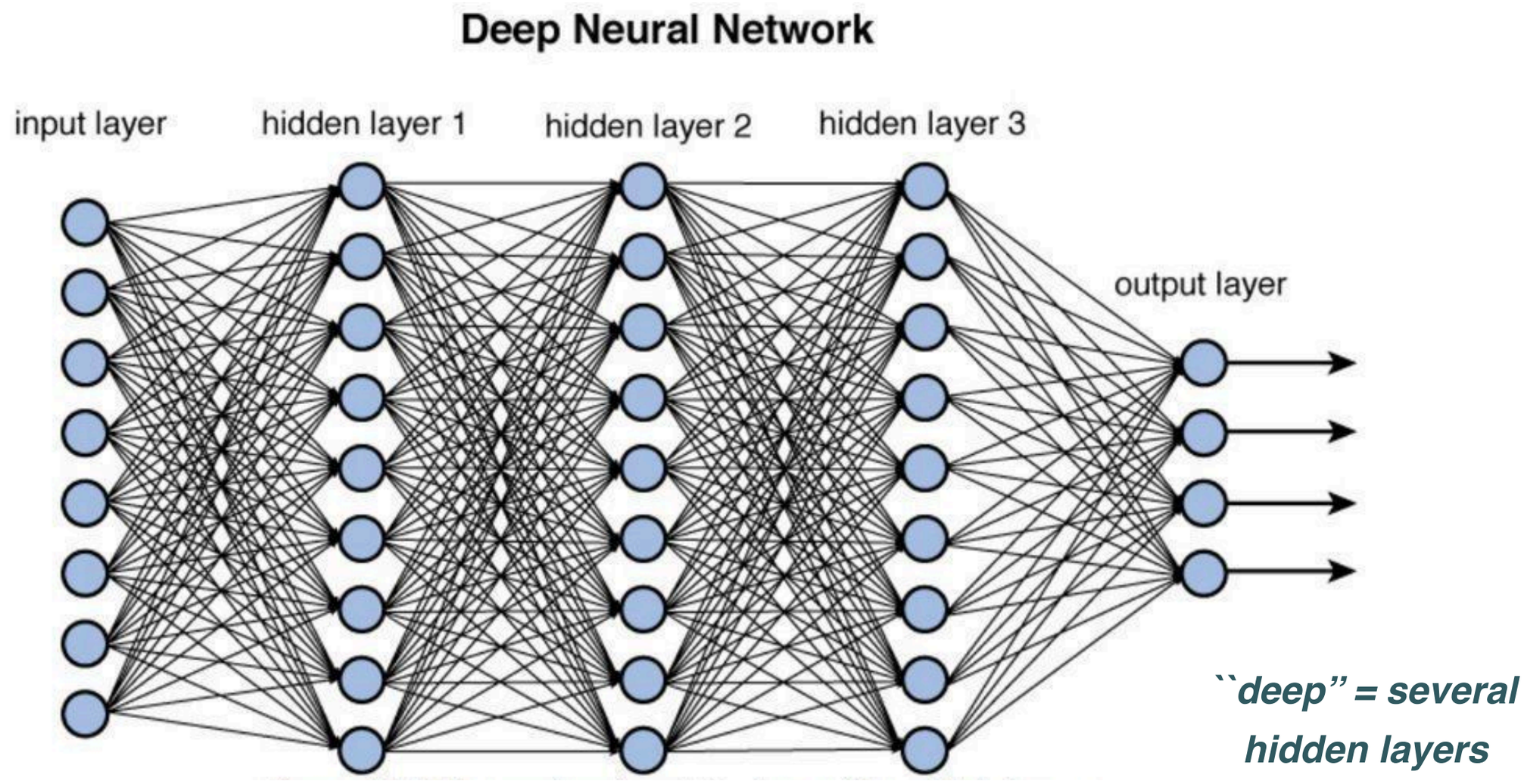
A neural network can thus be thought of a **complicated non-linear mapping** between the inputs and the outputs that depends on the parameters (weights and bias) of each neuron

What is Deep Learning? Can you guess now?

Deep Networks

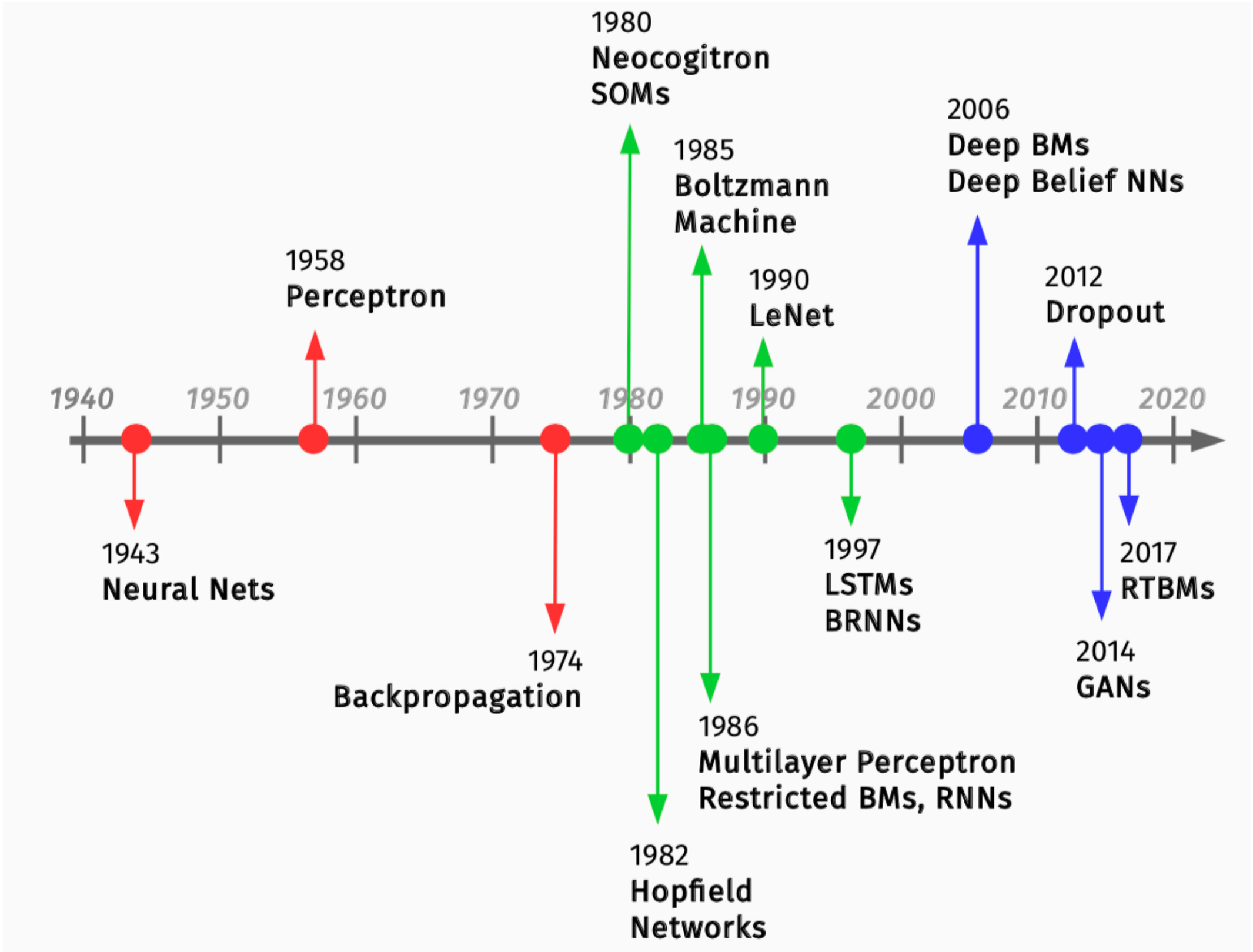
A neural network can thus be thought of a **complicated non-linear mapping** between the inputs and the outputs that depends on the parameters (weights and bias) of each neuron

We can make a NN **deep** by adding hidden layers, which greatly expands their **representational power**, also known as **expressivity**



``deep'' = several hidden layers

A timeline of neural networks



NN training

As standard in **Supervised Learning**, the first step to train a NN is to specify a **cost function**

$$\left(\boldsymbol{x}_i, y_i\right) \ , \quad i = 1,\ldots,n \longrightarrow \hat{y}_i\left(\boldsymbol{\theta}\right) \ , \quad i = 1,\ldots,n$$

for each of the n data points … *… the output of the NN provides the model prediction*

the loss function depends on whether the NN should provide **continuous or categorical** (discrete) predictions. For continuous data we can have the **mean square error**

$$E(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\left(\boldsymbol{\theta}\right)\right)^2$$

for categorical data we use the **cross-entropy**, which for binary (true/false) classification is

$$E(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i \ln \hat{y}_i(\boldsymbol{\theta}) + (1 - y_i)\ln(1 - \hat{y}_i(\boldsymbol{\theta}))\right)$$

where the true labels satisfy $y_i \in \{0,1\}$

NN training are based on a specific version of GD methods: **backpropagation**

Backpropagation

For deep NNs the brute-force evaluation of the **gradients of the cost function** is impractical

GD method:

$$\mathbf{v}_t = \eta_t \nabla_\theta E(\boldsymbol{\theta}_t)\,, \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$$

learning rate

gradient of cost function

*update of model parameters between iterations **t** and **t+1***

Why? What prevents us from using simple GD to train NNs?

Backpropagation

For deep NNs the brute-force evaluation of the **gradients of the cost function** is impractical

GD method:

$$\mathbf{v}_t = \eta_t \nabla_\theta E(\boldsymbol{\theta}_t)\,, \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$$

learning rate

gradient of cost function

*update of model parameters between iterations **t** and **t+1***

instead one uses **backpropagation**, which cleverly exploits the **layered structure** of NNs

we will use the following notation:

- A neural network with ***L* layers**, labelled as *l=1,…,L*
- $\omega_{jk}^{(l)}$**: weights** connecting *k*-th neuron in the *(l-1)*-th layer to *j*-th neutron in the *l*-th layer
- $b_j^{(l)}$ **: bias** of the *j*-th neuron in the *l*-th layer
- $a_j^{(l)}$**: activation state** of the *j*-th neuron in the *l*-th layer

Backpropagation

feed-forward NNs: the activation state of the *j*-th neuron in the *l*-th layer is a (nonlinear) function of the activation states of all the neutrons in the *(l-1)*-th layer

$$a_j^{(l)} = \sigma\left(\sum_{k=1}^{n_{l-1}} \omega_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}\right) \equiv \sigma(z_j^{(l)}), \quad j = 1,\ldots,n_l$$

the cost function of the NN therefore depends on:

- $a_j^{(L)}$: **activation states** of the neurons on the **output layer *L*** (directly)
- $a_j^{(l)}$: **activation states** of the neurons on the hidden layers ***l* <*L*** (indirectly)

we define the **error** associated to the *j*-th neuron in the output layer and hidden layers as

$$\Delta_j^{(L)} \equiv \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(L)}} \qquad \Delta_j^{(l)} \equiv \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(l)}} = \frac{\partial E(\boldsymbol{\theta})}{\partial a_j^{(l)}} \frac{da_j^{(l)}}{dz_j^{(l)}}$$

here assume that output layer is linear: ensures that model predictions are unbounded

Backpropagation

$$\Delta_j^{(L)} = \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(L)}} \qquad \Delta_j^{(l)} = \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(l)}} = \frac{\partial E(\boldsymbol{\theta})}{\partial a_j^{(l)}} \sigma'\left(z_j^{(l)}\right)$$

to derive the backpropagation equations, we will use the chain rule & NN layer structure

$$\Delta_j^{(l)} = \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \frac{\partial E(\boldsymbol{\theta})}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \Delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$

where here we exploit the **NN layered structure:** activation states of neurons in *(l+1)*-th layer depends only on activation states of neurons in *l*-th layer

$$\Delta_j^{(l)} = \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \Delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \left(\sum_{k}^{n_{l+1}} \Delta_k^{(l+1)} \omega_{kj}^{(l+1)} \right) \sigma'\left(z_j^{(l)}\right)$$

Backpropagation

$$\Delta_j^{(L)} = \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(L)}} \qquad \Delta_j^{(l)} = \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(l)}} = \frac{\partial E(\boldsymbol{\theta})}{\partial a_j^{(l)}} \sigma'\left(z_j^{(l)}\right)$$

to derive the backpropagation equations, we will use the chain rule & NN layer structure

$$\Delta_j^{(l)} = \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \frac{\partial E(\boldsymbol{\theta})}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \Delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$

where here we exploit the **NN layered structure:** activation states of neurons in *(l+1)*-th layer depends only on activation states of neurons in *l*-th layer

$$\Delta_j^{(l)} = \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \Delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \left(\sum_{k}^{n_{l+1}} \Delta_k^{(l+1)} \omega_{kj}^{(l+1)} \right) \sigma'\left(z_j^{(l)}\right)$$

$$z_k^{(l+1)} = \sum_{k'=1}^{n_l} \omega_{jk'}^{(l+1)} a_{k'}^{(l)} + b_j^{(l+1)} = \sum_{k'=1}^{n_l} \omega_{jk'} \sigma(z_{k'}^{l}) + b_j^{(l+1)}$$

Backpropagation

$$\Delta_j^{(L)} = \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(L)}} \qquad \Delta_j^{(l)} = \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(l)}} = \frac{\partial E(\boldsymbol{\theta})}{\partial a_j^{(l)}} \sigma'\left(z_j^{(l)}\right)$$

to derive the backpropagation equations, we will use the chain rule & NN layer structure

$$\Delta_j^{(l)} = \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \frac{\partial E(\boldsymbol{\theta})}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \Delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$

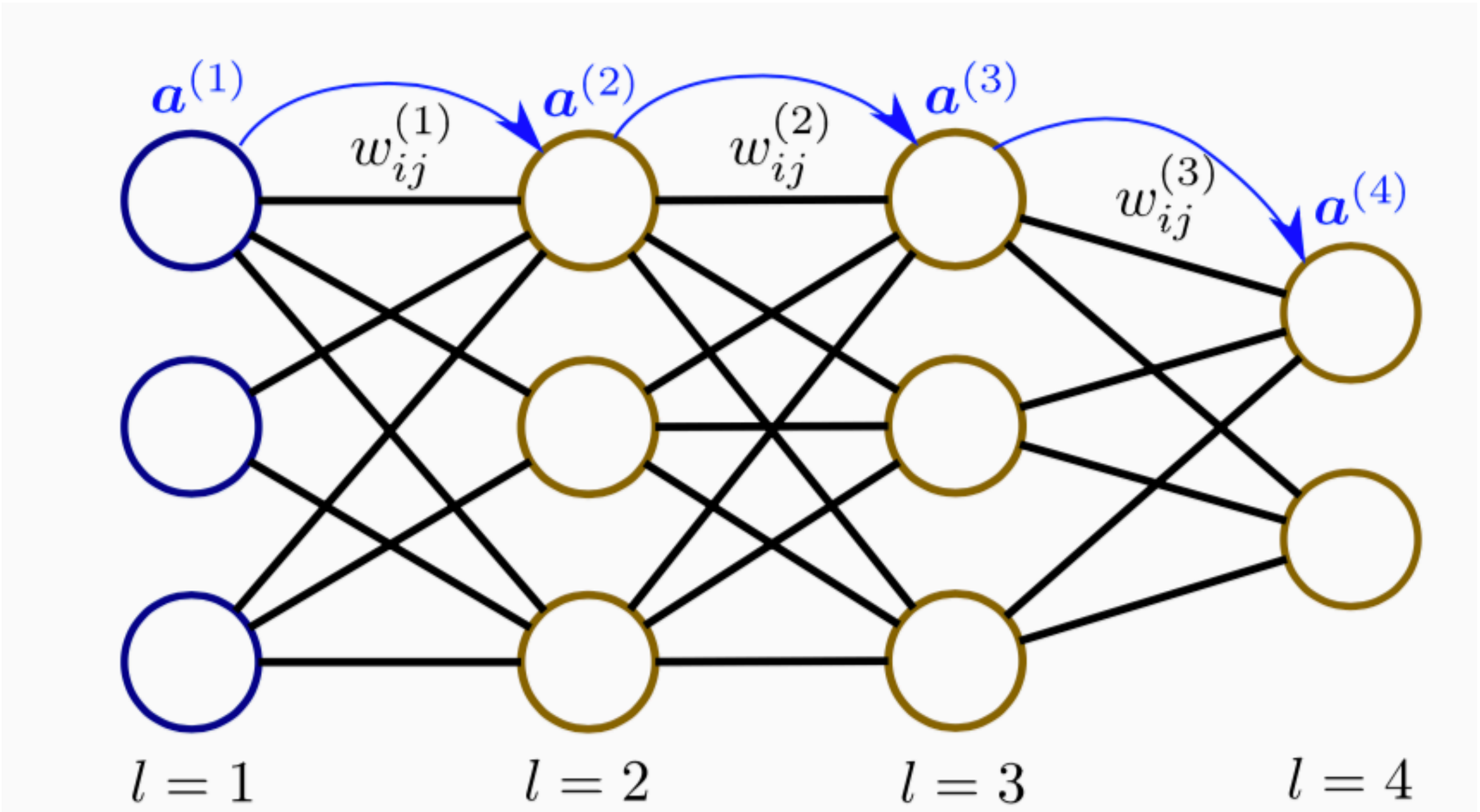
where here we exploit the **NN layered structure:** activation states of neurons in *(l+1)*-th layer depends only on activation states of neurons in *l*-th layer

$$\Delta_j^{(l)} = \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \Delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \left(\sum_{k}^{n_{l+1}} \Delta_k^{(l+1)} \omega_{kj}^{(l+1)} \right) \sigma'\left(z_j^{(l)}\right)$$

$$\frac{\partial E}{\partial \omega_{jk}^{(l)}} = \frac{\partial E}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial \omega_{jk}^{(l)}} = \Delta_j^{(l)} a_k^{(l-1)}$$

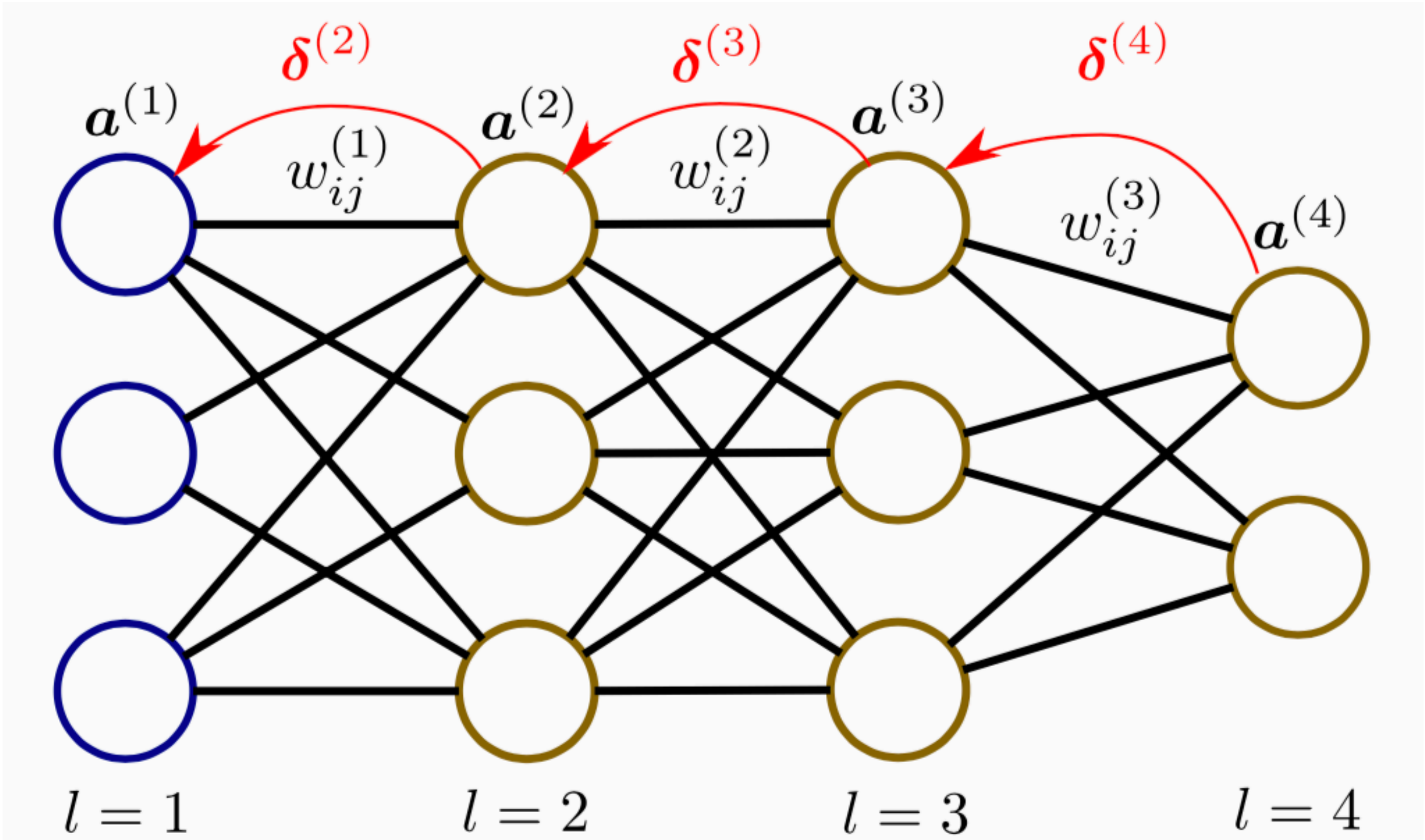
Backpropagation

first evaluate the activation states of all neurons using **forward propagation …**



Backpropagation

then **backpropagate** to evaluate the **errors Δ** and thus the gradients over model parameters



Backpropagation

we now have all the ingredients to deploy the **backpropagation algorithm for NN training**

- ***(1)*** Evaluate activation state of neurons in input layer $a_j^{(1)}, \quad j = 1,\ldots,n_1$

- ***(2)*** **Feed-forward:** evaluate activation states for each subsequent layer

$$a_j^{(l)} = \sigma\left(\sum_{k=1}^{n_l} \omega_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}\right) \equiv \sigma(z_j^{(l)}) \quad , l = 2,\ldots,L$$

- ***(3)*** With this information evaluate error on network's output layer

$$\Delta_j^{(L)} = \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(L)}}$$

here enters dependence on cost function

- ***(4)*** **Backpropagate** this error to evaluate the errors on all hidden layers

$$\Delta_j^{(l)} = \left(\sum_{k}^{n_{l+1}} \Delta_k^{(l+1)} \omega_{kj}^{(l+1)}\right) \sigma'\left(z_j^{(l)}\right)$$

error in j-th neuron in (l)-th layer

error in k-th neuron in (l+1)-th layer

Backpropagation

we now have all the ingredients to deploy the **backpropagation algorithm for NN training**

- ***(5)*** Evaluate **gradients of the model parameters** associated to the gradient of cost function

$$\frac{\partial E}{\partial \omega_{jk}^{(l)}} = \Delta_j^{(l)} a_k^{(l-1)} \qquad \frac{\partial E}{\partial b_j^{(l)}} = \Delta_j^{(l)}$$

extremely efficient way of calculating the gradients of the model parameters, requiring only a **single forward and backward pass** of the neural network

$$\Delta_j^{(L)} = \frac{\partial E(\boldsymbol{\theta})}{\partial z_j^{(L)}}$$

at most j numerical derivatives need to be evaluated, even in NNs with millions of parameters

Backpropagation

we now have all the ingredients to deploy the **backpropagation algorithm for NN training**

- ***(5)*** Evaluate **gradients of the model parameters** associated to the gradient of cost function

$$\frac{\partial E}{\partial \omega_{jk}^{(l)}} = \Delta_j^{(l)} a_k^{(l-1)} \qquad \frac{\partial E}{\partial b_j^{(l)}} = \Delta_j^{(l)}$$

extremely efficient way of calculating the gradients of the model parameters, requiring only a **single forward and backward pass** of the neural network

with this info one can carry out with Gradient Descent and **update the model parameters**

$$\mathbf{v}_t = \eta_t \nabla_\theta E(\boldsymbol{\theta}_t)\,, \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$$

note that even with BP training of large (deep) networks can be **computationally intensive!**

Further complication if cost function **depends non-trivially on NN output**

$$E(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \left(\mathscr{F}_i^{(\mathrm{dat})} - \mathscr{F}_i^{(\mathrm{model})}(\hat{\boldsymbol{y}}; \boldsymbol{\theta}) \right)^2$$

e.g. model requires integral of NN output

NN initialisation

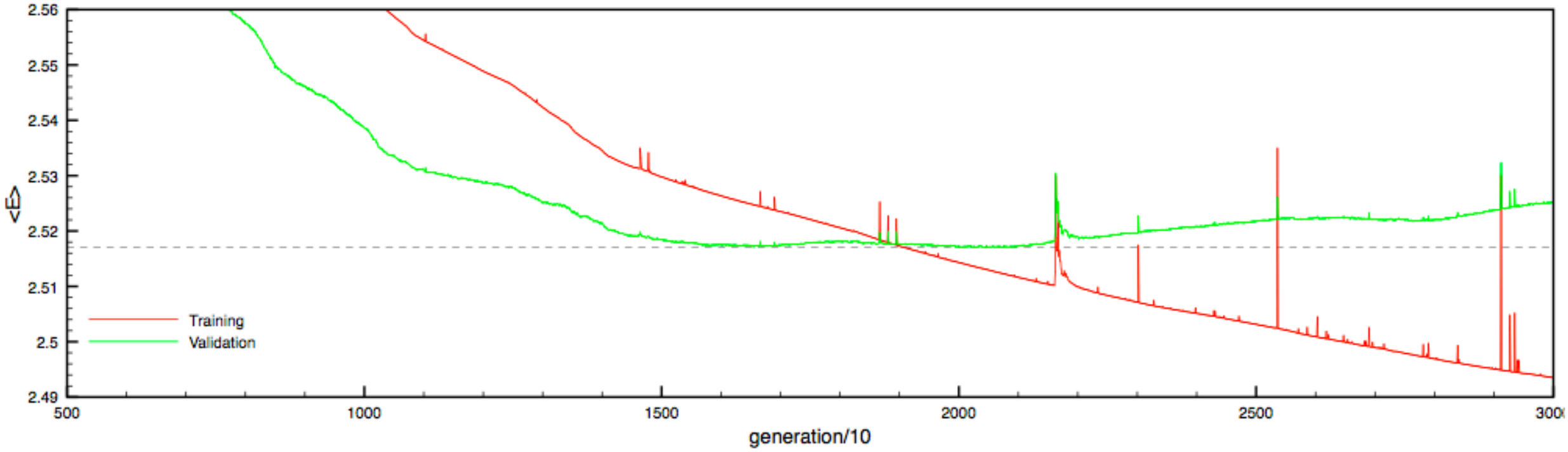
A further subtlety concerning NN training is that sometimes **a clever initialisation of the model parameters** helps in the learning process

- **zero:** all weights set to zero (initial complexity equivalent to single neuron)
- **random:** breaks parameter symmetry
- **glorot/xavier:** weights distributed randomly with Gaussian with variance based on in/out size of the neutron
- **he:** random initialisation avoiding the saturation region of the activation function

in many cases trial-and-error required to assess what works best

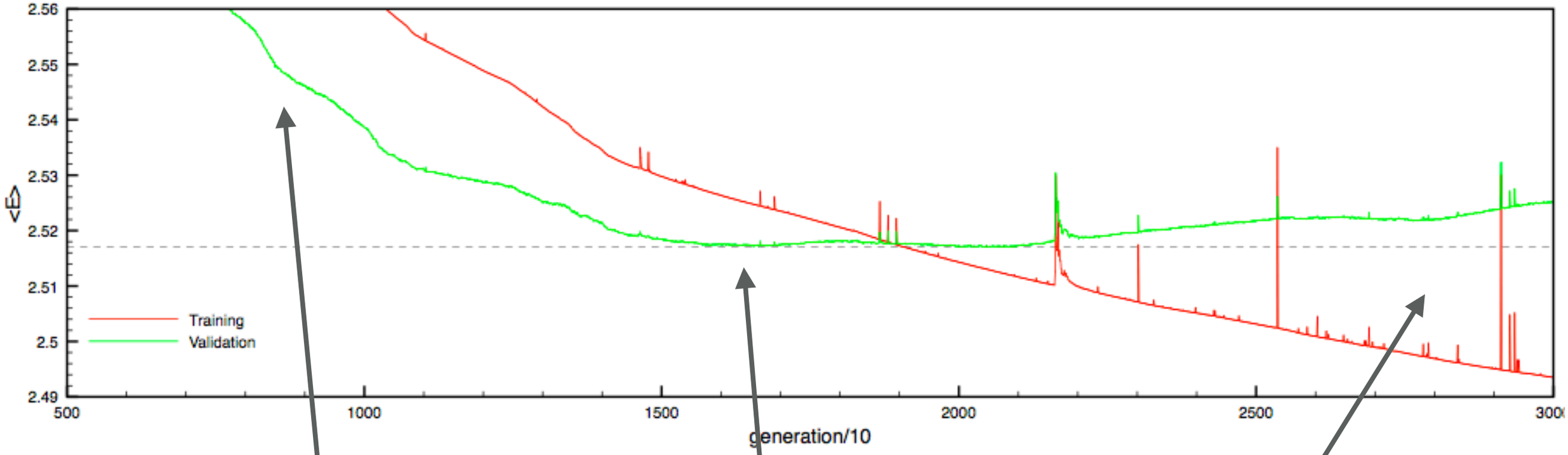
NN regularisation: cross-validation

Neural networks provide extremely flexible models to describe complex datasets, but one should **avoid overfitting**, else the model will be **unable to generalise**



NN regularisation: cross-validation

Neural networks provide extremely flexible models to describe complex datasets, but one should **avoid overfitting**, else the model will be **unable to generalise**



underlearning

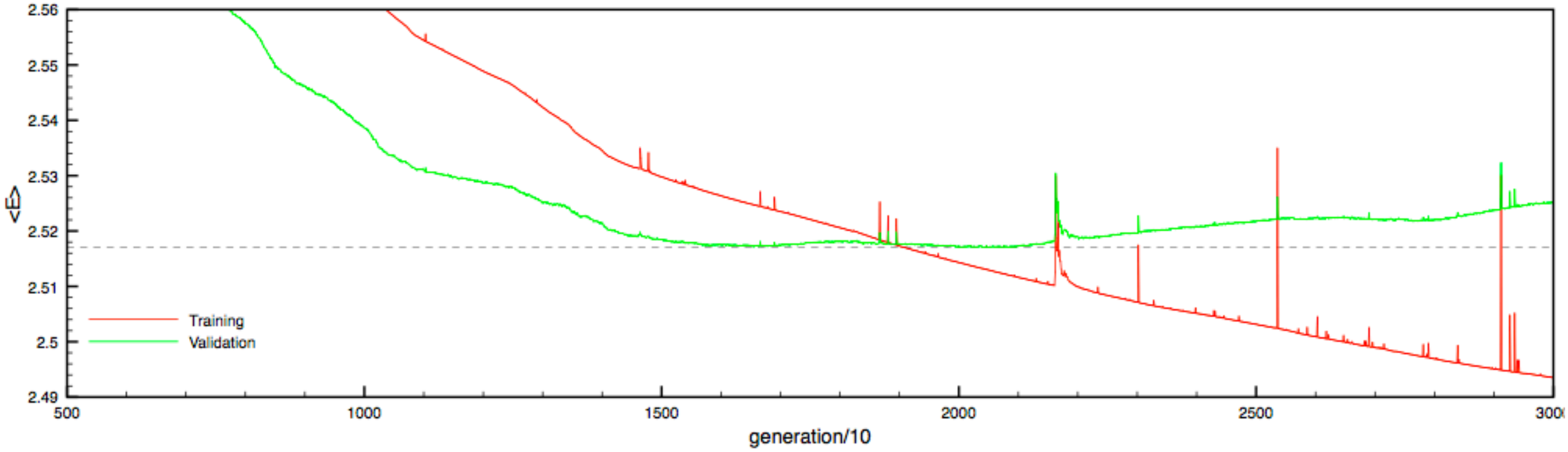
Optimal stopping

overfitting

recall the bias/variance tradeoff!

NN regularisation: cross-validation

Neural networks provide extremely flexible models to describe complex datasets, but one should **avoid overfitting**, else the model will be **unable to generalise**

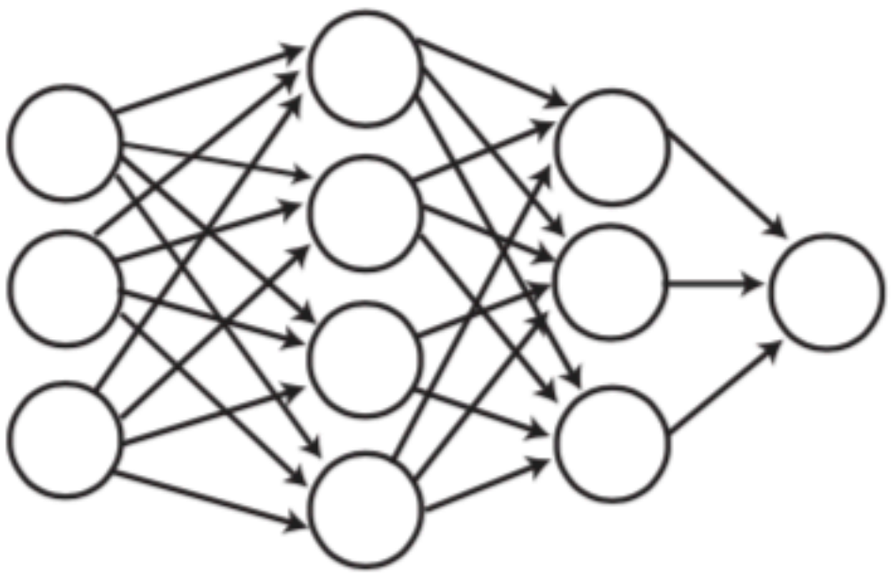


Cross-validation look-back stopping: train the NN for a large fixed number of iterations. Then look back and determine the point at the training where the out-of-sample error was the smallest. Take as best-fit NN parameters those corresponding to that point of the training

NN regularisation: Dropout

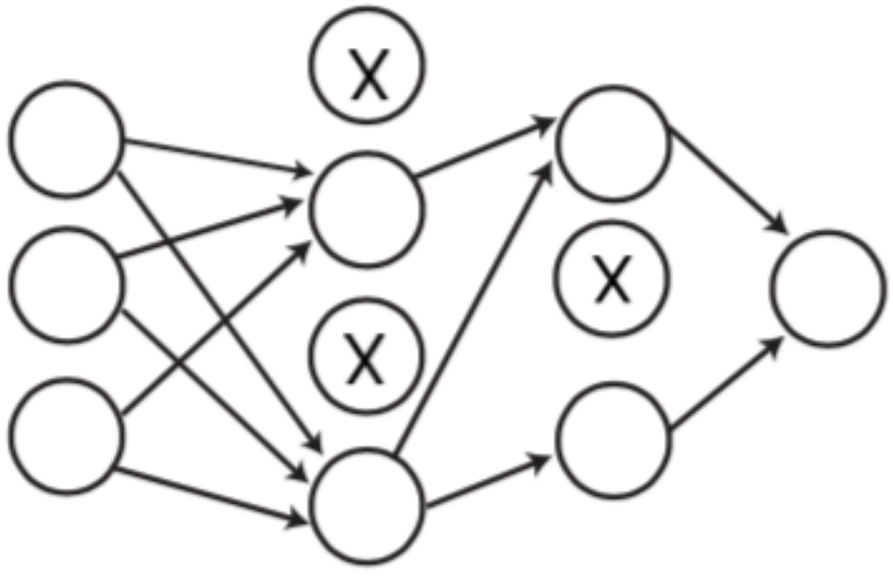
- Dropout is one of the standard **regularisation procedures** that aim to avoid overfitting when training deep NNs

Standard Neural Net



- During the training procedure, neurons are randomly "**dropped out**" of the neural network with some probability *p* giving rise to a thinned network, and the GD gradients are computed only there

After applying Dropout



- Dropout prevents overfitting by **reducing correlations among neurons** and reducing the variance

effective reduction of model complexity

Hyperoptimisation

In most Machine Learning applications, the model has several parameters which are typically **adjusted by hand** (trial and error) rather than algorithmically:

- Network architecture: number of layers of neurons per layer, activation functions, …
- Choice of minimiser (which of the GD variants?)
- Learning rate, momentum, memory, size of mini-batches, ….
- Regulariation parameters, stopping, dropout rate, patience, …

one can avoid the need of subjective choice by means of **an hyperoptimisation procedure**, where all model and training/stopping parameters are determined algorithmically

Such hyperoptimisation requires introducing a **reward function** to grade the model. Note that this is different from the **cost function:** the latter is optimised separately model by model (e.g. for each NN architecture) while the former compares between all optimised models

e.g. cost function $C = E_{\text{tr}}$ $R = E_{\text{val}}$ ***reward function***

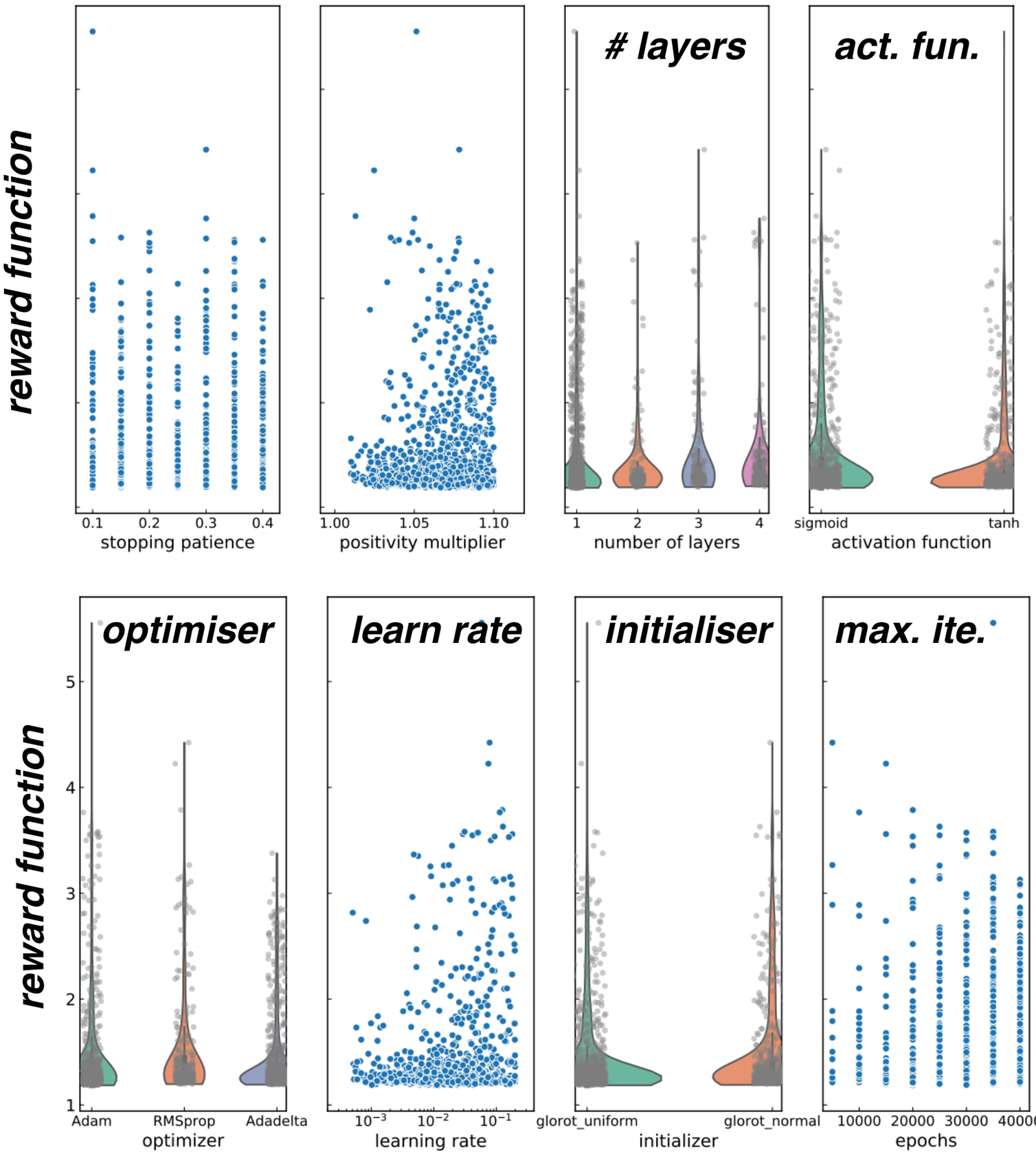
Hyperoptimisation

- In a hyperparameter scan one can compare the performance of **hundreds or thousands** of parameter combinations

- Some choices are **discrete** (type of minimiser, # of layers) others are **continuous** (learning rate)

- One can also **visualise** which choices are more crucial and which ones less important

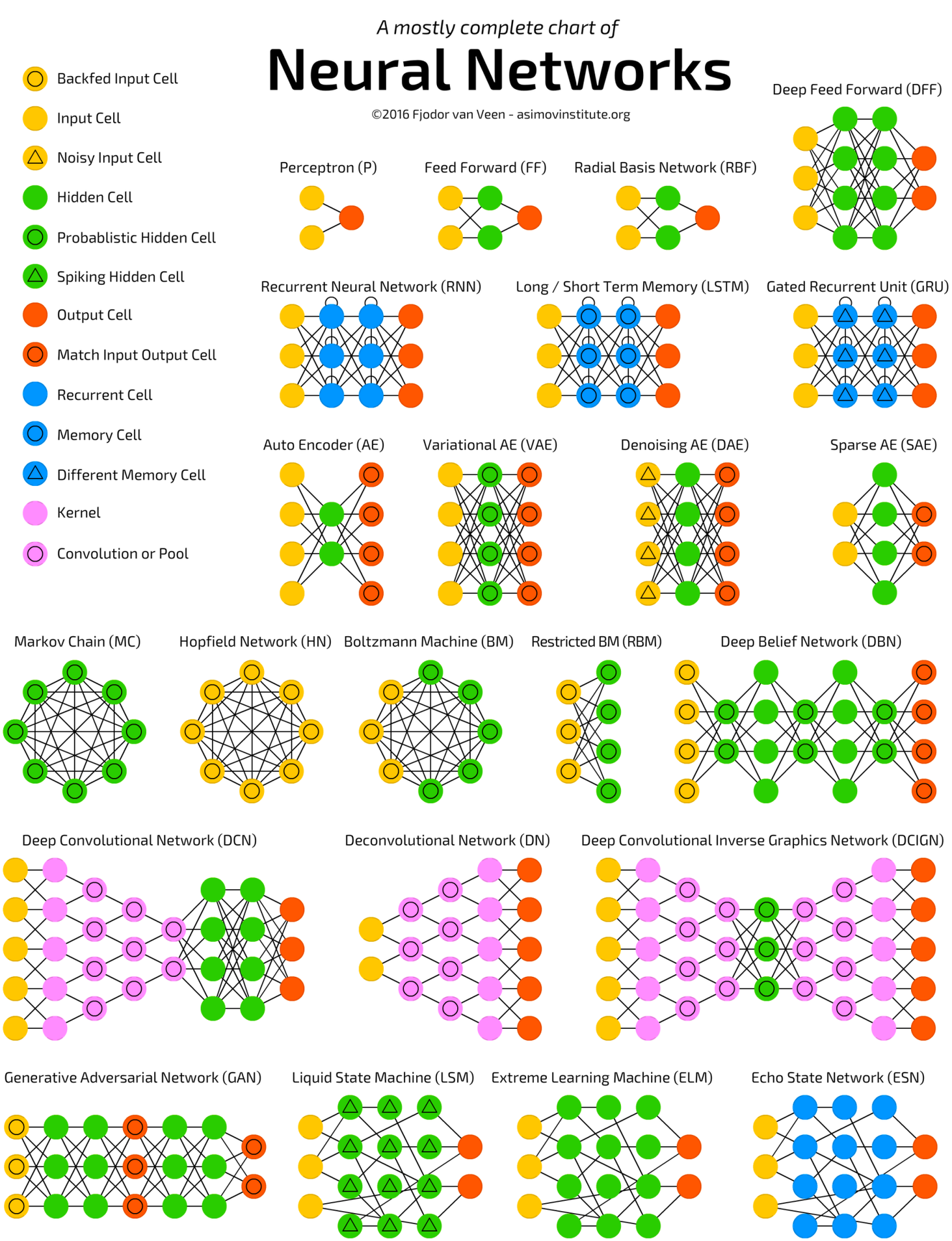
- The violin plots are the **KDE-reconstructed probability distributions** for the hyper-parameters



- A **large variety of neural network architectures** have been proposed: we will only study some of them in this course

- They differ in: number of layers and neurons, role of the neurons, connections between neurons and layers, ….

- Each architecture in general has associated **different training and regularisation strategies**: no fit-for-all methods available!



Case Study I:

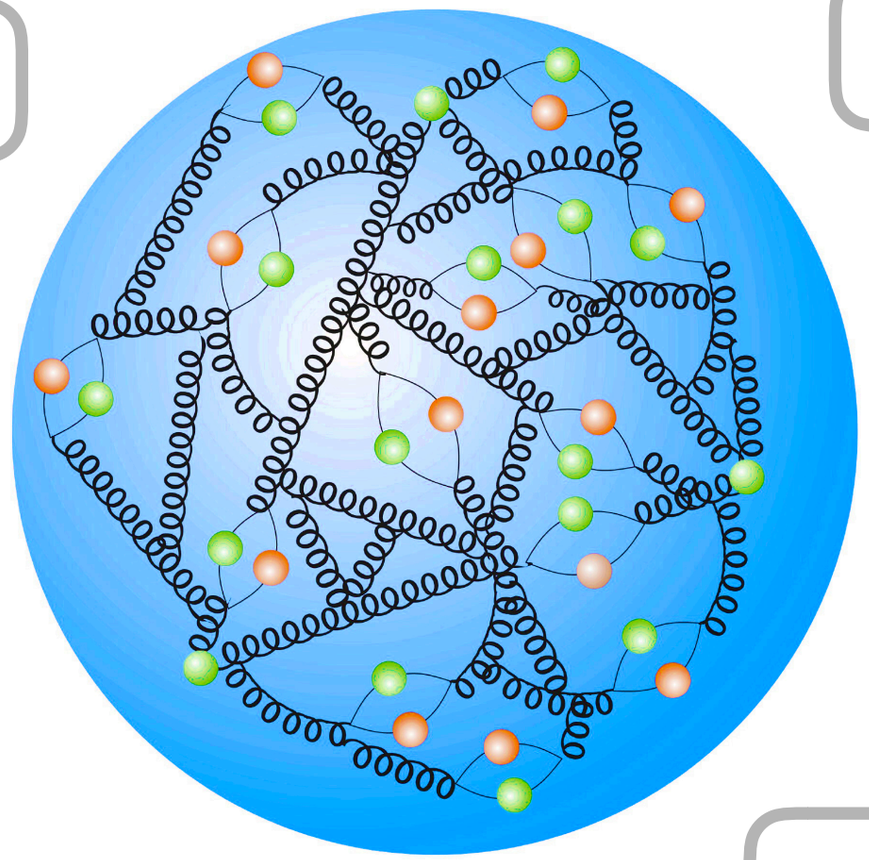
Protron Structure from Neural Networks

The many faces of the proton

QCD bound state of **quarks** and **gluons**

Origin of mass?

Origin of spin?



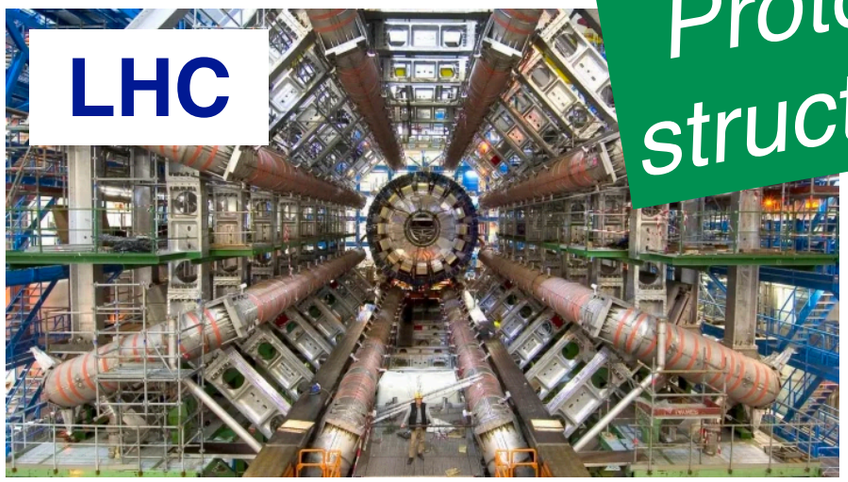
Gluon-dominated matter?

3D imaging?

Heavy quark content?

Nuclear modifications?

From colliders to the cosmos

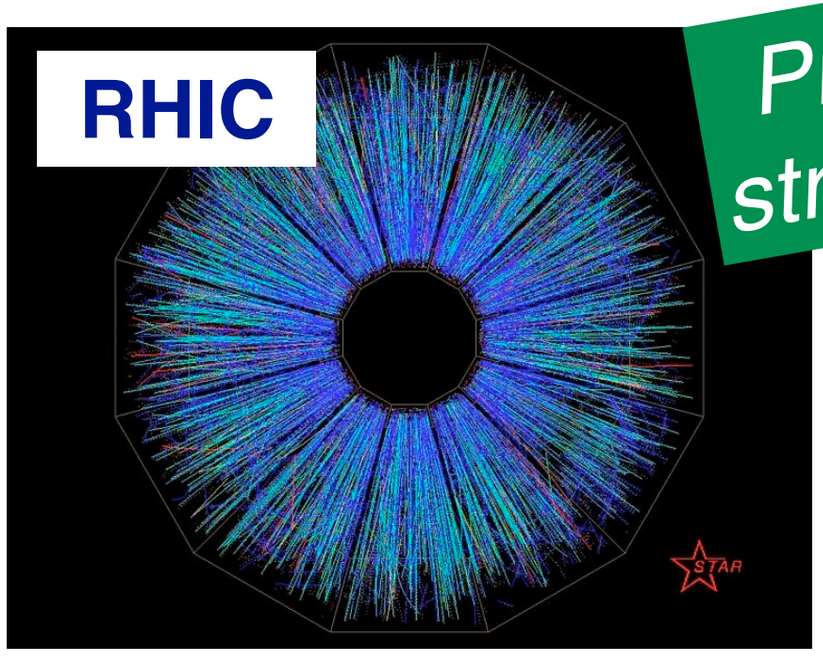


Proton structure

New **elementary particles** beyond the **Standard Model?**

Proton structure

Origins and properties of **cosmic neutrinos**?

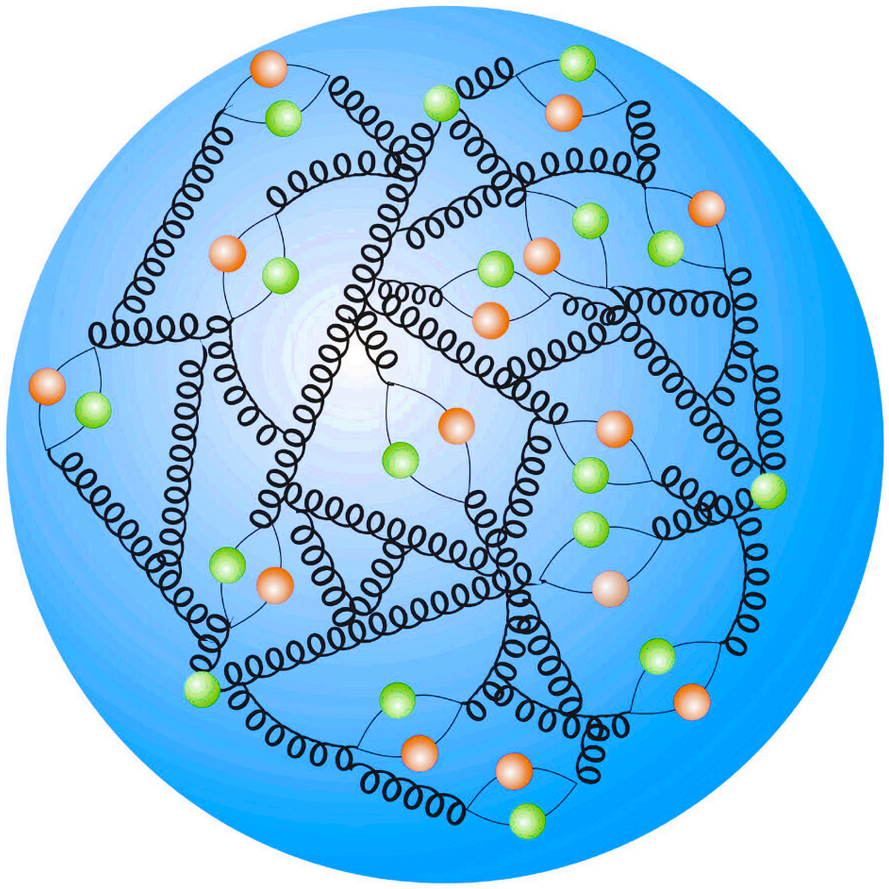


Proton structure

Nature of **Quark-Gluon Plasma** in **heavy-ion collisions?**

Parton Distributions

Proton energy divided among constituents: **quarks** and **gluons**



Parton Distribution Functions (PDFs)

Determine from **data:**

Global QCD analysis

Mass? Spin?

Heavy quark content?

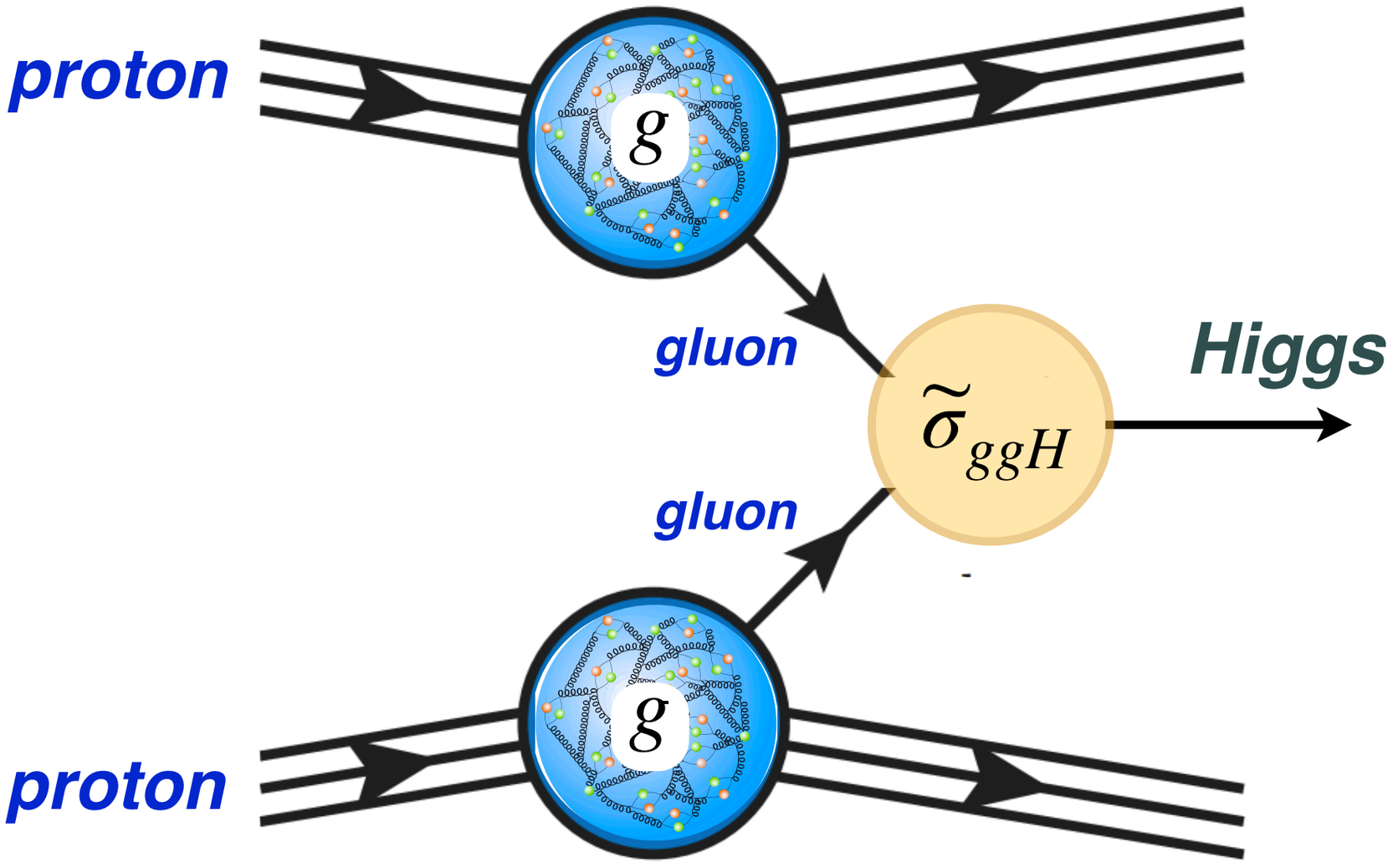
Novel QCD dynamics?

Theoretical predictions for LHC, RHIC, IceCube?

Parton Distributions

$$N_{\mathrm{LHC}}(H) \sim g \otimes g \otimes \tilde{\sigma}_{ggH}$$

Parton Distributions



All-order structure: **QCD factorisation theorems**

Parton Distributions

$$g(x, Q)$$

Probability of **finding a gluon inside a proton**, carrying a fraction ***x*** of the proton momentum, when probed with energy ***Q***

x: fraction of proton momentum carried by gluon

Energy of hard-scattering reaction**:** inverse of resolution length

Dependence on *x* fixed by **non-perturbative QCD dynamics**: extract from experimental data

- **Energy conservation**: momentum sum rule

$$\int_0^1 dx\, x \left(\sum_{i=1}^{n_f} \left[q_i((x, Q^2) + \bar{q}_i(x, Q^2) \right] + g(x, Q^2) \right) = 1$$

- **Quark number conservation**: valence sum rules

$$\int_0^1 dx\, \left(u(x, Q^2) + \bar{u}(x, Q^2) \right) = 2$$

Parton Distributions

$$g(x, Q)$$

Probability of **finding a gluon inside a proton**, carrying a fraction ***x*** of the proton momentum, when probed with energy ***Q***

x: fraction of proton momentum carried by gluon

Energy of hard-scattering reaction**:** inverse of resolution length

Dependence on ***Q*** fixed by **perturbative QCD dynamics**: computed up to $\mathcal{O}\left(\alpha_s^4\right)$

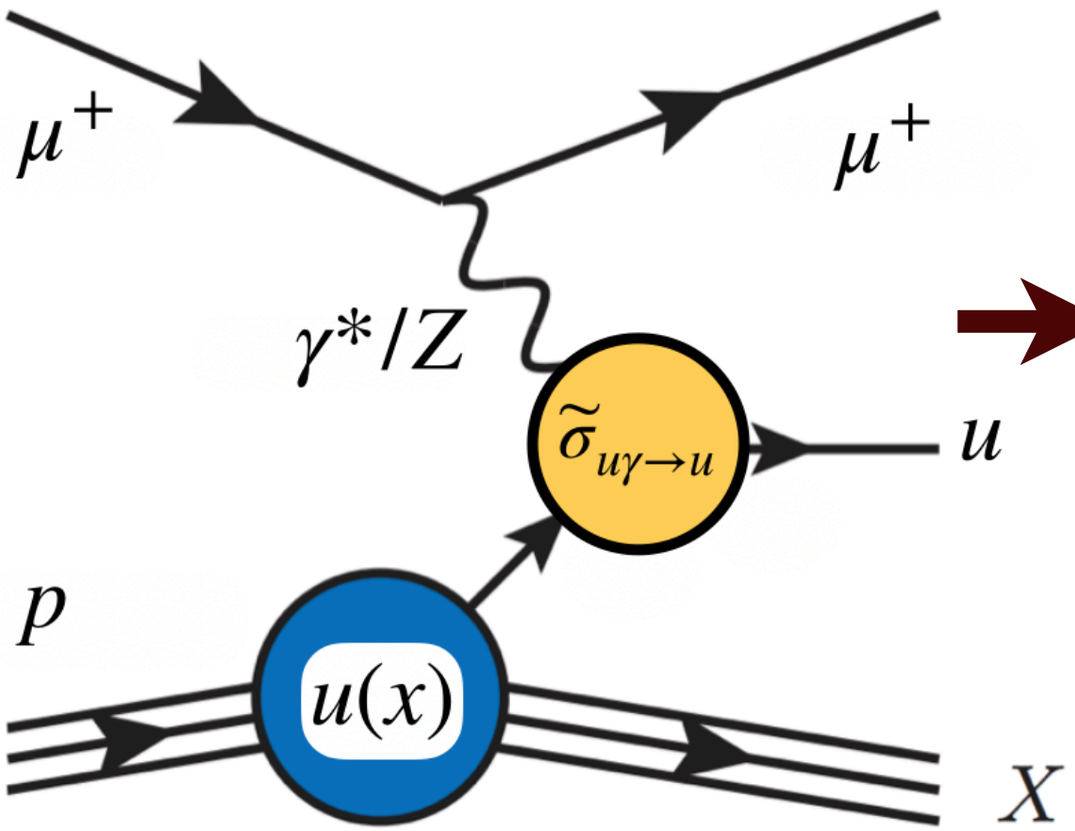
$$\frac{\partial}{\partial \ln Q^2} q_i(x, Q^2) = \int_x^1 \frac{dz}{z} P_{ij}\left(\frac{x}{z}, \alpha_s(Q^2)\right) q_j(z, Q^2)$$

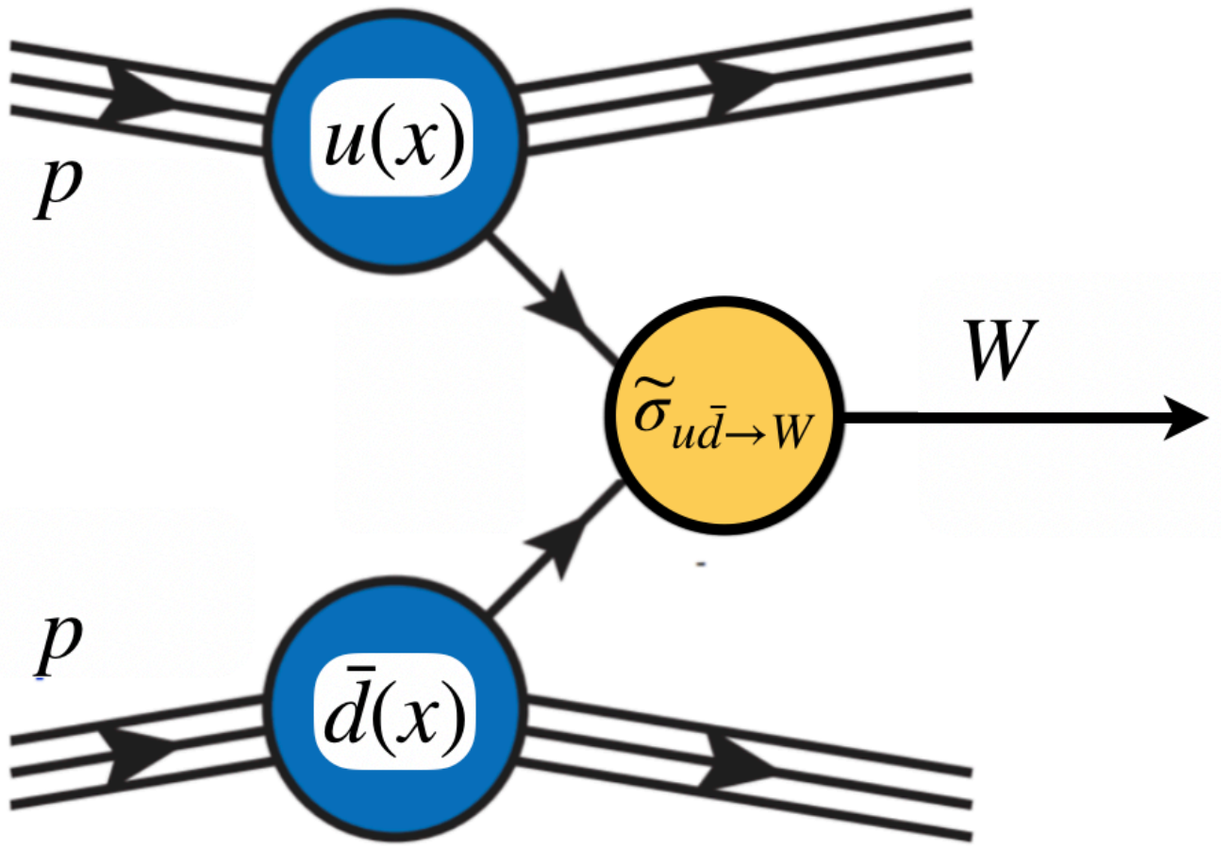
DGLAP *parton evolution equations*

The Global QCD analysis paradigm

QCD factorisation theorems**: PDF universality**

$$\sigma_{l\,p\to\mu\,X} = \widetilde{\sigma}_{u\gamma\to u} \otimes u(x) \quad \rightarrow \quad \sigma_{p\,p\to W} = \widetilde{\sigma}_{u\bar{d}\to W} \otimes u(x) \otimes \bar{d}(x)$$

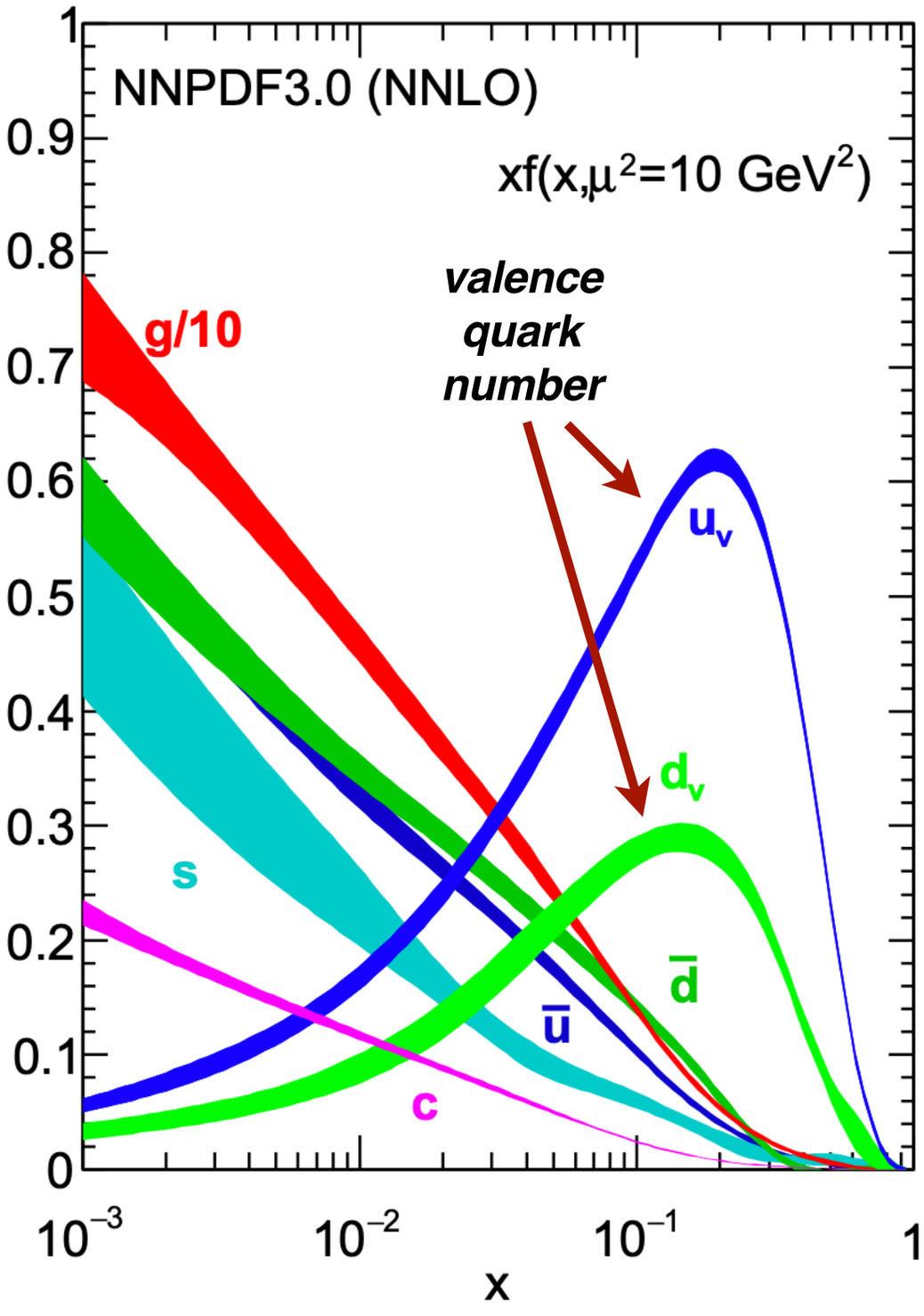


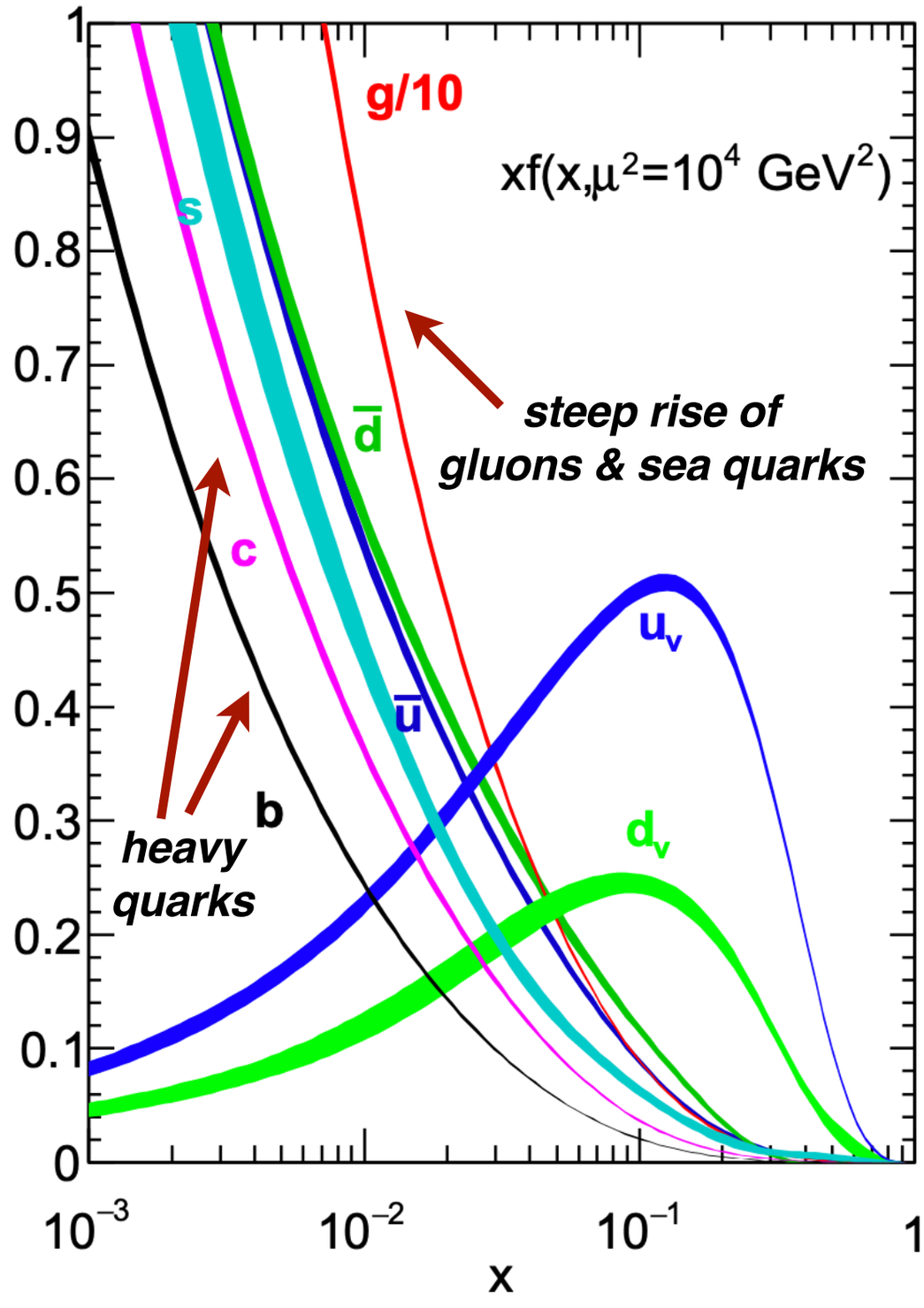


*Determine PDFs from **deep-inelastic scattering**…*

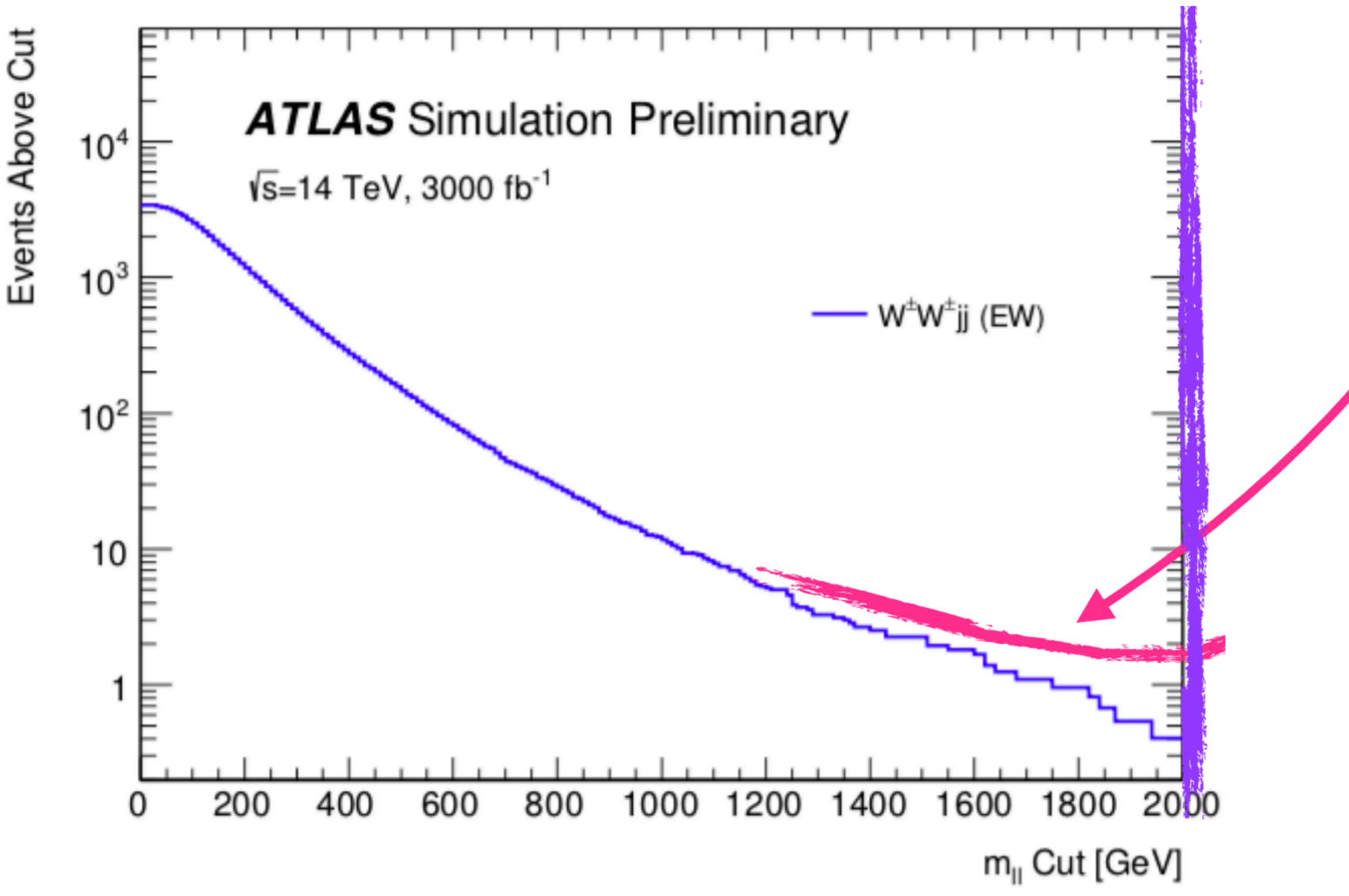
*… and use them to compute predictions for **proton-proton collisions***

A proton structure snapshop





Why do we need better PDFs?

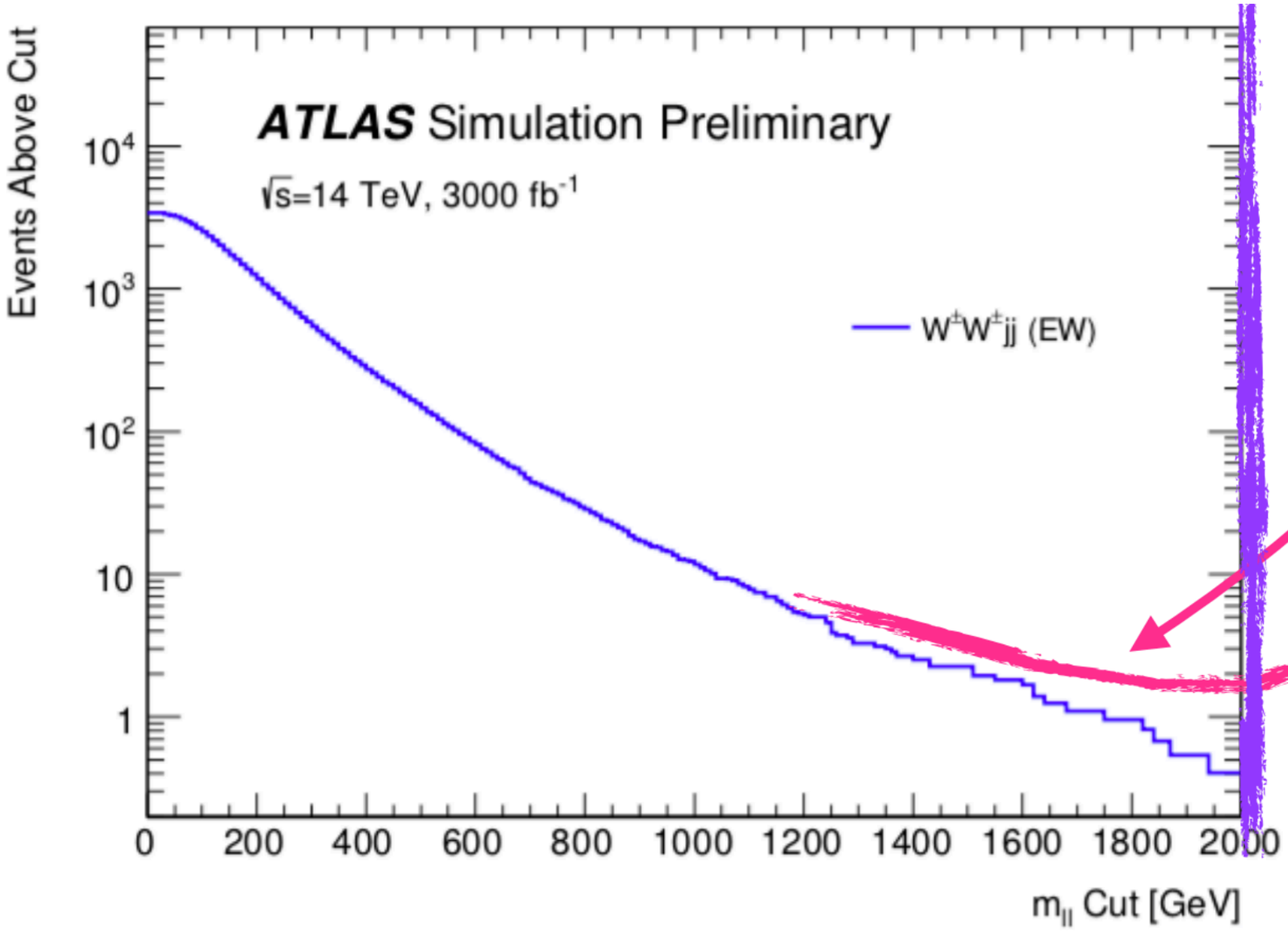


Deviation from SM prediction in high energy tails?

CA Lee, HL/HE-LHC Jamboree, 1 March 2019

Why do we need better PDFs?

Deviation from SM prediction in high energy tails?

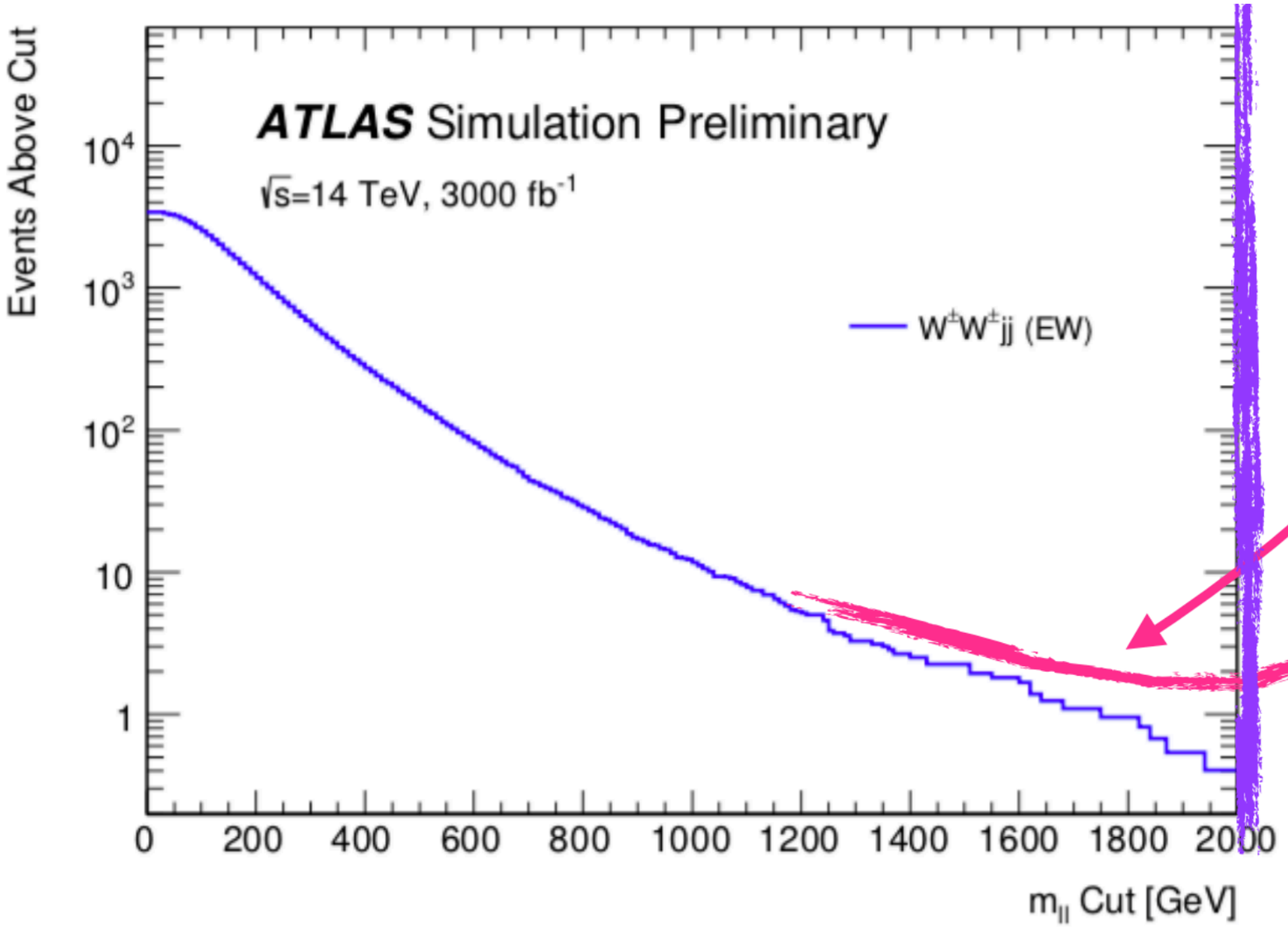


CA Lee, HL/HE-LHC Jamboree, 1 March 2019

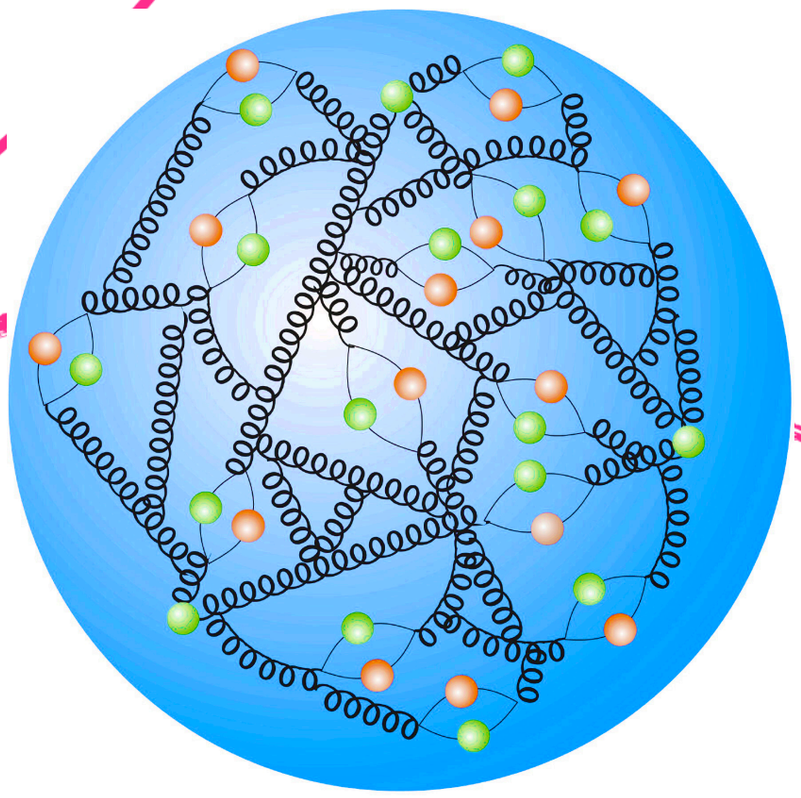
SMEFT interpretation: from a massive particle at high energies …

Why do we need better PDFs?

Deviation from SM prediction in high energy tails?

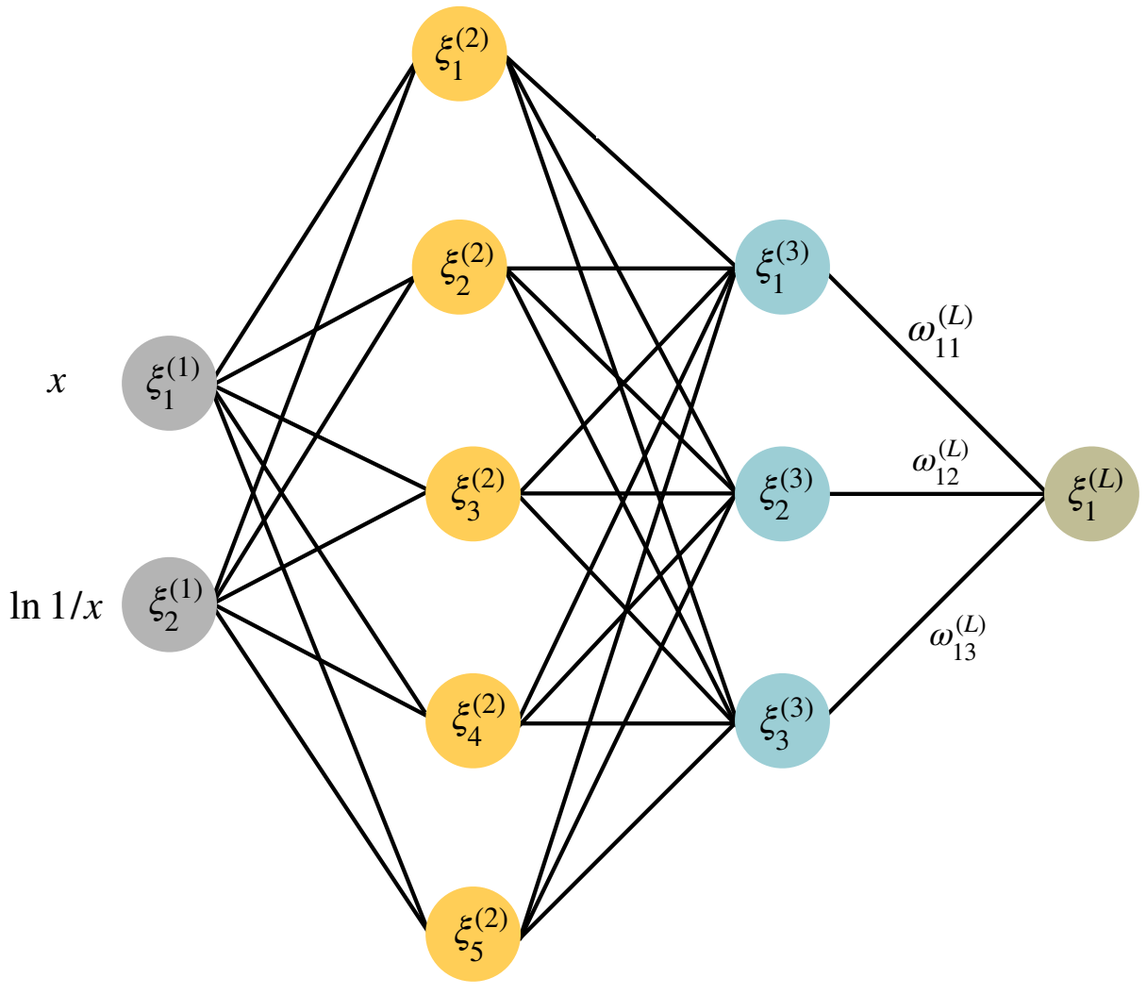


CA Lee, HL/HE-LHC Jamboree, 1 March 2019



…or reflecting our limited understating of proton structure?

Artificial Neural Networks and PDFs



- **Neural Networks** can be used universal unbiased interpolants to **parametrise PDFs**
- Removes model dependence: **unbiased learning** the physical laws from data
- Highly **redundant parametrisation:** identical results if O(10) increase in # free params

	Proton PDFs	***Nuclear PDFs***
Traditional	$g(x) \simeq x^{-b}(1-x)^c$	$R_g(x,A) \simeq (1+bx+cx^2) \times A^d$
Neural Nets	$g(x) \simeq \mathrm{NN}(x)$	$R_g(x,A) \simeq \mathrm{NN}(x,A)$

x: *proton's* ***energy fraction*** *carried by gluons*

A: *number of protons + neutrons*

Monte Carlo method for error propagation

- Generate a large sample of **Monte Carlo replicas** to construct the **probability distribution** in the space of experimental data

$$\mathcal{O}_i^{(\mathrm{art})(k)} = S_{i,N}^{(k)} \mathcal{O}_i^{(\mathrm{exp})} \left(1 + \sum_{\alpha=1}^{N_{\mathrm{sys}}} r_{i,\alpha}^{(k)} \sigma_{i,c}^{(\mathrm{sys})} + r_i^{(k)} \sigma_i^{(\mathrm{stat})} \right) , \quad k = 1, \ldots, N_{\mathrm{rep}}$$

- Determine the PDF NN parameters **replica-by-replica** by minimising a cost function

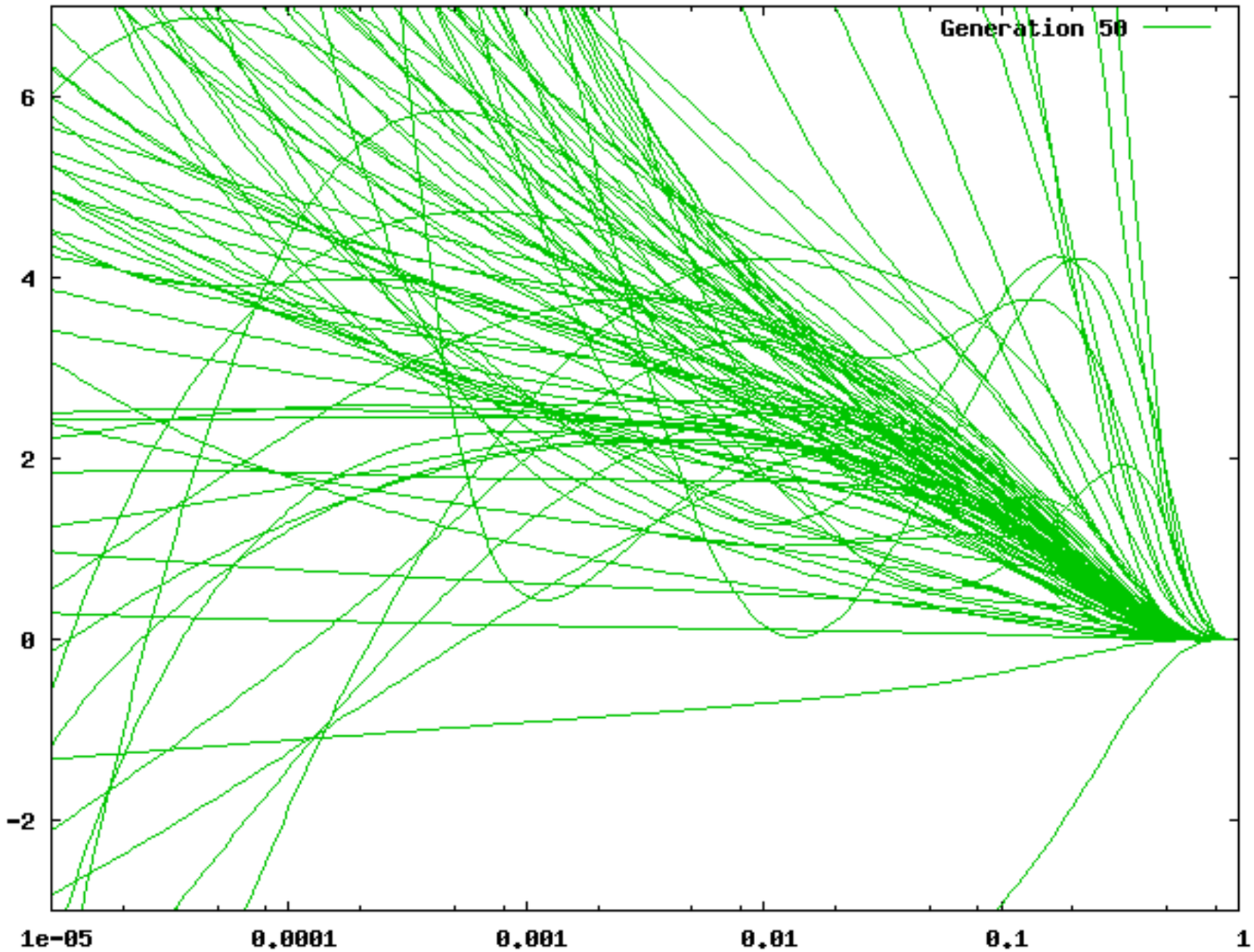
$$E(\{\boldsymbol{\theta}^{(k)}\}) \equiv \frac{1}{N_{\mathrm{dat}}} \sum_{i,j=1}^{N_{\mathrm{dat}}} \left(\mathcal{O}_i^{(\mathrm{th})} \left(q_l(\{\boldsymbol{\theta}^{(k)}\}) \right) - \mathcal{O}_i^{(\mathrm{art})(k)} \right) (\mathrm{cov}^{-1})_{ij} \left(\mathcal{O}_j^{(\mathrm{th})} \left(q_l(\{\boldsymbol{\theta}^{(k)}\}) \right) - \mathcal{O}_j^{(\mathrm{art})(k)} \right)$$

- The ensemble of trained neural networks provides a representation of the **probability density in the space of PDFs**

$$\langle g(x,Q) \rangle = \frac{1}{N_{\mathrm{rep}}} \sum_{k=1}^{N_{\mathrm{rep}}} g(x,Q,\boldsymbol{\theta}^{(k)})$$

NN training

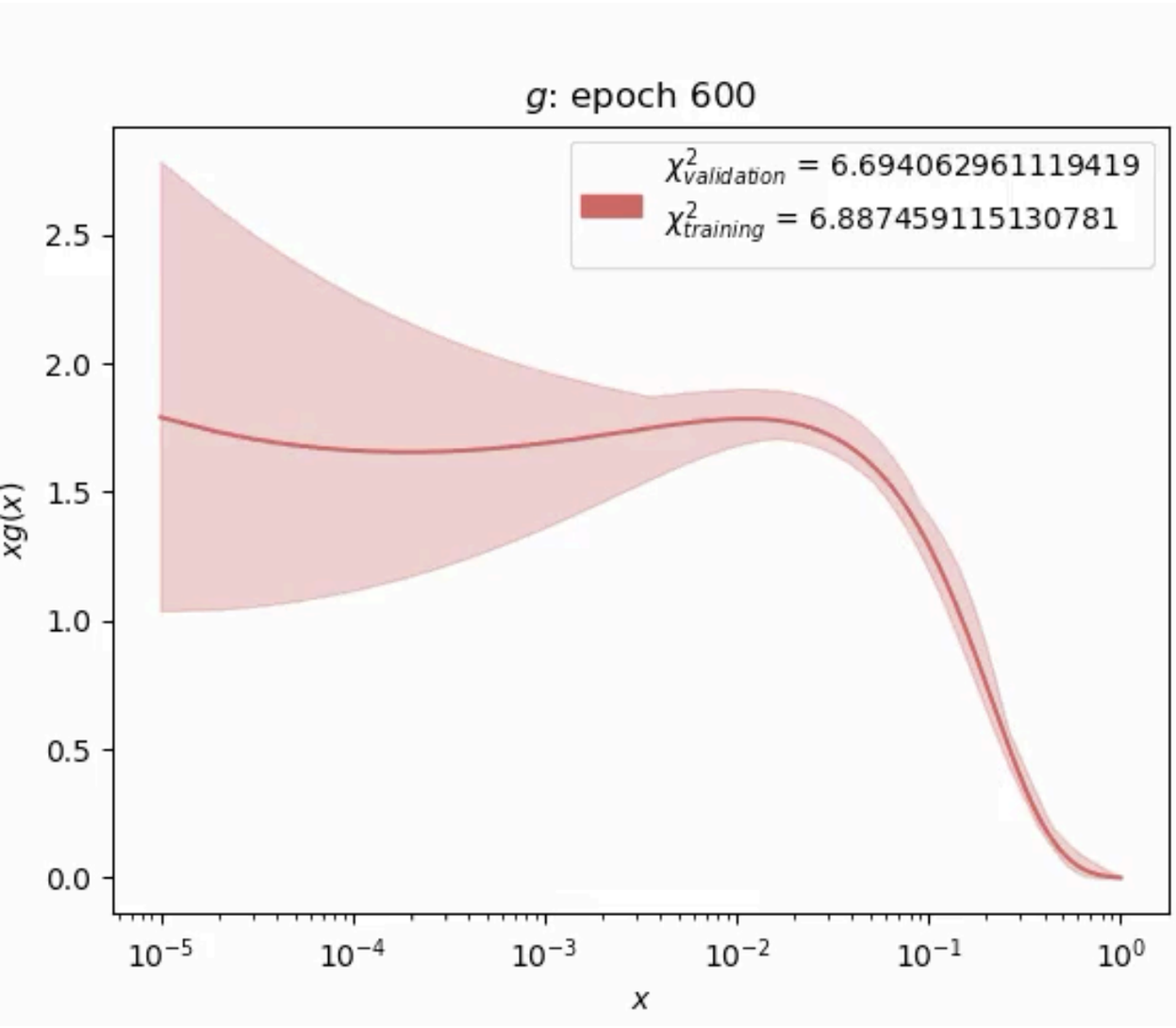
starting from **random parameters,** each member of the MC ensemble of NNs is trained under **convergence** (cross-validation stopping) achieved

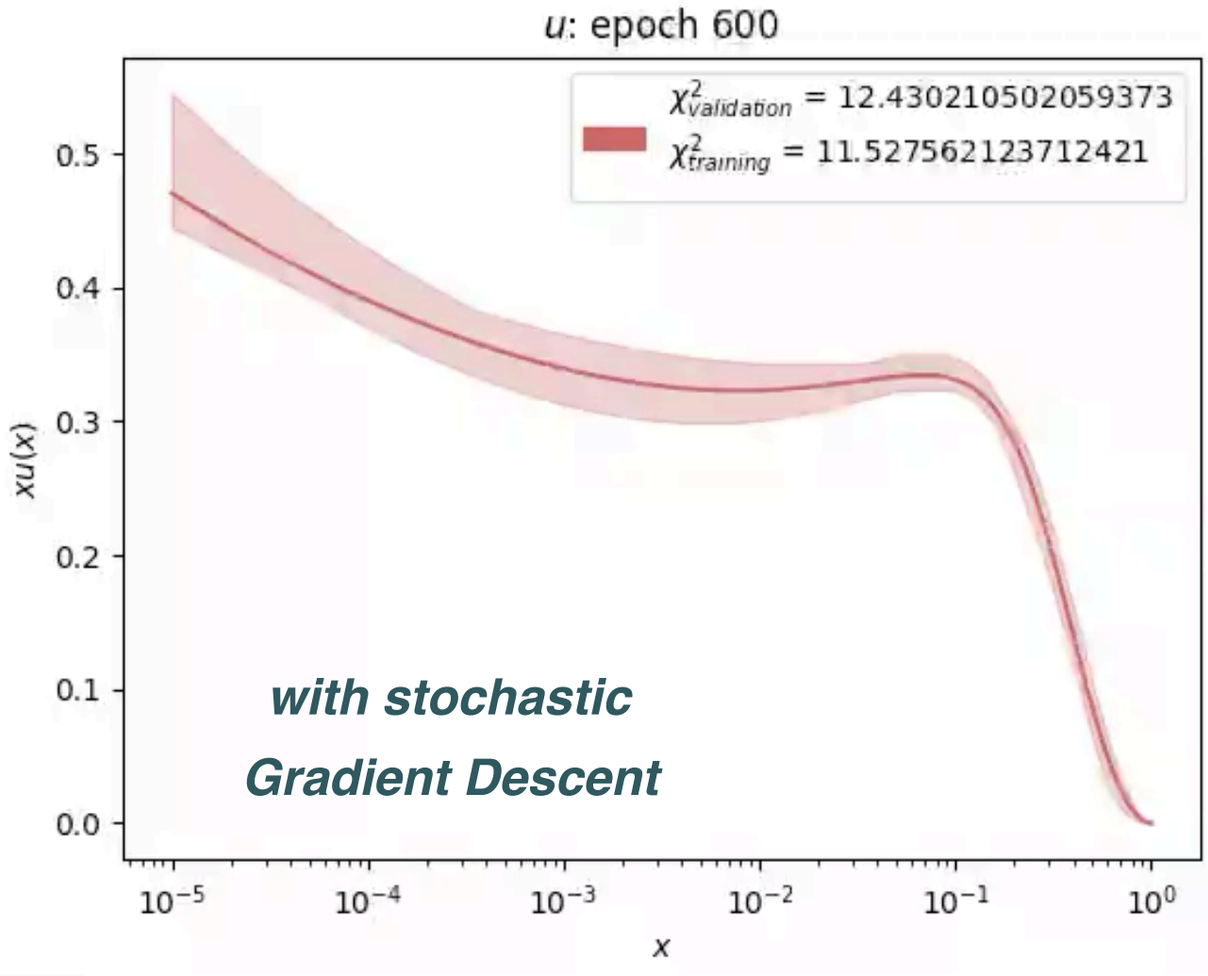


with Genetic Algorithms

NN training live

starting from **random parameters,** each member of the MC ensemble of NNs is trained under **convergence** (cross-validation stopping) achieved





quick initial decrease of the cost function, which slows as we approach the minimum

Tutorial 2 Exercise 2a

starting point is the **Python script** that you will find in

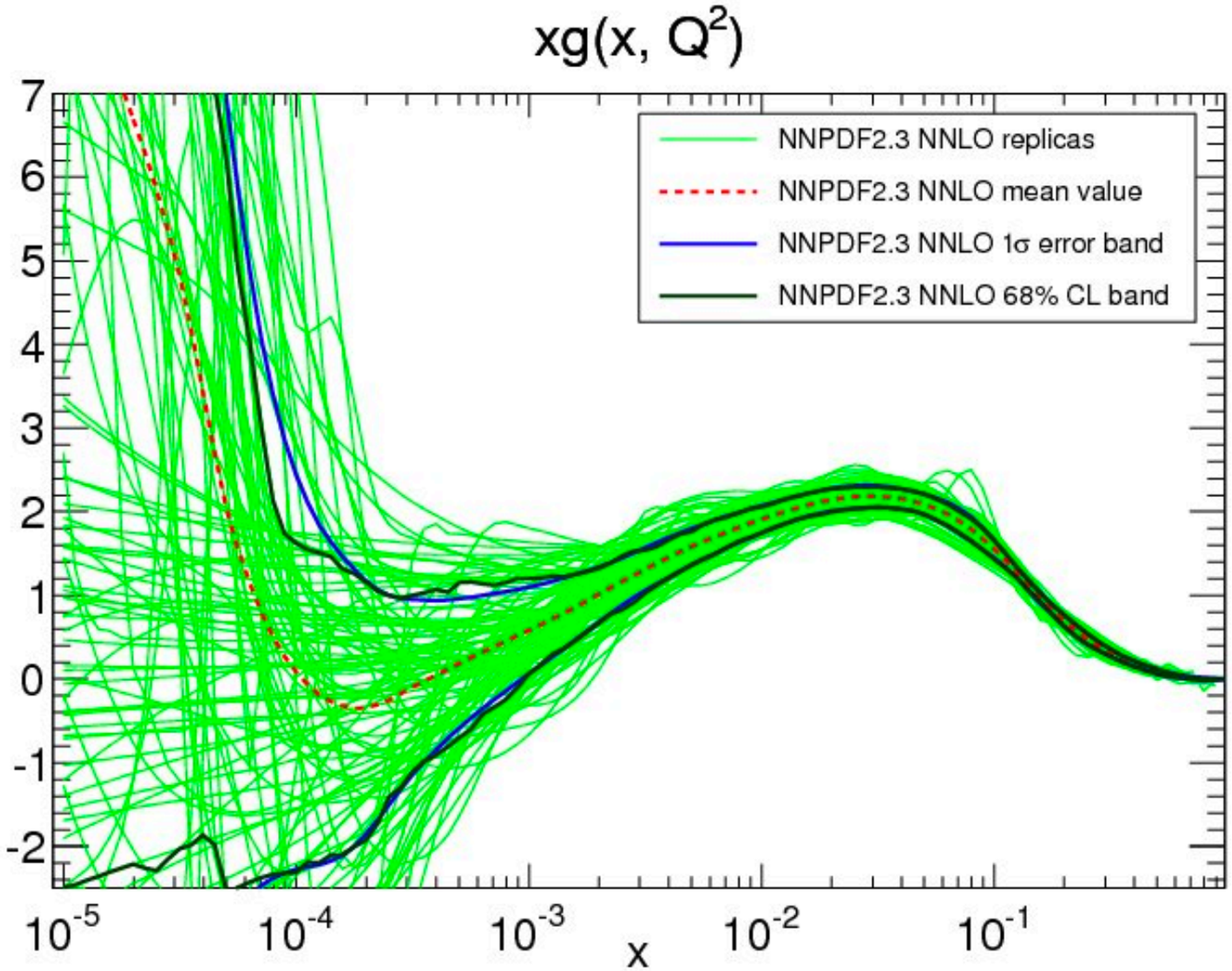
https://github.com/juanrojochacon/ml-ditp-attp/blob/master/Tutorials/Tutorial2/

- Train a **deep neural network** on pseudo-data for Parton Distribution Functions

- Study the dependence of the results with the choice of architecture

- Which minimiser settings lead to the fastest convergence?

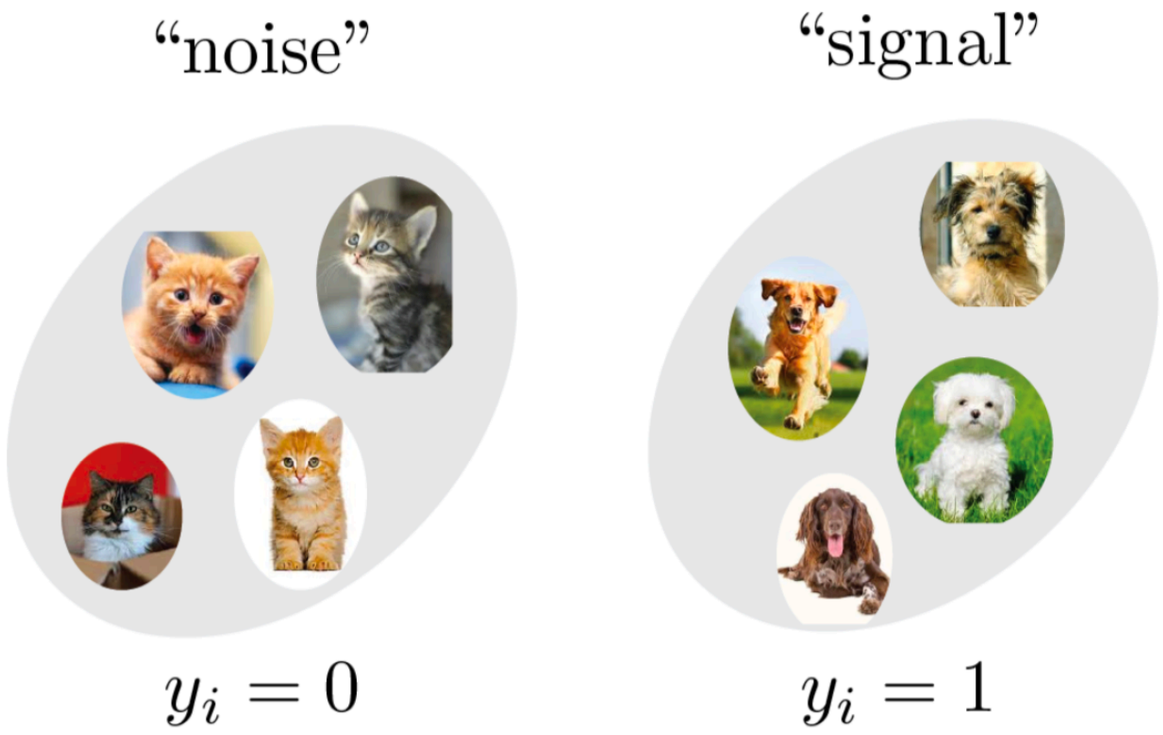
- What happens if you change the random seed? How to you interpret the results?



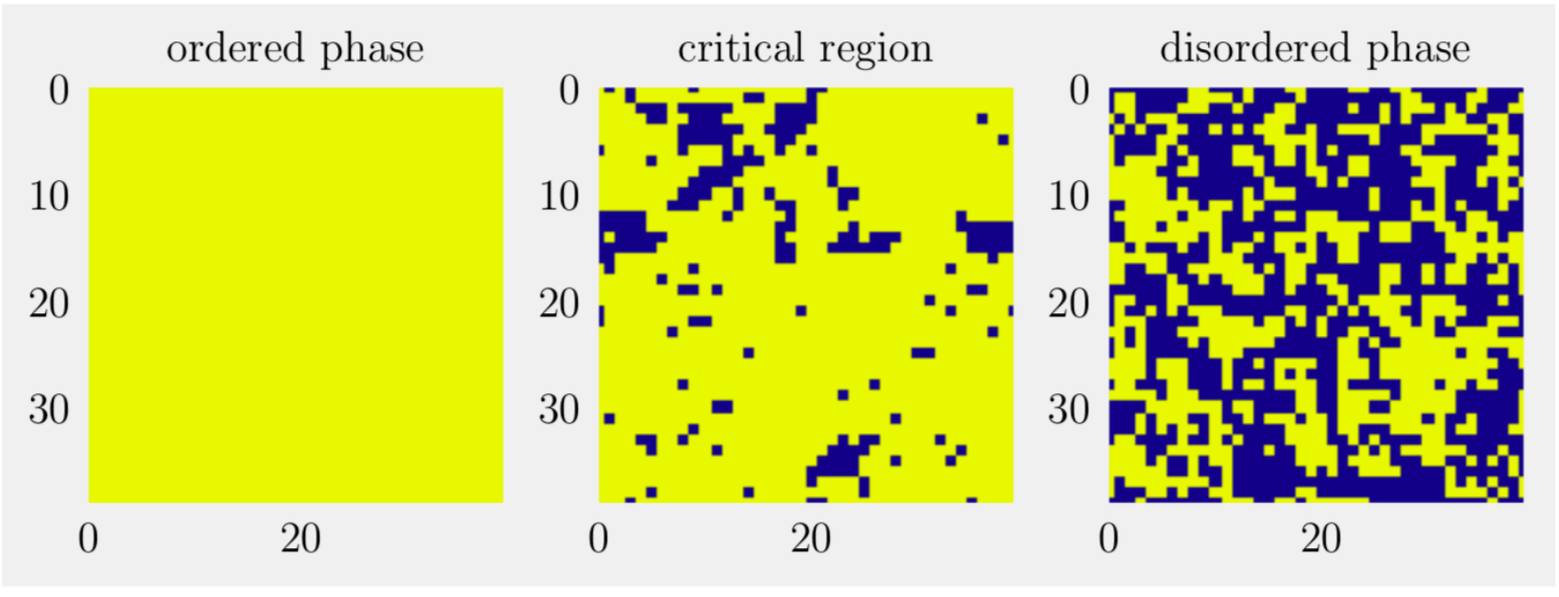
Supervised Learning and Classification

Logistic regression

Relevant for Machine Learning applications where outcomes are discrete variables, eg. **categories in classification problems**



can we tell apart cats from dogs?



can we identify the phase (ordered/disordered) of spin configurations in 2D Ising?

Logistic regression

In this class of problems, the **dependent variables** y_i are discrete and take values *m=0, …, M-1*, so the index *m* also labels the *M* categories

our goal is to **classify the *n* input samples**, each composed by *p* features, intro the ***M* possible categories** of the problem

A simple classifier is the **perceptron:** a linear classifier that categorises examples from a linear combination of the features

$$\sigma(s_i) = \text{sign}(s_i) = \text{sign}\left(\boldsymbol{x}_i^T \boldsymbol{\theta} + b_0\right)$$

p features of the samples

model parameters

a perceptron is a **hard classifier** where each sample is assigned to a category with 100% probability

more complex models can be used as classifiers, including NNs

Logistic regression

In many cases a **soft classifier,** that **outputs the probability** of a given category, is advantageous over a hard classifier

In **logistic regression** the probability that a data point $\boldsymbol{x}_i$ belongs to a category y_i is given by

$$P(y_i = 1 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{x}_i^T \boldsymbol{\theta}}} = \sigma(\boldsymbol{x}_i^T \boldsymbol{\theta})$$

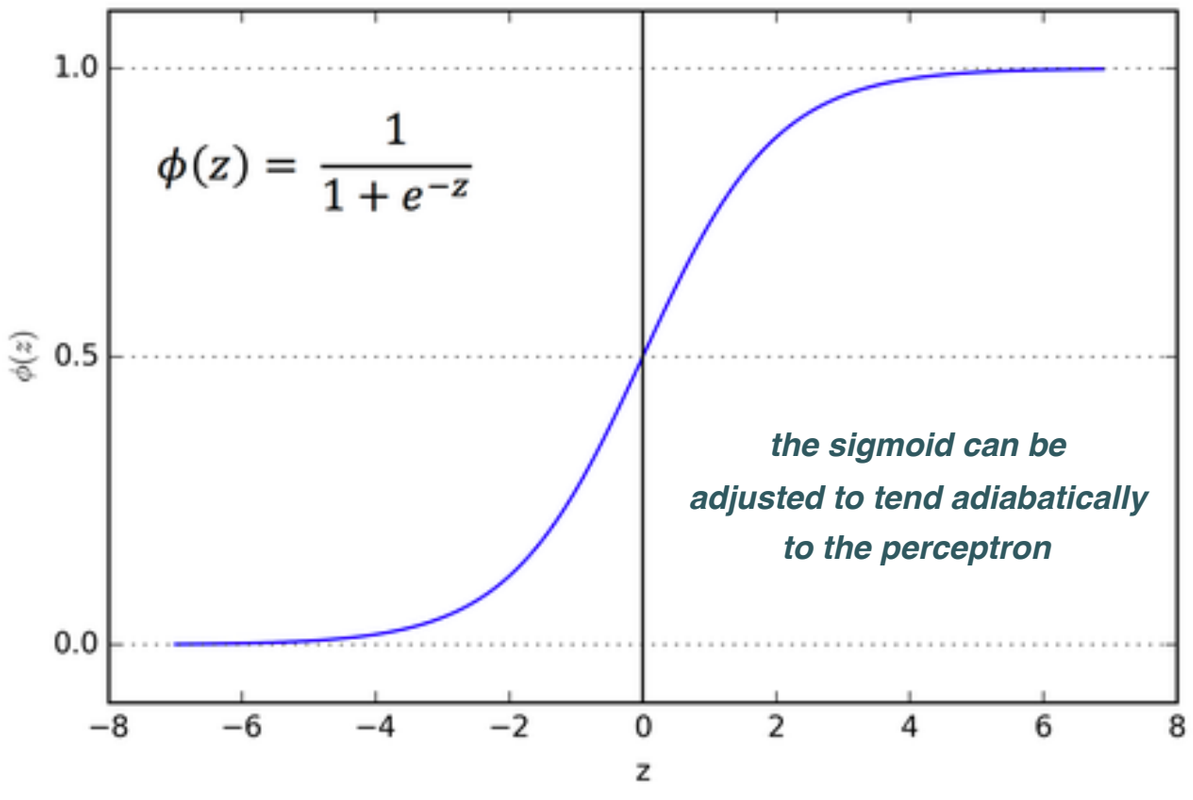
non-linear sigmoid function

$$P(y_i = 0 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = 1 - P(y_i = 1 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = 1 - \sigma(\boldsymbol{x}_i^T \boldsymbol{\theta}) = \sigma(-\boldsymbol{x}_i^T \boldsymbol{\theta})$$

where we have used the logistic (or sigmoid) function:

$$\sigma(s) = \frac{1}{1 + e^{-s}}$$

$$\sigma(-s) = 1 - \sigma(s)$$



Logistic regression

cost function for logistic regression from **Maximum Likelihood Estimation** (MLE):
choose parameters that maximise the probability of seeing the observed data

$$P(y_i \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \Big(P(y_i = 1 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta})\Big)^{y_i} \times \Big(P(y_i = 0 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta})\Big)^{1-y_i}$$

recovers limits when y_i=0, y_i=1

$$P(y_i \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \Big(\sigma(\boldsymbol{x}_i^T \boldsymbol{\theta})\Big)^{y_i} \times \Big(1 - \sigma(\boldsymbol{x}_i^T \boldsymbol{\theta})\Big)^{1-y_i}$$

assuming that all observations are Bernoulli **independent**, the total likelihood is

$$\mathscr{L}(\boldsymbol{\theta} \,|\, \mathscr{D}) = \prod_{i=1}^{n} P(y_i \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \prod_{i=1}^{n} \Big(\sigma(\boldsymbol{x}_i^T \boldsymbol{\theta})\Big)^{y_i} \times \Big(1 - \sigma(\boldsymbol{x}_i^T \boldsymbol{\theta})\Big)^{1-y_i}$$

note that this is soft classification: the probability is not only 0 or 1 but any value in between is possible

Logistic regression

cost function for logistic regression derived from Maximum Likelihood Estimation (MLE): choose parameters that maximise the probability of seeing the observed data

$$\mathscr{L}(\boldsymbol{\theta}\,|\,\mathscr{D}) = \prod_{i=1}^{n} P(y_i\,|\,\boldsymbol{x}_i,\boldsymbol{\theta}) = \prod_{i=1}^{n} \left(\sigma(\boldsymbol{x}_i^T\boldsymbol{\theta}\right)^{y_i} \times \left(1-\sigma(\boldsymbol{x}_i^T\boldsymbol{\theta}\right)^{1-y_i}$$

Since the cost function is the negative log-likelihood, we find that for logistic regression

$$E(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left(-y_i \log \sigma(\boldsymbol{x}_i^T\boldsymbol{\theta}) - (1-y_i)\log(1-\sigma(\boldsymbol{x}_i^T\boldsymbol{\theta}))\right)$$

which is known as the cross-entropy

The parameters of the model are determined by minimising the cross-entropy

$$\widehat{\theta} = \arg\,\min_{\theta}\left\{ \sum_{i=1}^{n} \left(-y_i \log \sigma(\boldsymbol{x}_i^T\boldsymbol{\theta}) - (1-y_i)\log(1-\sigma(\boldsymbol{x}_i^T\boldsymbol{\theta}))\right) \right\}$$

note that no analytic solution is possible, and numerical methods are required

one can use more complex models to parametrise the probabilities, such as NNs

Metrics for binary classification

there are other **useful metrics** that are used in ML binary classification problems

example classification problem: 34 training samples, of which

True Positives (TP) e.g. 8	False Positives (FP) e.g. 2
False Negatives (FN) e.g. 4	True Negatives (TN) e.g. 20

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{8 + 20}{8 + 20 + 2 + 4} = 0.823$$

however accuracy is not the right metric for classification problems

defined by a large disparity between classes, e.g. when TN >> TP

$$\text{Accuracy} \simeq \frac{\text{TN}}{\text{TN} + \text{FN}} + \dots$$

Metrics for binary classification

there are other **useful metrics** that are used in ML binary classification problems

example classification problem: 34 training samples, of which

True Positives (TP) e.g. 8	False Positives (FP) e.g. 2
False Negatives (FN) e.g. 4	True Negatives (TN) e.g. 20

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{8 + 20}{8 + 20 + 2 + 4} = 0.823$$

$$\text{Precision} = \frac{TP}{TP + FP} = 0.80$$

proportion of correct positive classifications

$$\text{Recall} = \frac{TP}{TP + FN} = 0.67$$

proportion of correct actual positive classifications

ROC curve

In most ML classification problems there is a **threshold** that can be varied to decide at what output of the model a given data point is assigned to each category:

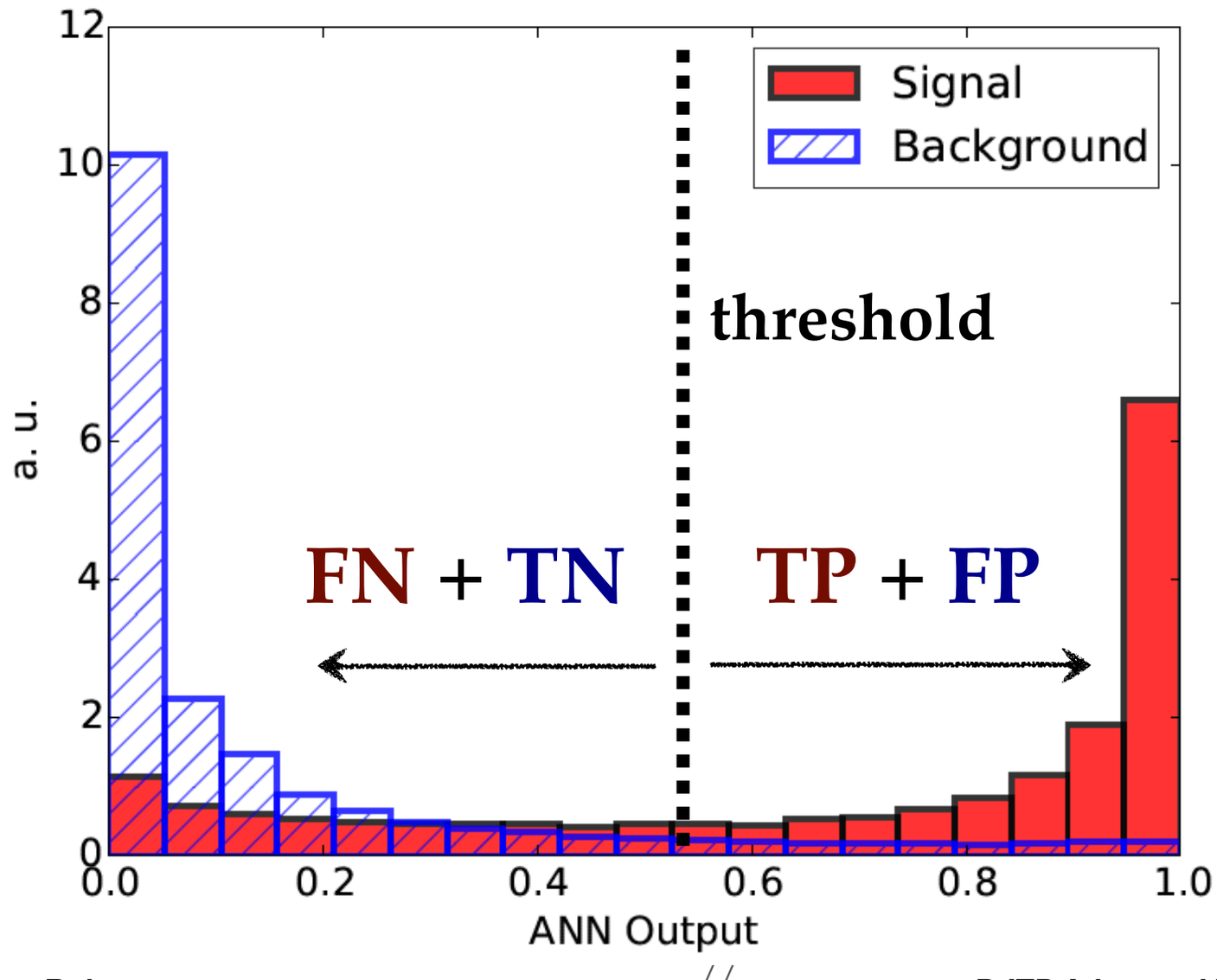
how should we select this threshold for classification?

ROC curve

In most ML classification problems there is a **threshold** that can be varied to decide at what output of the model a given data point is assigned to each category:

☑ A conservative threshold will **maximise TP**, but also FP might be large

☑ An aggressive threshold **reduces FP** but then FN might be large.



ROC curve

In most ML classification problems there is a **threshold** that can be varied to decide at what output of the model a given data point is assigned to each category:

- ☑ A conservative threshold will **maximise TP**, but also FP might be large
- ☑ An aggressive threshold **reduces FP** but then FN might be large.

effect of threshold is quantified by the **Receiver Operating Characteristic** (ROC) curve

$$\text{Recall} = \text{True Positive Rate} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

← *proportion of correctly classified positives*

vs

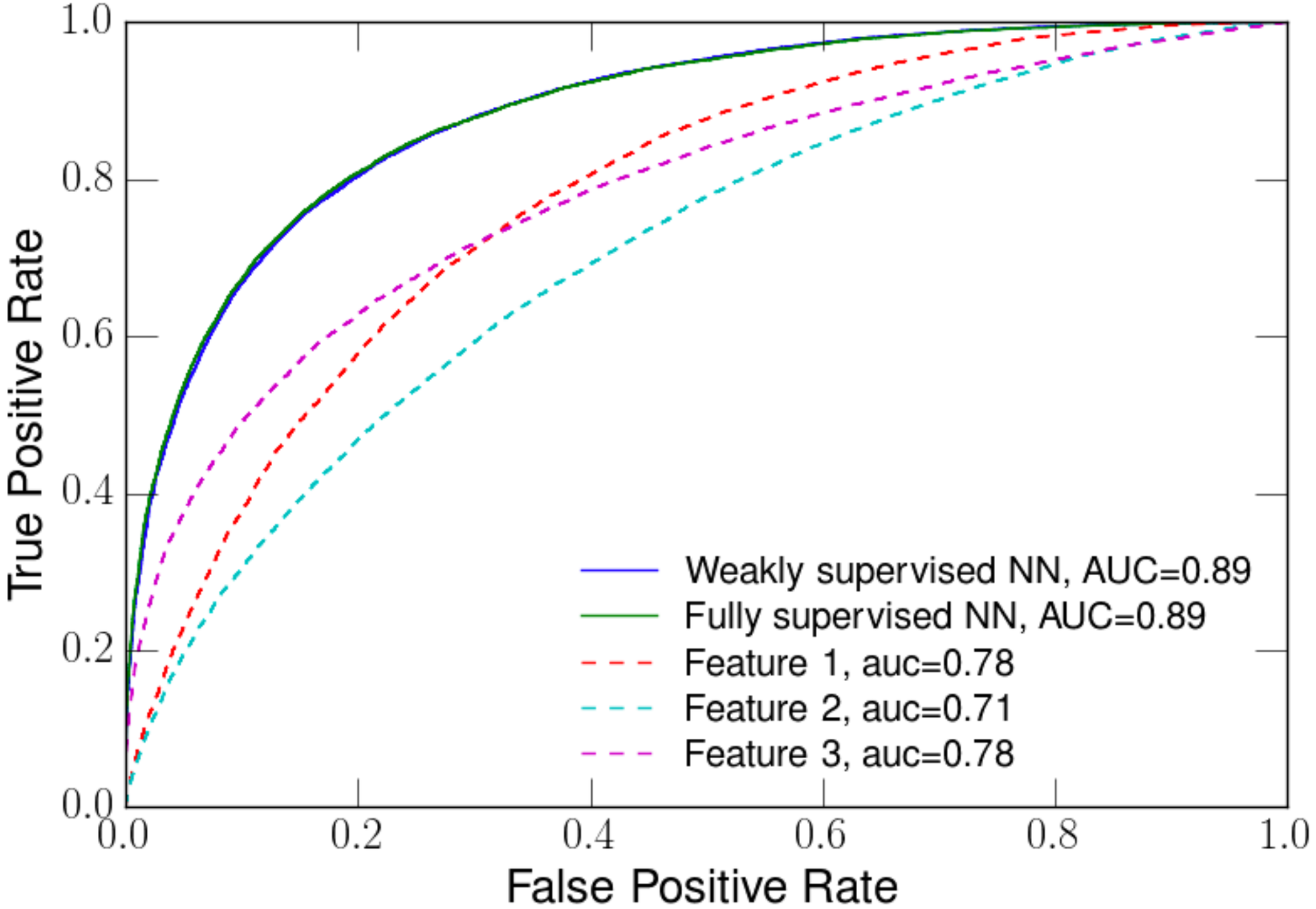
$$\text{False Positive Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

← *incorrectly classified negatives*

a good classifier should maximise TPR while minimising FPR

ROC curve

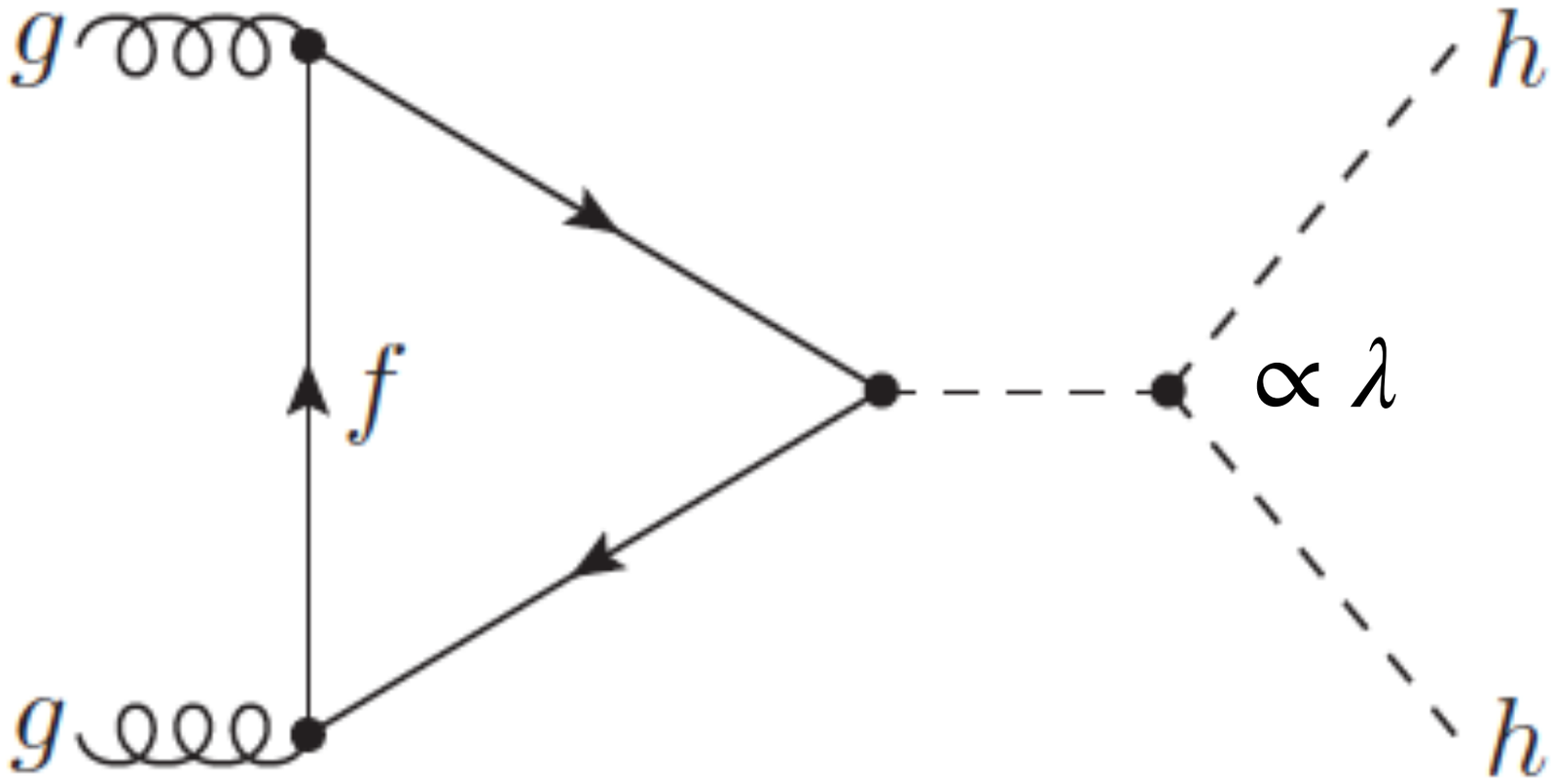
$$\text{Recall} = \text{True Positive Rate} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad \text{False Positive Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$



Case Study II:

Higgs Pair Production

Higgs Pair Production

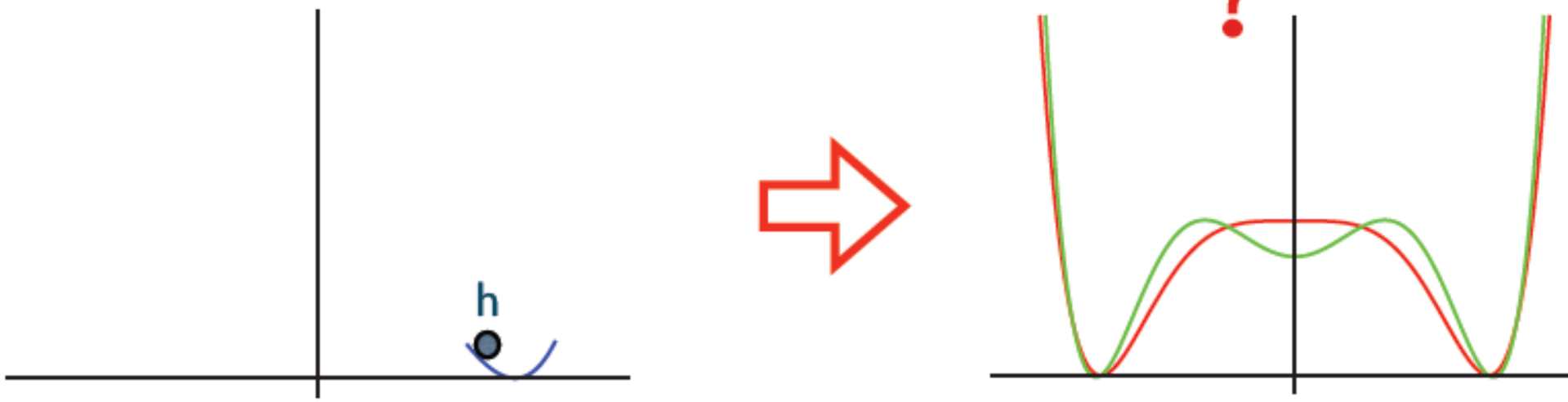


$$V(h) = m_h^2 h^\dagger h + \frac{1}{2}\lambda (h^\dagger h)^2$$

Due to small rates in the SM, only final states where at least one of the Higgs bosons decays into a **bb pair** are experimentally accessible at the LHC

Higgs Pair Production

- Current measurements in single Higgs production probe **Higgs potential close to minimum**
- Double Higgs production essential to **reconstruct full potential** and clarify EWSB mechanism
- In the SM the Higgs potential is fully *ad-hoc*: **many other EWSB mechanisms** conceivable



Higgs mechanism ***Coleman-Weinberg mechanism***

$$V(h) = m_h^2 h^\dagger h + \frac{1}{2}\lambda (h^\dagger h)^2 \qquad V(h) \to \frac{1}{2}\lambda (h^\dagger h)^2 \log\left[\frac{(h^\dagger h)}{m^2}\right]$$

Each possibility associated to **completely different EWSB mechanism**, with crucial implications for the **hierarchy problem**, the structure of quantum field theory, and **New Physics at the EW scale**

Arkani-Hamed, Han, Mangano, Wang, arxiv:1511.06495

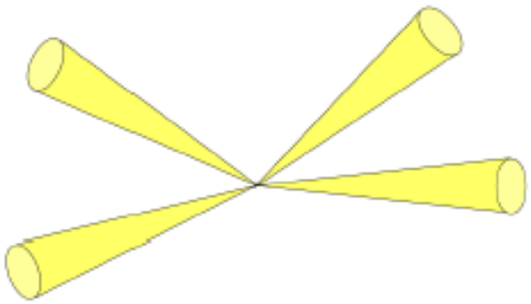
ggF di-Higgs in the 4b final state

- The **4b final state** offers largest rates but overwhelming QCD multijet backgrounds
- Made competitive requiring the **di-Higgs system to be boosted** and exploiting kinematic differences between signal and QCD background with **jet substructure**
- **Boosted b-tagging** by ghost-associating mass-drop tagged large-*R* jets with b-tagged small-*R* jets
- **Tagging these events** is challenging due to the very high rate of QCD multijets but doable
- **Scale-invariant tagging:** event-by-event classification depending on final state topology

Process	Generator	N_{evt}	σ_{LO} (pb)	K-factor
$pp \to hh \to 4b$	`MadGraph5_aMC@NLO`	1M	$6.2 \cdot 10^{-3}$	2.4 (NNLO+NNLL [18, 19])
$pp \to b\bar{b}b\bar{b}$	`SHERPA`	3M	$1.1 \cdot 10^{3}$	1.6 (NLO [63])
$pp \to b\bar{b}jj$	`SHERPA`	3M	$2.7 \cdot 10^{5}$	1.3 (NLO [63])
$pp \to jjjj$	`SHERPA`	3M	$9.7 \cdot 10^{6}$	0.6 (NLO [77])
$pp \to t\bar{t} \to b\bar{b}jjjj$	`SHERPA`	3M	$2.5 \cdot 10^{3}$	1.4 (NNLO+NNLL [78])

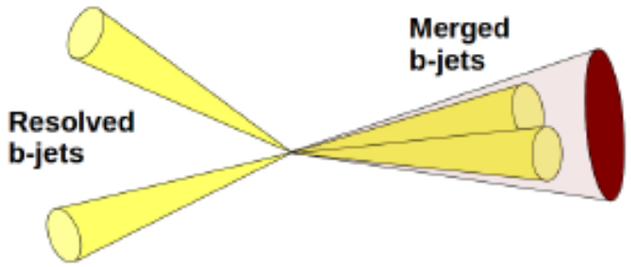
Analysis strategy

Resolved



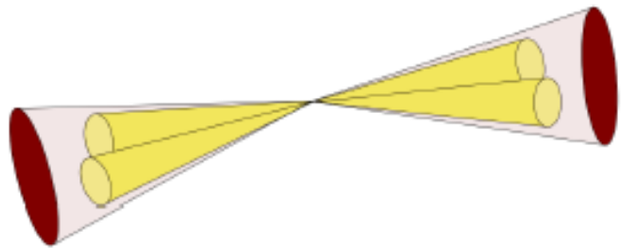
- ≥ 4 b-tagged small-R jets
- Higgs reconstruction from leading 4 jets
- Choice that minimises mass difference between dijet systems

Intermediate



- $= 1$ large-R jet (Higgs-tagged + b-tagged) (leading Higgs)
- ≥ 2 b-tagged small-R jets
- $\Delta R > 1.2$ w.r.t. large-R jet
- Higgs reconstruction from leading 2 small-R jets
- Choice that minimises mass difference of dijet system and large-R jet

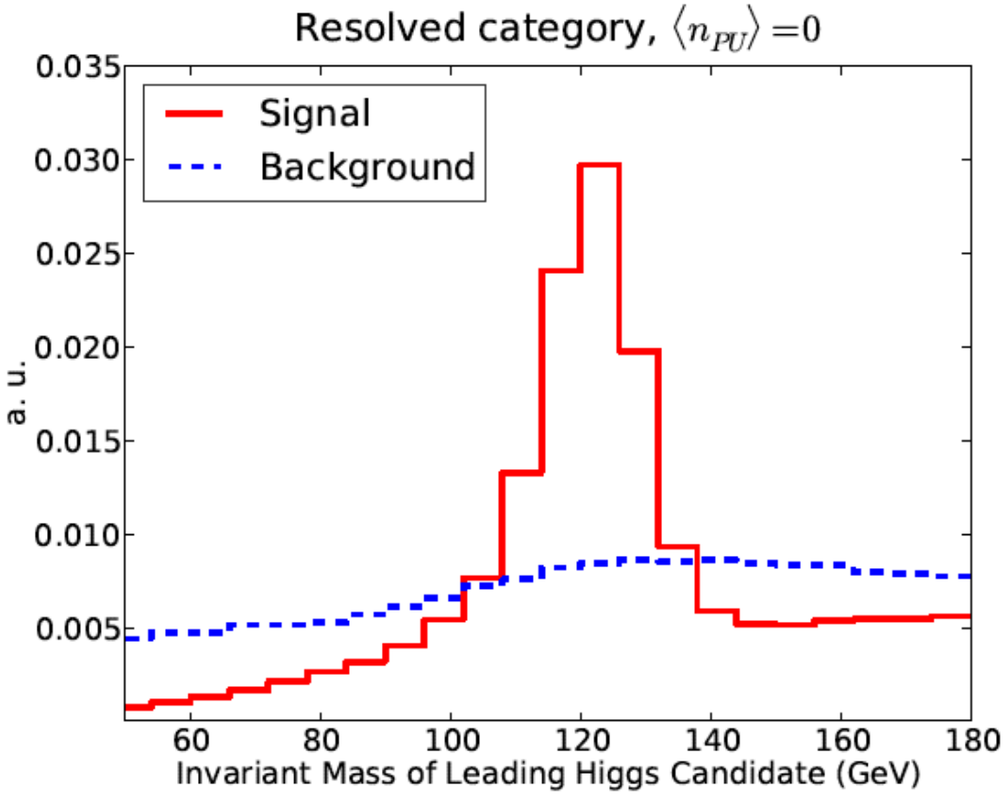
Boosted

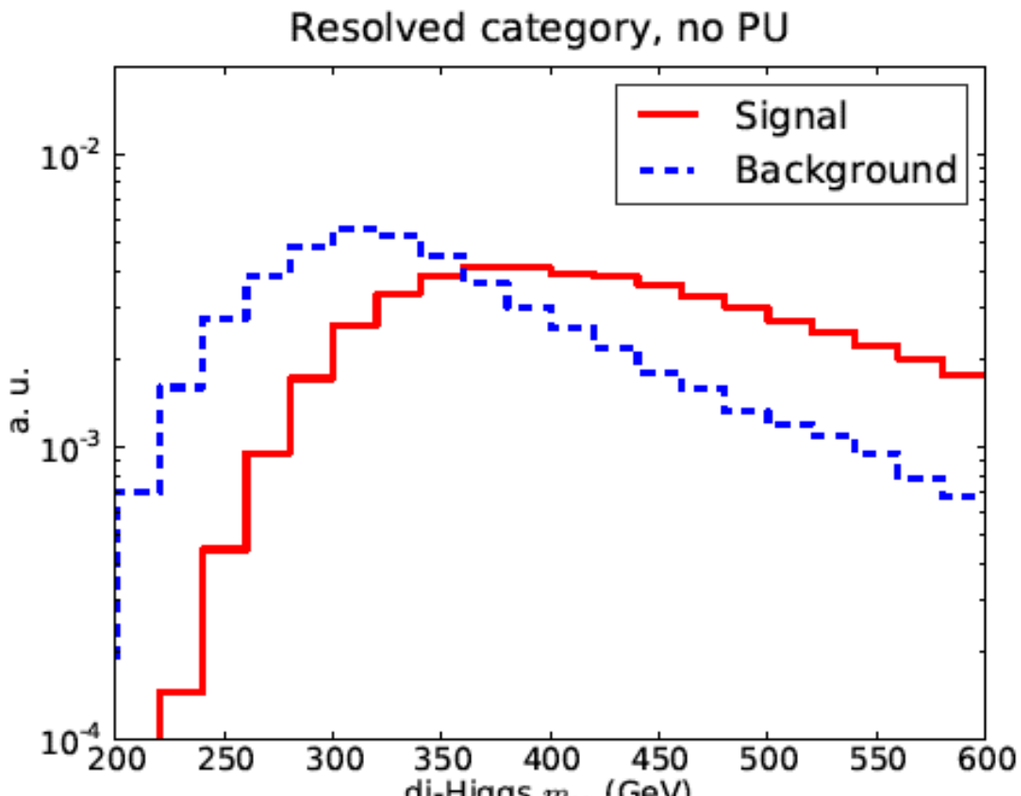


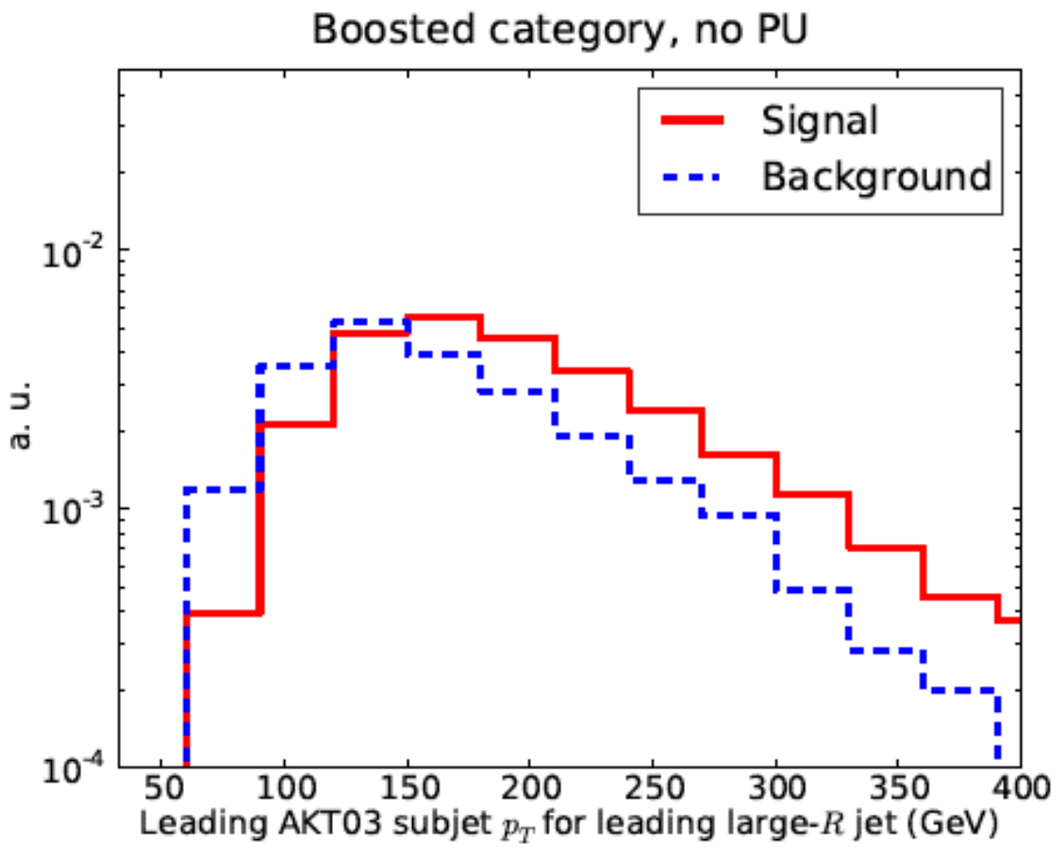
- ≥ 2 large-R jets (Higgs-tagged + b-tagged)
- Leading two jets taker as Higgs candidates

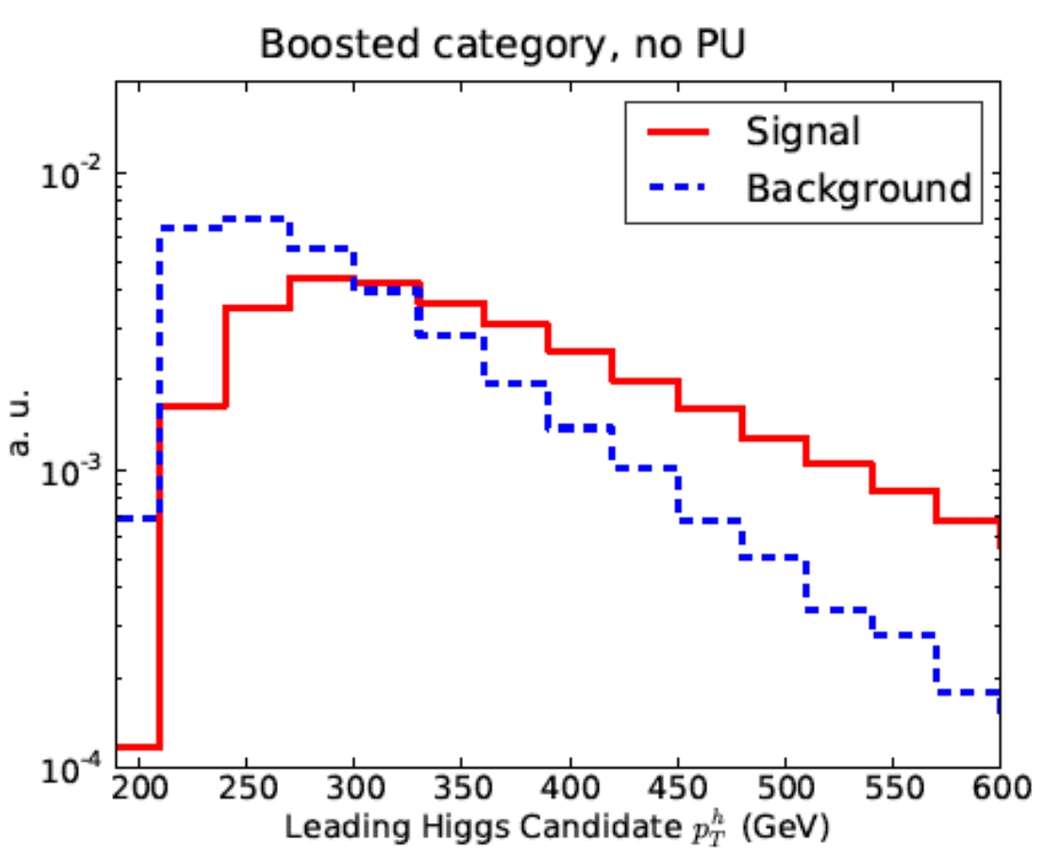
+ **Loose Higgs mass window cut**: $|m_{h,j} - 125\ \mathrm{GeV}| < 40\ \mathrm{GeV}\ , j = 1, 2$
+ **Rank categories** by $S/\sqrt{B}$ to make them **exclusive**: boosted > intermediate > resolved

Results of cut-based analysis

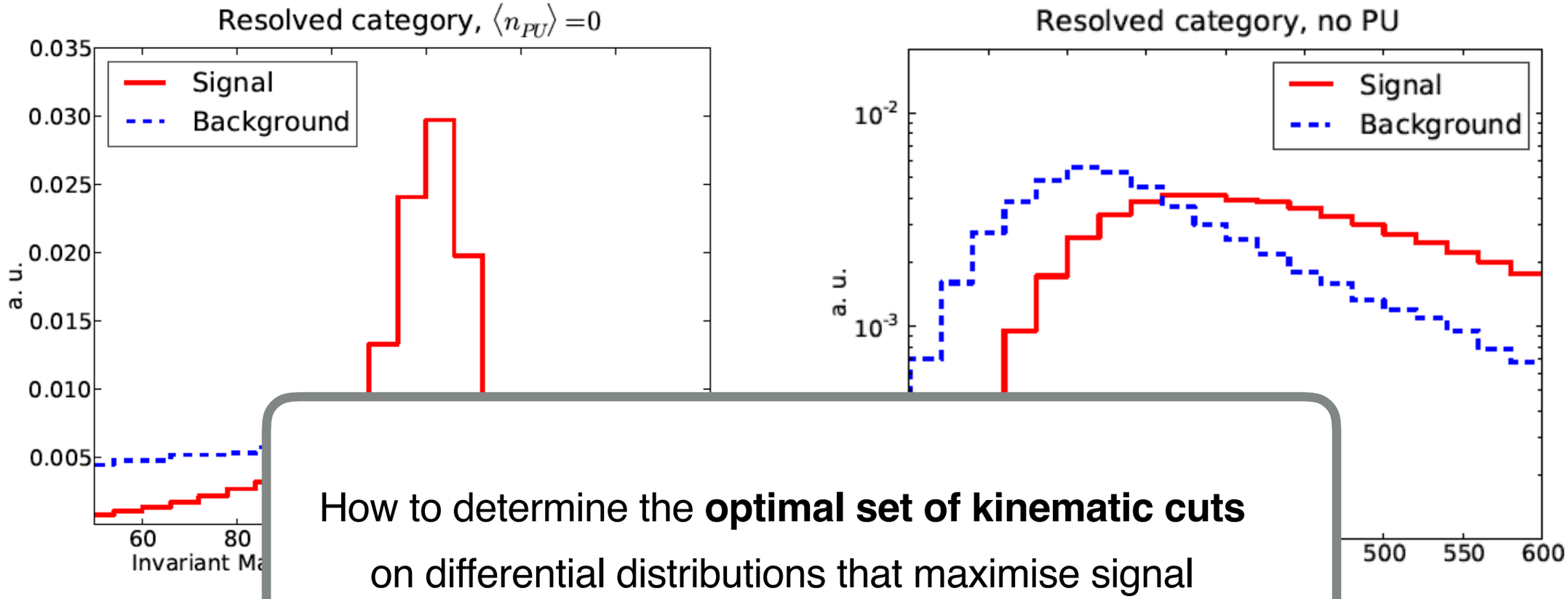




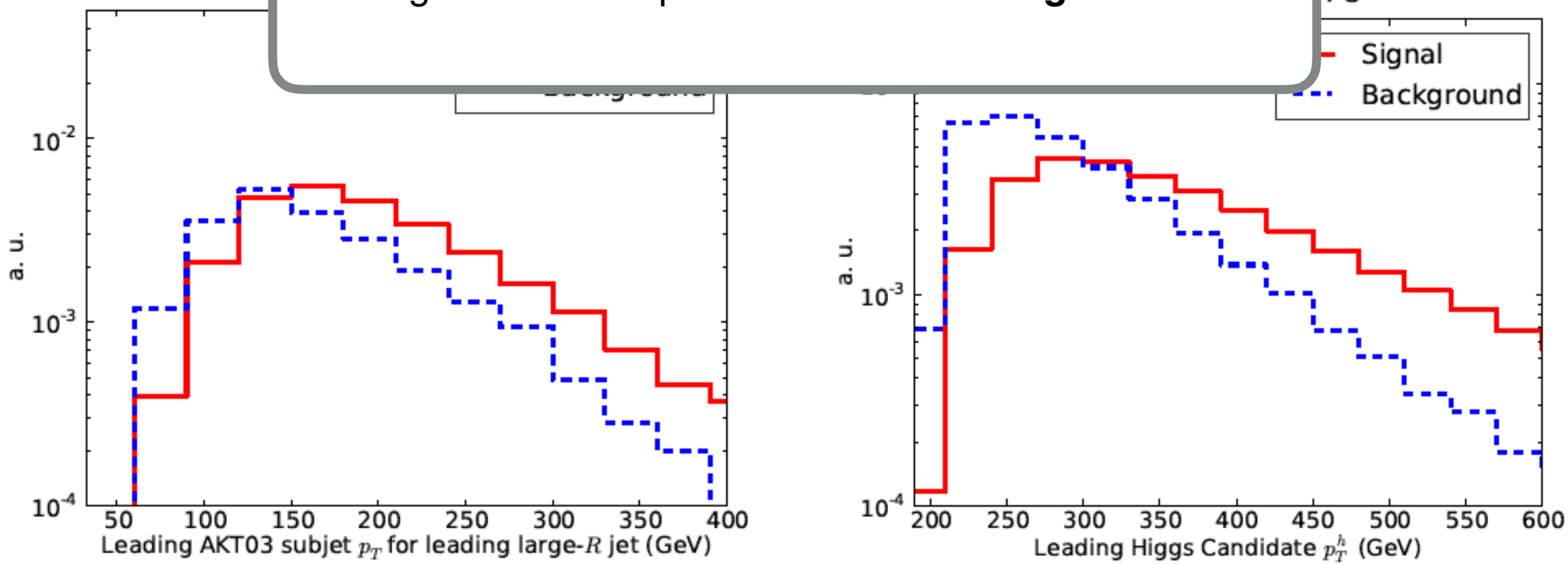




Results of cut-based analysis



How to determine the **optimal set of kinematic cuts** on differential distributions that maximise signal significance? Exploit **Machine Learning** tools!



Neural Network Discriminator

Input

Higgs pT

Higgs m

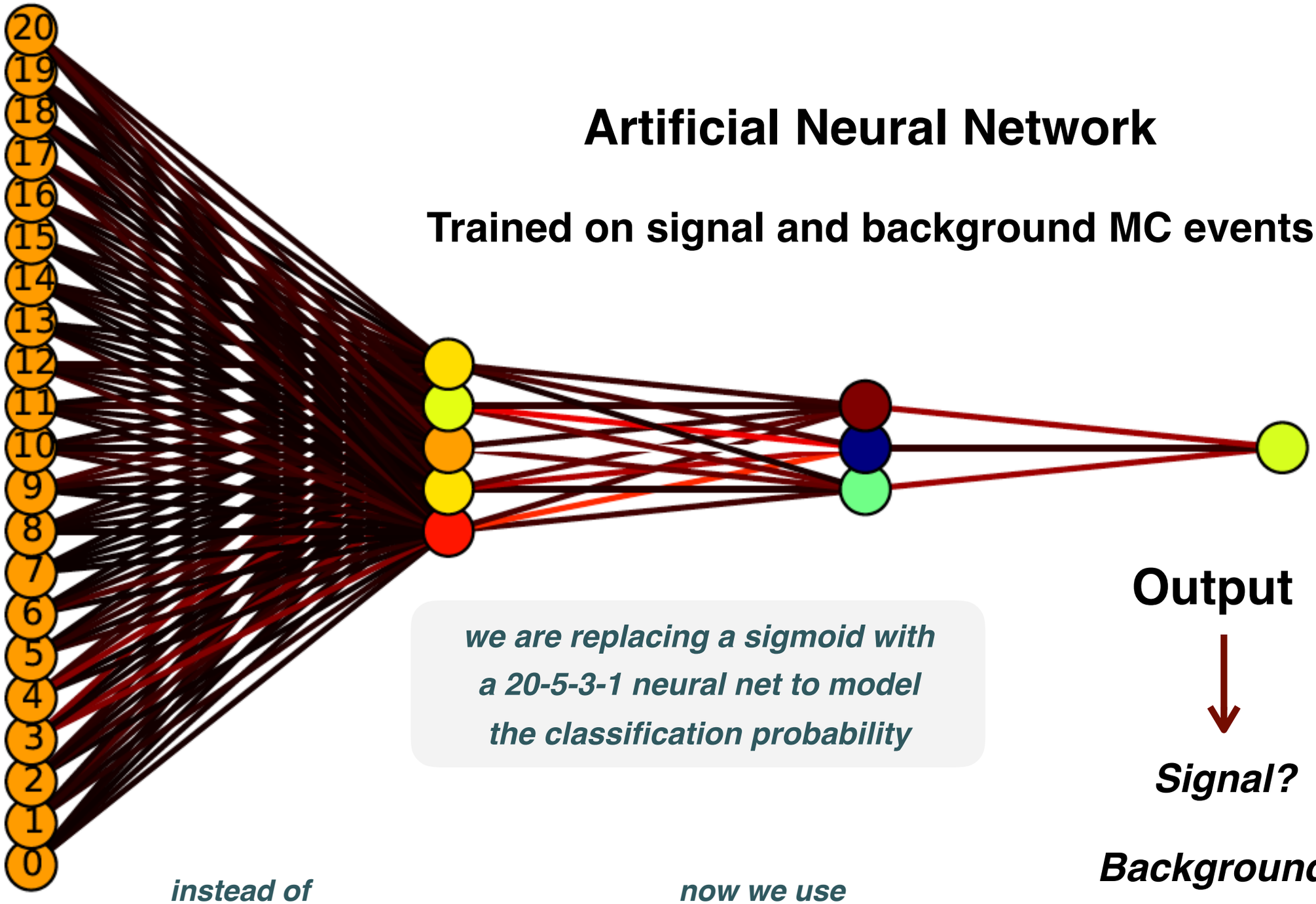
di-Higgs m

ECF

τ_{12}

Subjet pT

.....



Artificial Neural Network

Trained on signal and background MC events

we are replacing a sigmoid with a 20-5-3-1 neural net to model the classification probability

Output

Signal?

Background?

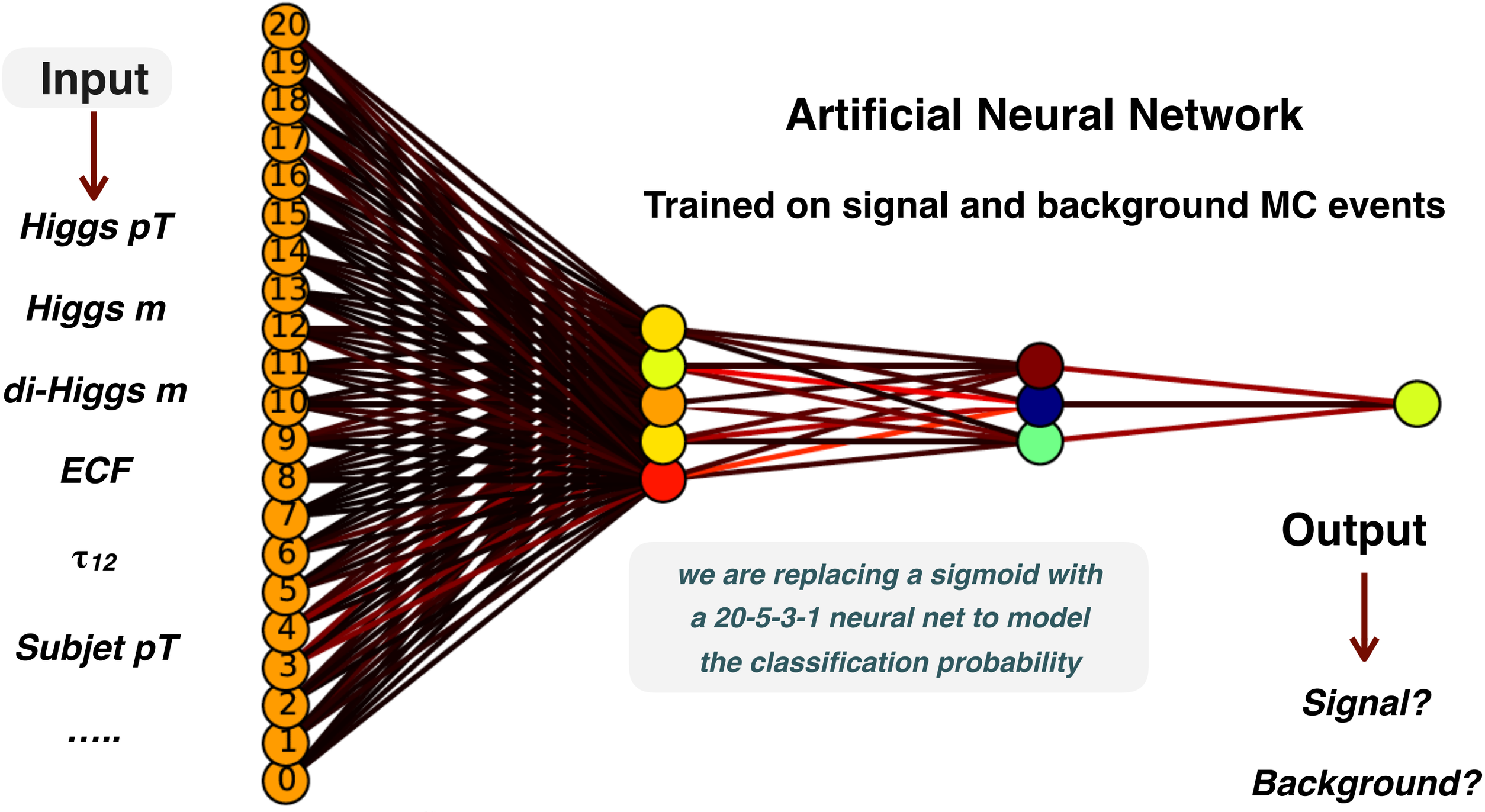
instead of

$$P(y_i = 1 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{x}_i^T \boldsymbol{\theta}}}$$

now we use

$$P(y_i = 1 \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}) = \mathrm{NN}(\boldsymbol{x}_i; \boldsymbol{\theta})$$

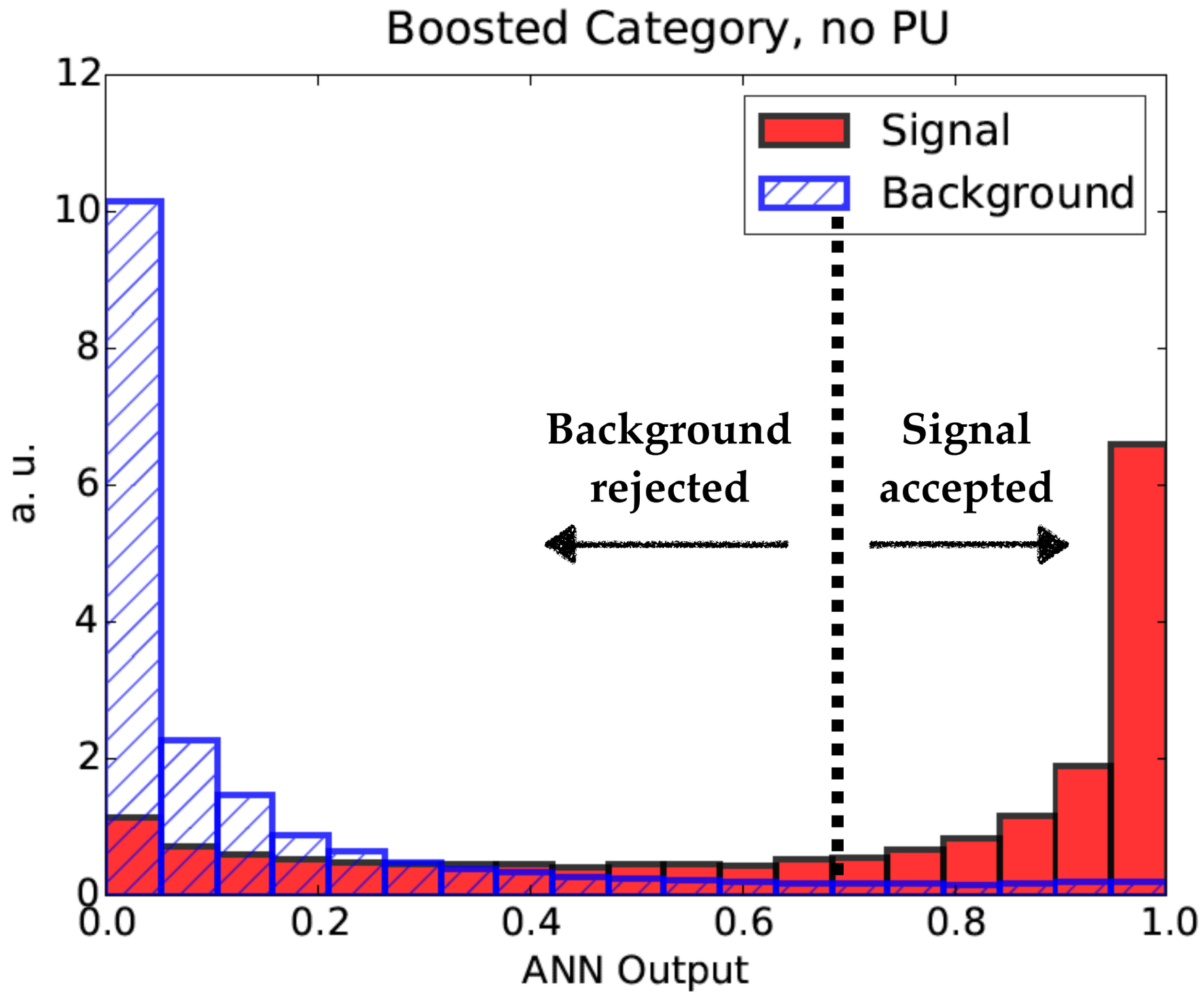
Neural Network Discriminator



$$\hat{\boldsymbol{\theta}} = \arg\,\min_{\theta}\left\{ \sum_{i=1}^{n} \left(-y_i \log \mathrm{NN}(\boldsymbol{x}_i;\boldsymbol{\theta})\right) - (1-y_i)\log(1-\mathrm{NN}(\boldsymbol{x}_i;\boldsymbol{\theta}))\right) \right\}$$

ML discriminator

Optimal signal/background discrimination from ML-driven combination of kinematical info

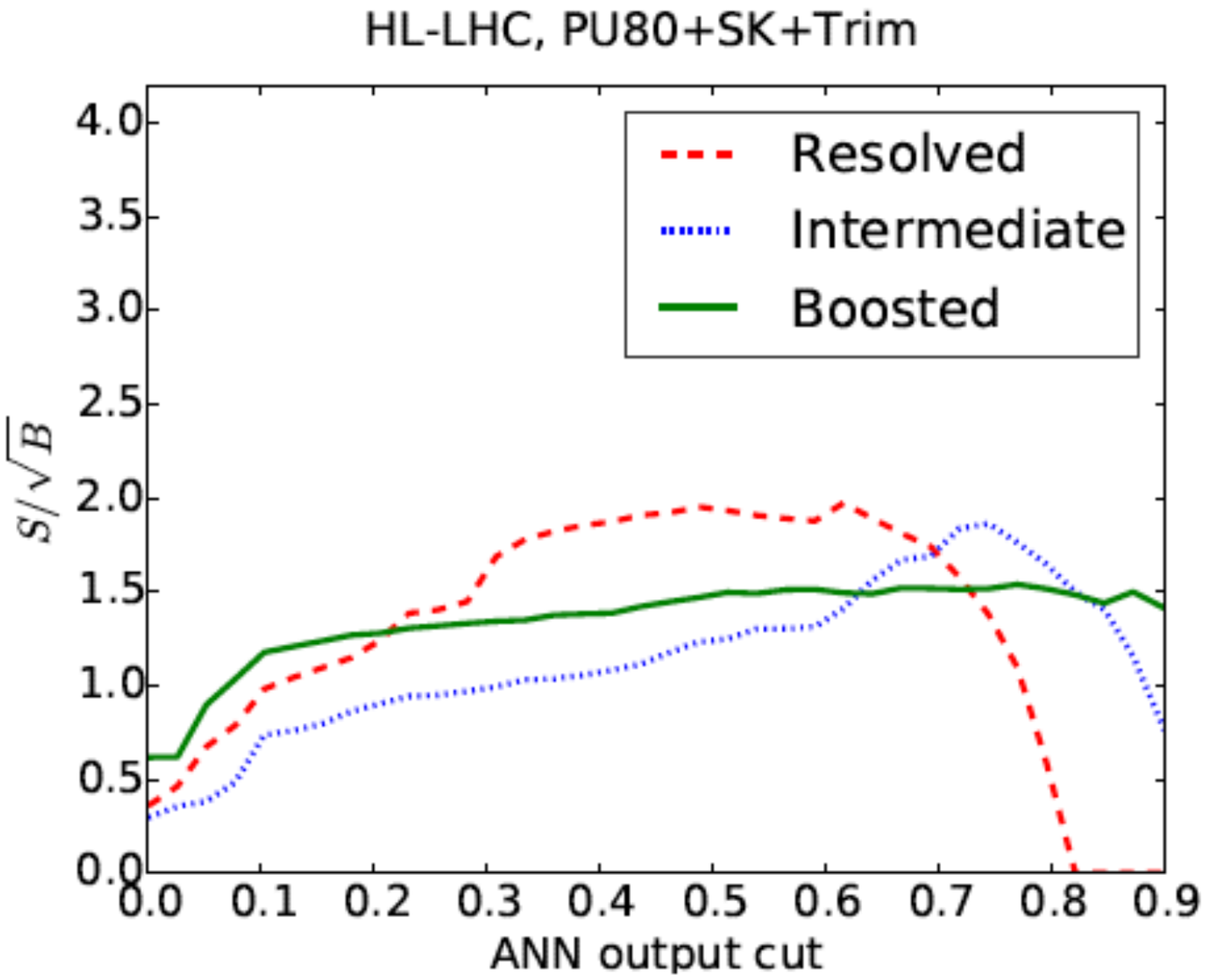


Results

The total combined significance is enough to **observe Higgs pair production in the 4b final state** at the HL-LHC. Substantial improvement if reducible backgrounds (fakes) can be eliminated

$$\left(\frac{S}{\sqrt{B}}\right)_{\text{tot}} \simeq 3.1\ (1.0) \qquad \left(\frac{S}{\sqrt{B_{4\text{b}}}}\right)_{\text{tot}} \simeq 4.7\ (1.5)\,, \quad \mathcal{L} = 3000\ (300)\,\text{fb}^{-1}$$

Pre-ML result

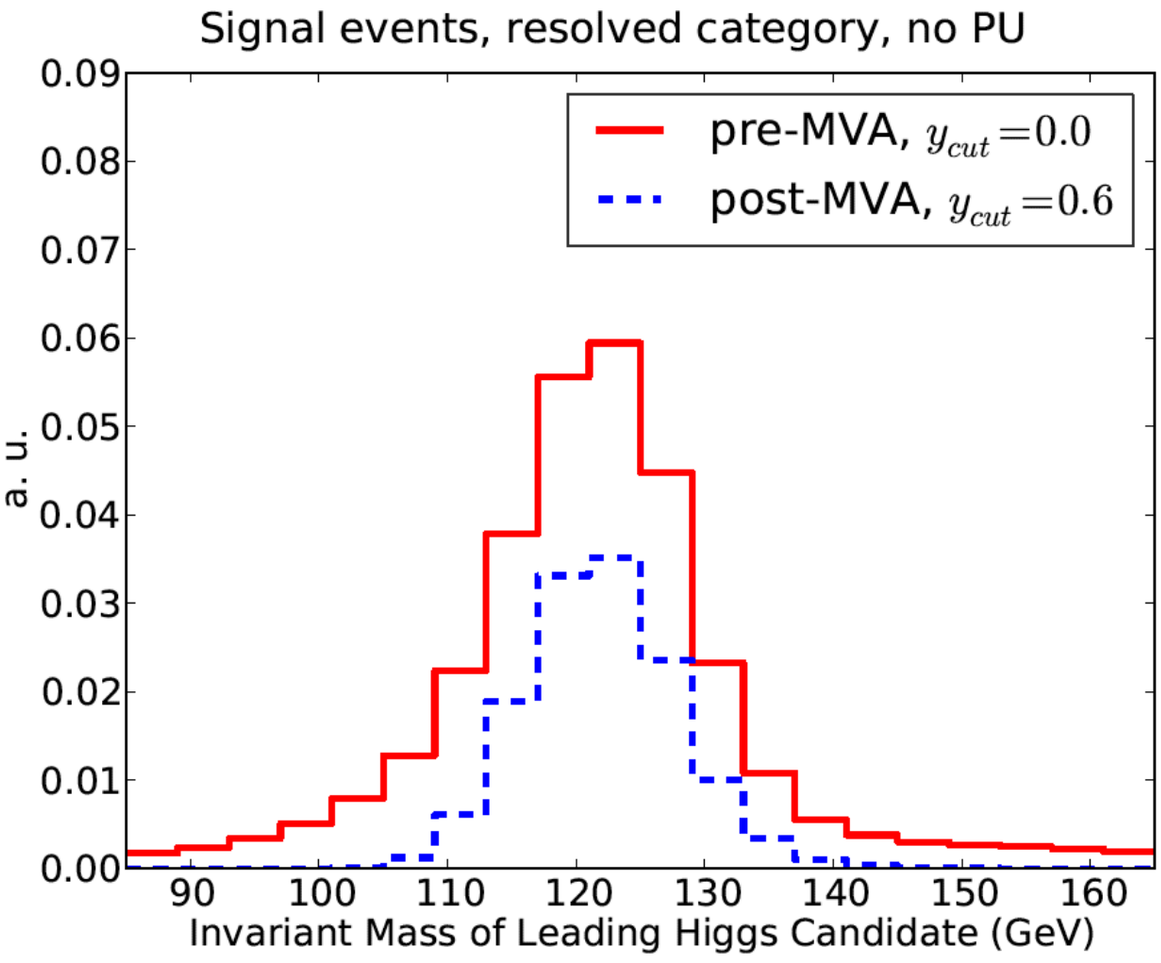


Opening the black box

- ML tools often criticised as **black boxes**, with little understanding of inner working
- ANNs are simply a **set of combined kinematical cuts**, nothing mysterious in them!
- Plot kin distributions after and before the ANN cut to determine the **effective kinematic cuts**
- This info enough to **perform a cut-based analysis** with similar signal significance

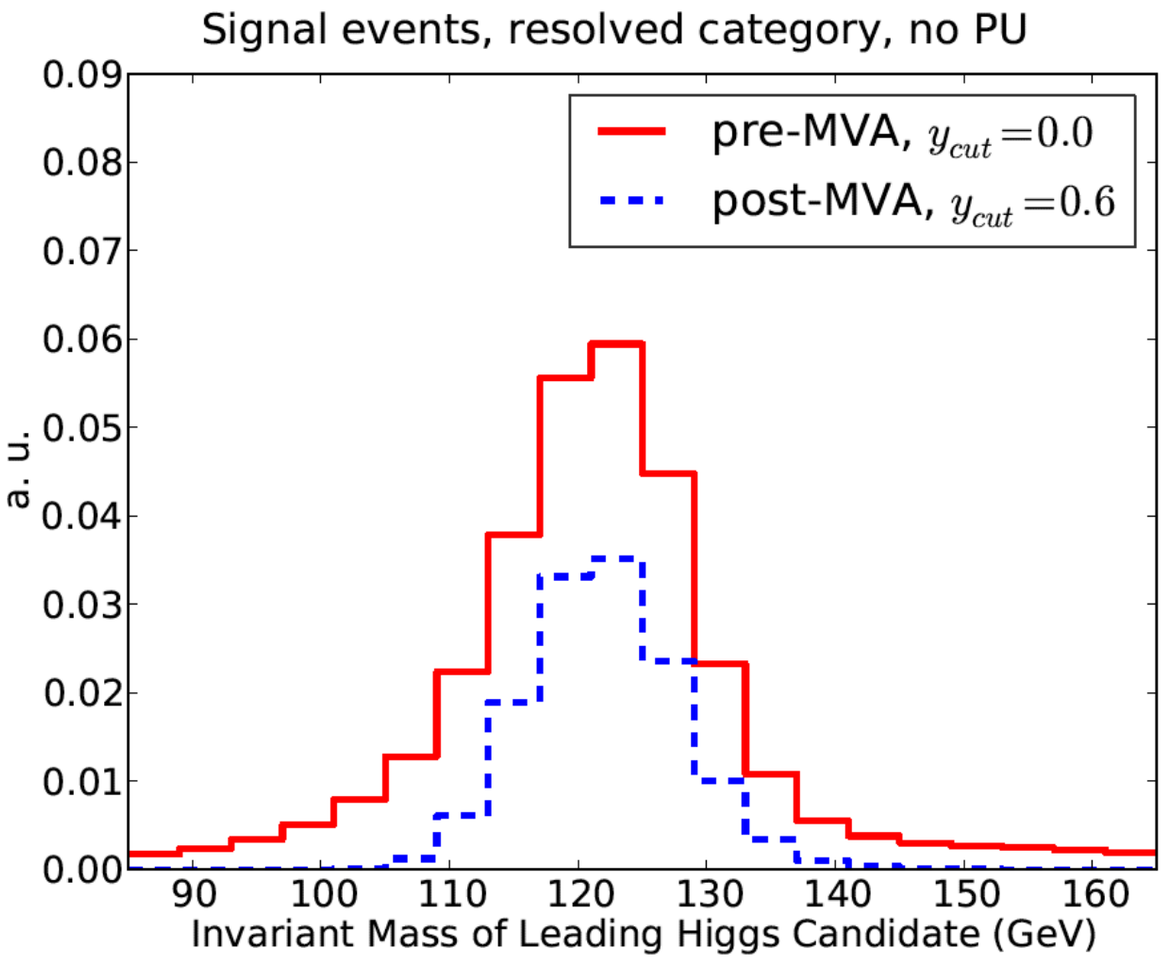
Opening the black box

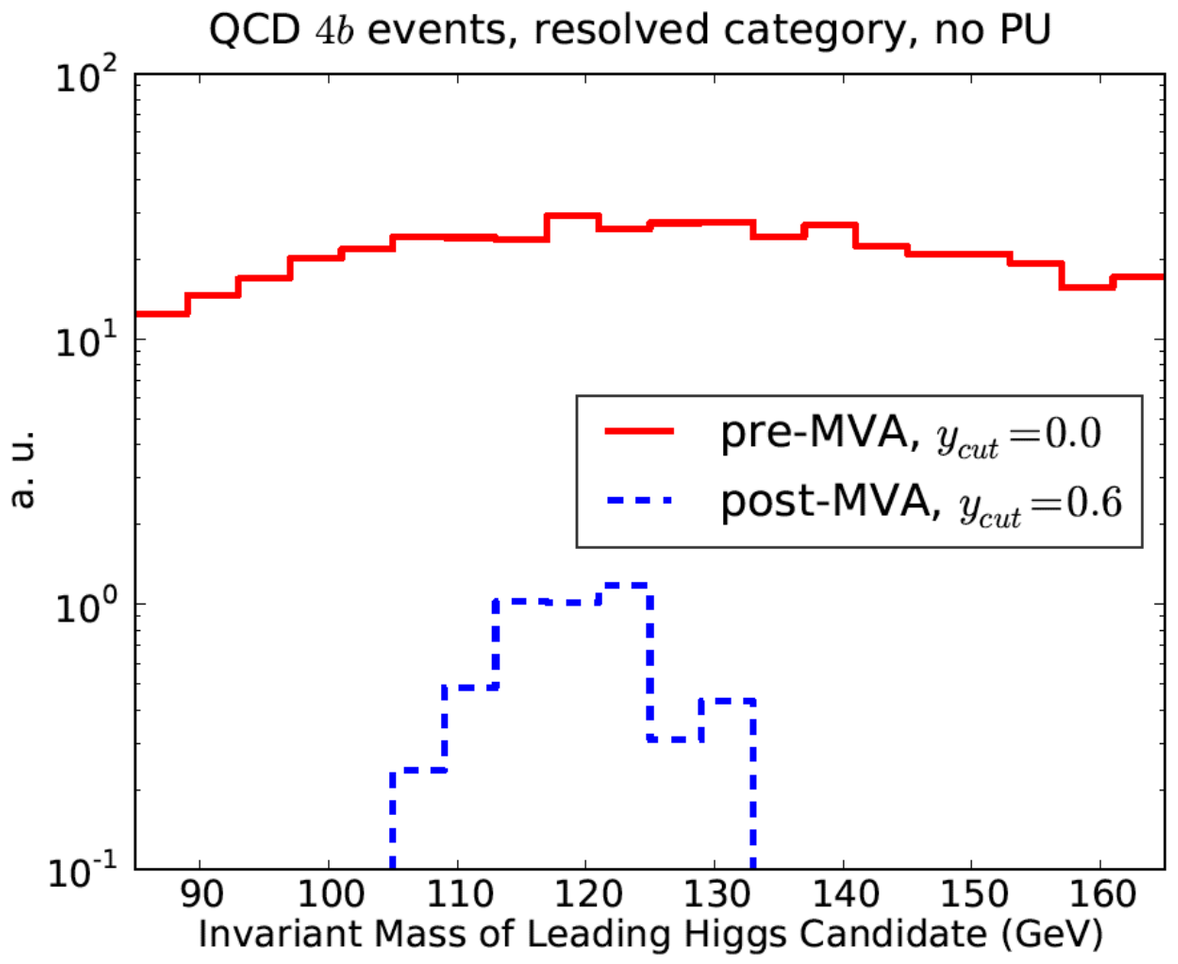
- ML tools often criticised as **black boxes**, with little understanding of inner working
- ANNs are simply a **set of combined kinematical cuts**, nothing mysterious in them!
- Plot kin distributions after and before the ANN cut to determine the **effective kinematic cuts**
- This info enough to **perform a cut-based analysis** with similar signal significance



Opening the black box

- ML tools often criticised as **black boxes**, with little understanding of inner working
- ANNs are simply a **set of combined kinematical cuts**, nothing mysterious in them!
- Plot kin distributions after and before the ANN cut to determine the **effective kinematic cuts**
- This info enough to **perform a cut-based analysis** with similar signal significance





*Higgs-like peak **sculpted** in QCD background*

Tutorial 2 Exercise 2b

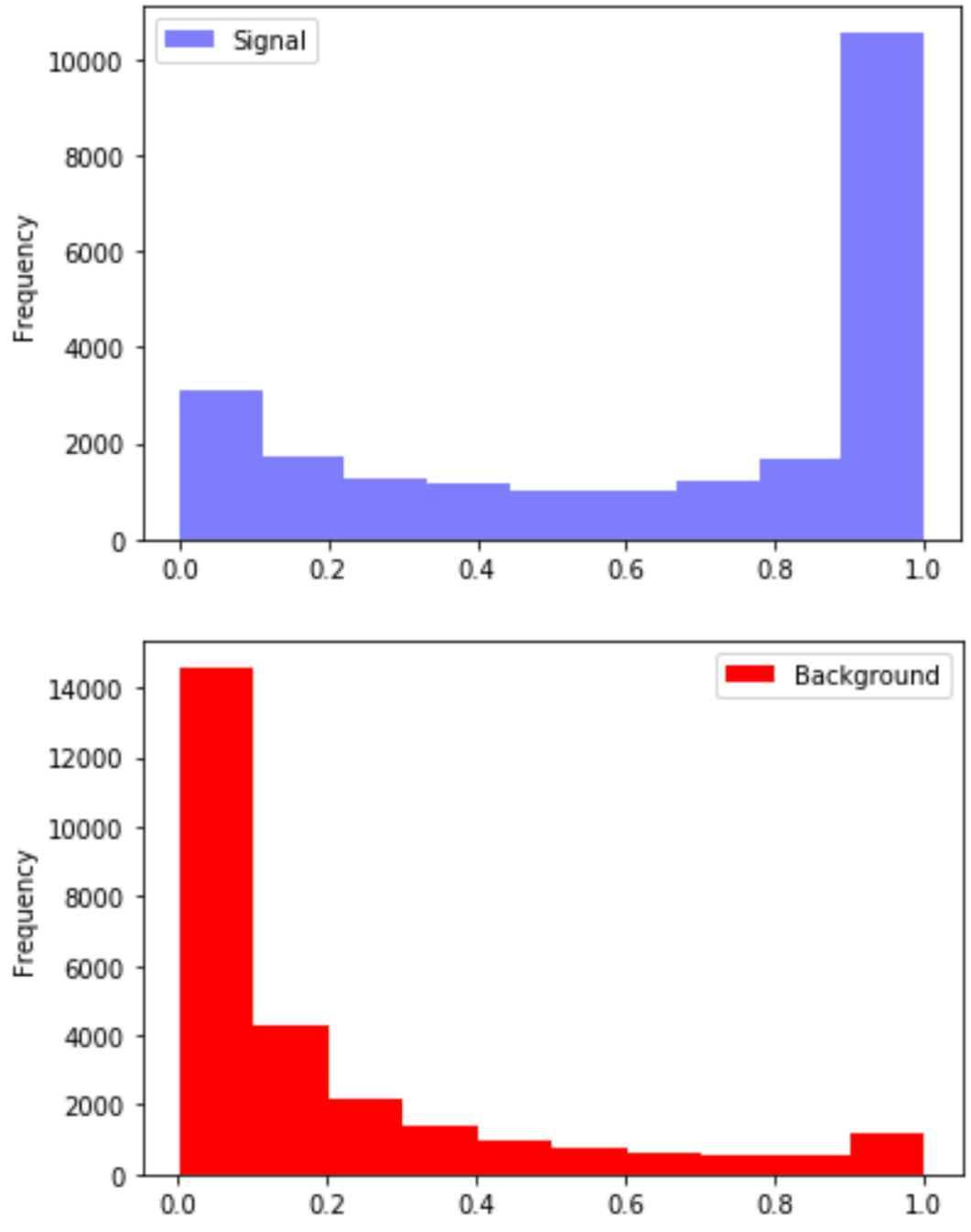
starting point is the **Python script** that you will find in

https://github.com/juanrojochacon/ml-ditp-attp/blob/master/Tutorials/Tutorial2/

- Train a **classifier** to classify signal events (supersymmetric particle production) with respect to background events at the LHC

- Determine the settings that lead to the **best discrimination** quantified by the largest ratio of TPR to NPR

- How does the performance of the classifier depend as we vary the size of the training dataset?



Statistical and Bayesian learning

Statistical Learning Theory

Our starting point is the **underlying law** ***y=f(x)*** which we aim to learn from a dataset (x_i,y_i), $i=1,\ldots,N$. We will also need an **hypothesis set** ***H*** containing all functions that we consider to be good candidates for the underlying law

The goal of **Statistical Learning Theory** is to determine a function from the hypothesis set *H* that approximates *f(x)* as best as possible, ideally in a strict mathematical limit

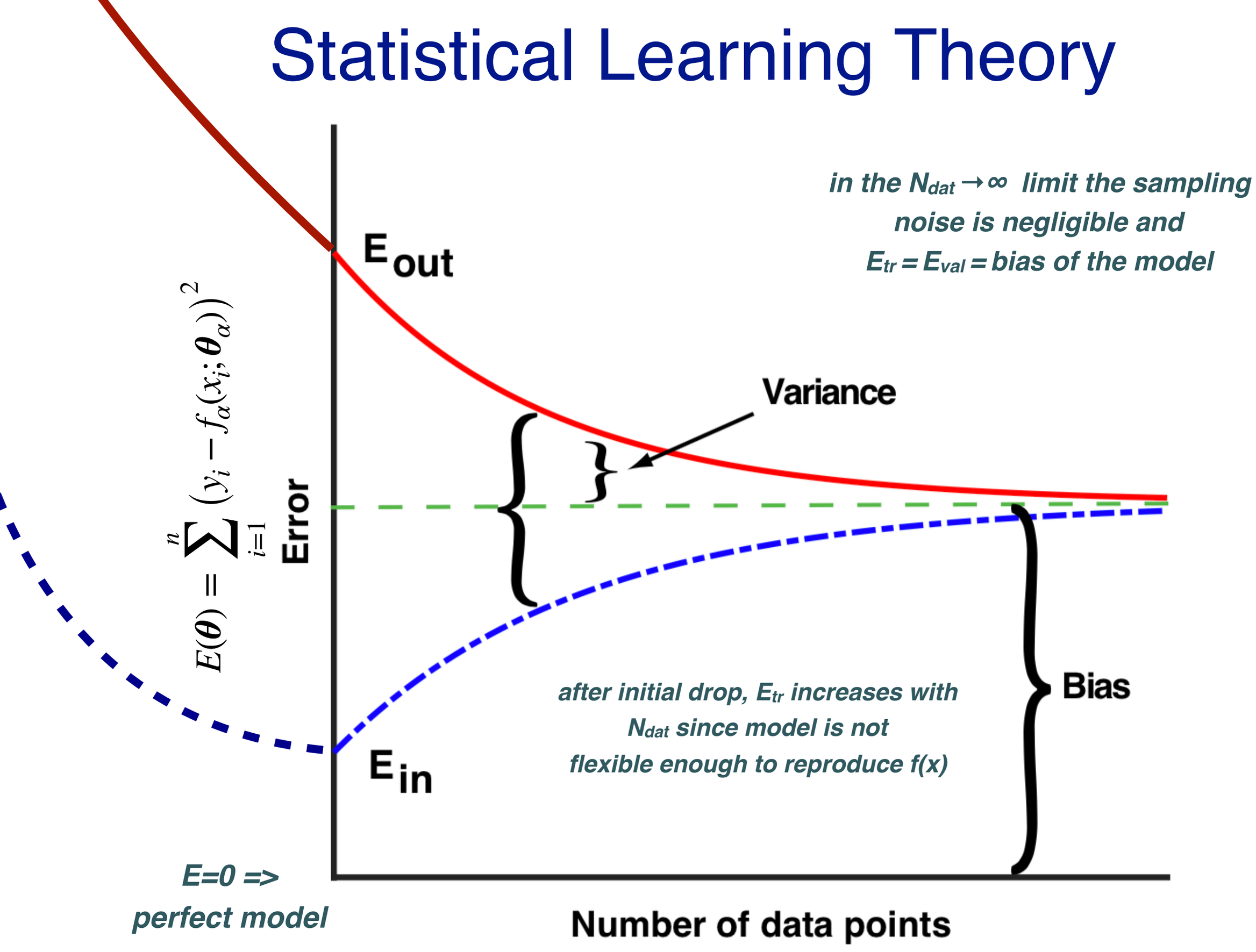
Here we will provide an **intuitive picture** of how Statistical Learning works

Consider the following situation:

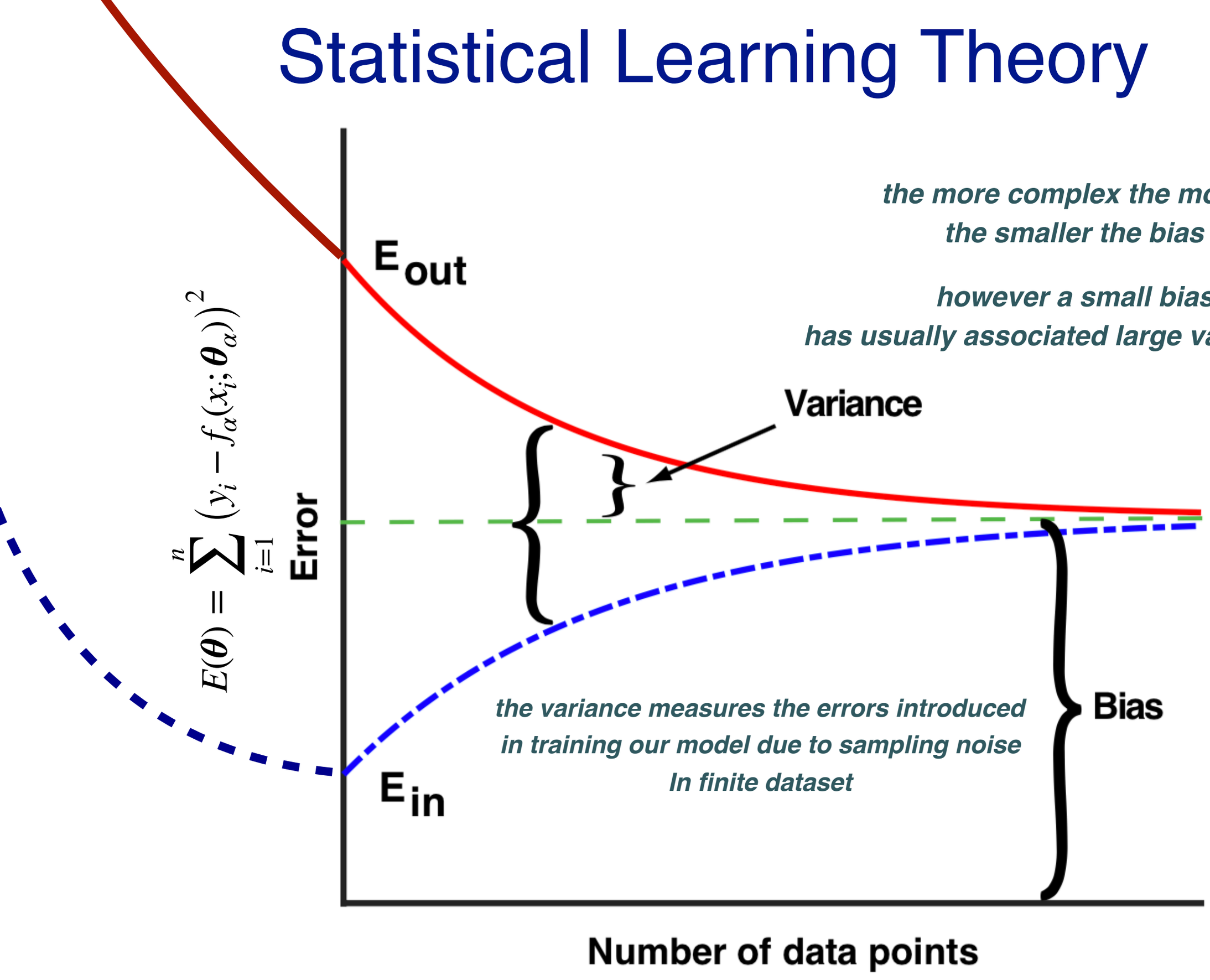
- The **underlying law** is so complex that we cannot aim to exactly reproduce *f(x)*
- We want to study the dependence of E_{tr} and E_{val} with the number of data points

This setting allows us to present one of the most important concepts in the theory of Machine Learning: **the bias-variance tradeoff**

Statistical Learning Theory



Statistical Learning Theory



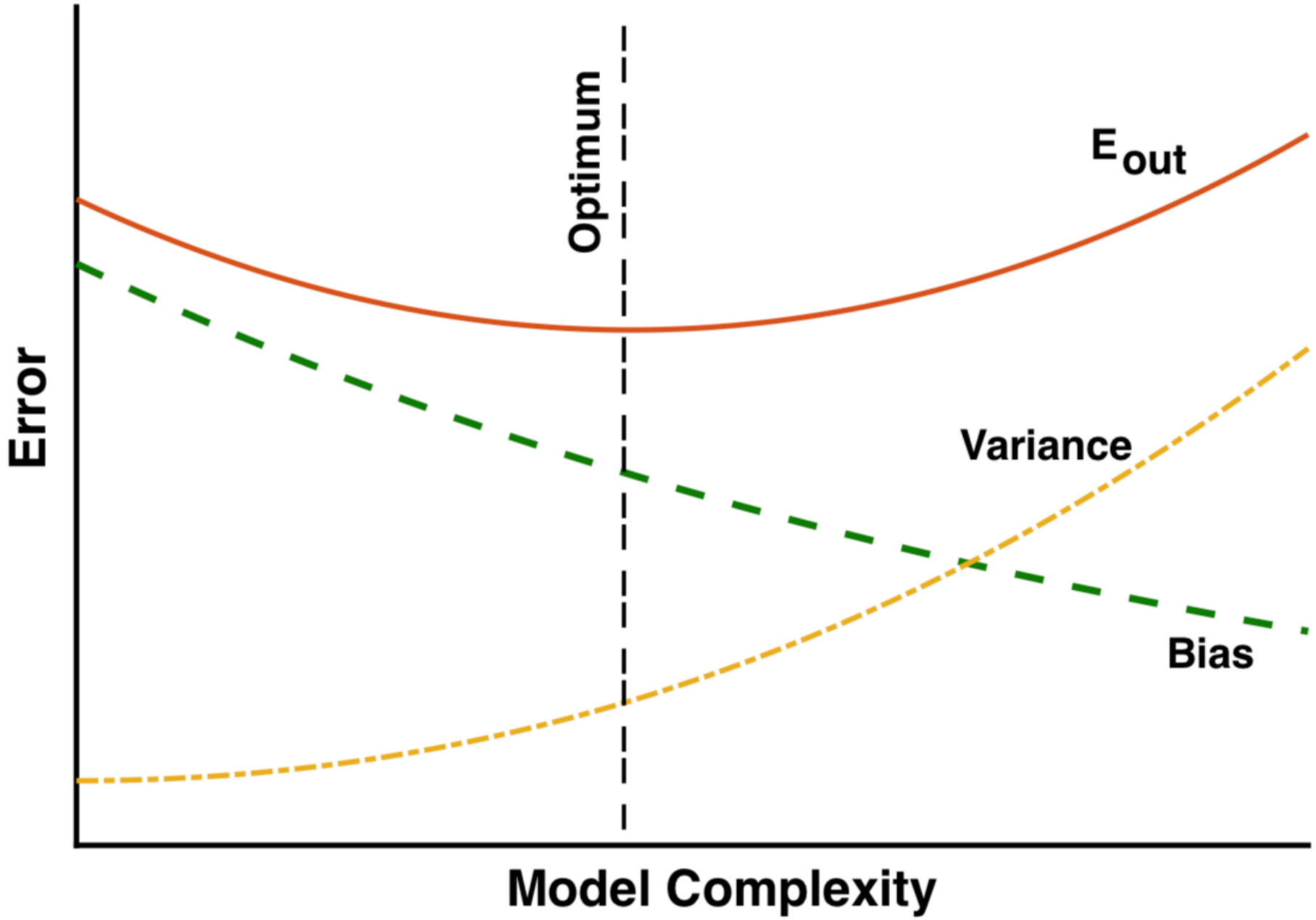
the more complex the model,
the smaller the bias

however a small bias
has usually associated large variance ….

the variance measures the errors introduced
in training our model due to sampling noise
In finite dataset

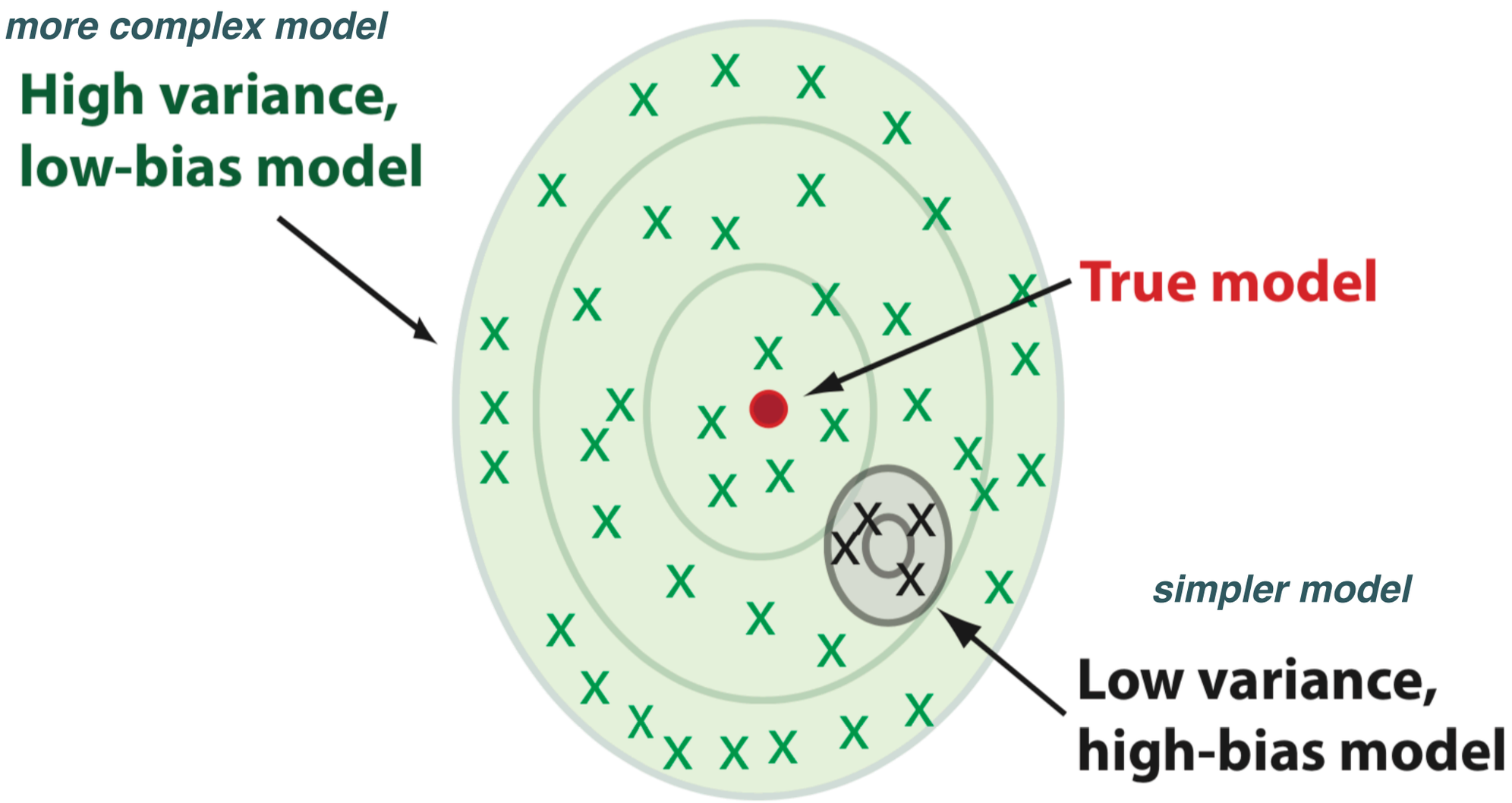
The bias-variance tradeoff

Optimal model performance (measured by minimising the generalisation error E_{val}) is typically achieved at intermediate levels of model complexity: the **bias-variance tradeoff**



The bias-variance tradeoff

Optimal model performance (measured by minimising the generalisation error E_{val}) is typically achieved at intermediate levels of model complexity: the **bias-variance tradeoff**



The bias-variance tradeoff

we can provide further insight on the bias-variance tradeoff with a simple scenario

assume a **model** extracted from a **training set** $\longrightarrow$ $\hat{y}(\boldsymbol{x}_{\rm tr})$

and that the true underlying law is given by

$$Y = y(\boldsymbol{X}) + \epsilon\,, \qquad y(\boldsymbol{x}_{\rm tr}) = E(Y, \boldsymbol{X} = \boldsymbol{x}_{\rm tr})$$

zero-mean, fixed variance noise

take a data point **outside the training set, *(x₀,y₀)***, and compute **its variance** in our model

$$E[\left(Y(x_0) - \hat{y}(x_0)\right)^2] = \left(y_0^2 - 2y_0 E[\hat{y}(x_0)] + E[\hat{y}^2(x_0)] + E[\epsilon^2]\right)$$

underlying law *model*

where we set to zero terms linear in the stochastic noise

The bias-variance tradeoff

we can provide further insight on the bias-variance tradeoff with a simple scenario

assume a **model** extracted from a **training set** $\longrightarrow$ $\hat{y}(\boldsymbol{x}_{\rm tr})$

and that the true underlying law is given by

$$Y = y(\boldsymbol{X}) + \epsilon\,, \qquad y(\boldsymbol{x}_{\rm tr}) = E(Y, \boldsymbol{X} = \boldsymbol{x}_{\rm tr})$$

zero-mean, fixed variance noise

take a data point **outside the training set, *(x_0,y_0)***, and compute **its variance** in our model

$$E[\left(Y(x_0) - \hat{y}(x_0)\right)^2] = \left(\text{Bias}[\hat{y}(x_0)]\right)^2 + \text{Var}[\hat{y}(x_0))] + \text{Var}(\epsilon)$$

$$\text{Bias}[\hat{y}(x_0))] = E[\hat{y}(x_0)] - y_0$$

$$\text{Var}[\hat{y}(x_0))] = E[\hat{y}^2(x_0)] - E[\hat{y}(x_0)]^2$$

the **model generalisation power** decreases both with its bias and its variance

Bayesian Inference

Bayesian inference a method of statistical inference in which **Bayes' theorem** is used to **update the probability for an hypothesis** as more information becomes available

in Bayesian statistics there are two main ingredients:

likelihood function

$$p(\boldsymbol{X}\,|\,\boldsymbol{\theta})$$

probability of observing the dataset $\boldsymbol{X}$ given model parameters $\boldsymbol{\theta}$

prior distribution

$$p(\boldsymbol{\theta})$$

knowledge of model parameters before we see the data **X**

which are used to compute the **posterior distribution** using **Bayes' Theorem**

$$p(\boldsymbol{\theta}\,|\,\boldsymbol{X}) = \frac{p(\boldsymbol{X}\,|\,\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int d\boldsymbol{\theta}' p(\boldsymbol{X}\,|\,\boldsymbol{\theta}')p(\boldsymbol{\theta}')}$$

probability of the model parameters $\boldsymbol{\theta}$ after observing the dataset $\boldsymbol{X}$

Bayesian Inference

Many common statistical features such as least-squares fitting can be understood as carrying out a **Maximum Likelihood Estimation** (MLE)

In MLE one selects model parameters that **maximise likelihood** of observed data

$$\hat{\boldsymbol{\theta}} = \arg\ \max_{\boldsymbol{\theta}} \log p(\boldsymbol{X}\,|\,\boldsymbol{\theta})$$

A central ingredient of Bayesian Inference is related to the **choice of prior**

no knowledge on model parameters ⟶ ***uninformative prior***

some knowledge on model parameters ⟶ ***informative prior***

an informative prior tends to decrease the variance of posterior distribution while potentially, increasing bias

e.g. Gaussian prior $p(\boldsymbol{\theta}, \lambda) = \prod_j \sqrt{\frac{\lambda}{2\pi}} e^{-\lambda\theta_j^2}$ ***justified from belief that many parameters will be small***

Bayesian Inference

once we have computed using Bayesian Inference the posterior distribution, we can evaluate the **optimal values** of the model parameters

posterior mean $\longrightarrow$ $\langle \boldsymbol{\theta} \rangle = \int d\boldsymbol{\theta}\, \boldsymbol{\theta}\, p(\boldsymbol{X}\,|\,\boldsymbol{\theta})$

posterior mode $\longrightarrow$ $\hat{\boldsymbol{\theta}} = \arg\ \max_{\theta} p(\boldsymbol{X}\,|\,\boldsymbol{\theta})$

If the prior depends on **hyperparameters such as λ** then one needs also to determine a suitable range for them (a hierarchical prior)

Bayesian methods are very powerful in Machine Learning: Bayesian networks, multinomial & Gaussian naive Bayes….