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

Today's lecture

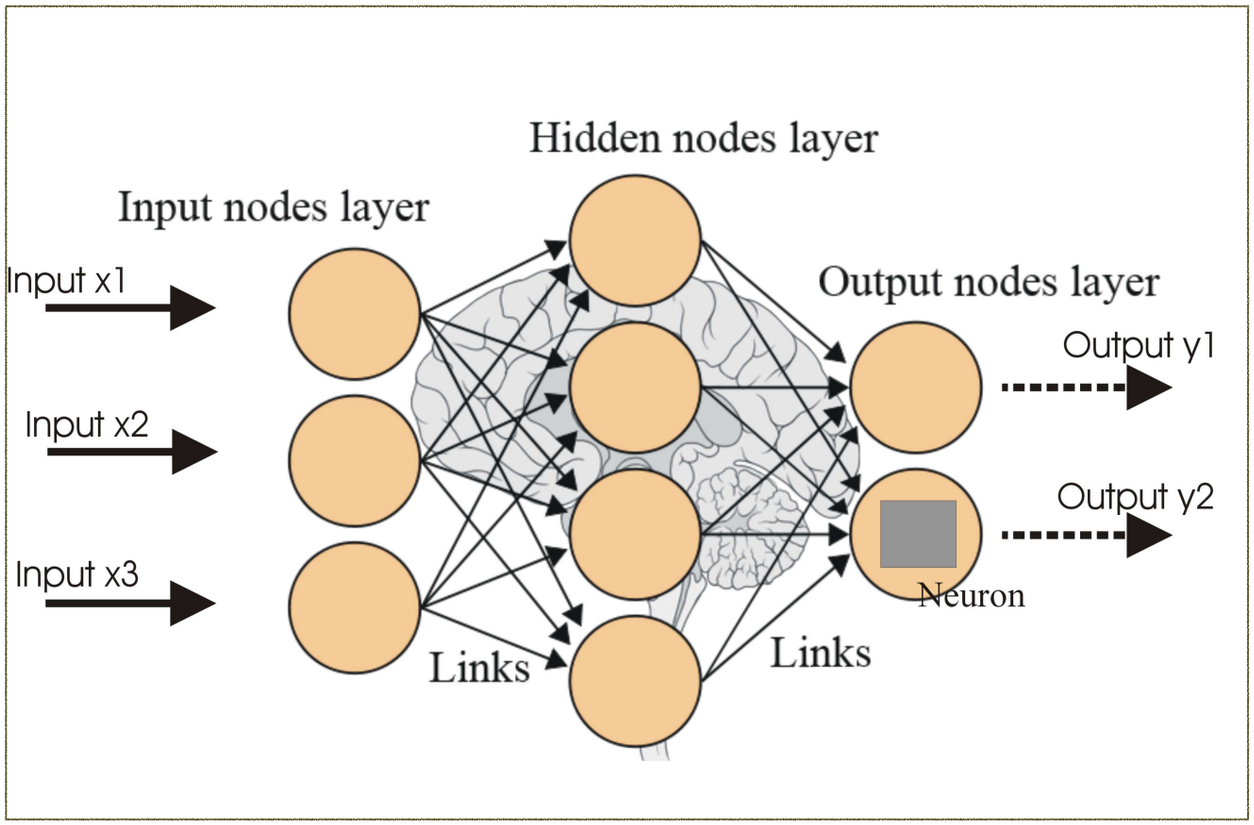
- (Deep) Neural Networks
- Backpropagation in Gradient Descent
- Regularisation of Neural Networks (cross-validation)
- Case Study I: the proton structure from neural nets
- Supervised Learning for Classification: logistic regression

Tutorial 2: (a) Neural Network Training

(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**, ...



Neural Networks

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

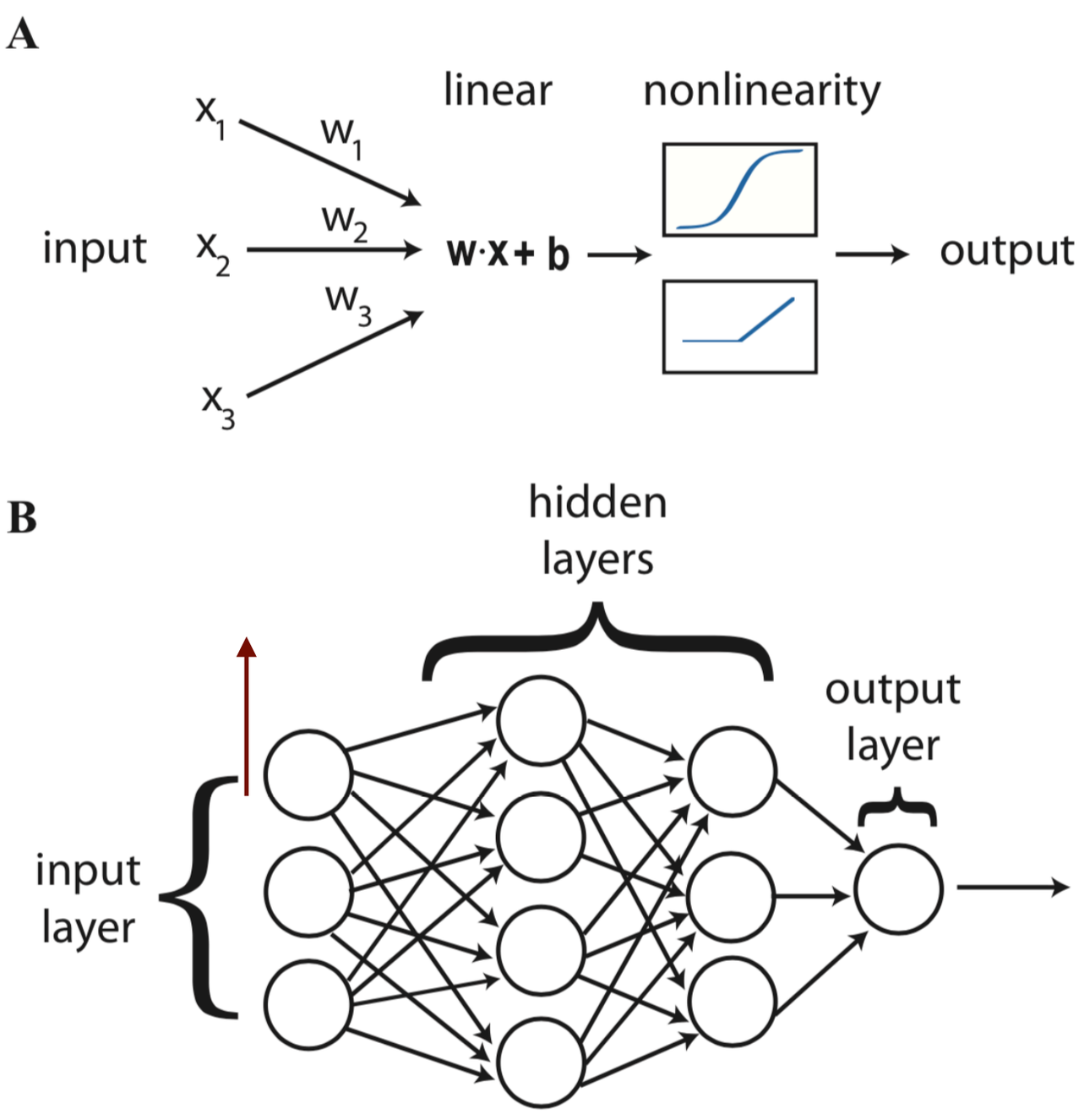
- The basic unit of a NN is the **neuron,** a transformation of a set of *d* input features into a scalar output

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

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

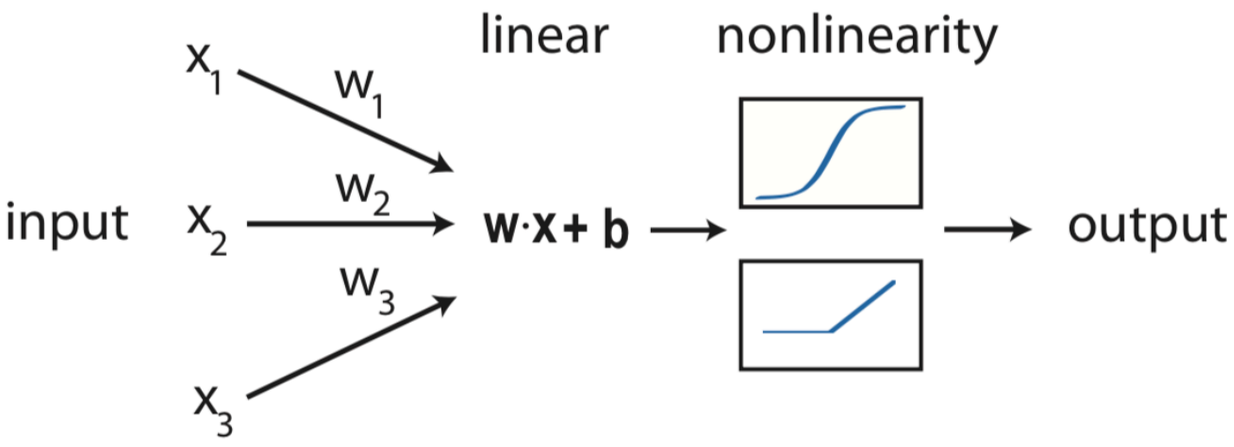
$$z^{(i)} = \boldsymbol{x}^T \cdot \boldsymbol{\theta}^{(i)} + \boldsymbol{\theta}_0^{(i)}$$

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

a **non-linear transformation**

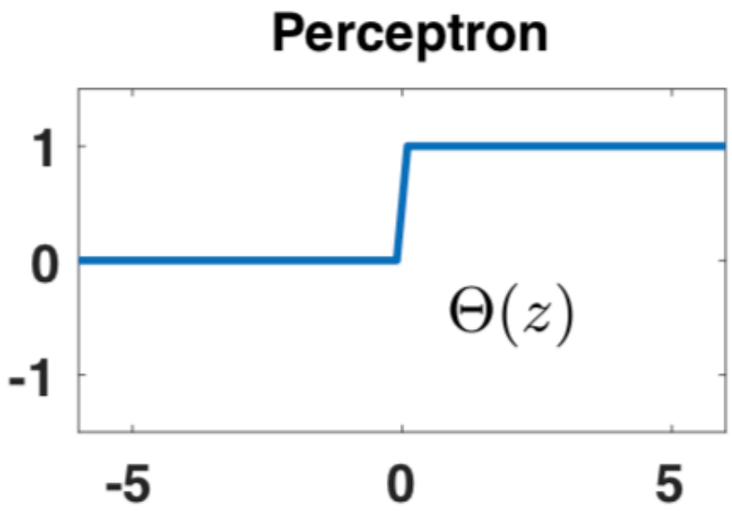
$$a^{(i)}(\boldsymbol{x}) = \sigma_i(z^{(i)})$$

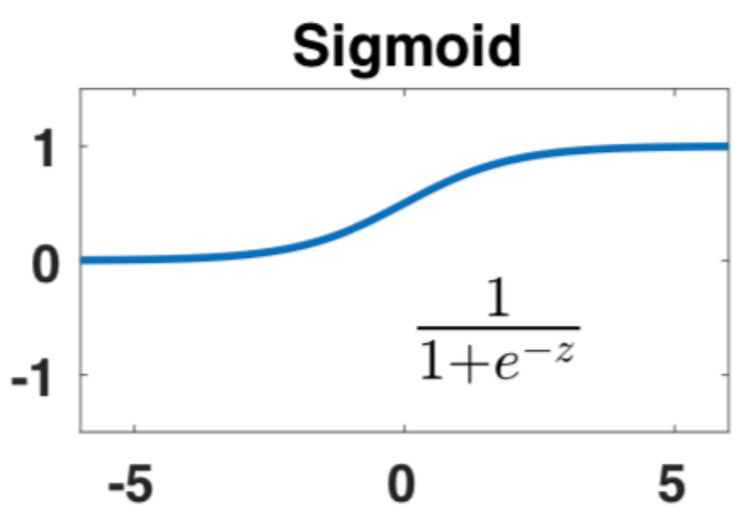
i-th neuron activation state — *i-th neuron activation function (non-linear)*

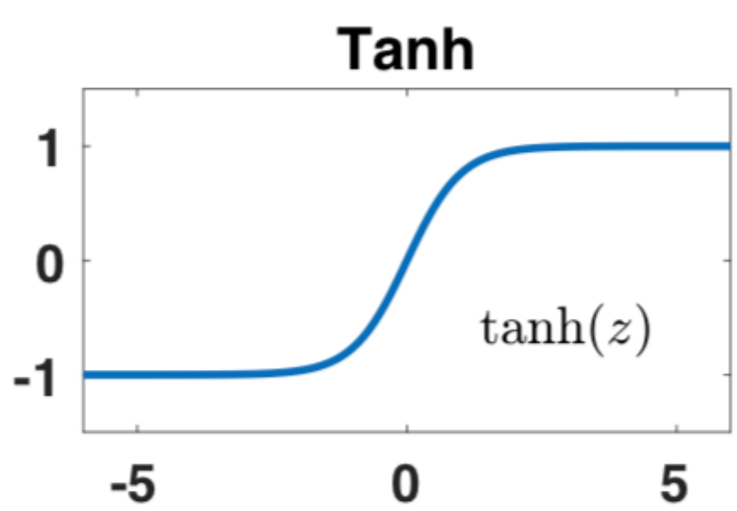


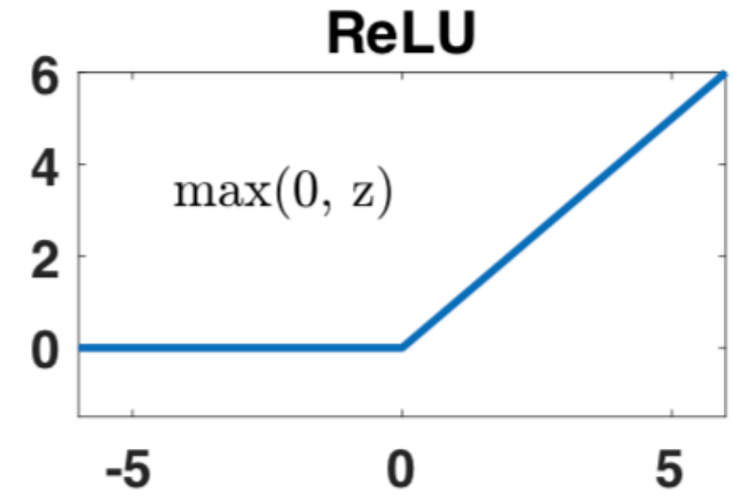
the choice of non-linear activation function affects the **computational and training properties** of the neural nets, since they modify the output gradients required for GD training

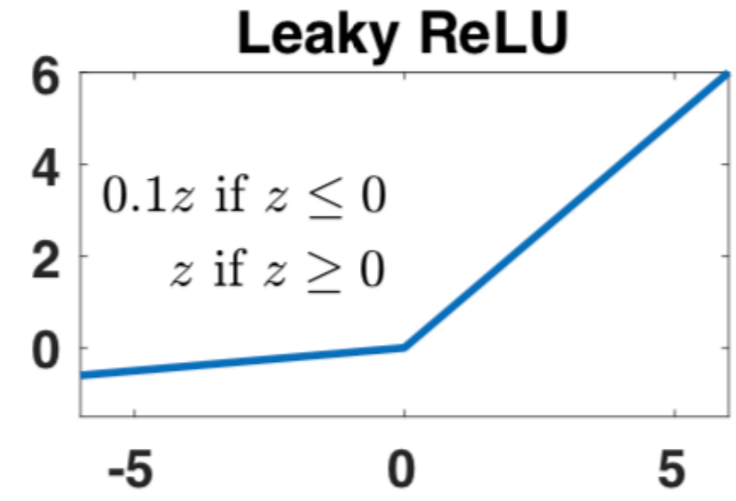
Activation functions

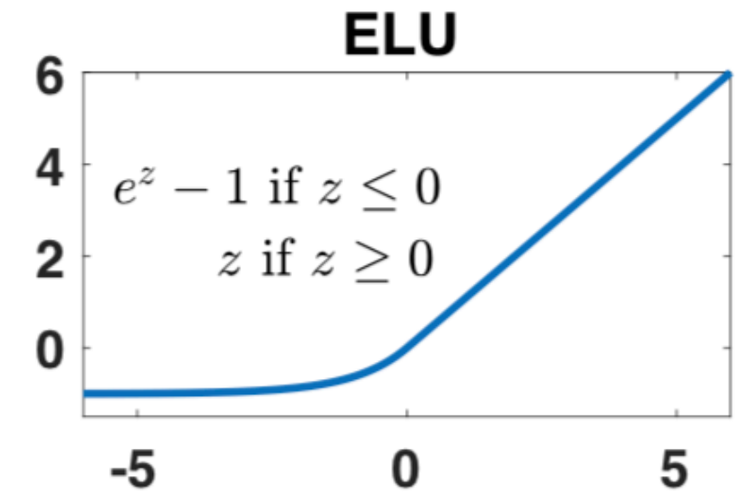












- Activation functions can be classified between those that saturate at large inputs (*e.g.* sigmoid) and those that **do not saturate at large inputs** (*e.g.* Rectified Linear Units ReLU)

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

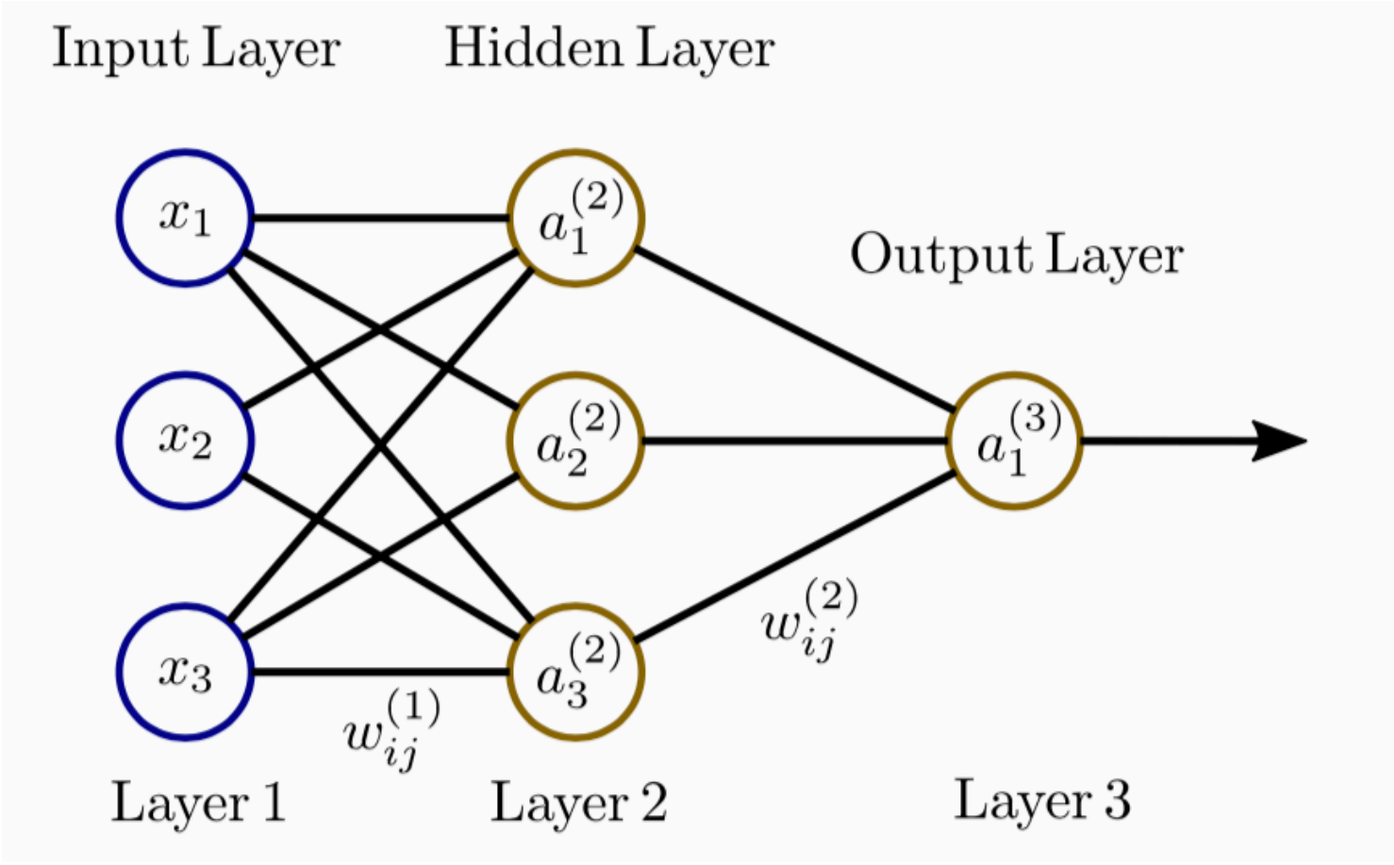
sigmoid $\longrightarrow \left.\frac{d\sigma}{dz}\right|_{z\gg 1} = 0$

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

vanishing gradients are problematic for deep networks

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

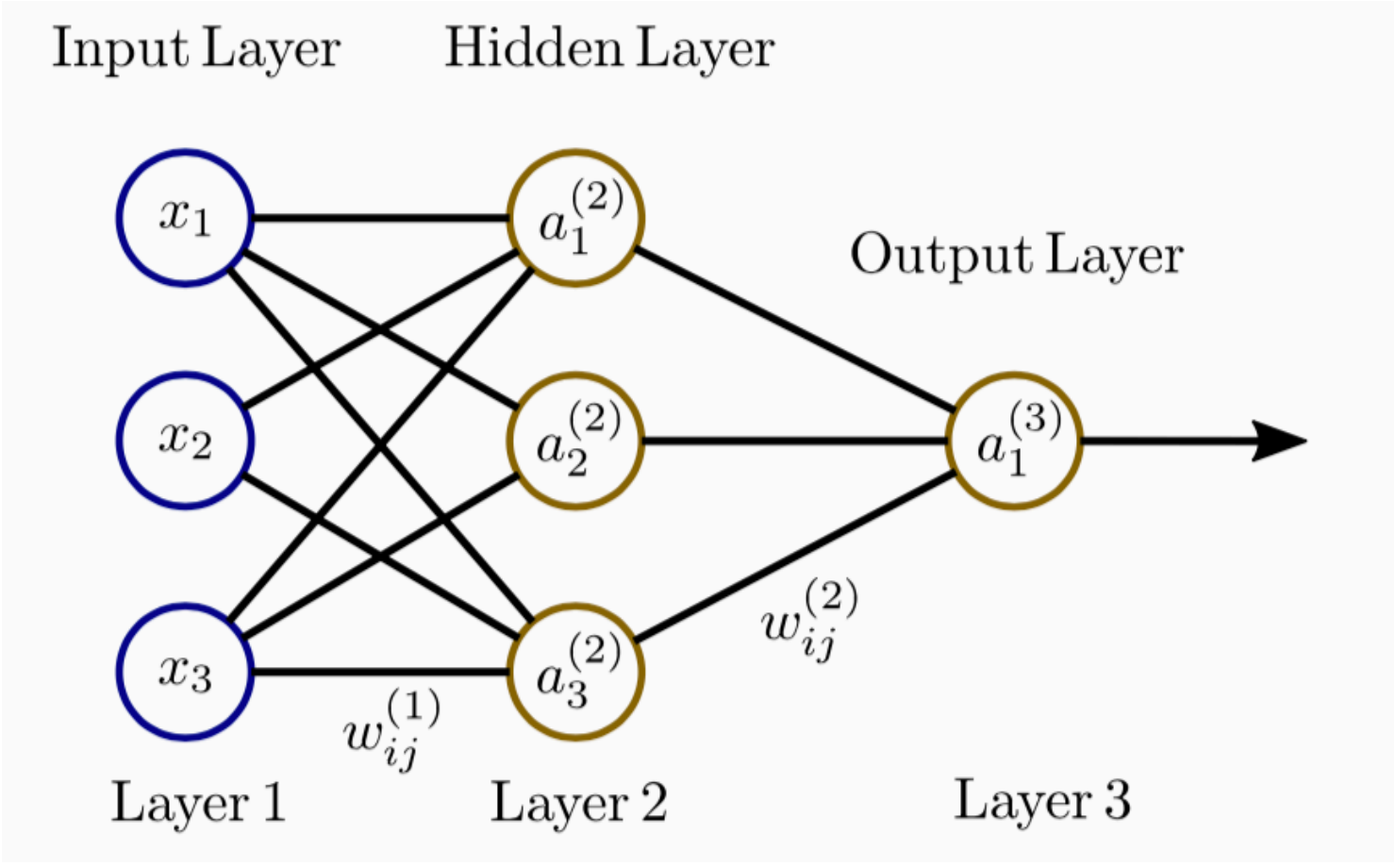
$$a_1^{(2)} = \sigma\left(x_1\theta_{11}^{(1)} + x_2\theta_{12}^{(1)} + x_3\theta_{13}^{(1)} + \theta_{10}^{(1)}\right)$$

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

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

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

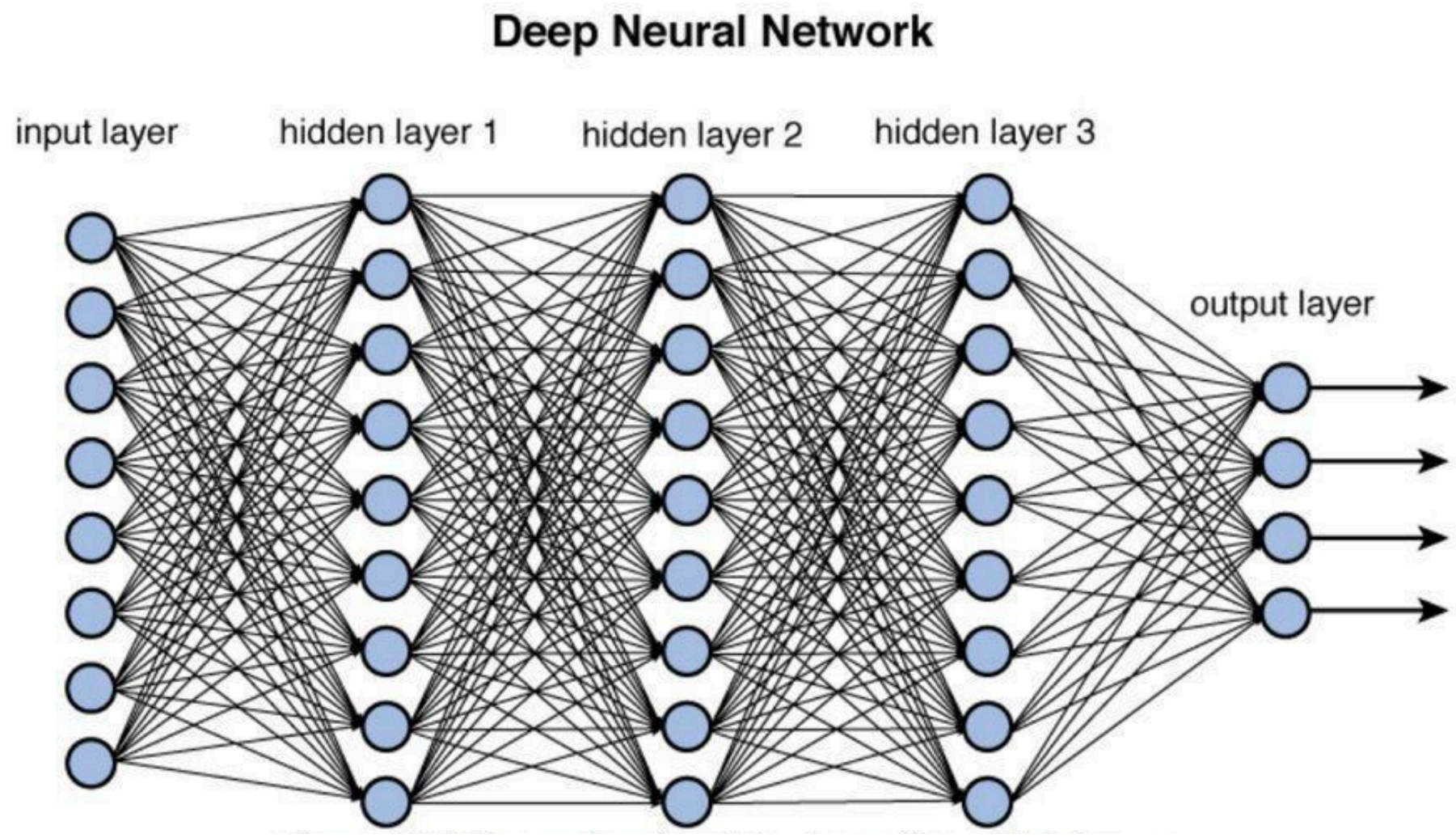
$$a_1^{(3)} = \sigma\left(a_1^{(2)}\theta_{11}^{(2)} + a_2^{(2)}\theta_{12}^{(2)} + a_3^{(2)}\theta_{13}^{(2)} + \theta_{10}^{(2)}\right)$$

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

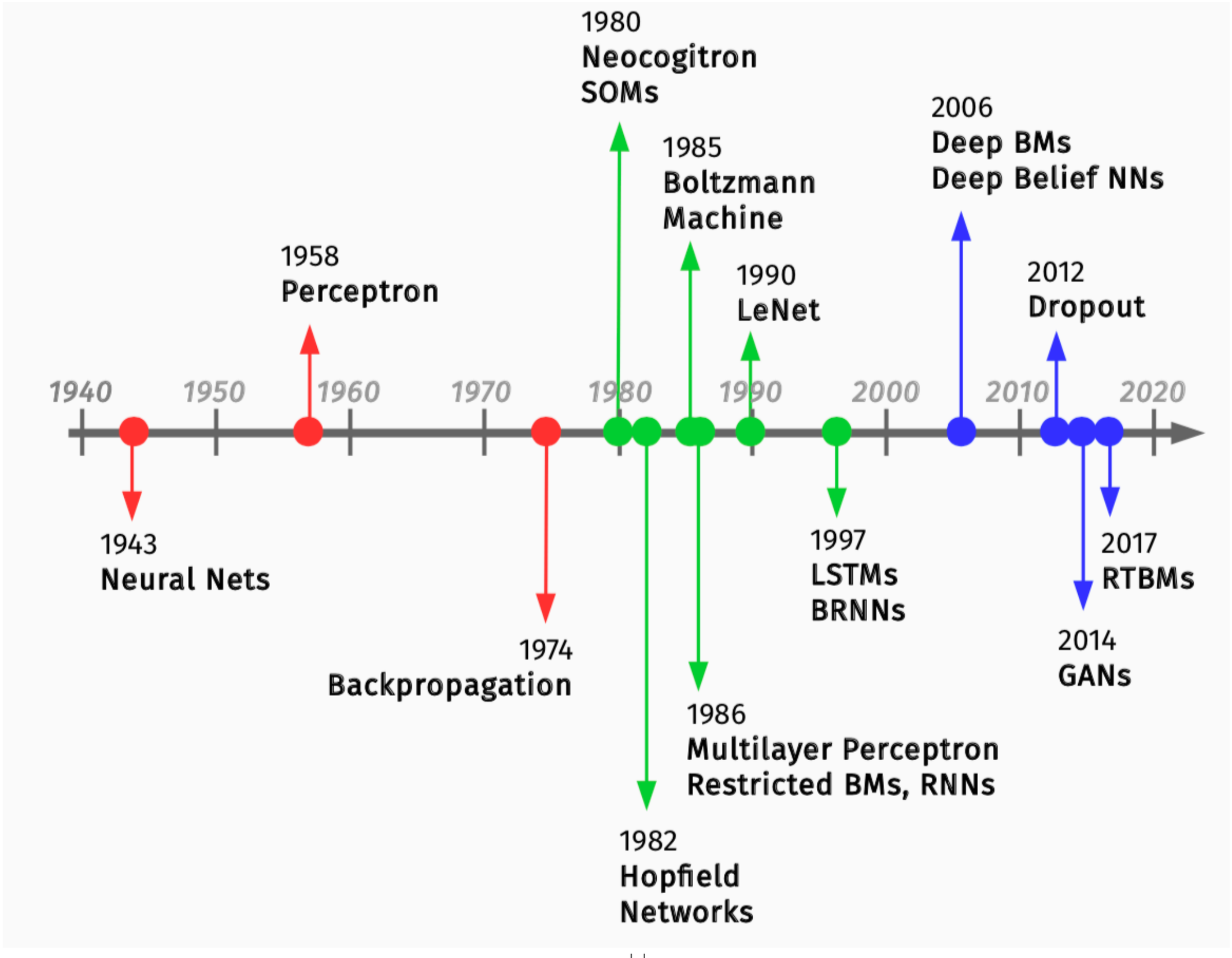
Deep Networks

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**

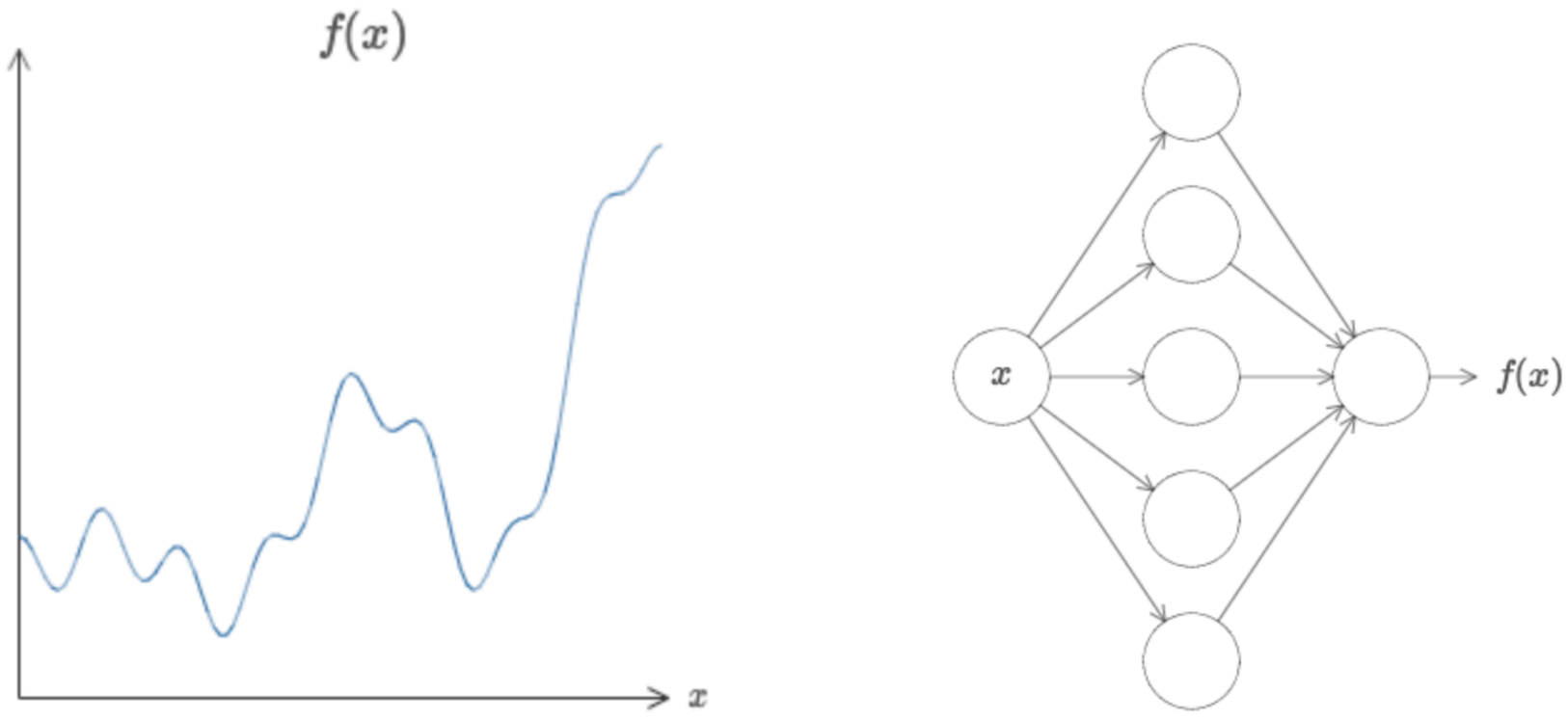


A timeline of neural networks



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

NN training

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

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

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***

instead one can use **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

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

the cost function of the NN therefore depends on:

- $a_j^{(L)}$: **activation states** of the neurons on the **output layer** (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

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

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 rest of back propagation equations, we will use the chain rule

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

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

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

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

Backpropagation

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

- ***(1)*** Evaluate activation state of neurons in input layer $a_j^{(1)}$

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

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

- ***(3)*** With this information evaluate error on neurons output layer

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

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

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

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**

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

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

Hyperoptimisation

In many 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 and number of neurons per layer
- Learning rate and dropout rate

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 hyper optimisation 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

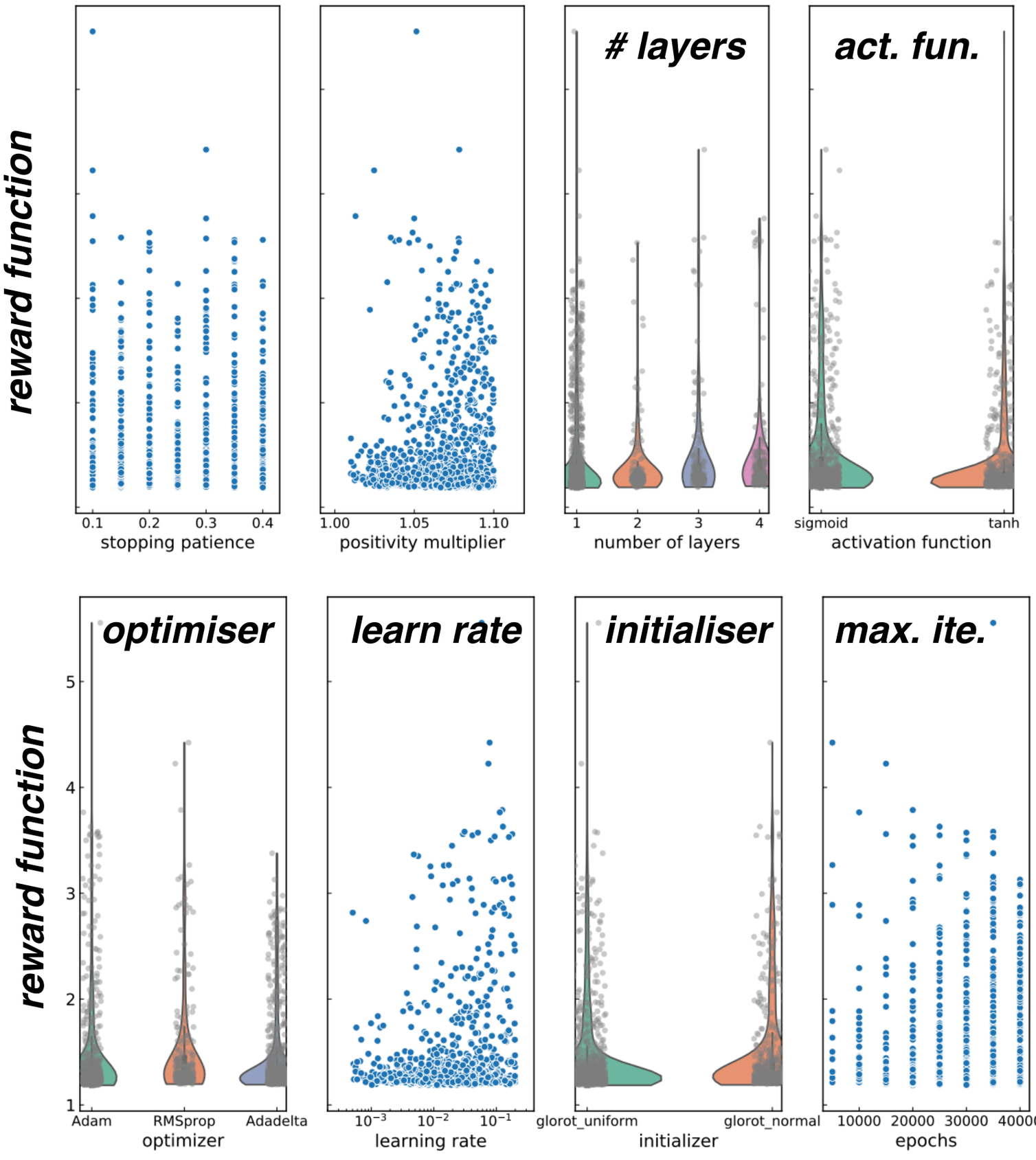
Hyperparameter scan

- 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



Juan Rojo

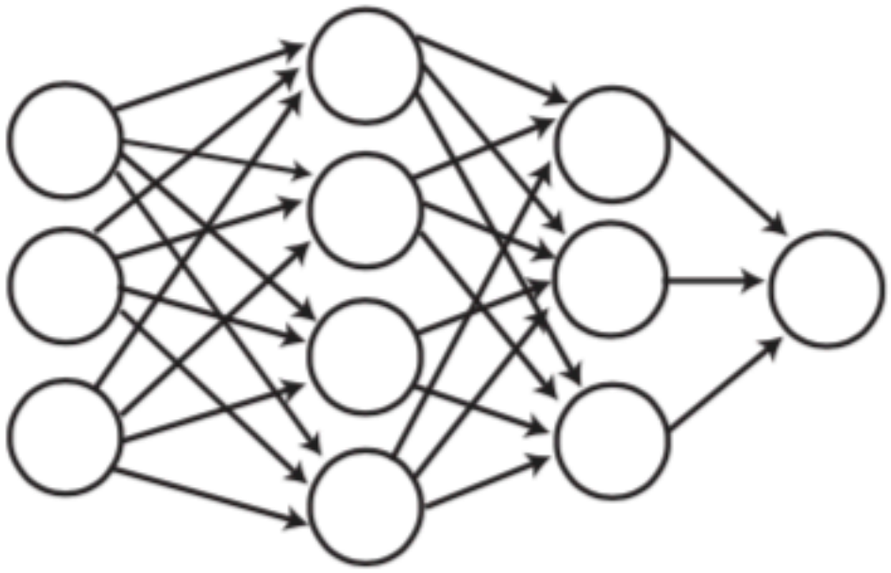
NN regularisation: cross-validation

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

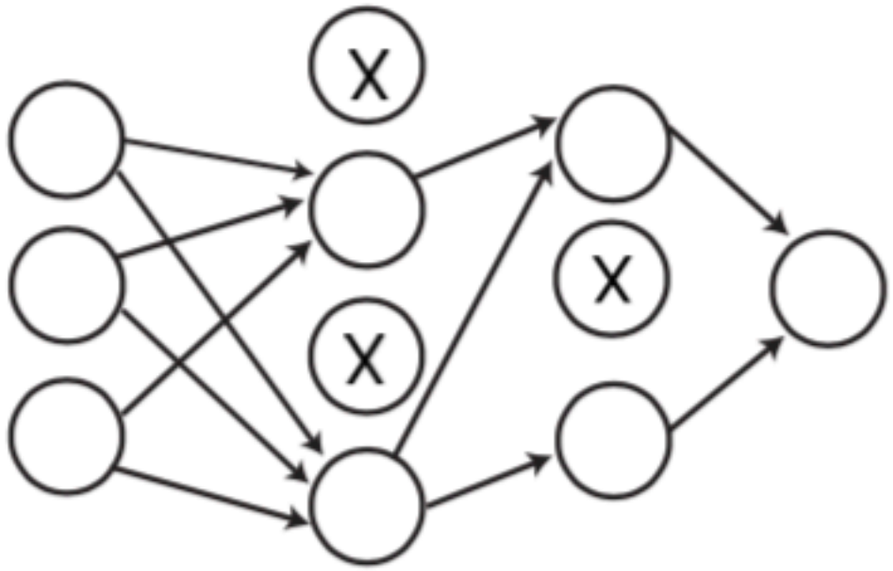
NN regularisation: Dropout

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

Standard Neural Net



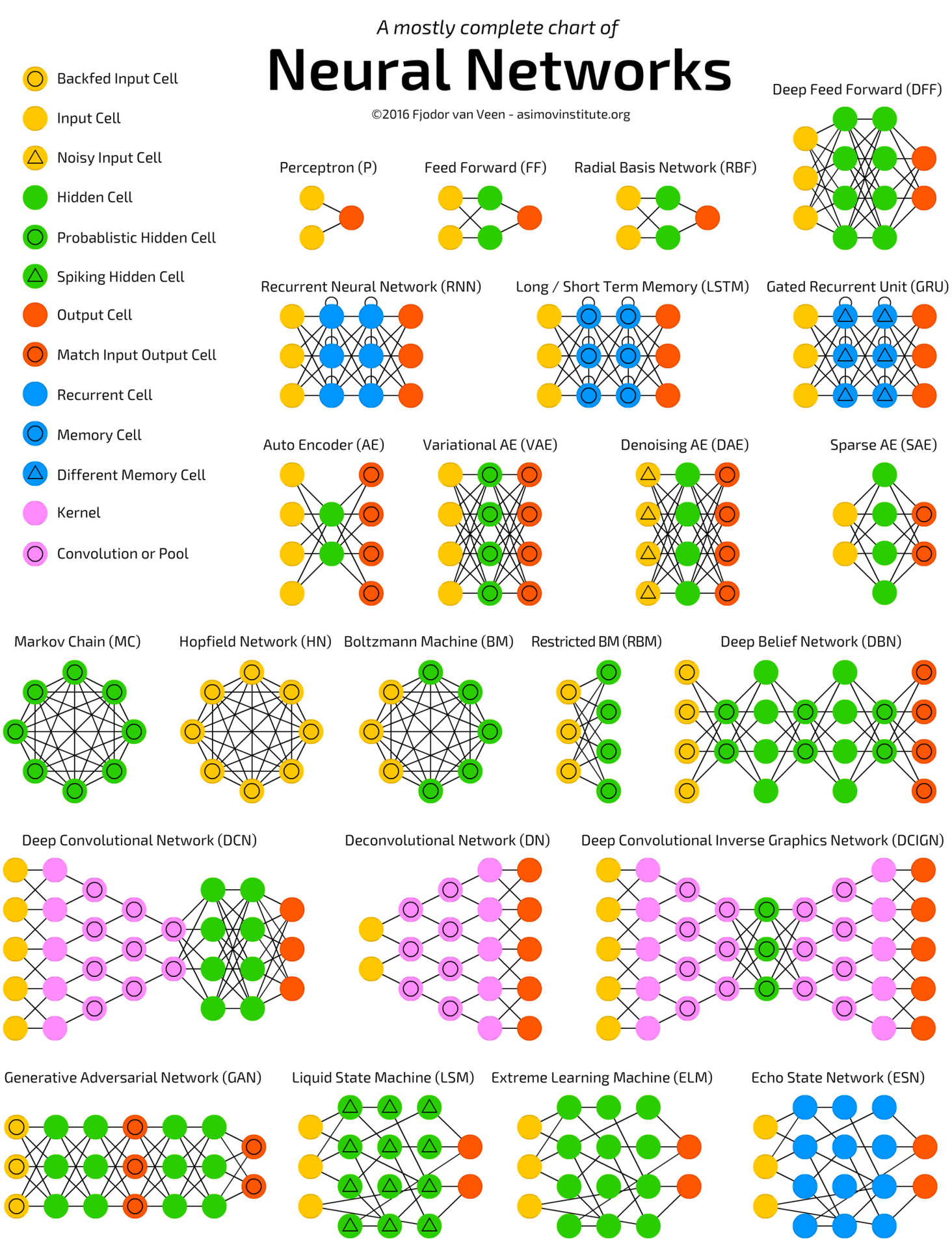
After applying Dropout



- 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:

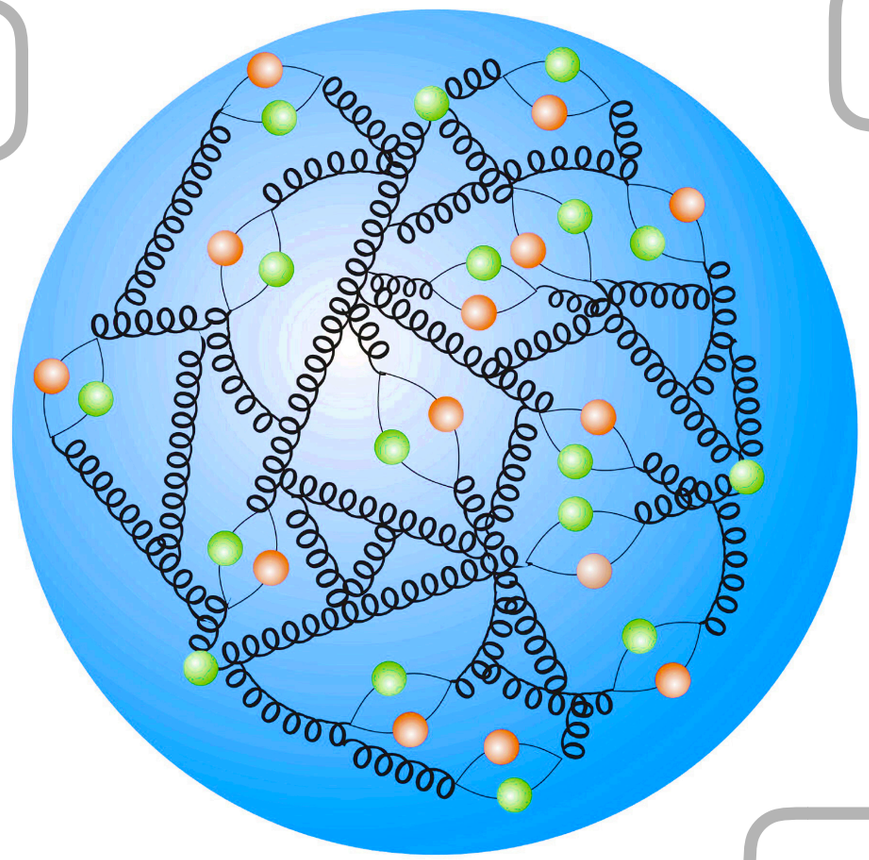
Protron Structure from Neural Networks

The many faces of the proton

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

Origin of mass?

Origin of spin?



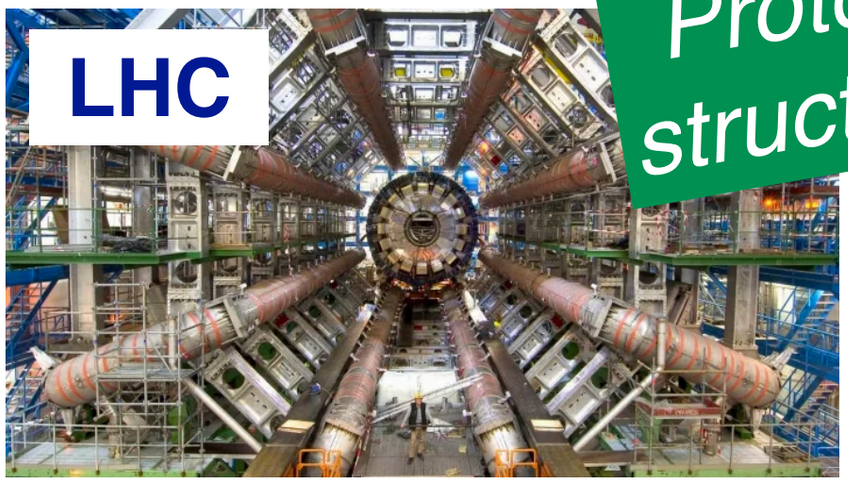
Gluon-dominated matter?

3D imaging?

Heavy quark content?

Nuclear modifications?

From colliders to the cosmos

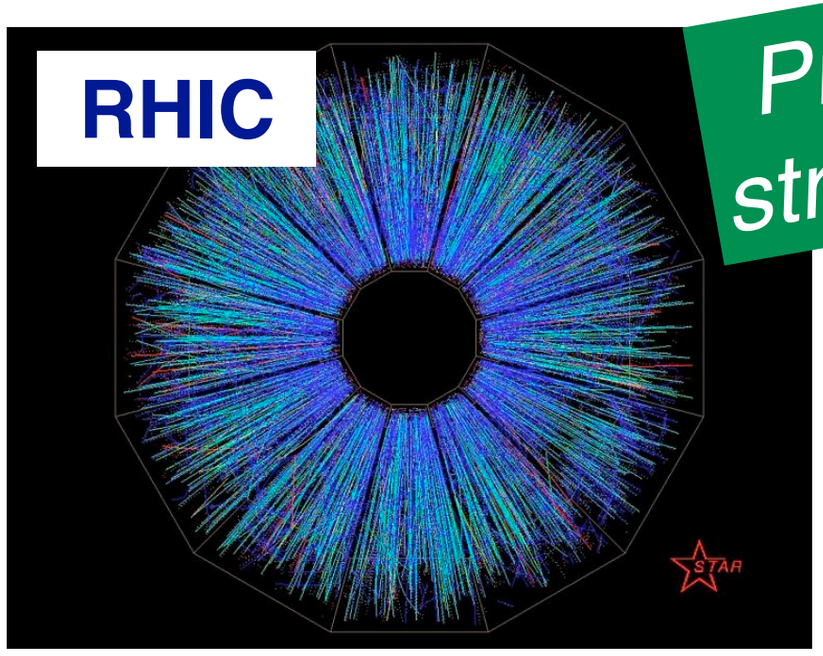


Proton structure

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

Proton structure

Origins and properties of **cosmic neutrinos**?

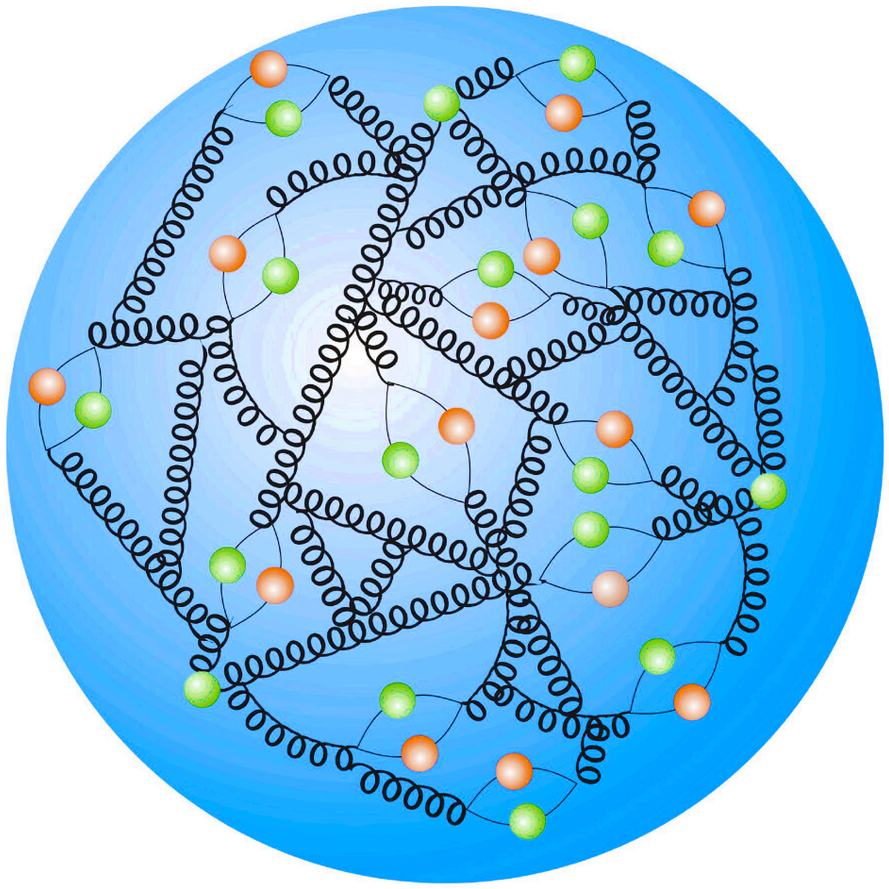


Proton structure

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?

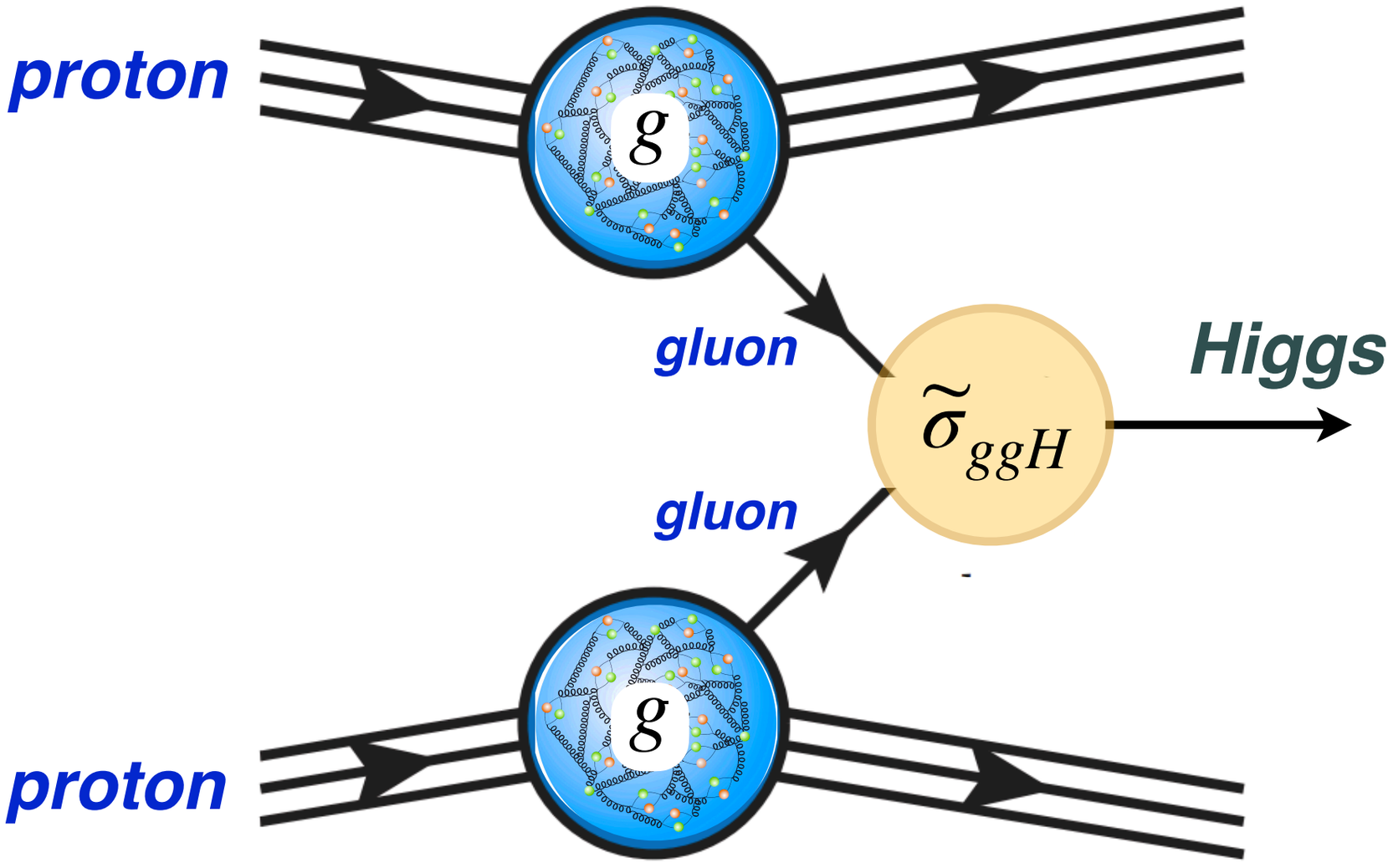
Novel QCD dynamics?

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 $\mathcal{O}\left(\alpha_s^4\right)$

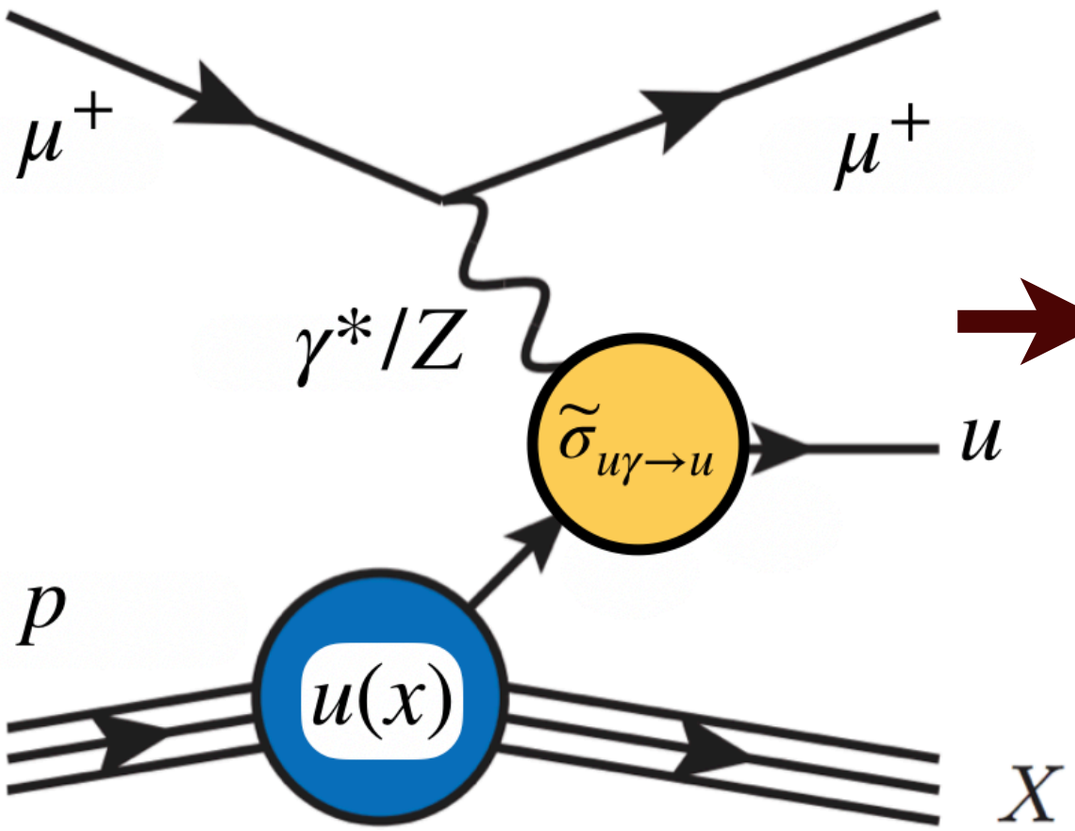
$$\frac{\partial}{\partial \ln Q^2} q_i(x, Q^2) = \int_x^1 \frac{dz}{z} P_{ij}\left(\frac{x}{z}, \alpha_s(Q^2)\right) q_j(z, Q^2)$$

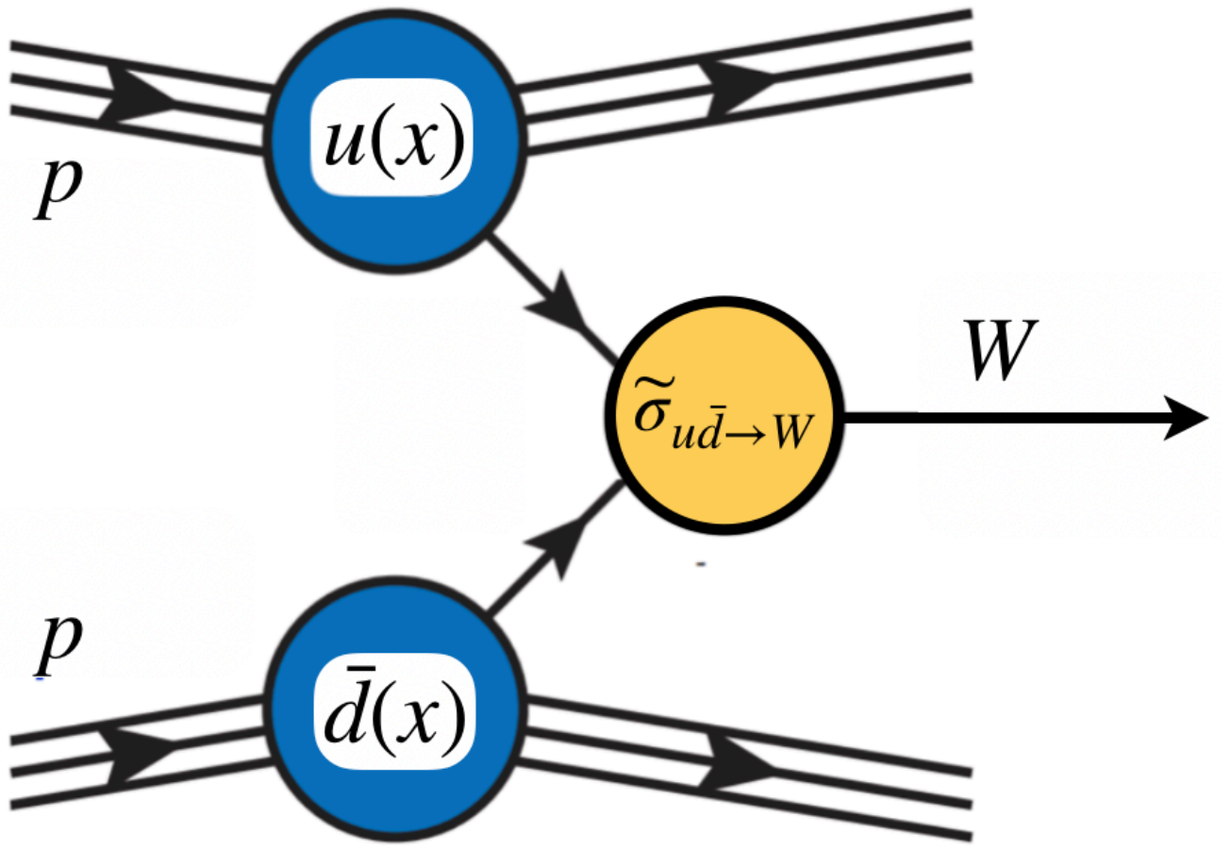
DGLAP *parton evolution equations*

The Global QCD analysis paradigm

QCD factorisation theorems**: PDF universality**

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

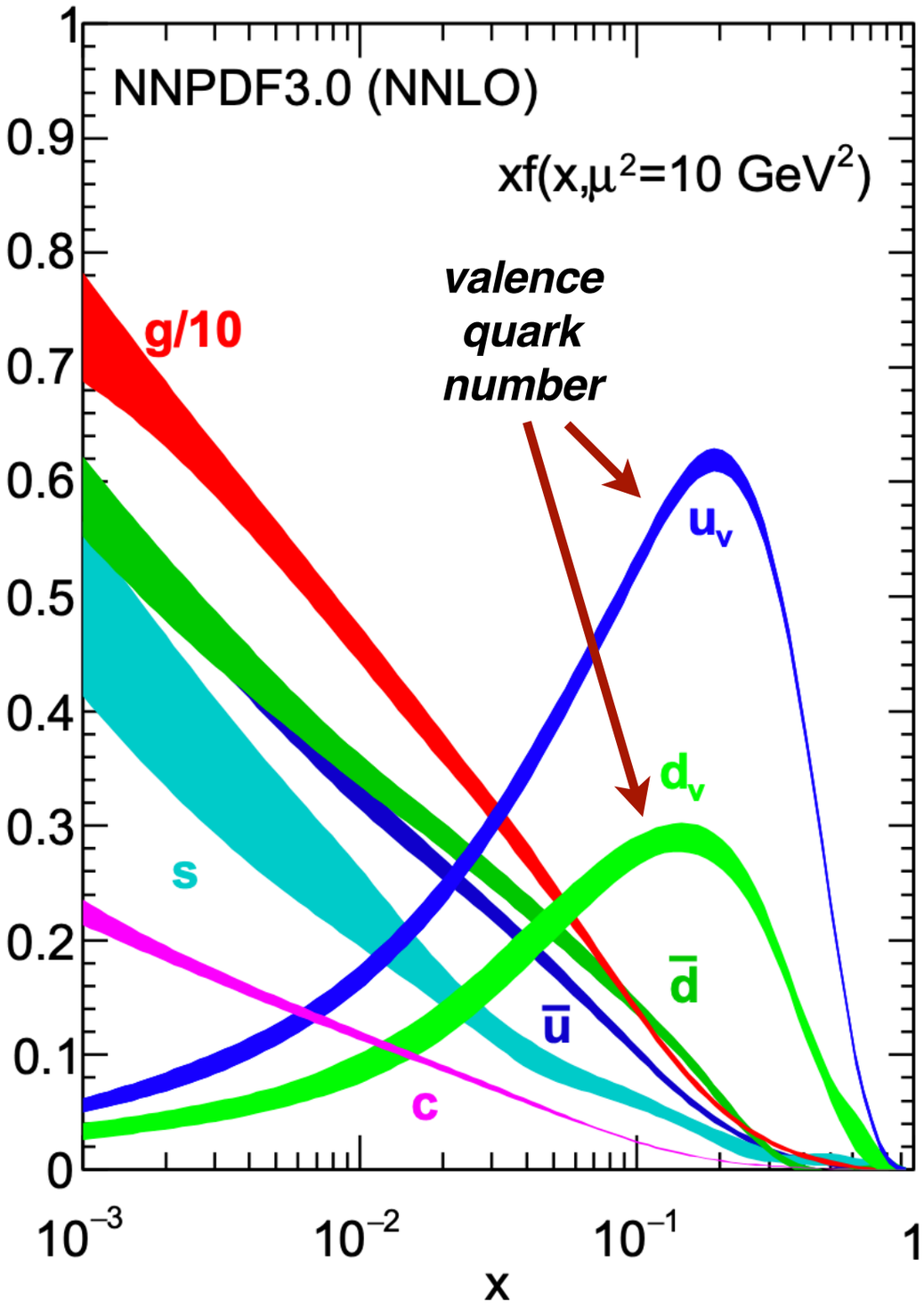


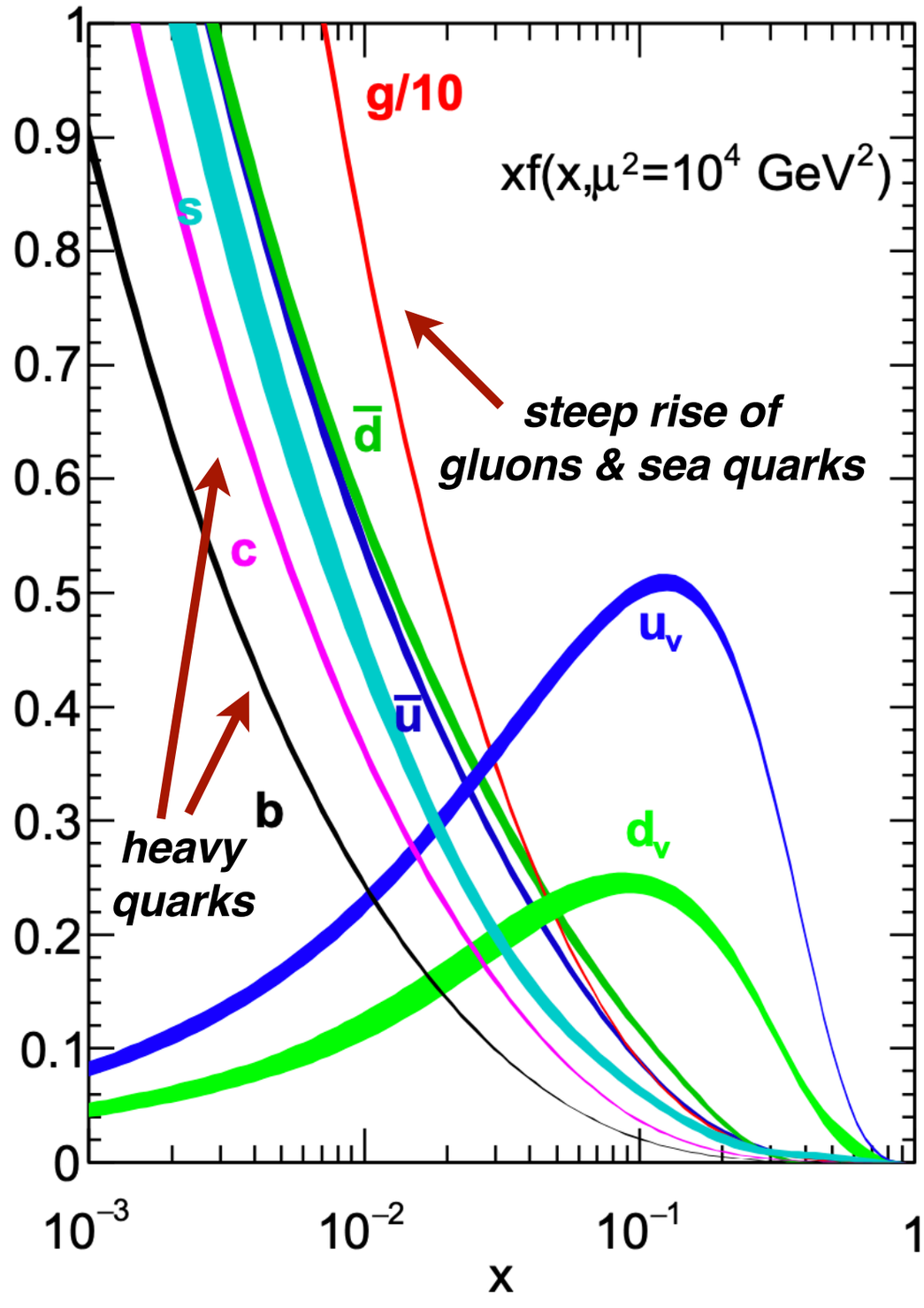


*Determine PDFs from **deep-inelastic scattering**…*

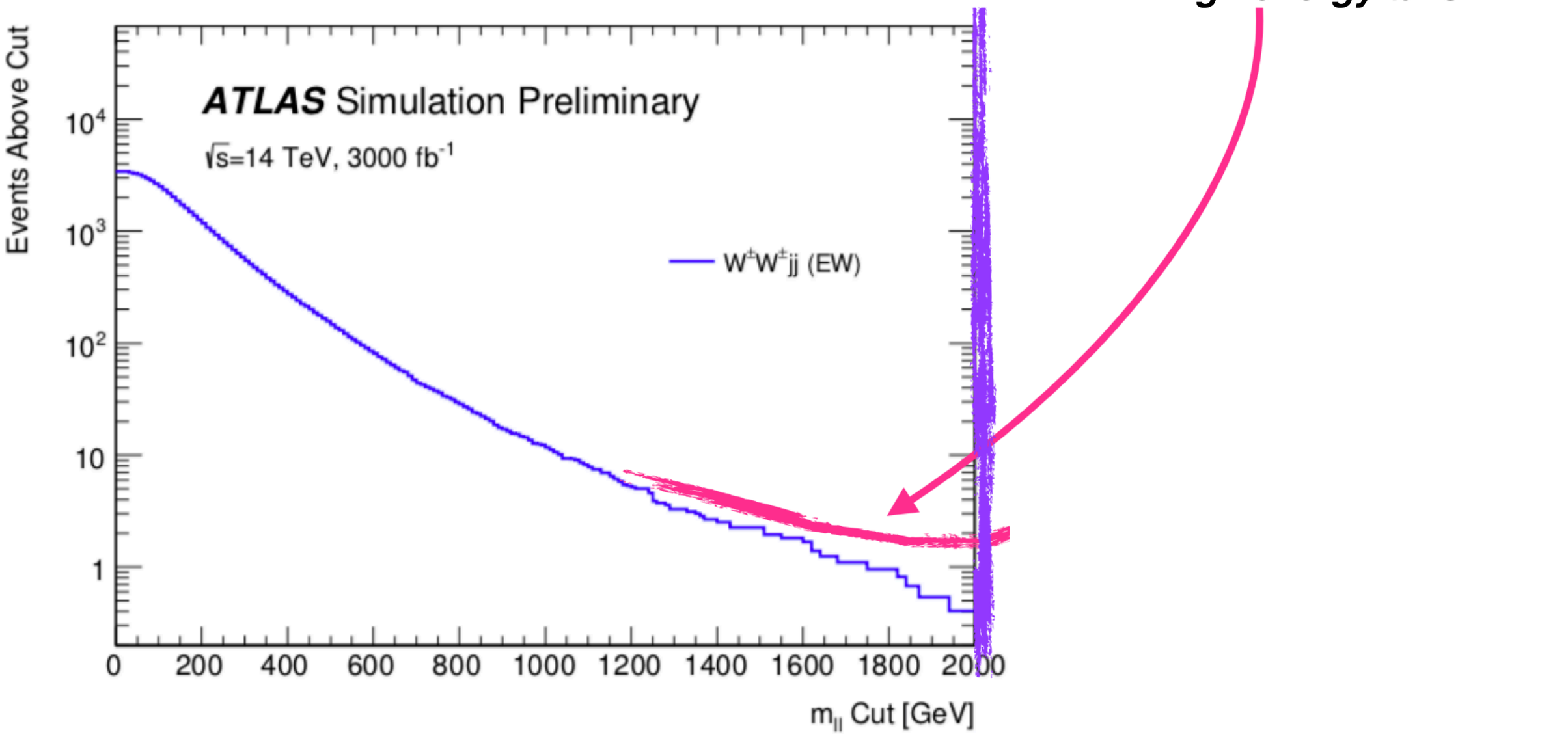
*… and use them to compute predictions for **proton-proton collisions***

A proton structure snapshop





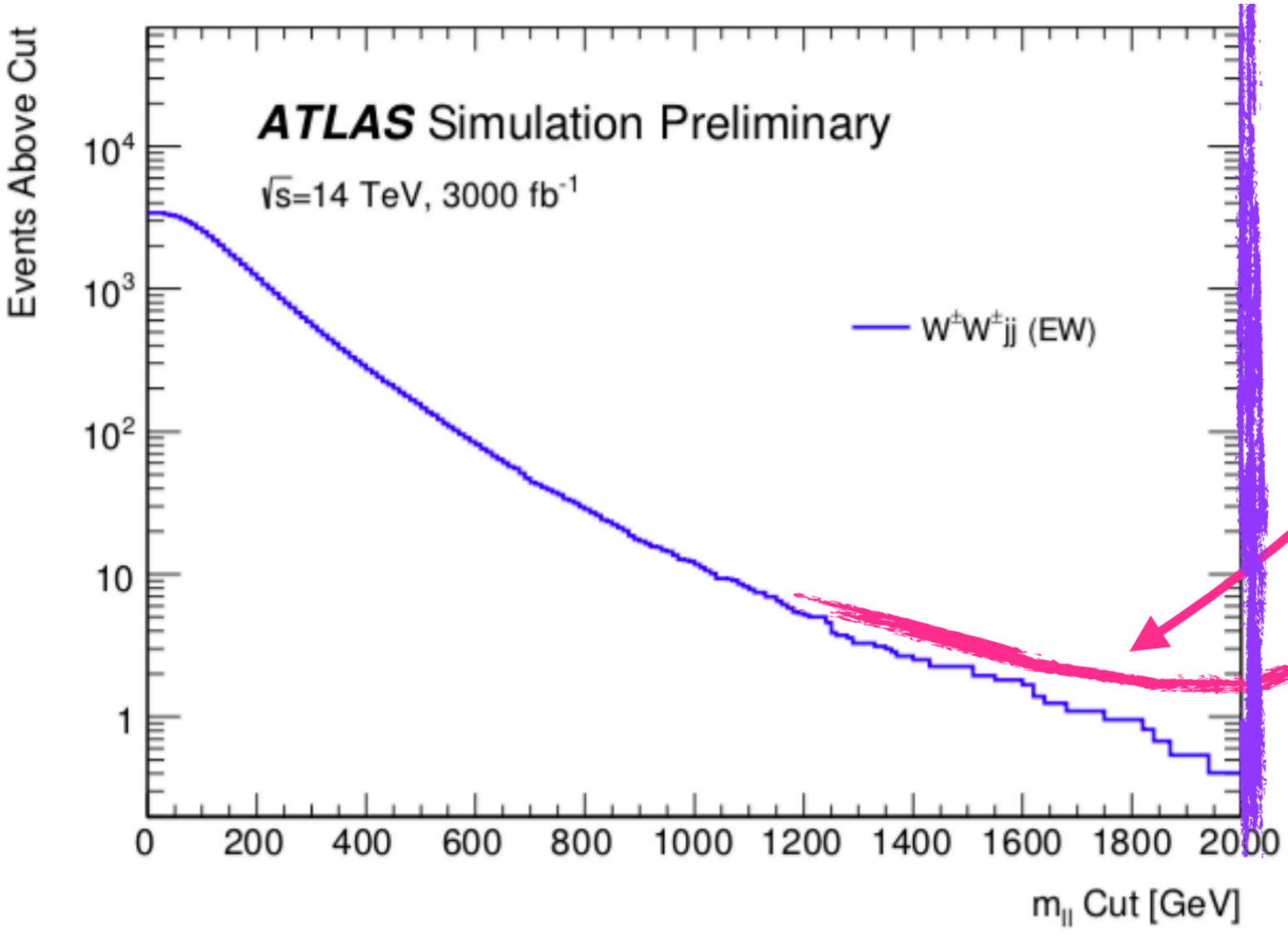
Why do we need better PDFs?



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

Why do we need better PDFs?

Deviation from SM prediction in high energy tails?

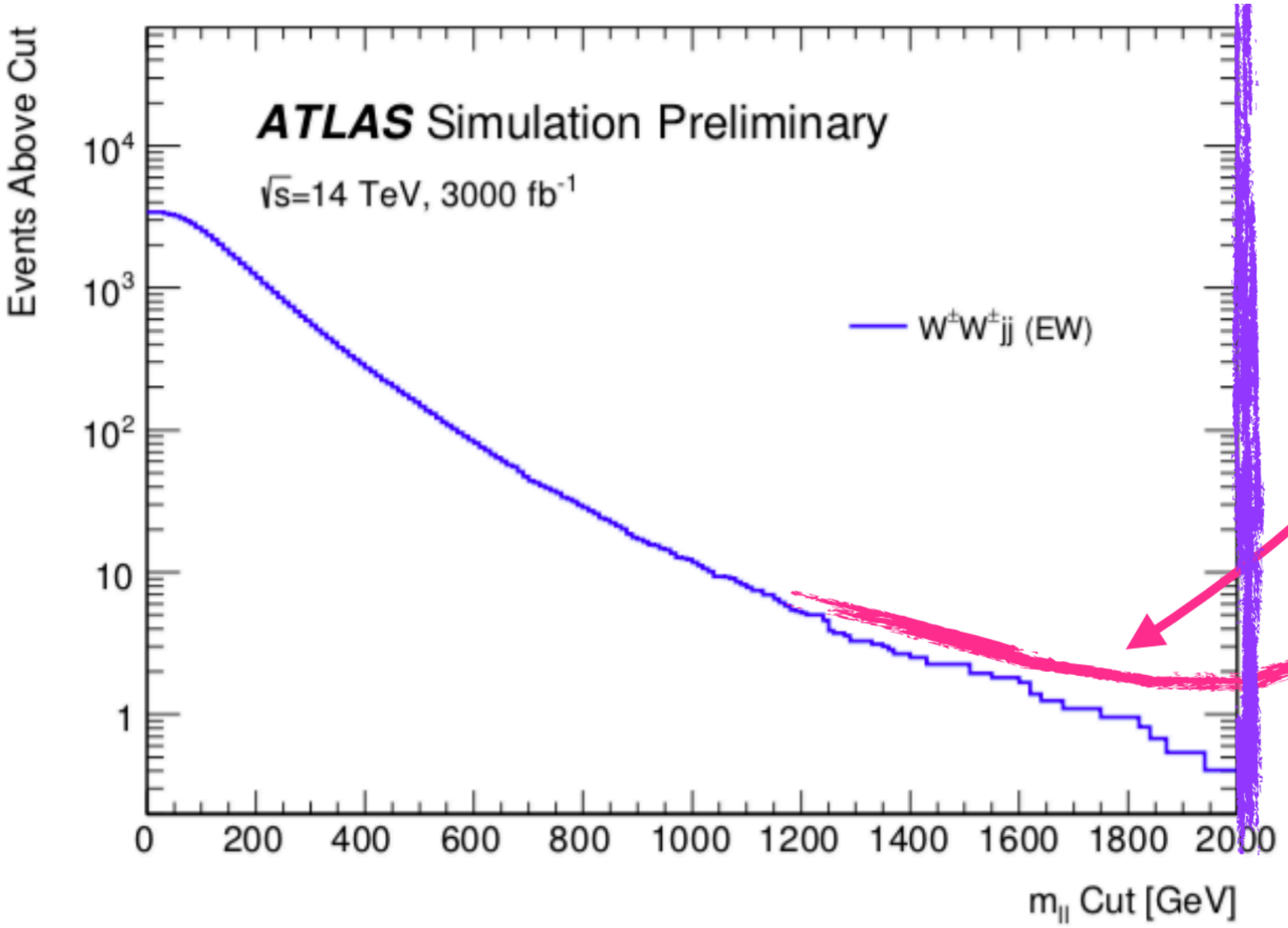


CA Lee, HL/HE-LHC Jamboree, 1 March 2019

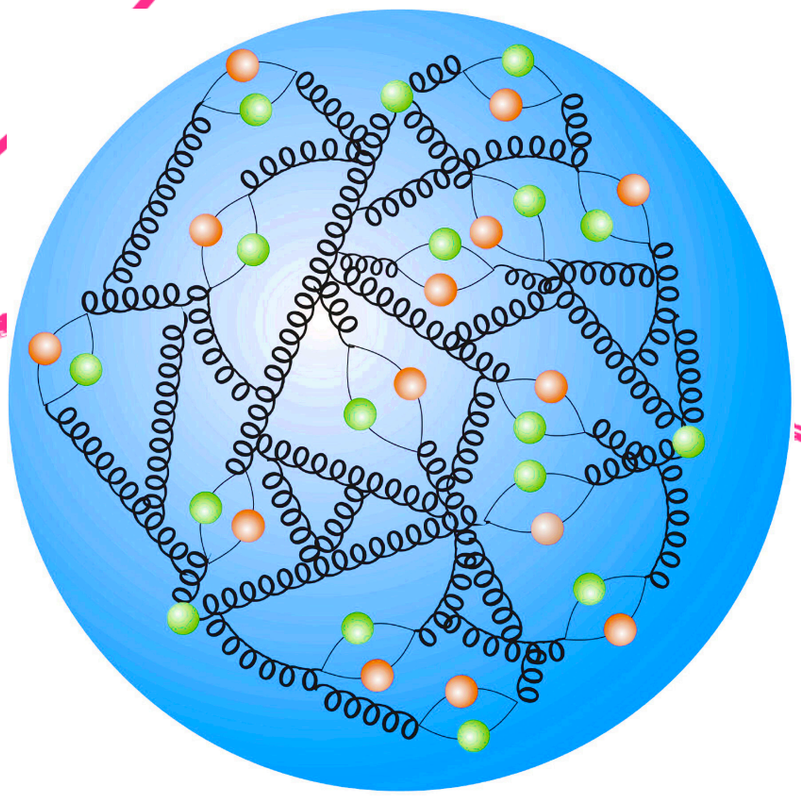
SMEFT interpretation: from a massive particle at high energies …

Why do we need better PDFs?

Deviation from SM prediction in high energy tails?



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

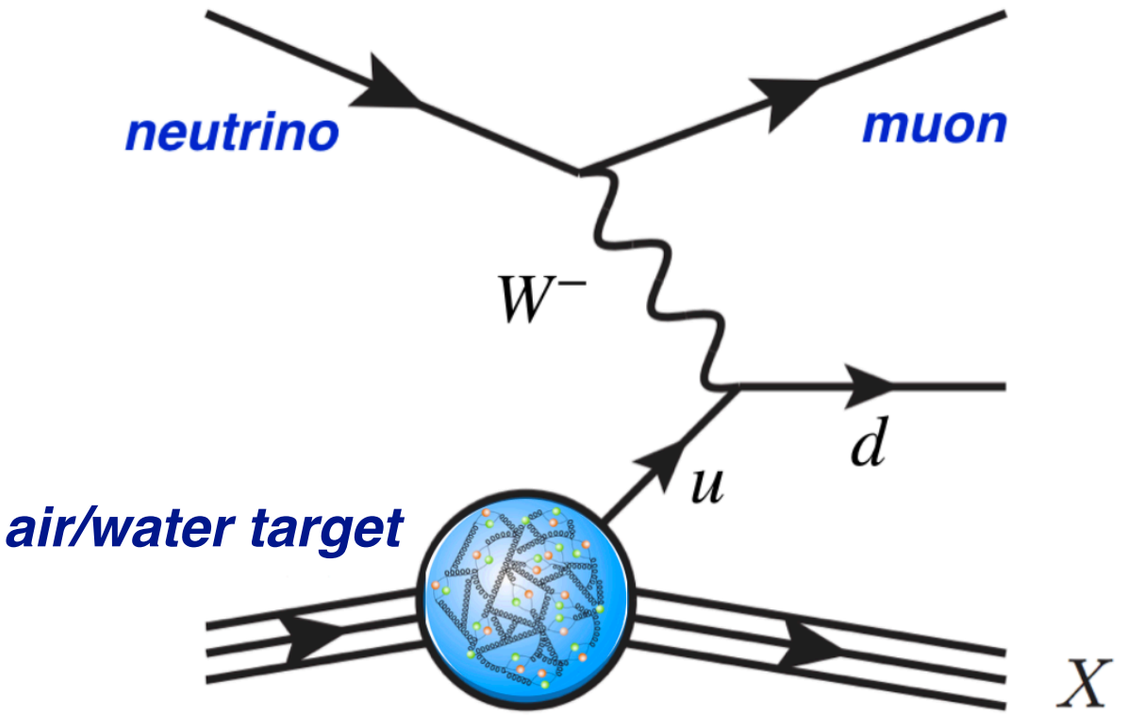


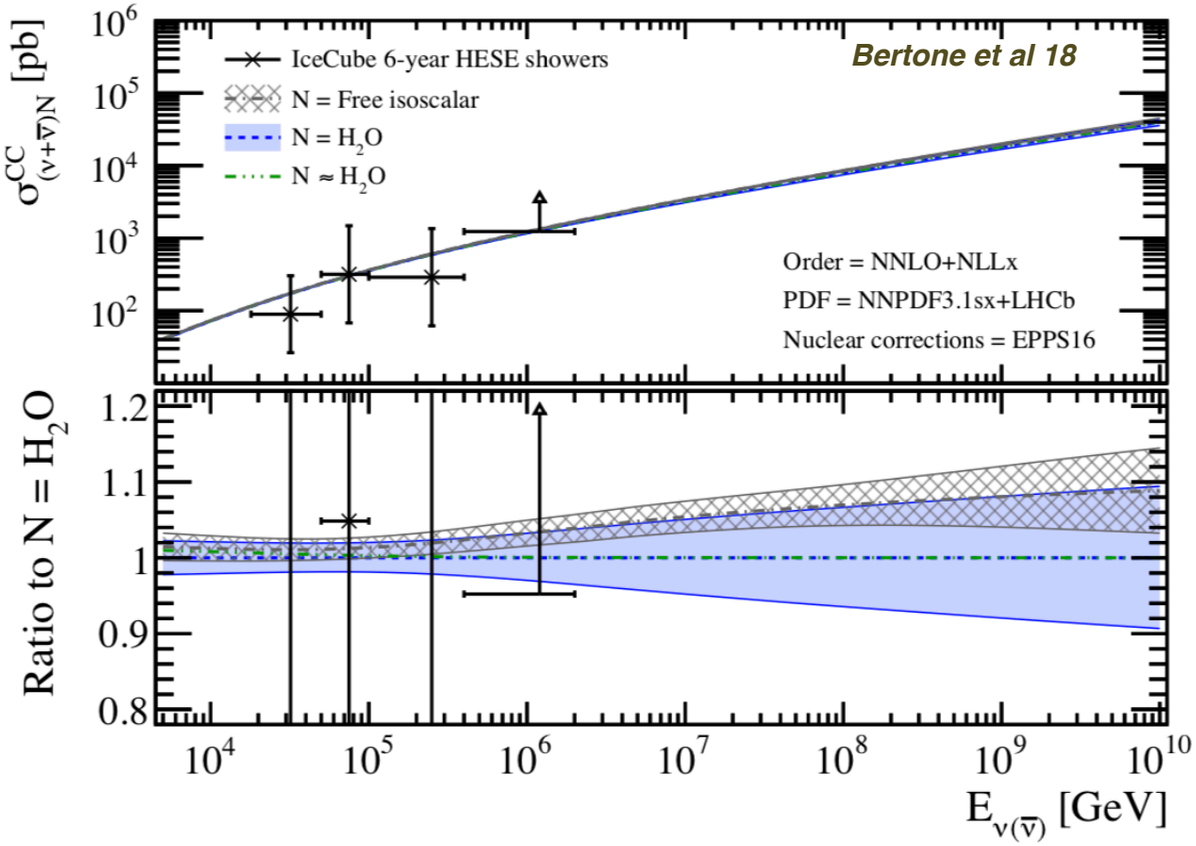
…or reflecting our limited understating of proton structure?

Neutrino telescopes as QCD microscopes

Ultra-high energy (cosmic) neutrino *- nucleus scattering:*

unique probe of ***small-x PDFs and QCD***



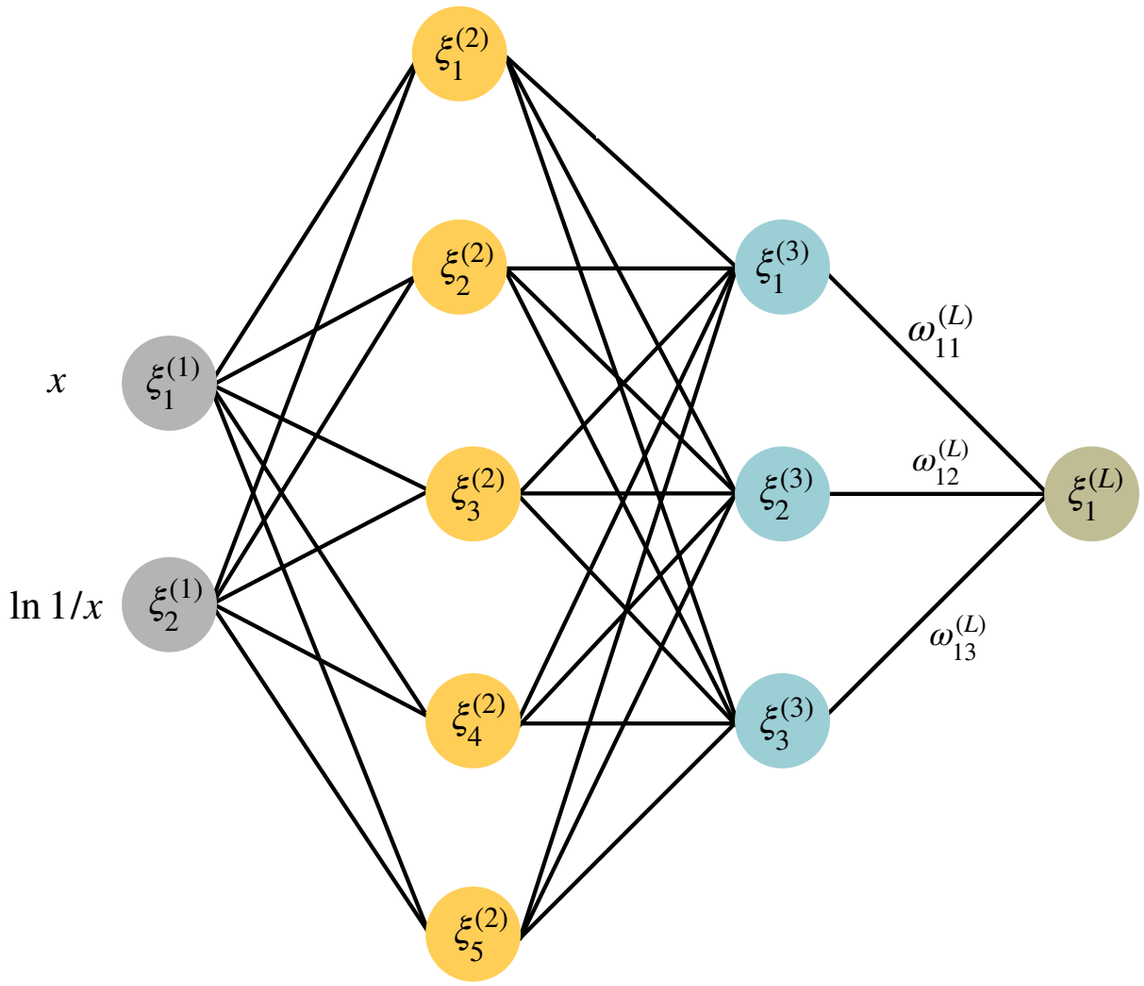


Sensitive to **small-*x* quarks** (and gluons via evolution)

down to $\mathbf{x \simeq 10^{-8}}$ at $\mathbf{Q \simeq M_W}$

Bertone, Gauld, JR 18

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	$g(x) \simeq \mathrm{NN}(x)$	$R_g(x,A) \simeq \mathrm{NN}(x,A)$

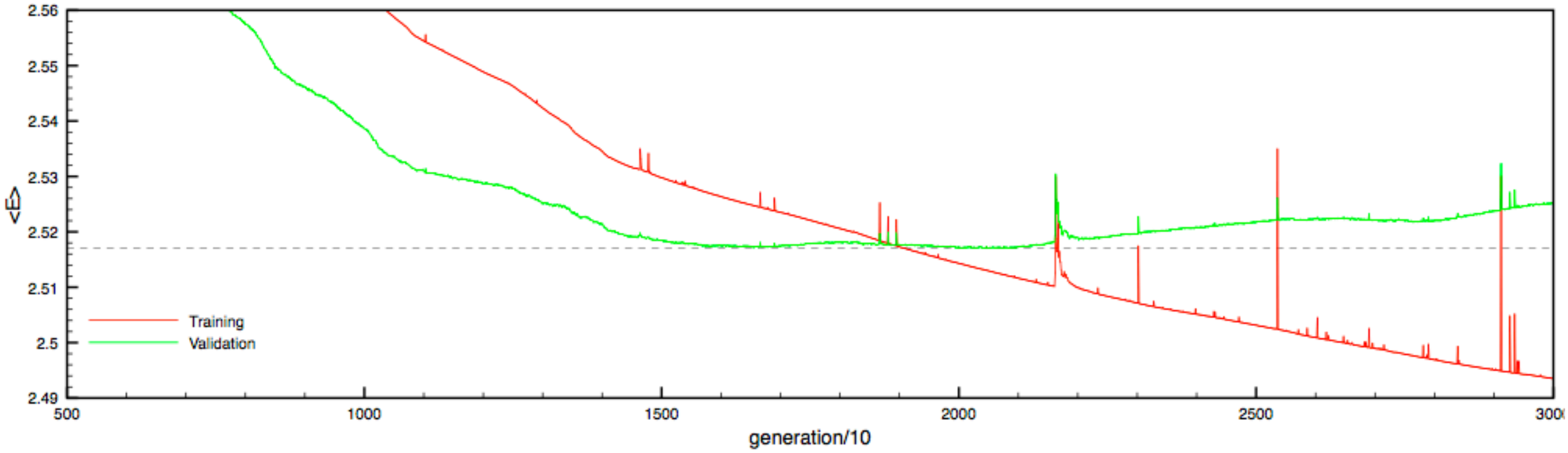
x: *proton's* ***energy fraction*** *carried by gluons*

A: *number of protons + neutrons*

Cross-validation stopping

separate input dataset into **training set** (used to minimise the cost function)

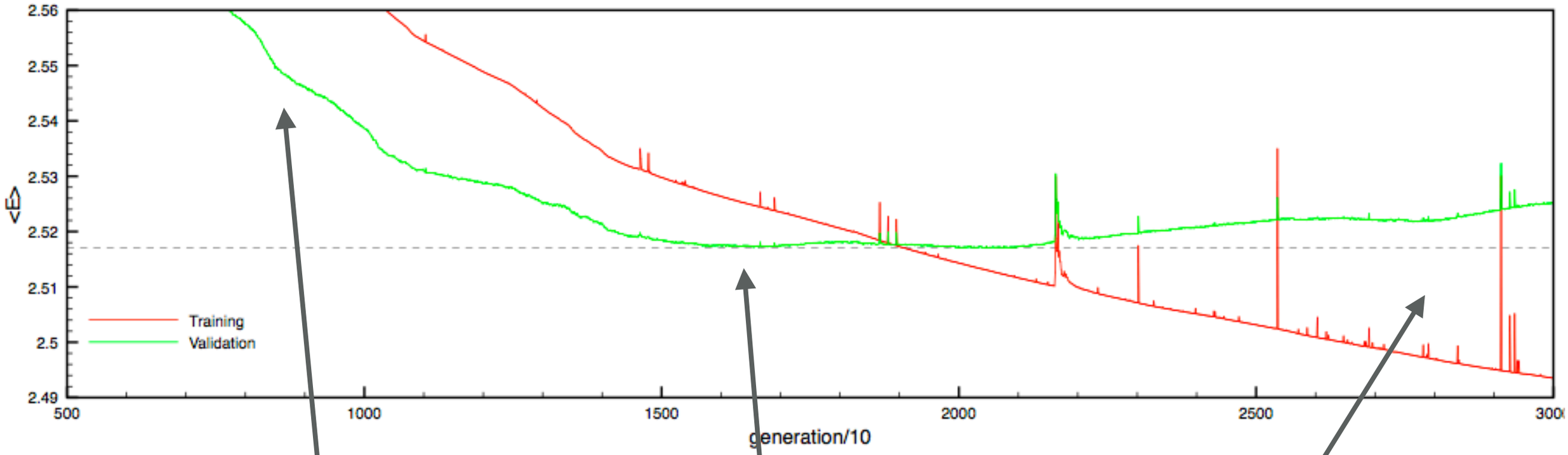
and **validation set** (used to monitor the model ability to generalise)



Cross-validation stopping

separate input dataset into **training set** (used to minimise the cost function)

and **validation set** (used to monitor the model ability to generalise)



underlearning

Optimal stopping

overfitting

recall the bias/variance tradeoff!

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

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

- Construct theory calculations where the SM is **extended by SMEFT corrections**

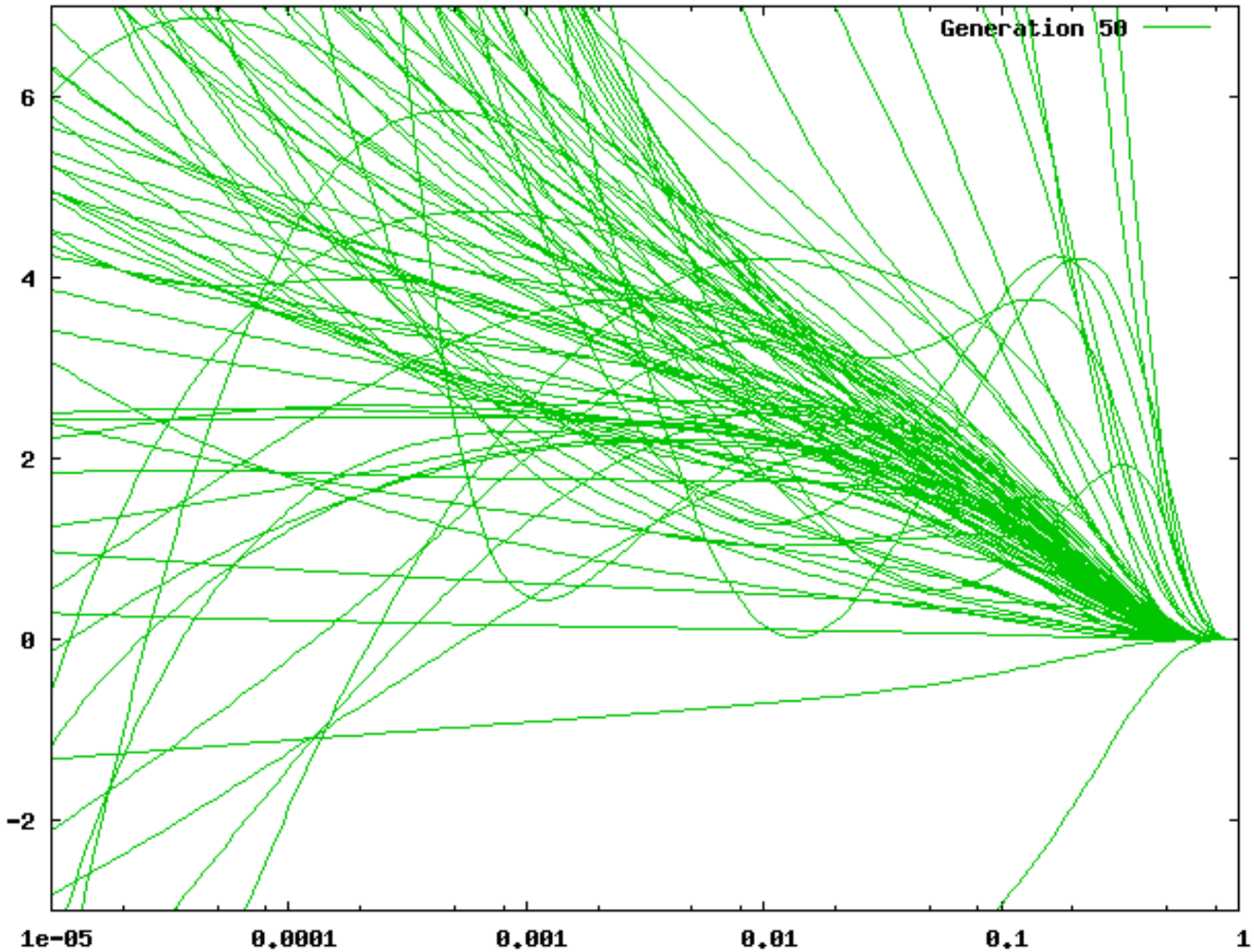
$$\mathcal{O}_i^{\mathrm{th}} \left(\{c_n\} \right) = \sigma_{\mathrm{SM},i} + \sum_{n=1}^{N_{\mathrm{op}}} \widetilde{\sigma}_{i,n} \frac{c_n}{\Lambda^2} + \sum_{n,m=1}^{N_{\mathrm{op}}} \widetilde{\sigma}_{i,nm} \frac{c_n c_m}{\Lambda^4} , \quad i = 1 \ \ldots, N_{\mathrm{dat}}$$

- Determine the SMEFT coefficients **replica-by-replica** by minimising a cost function

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

NN training

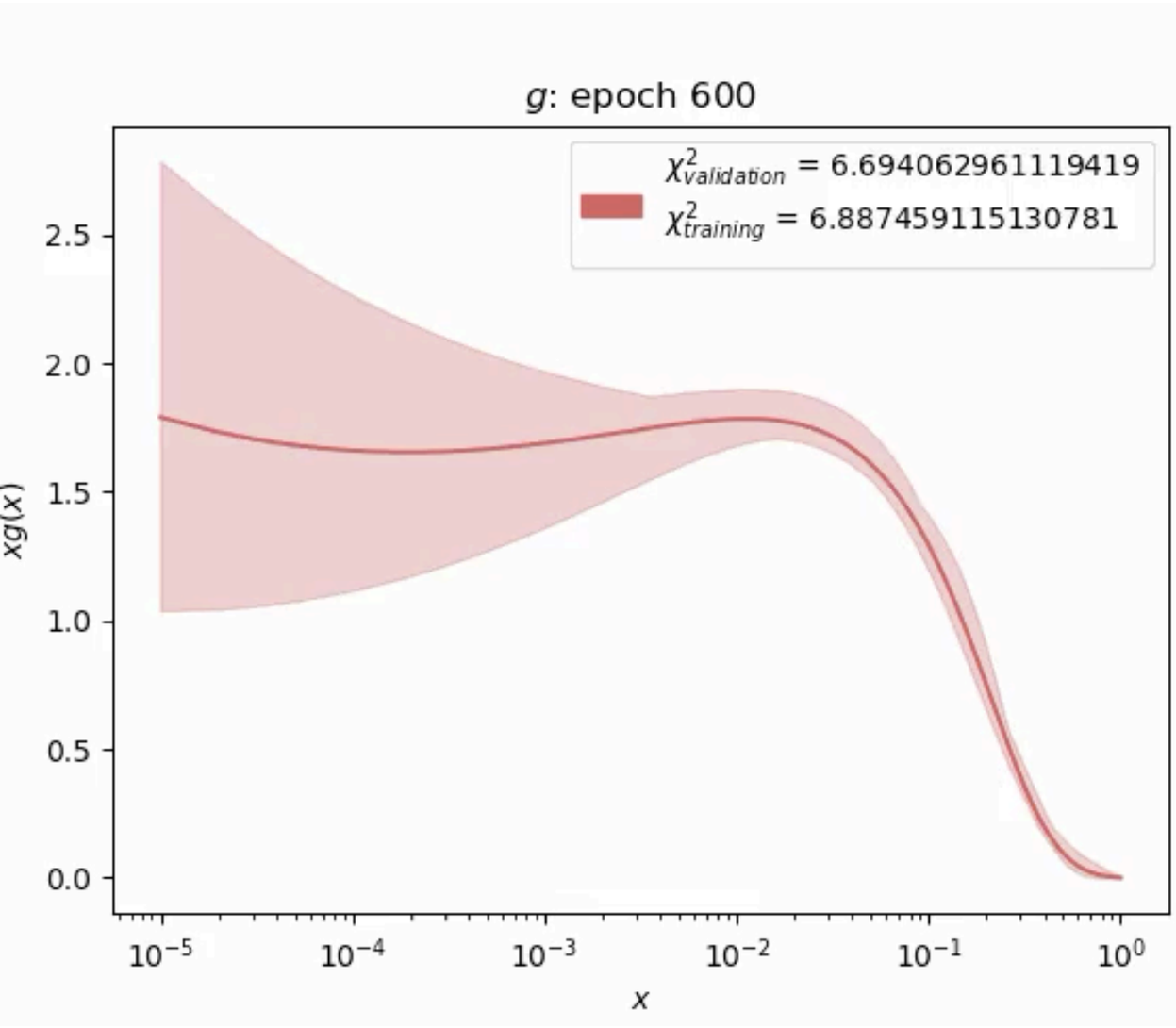
starting from **random parameters,** each member of the MC ensemble of NNs is trained under **convergence** (cross-validation stopping) achieved

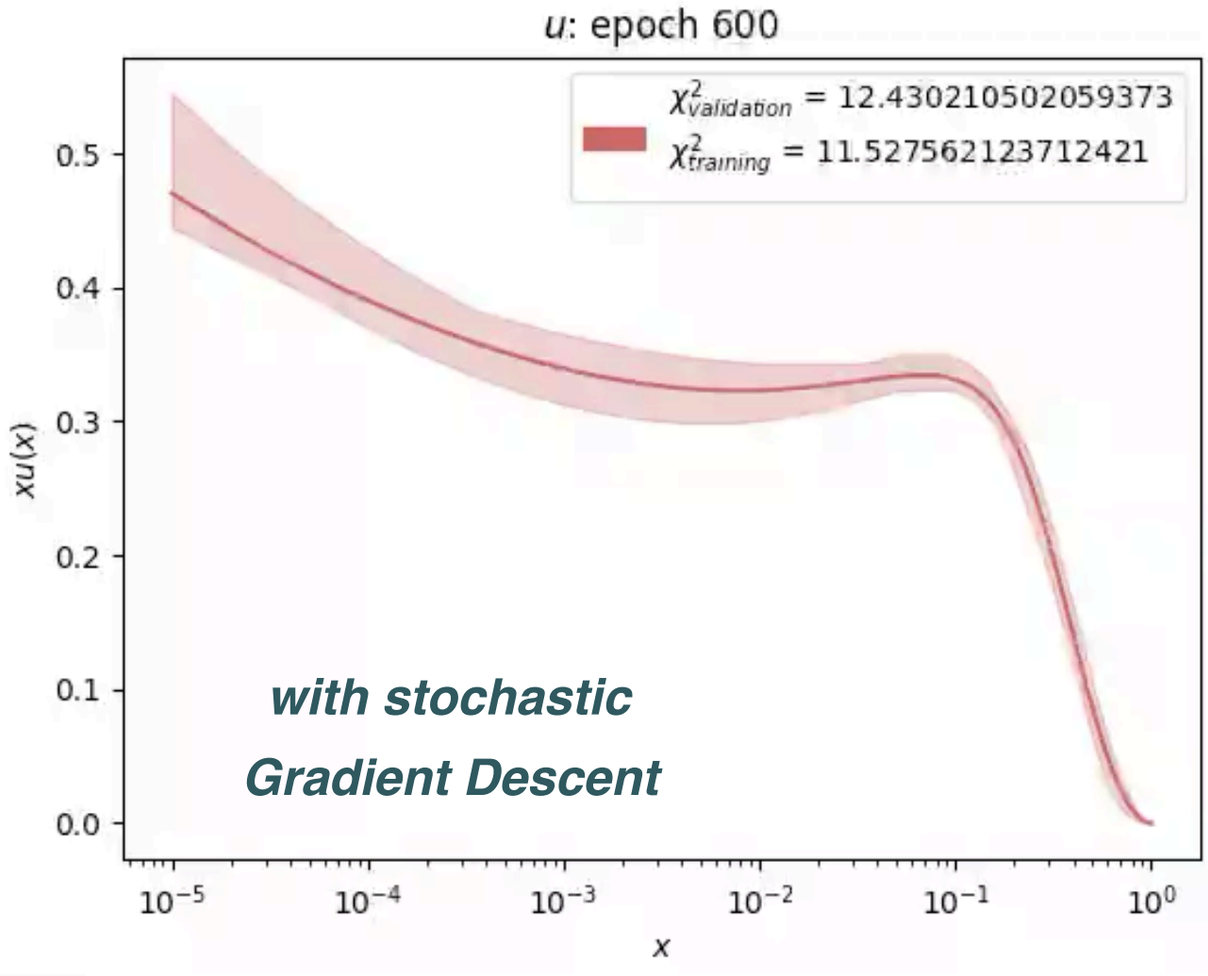


with Genetic Algorithms

NN training live

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



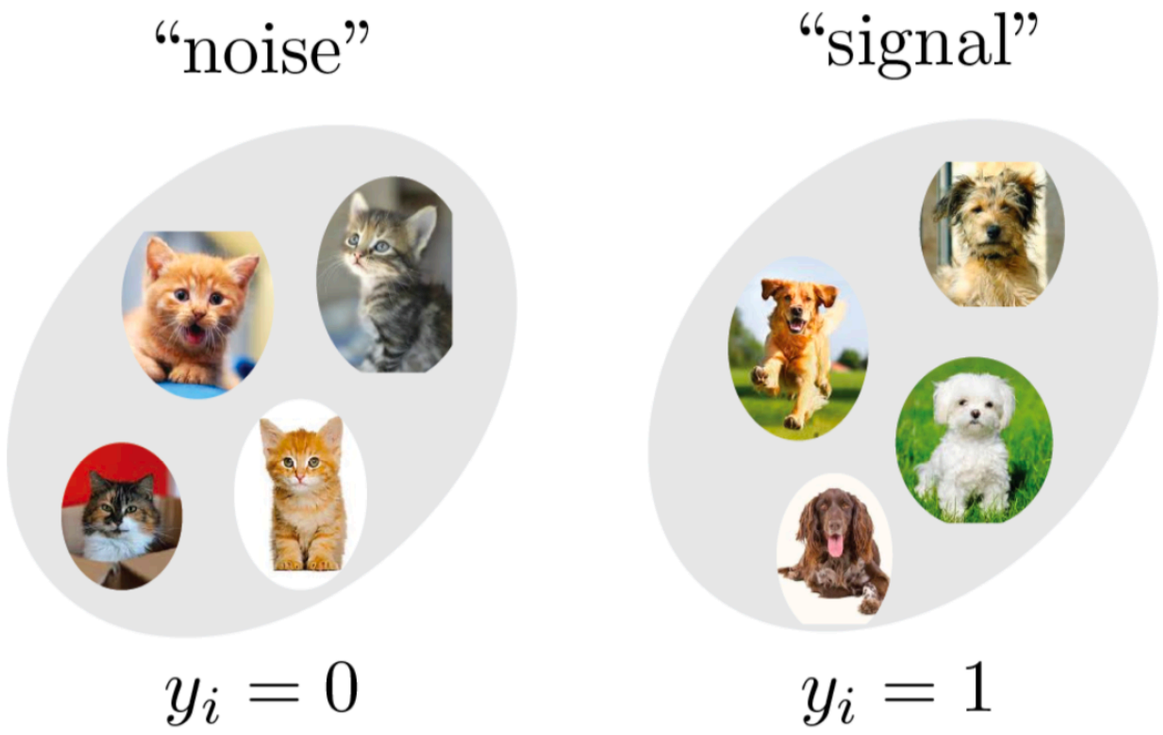


quick initial decrease of the cost function, which slows as we approach the minimum

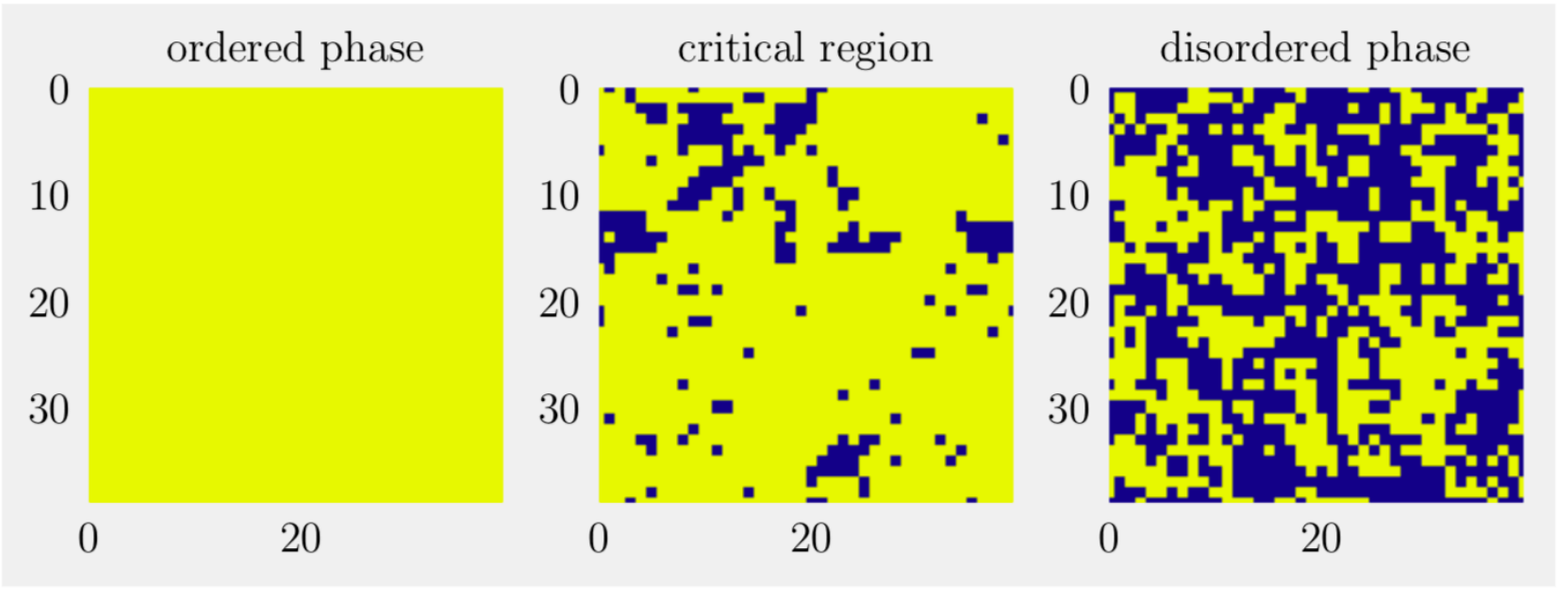
Supervised Learning and Classification

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 configuration 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,

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

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

features of the samples

model parameters

A perceptron is a **hard classifier** is assigned to a category

Logistic regression

In many cases a **soft classifier,** that outpouts 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

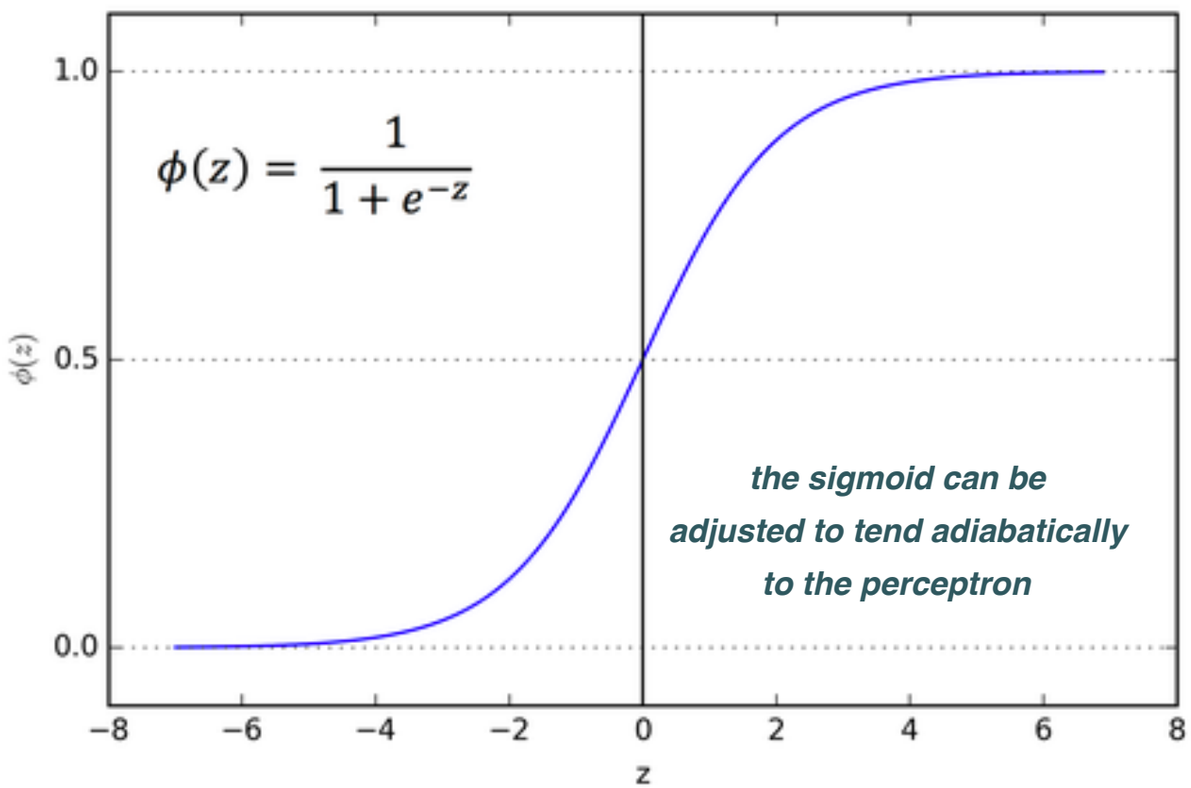
$$P(y_i = 1 \,|\, \mathbf{x}_i, \boldsymbol{\theta}) = \frac{1}{1 + e^{-\mathbf{x}_i^T \boldsymbol{\theta}}} = \sigma(\mathbf{x}_i^T \boldsymbol{\theta})$$

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

Where we have defined the logistic (or sigmoid) function:

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

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



Logistic regression

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

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

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

$$E(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left(-y_i \ln \sigma(\mathbf{x_i^T}\boldsymbol{\theta}) - (1-y_i)\ln[1-\sigma(\mathbf{x_i^T}\boldsymbol{\theta}]\right)$$

which is known as the **cross-entropy**

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

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

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

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, eg. TN >> TP

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

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

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

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

proportion of correct positive classifications

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

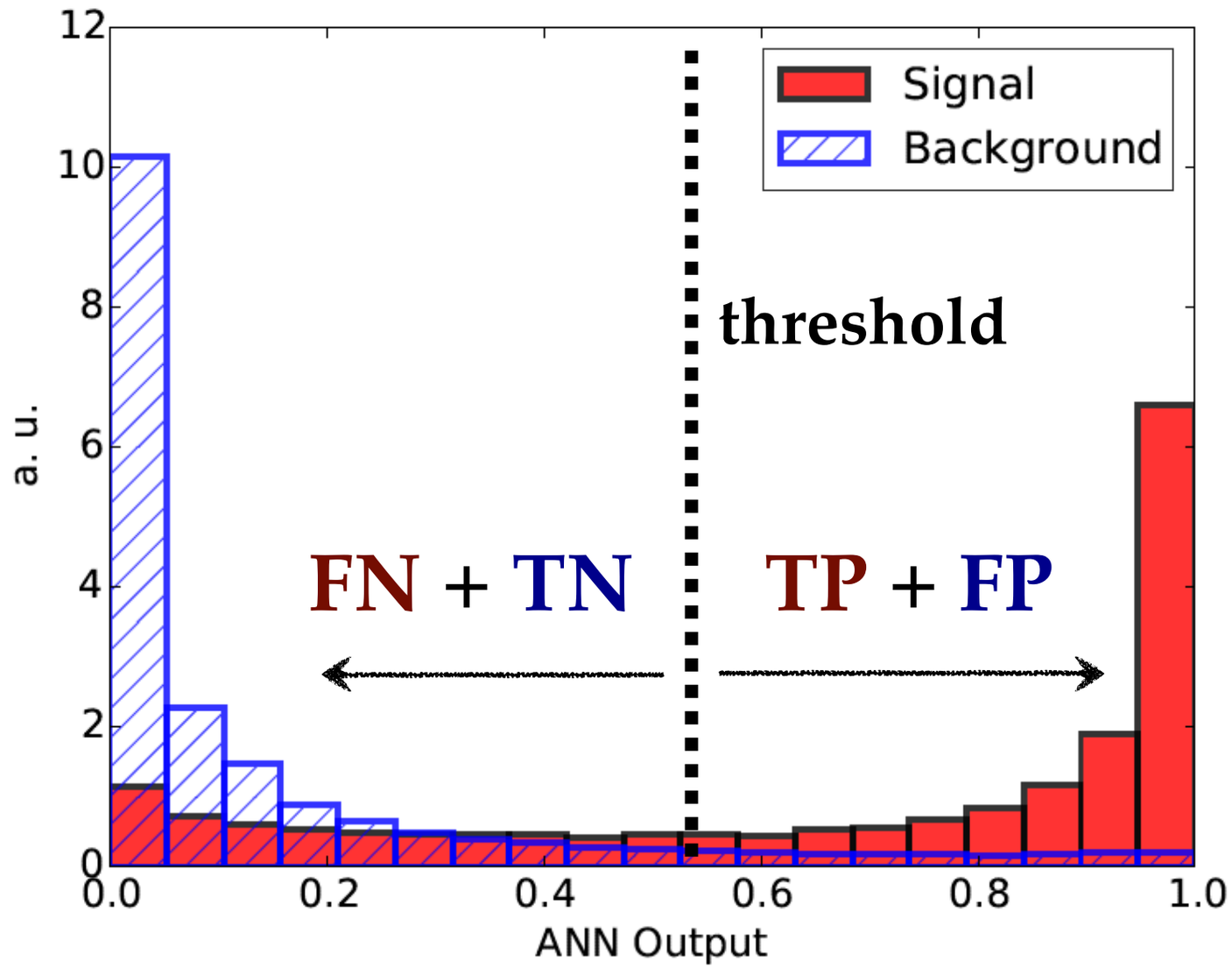
proportion of correct actual positive classifications

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:

☑ 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

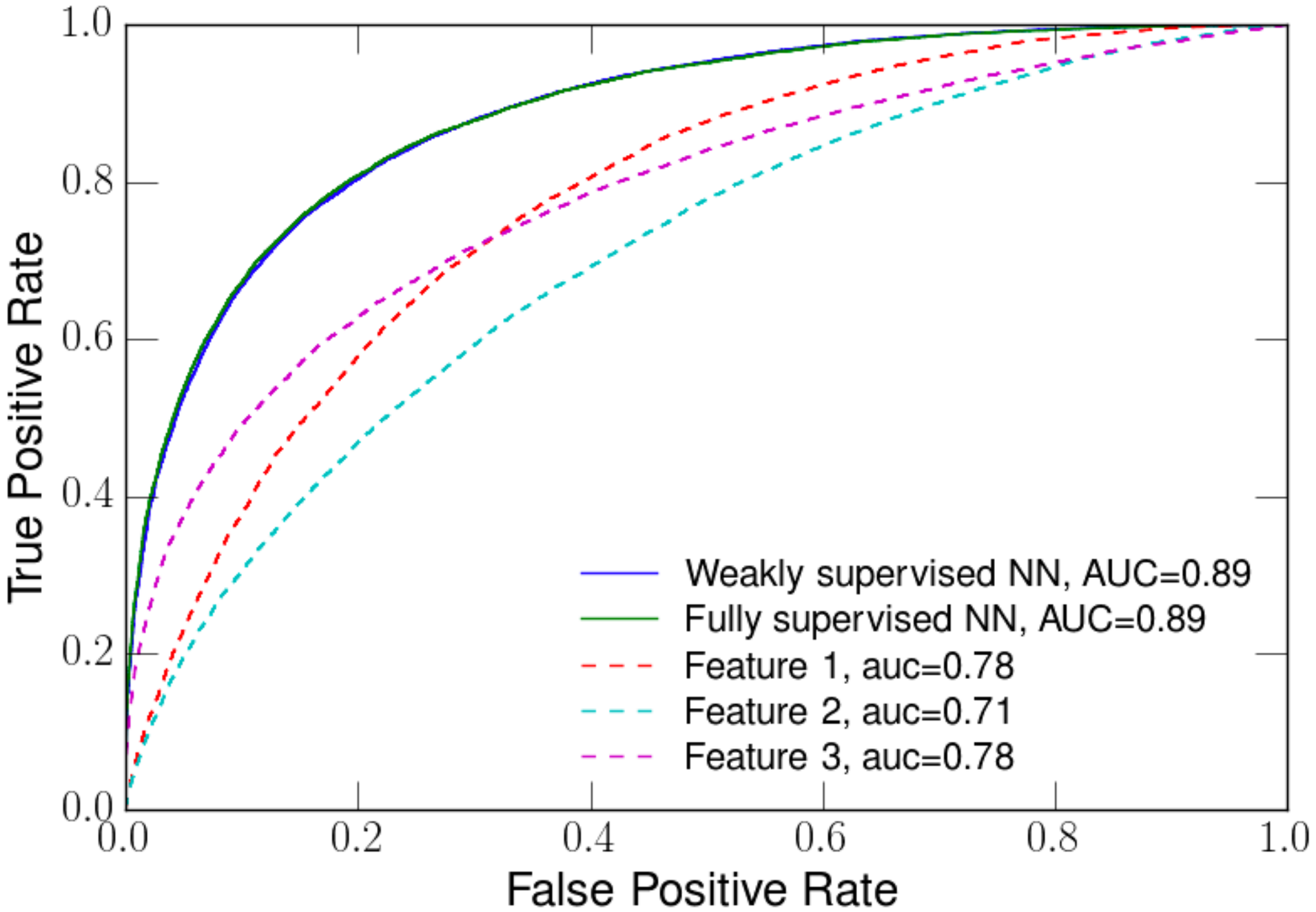
$$\text{False Positive Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

← *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:

The Ising Model in 2D

The Ising Model in 2D

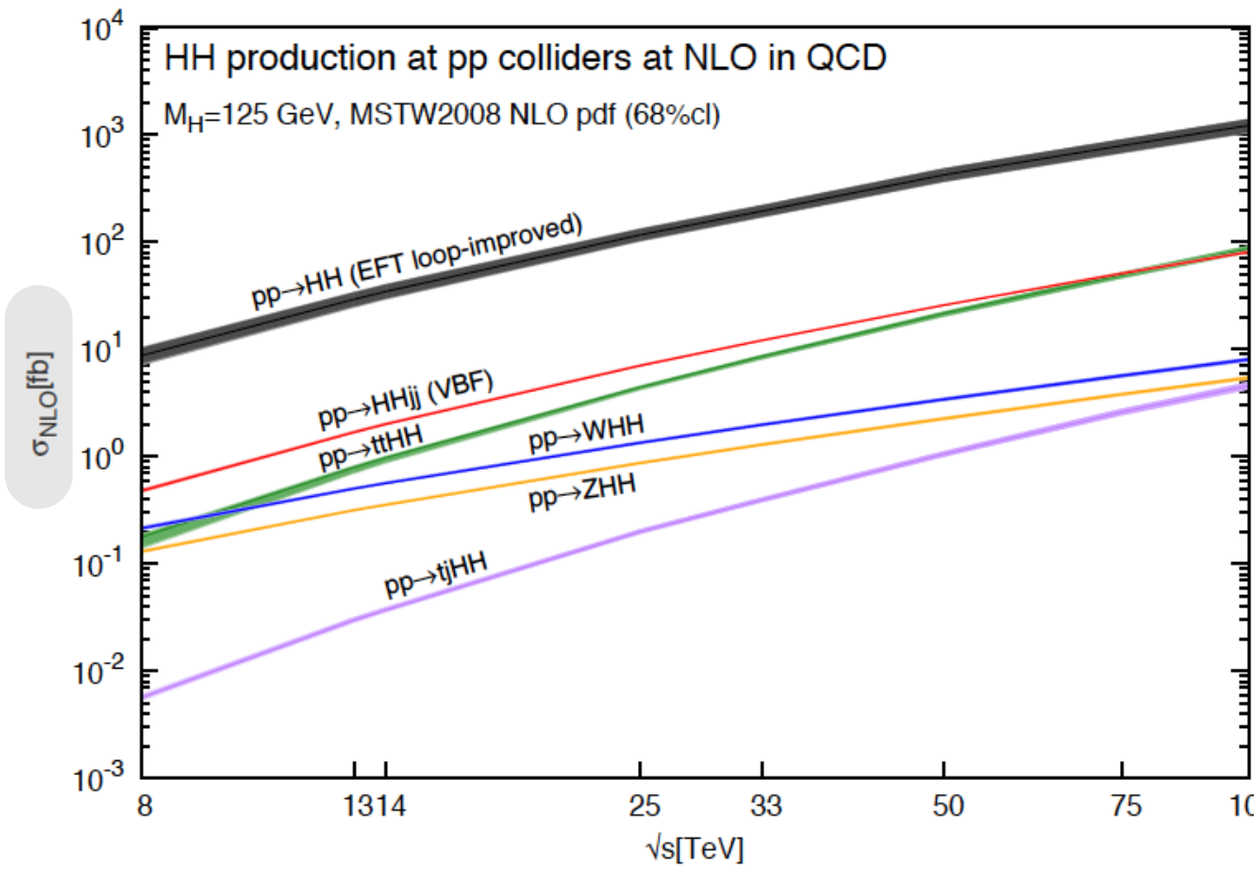
This model describes **ferromagnetism** in statistical mechanics

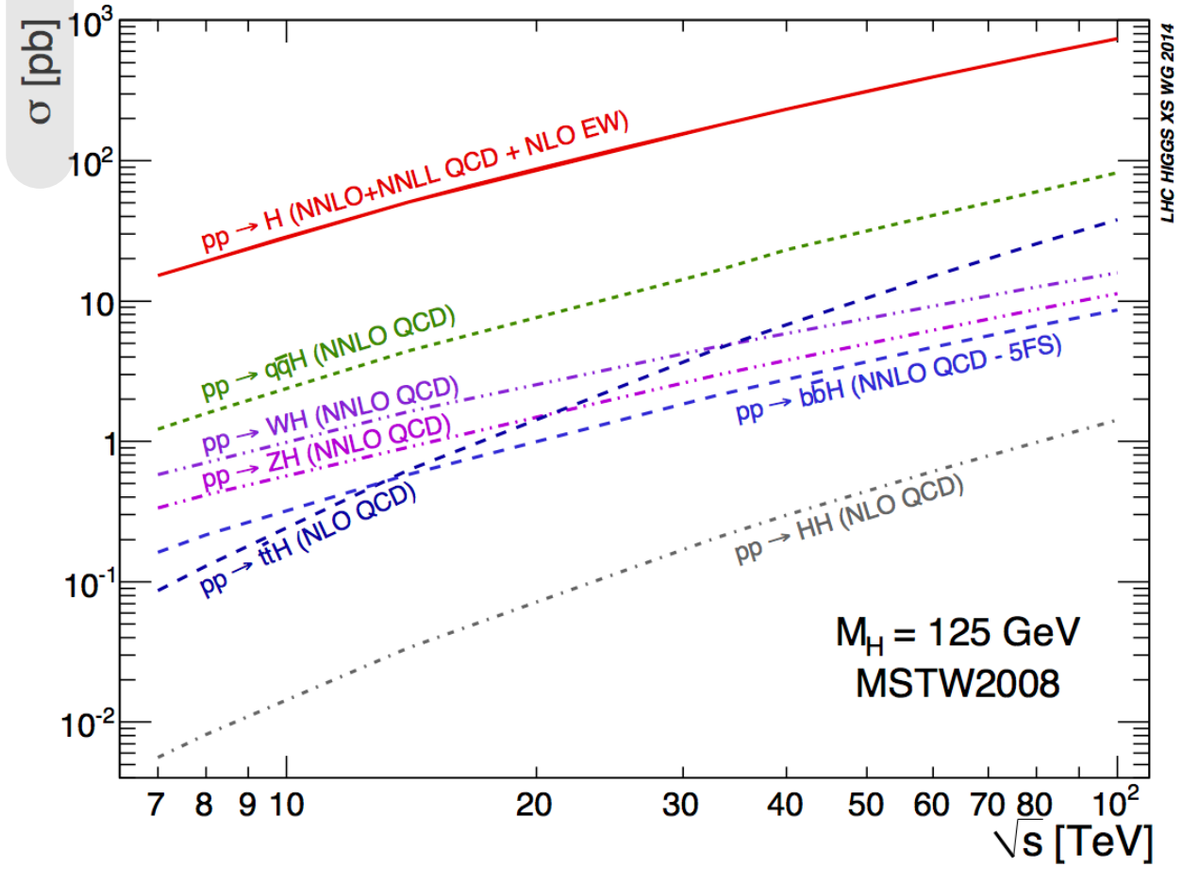
Case Study III:

Higgs Pair Production

Higgs Pair Production

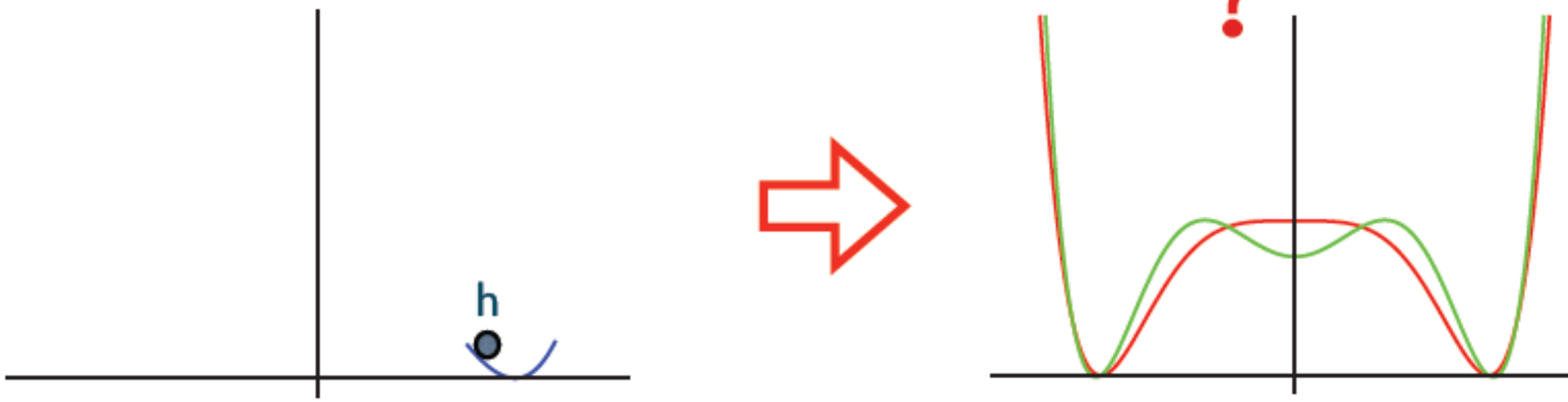
- Double Higgs production allows accessing **crucial components of the Higgs sector**:
  - ☑ Reconstructing the full **electroweak symmetry breaking potential**
  - ☑ Probing the **Higgs self-interaction**
  - ☑ Asessing the **doublet nature** of the Higgs by means of the **hhVV coupling**
- In the SM, **hh rates are small: even** in the gluon-fusion production mode, the **cross-section at 14 TeV** is only **40 fb,** and suppressed by branching fractions





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



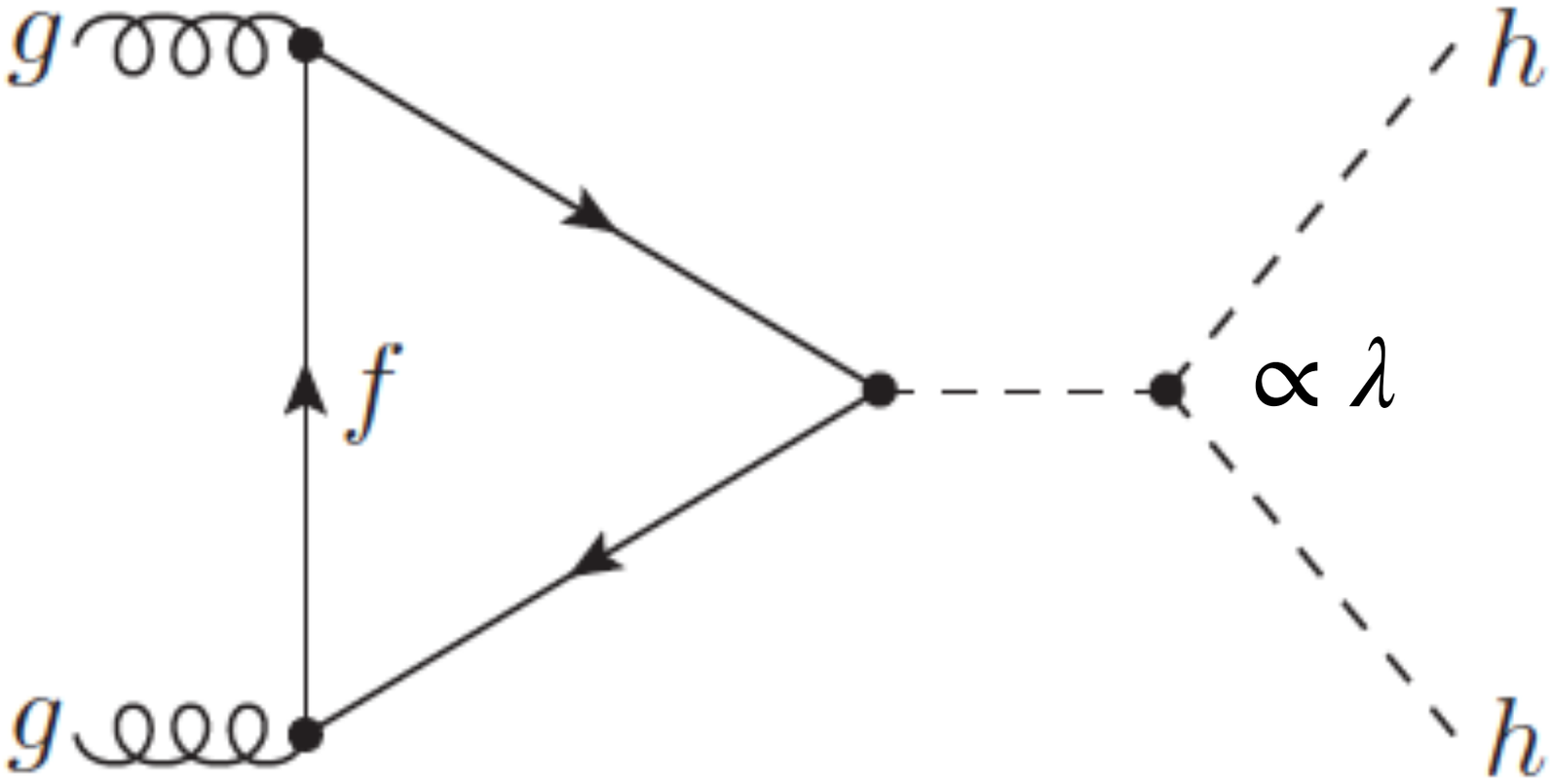
Higgs mechanism ***Coleman-Weinberg mechanism***

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

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

Higgs Pair Production



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

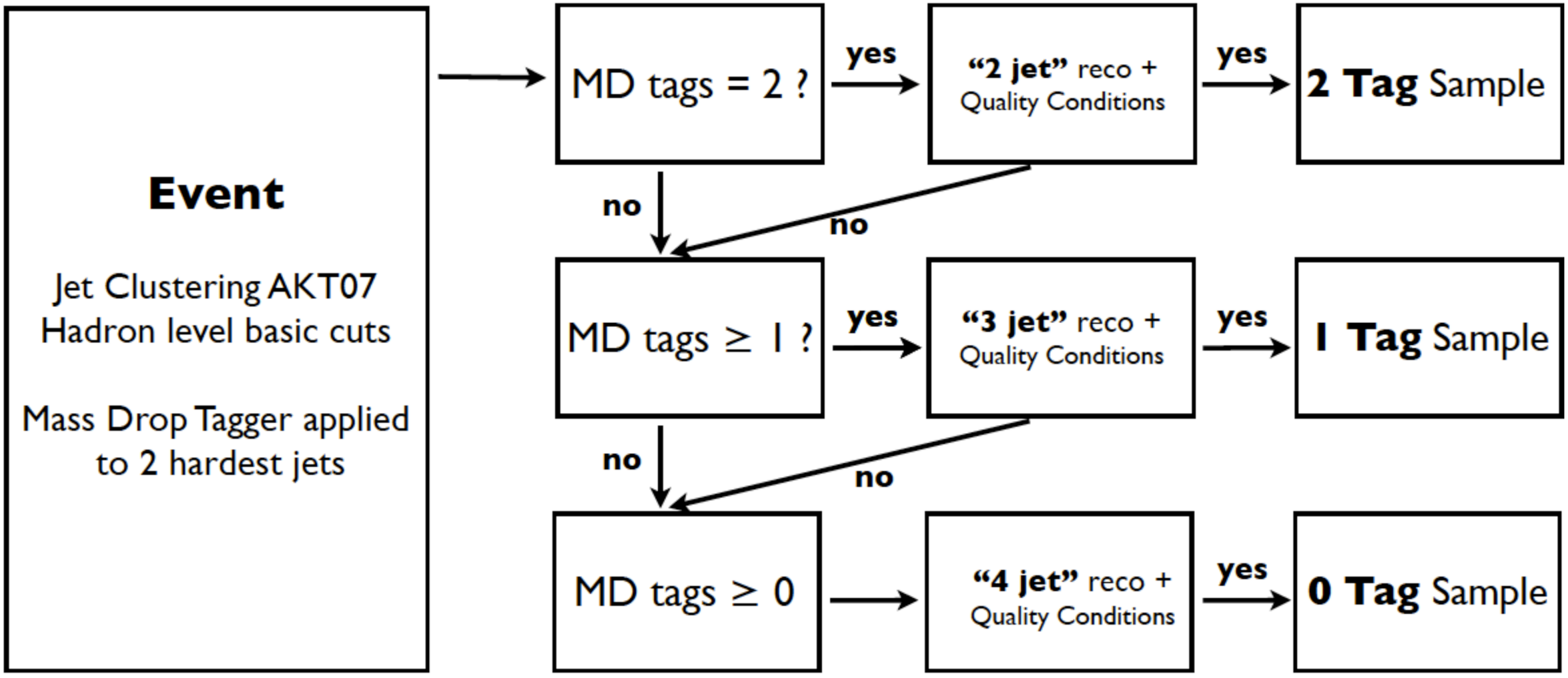
Due to small rates in the SM, only final states where at least one of the Higgs bosons decays into a **bb pair** are experimentally accessible at the LHC

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

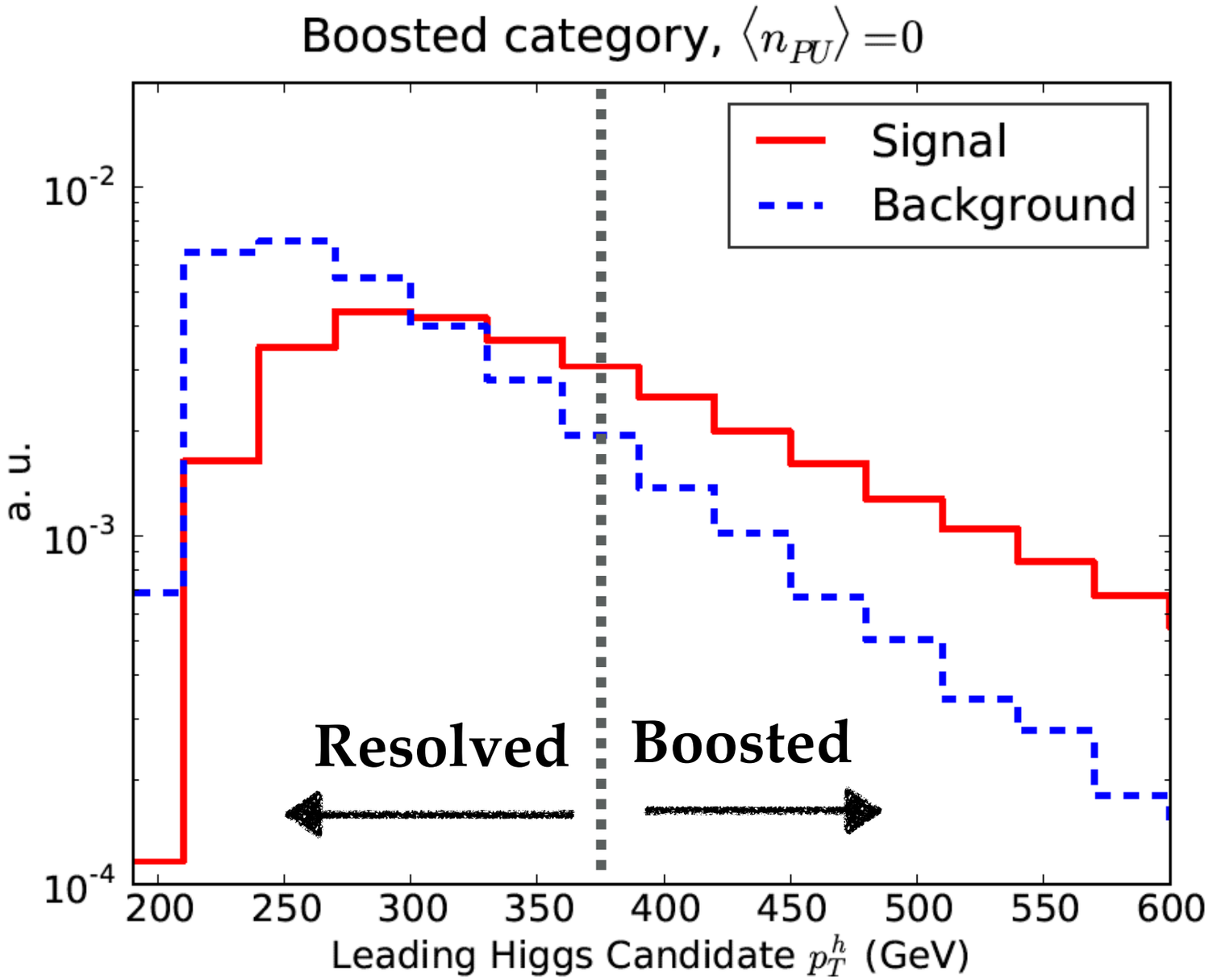
Process	Generator	N_{evt}	σ_{LO} (pb)	K-factor
$pp \to hh \to 4b$	`MadGraph5_aMC@NLO`	1M	$6.2 \cdot 10^{-3}$	2.4 (NNLO+NNLL [18, 19])
$pp \to b\bar{b}b\bar{b}$	`SHERPA`	3M	$1.1 \cdot 10^{3}$	1.6 (NLO [63])
$pp \to b\bar{b}jj$	`SHERPA`	3M	$2.7 \cdot 10^{5}$	1.3 (NLO [63])
$pp \to jjjj$	`SHERPA`	3M	$9.7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

ggF di-Higgs in the 4b final state



Similar contributions from the **boosted, intermediate,** and **resolved** categories to the overall signal efficiency

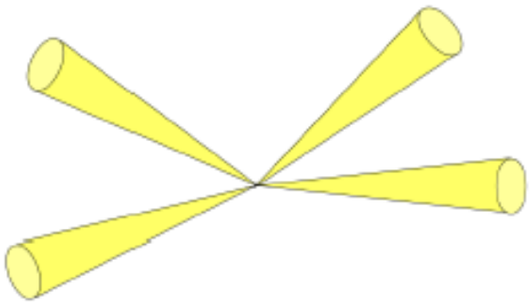
Resolved vs boosted



Similar contributions from the **boosted, intermediate,** and **resolved** categories to the overall signal efficiency

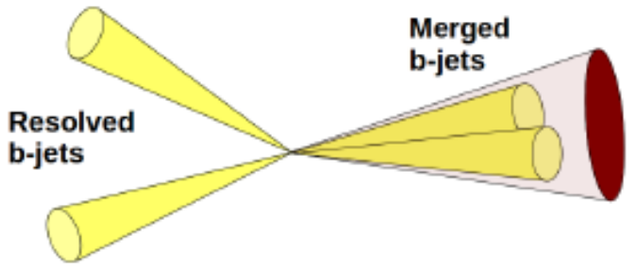
Analysis strategy

Resolved



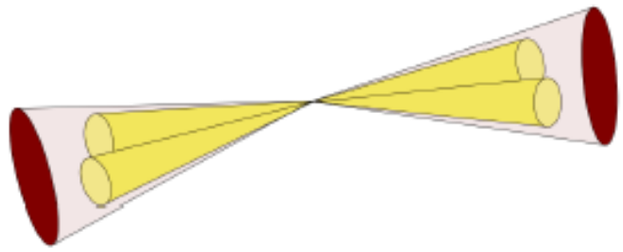
- ≥ 4 b-tagged small-R jets
- Higgs reconstruction from leading 4 jets
- Choice that minimises mass difference between dijet systems

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

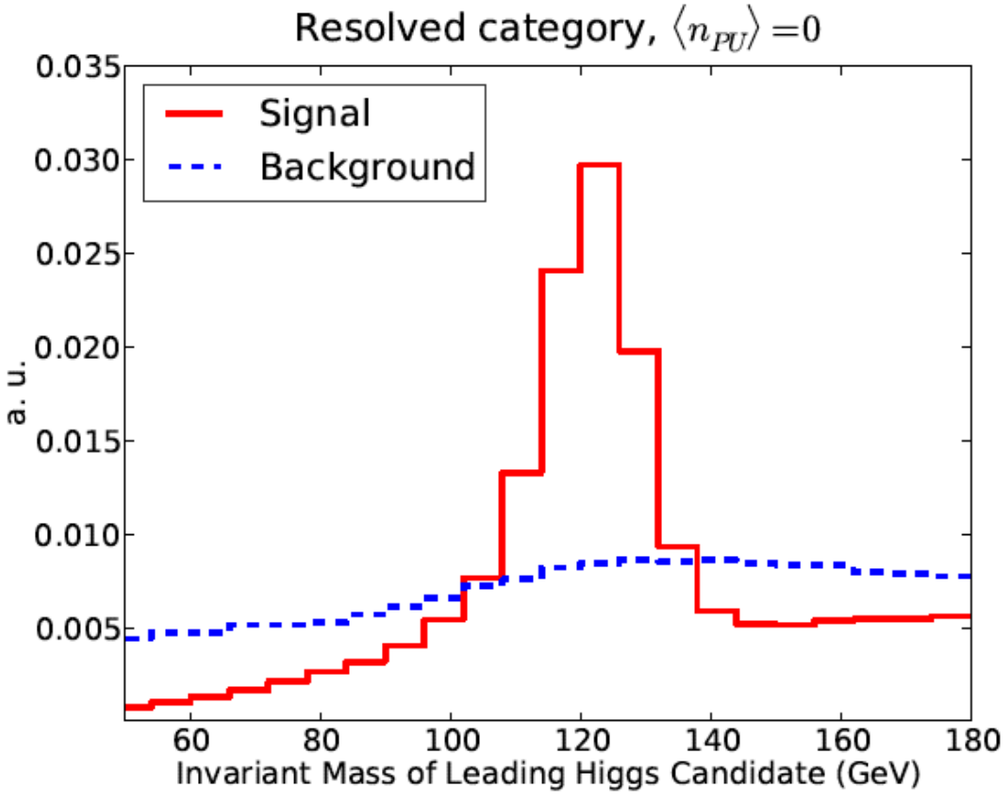
Boosted

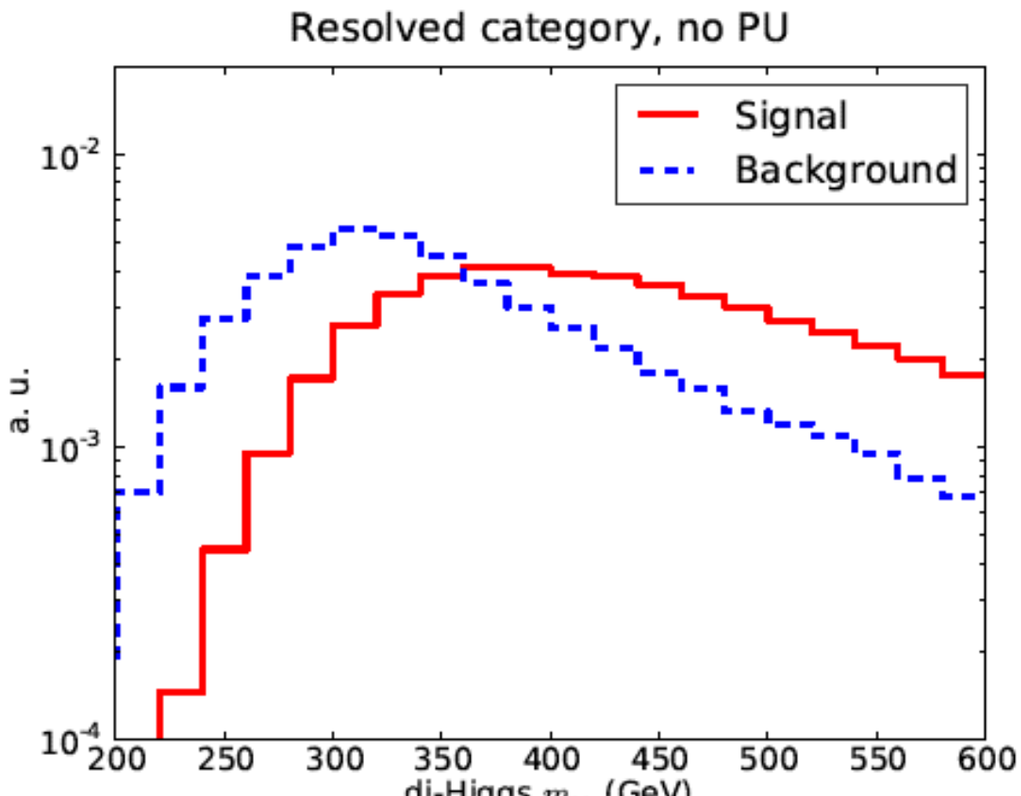


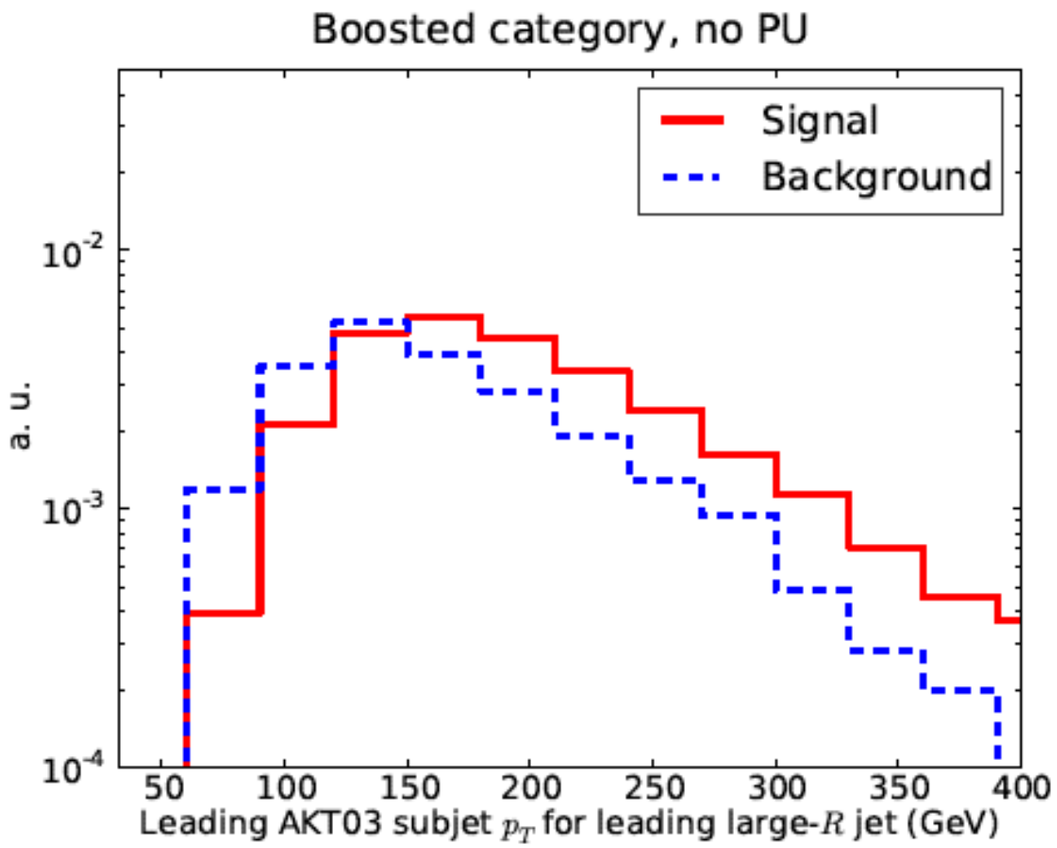
- ≥ 2 large-R jets (Higgs-tagged + b-tagged)
- Leading two jets taker as Higgs candidates

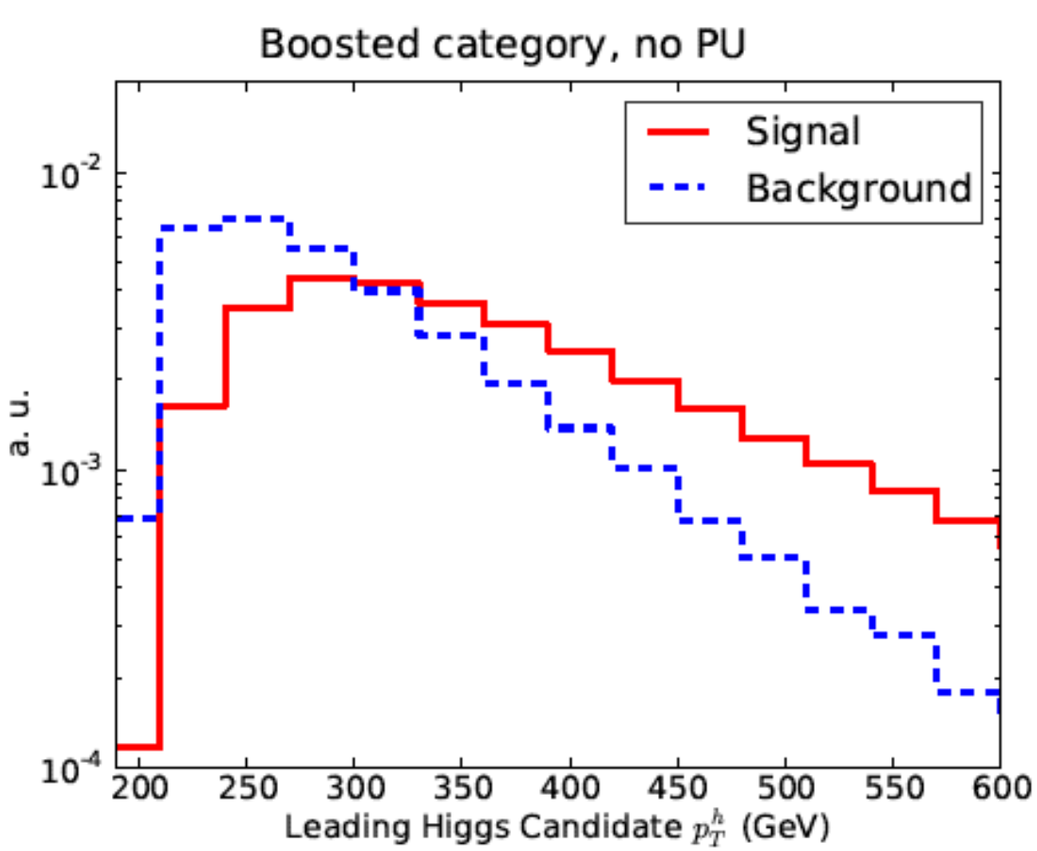
+ **Loose Higgs mass window cut**: $|m_{h,j} - 125\ \mathrm{GeV}| < 40\ \mathrm{GeV}\ , j = 1, 2$
+ **Rank categories** by $S/\sqrt{B}$ to make them **exclusive**: boosted > intermediate > resolved

Results of cut-based analysis

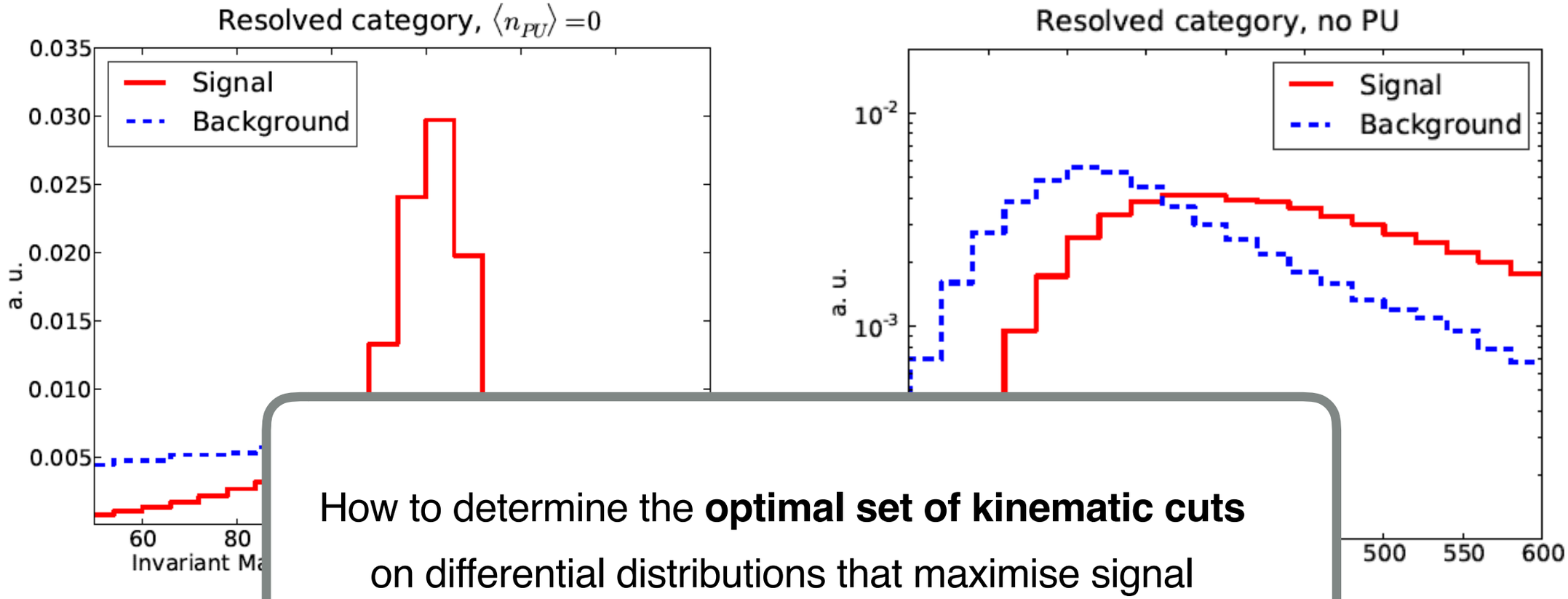




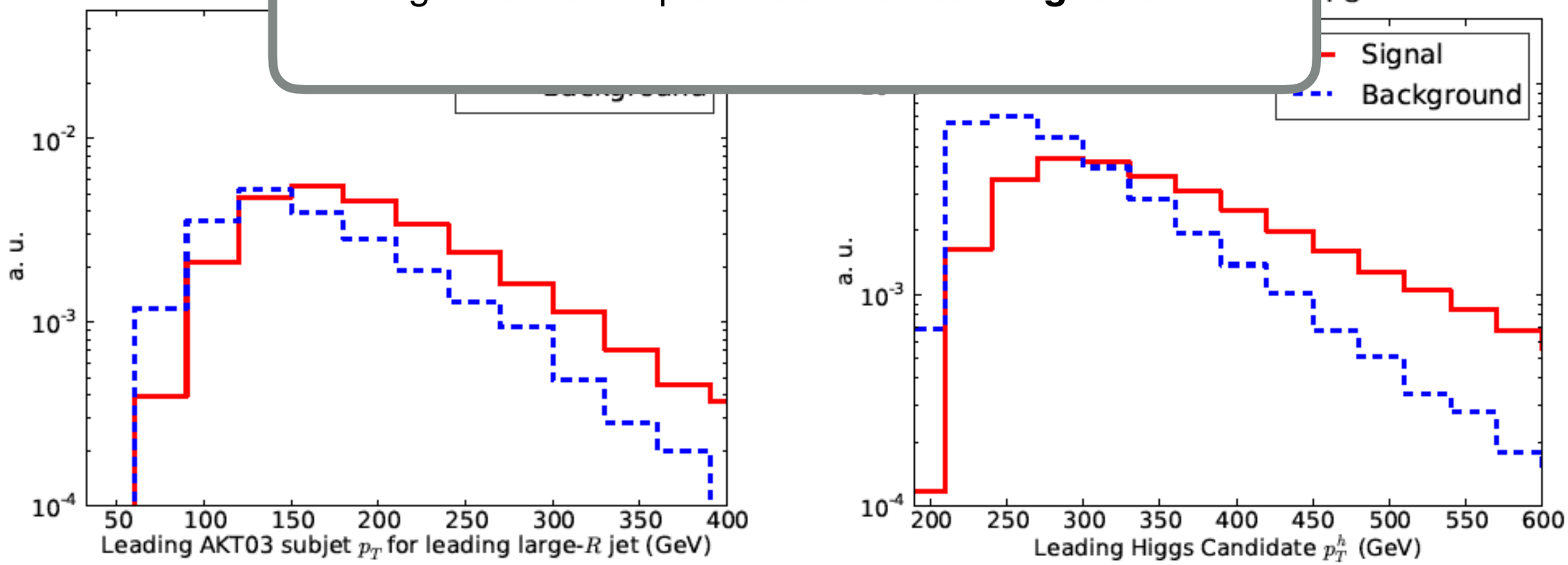




Results of cut-based analysis



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

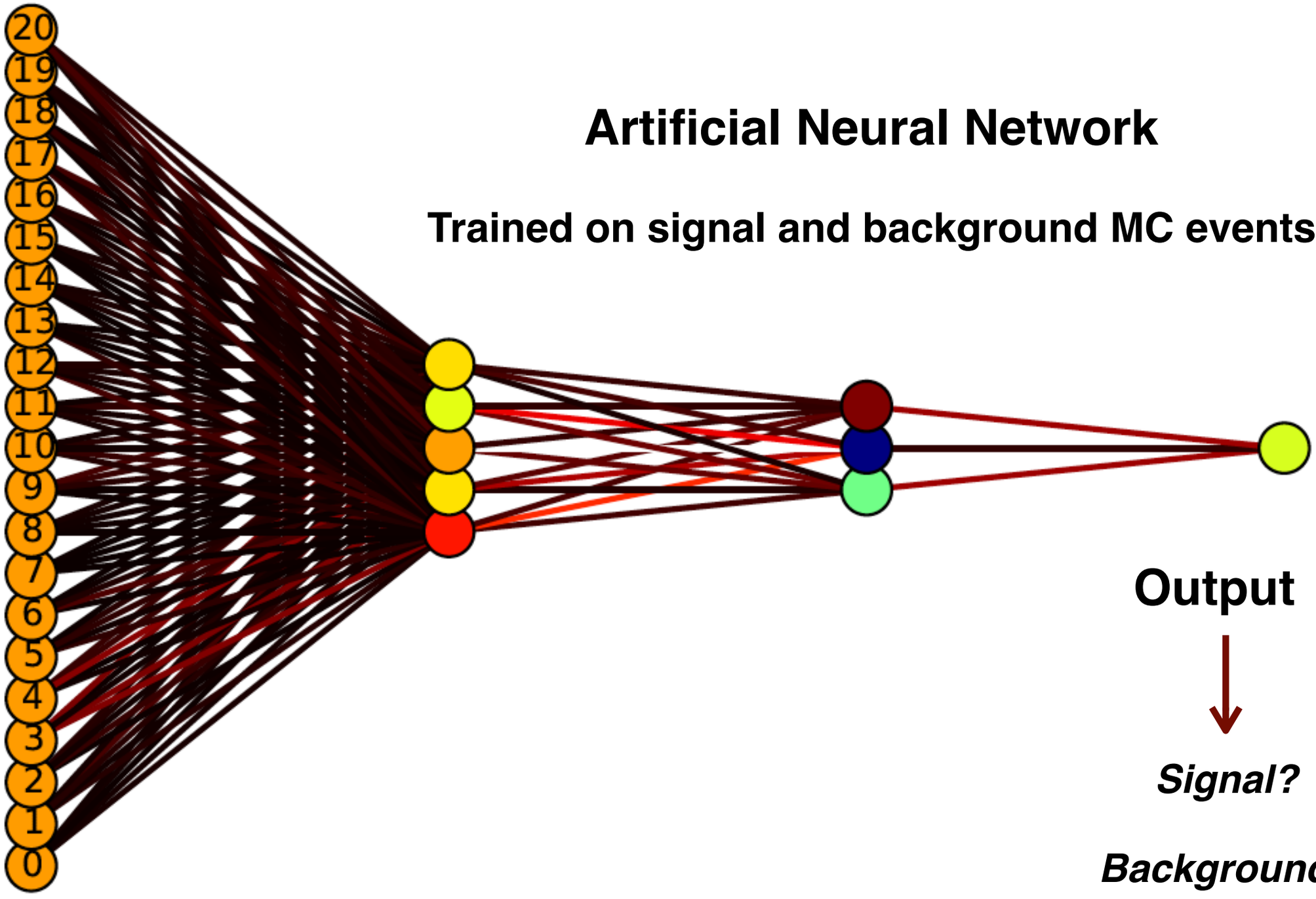
di-Higgs m

ECF

τ_{12}

Subjet pT

.....



Artificial Neural Network

Trained on signal and background MC events

Output

Signal?

Background?

ML discriminator

Given a set of N_{var} kinematic variables $\{k_i\}$ associated to event i, and a set of ANN weights $\{\omega\}$, the ANN output y_i interpreted as probability that event originates from signal process

$$y_i = P(y'_i = 1 | \{k\}_i, \{\omega\}) \, ,$$

with y'_i the true MC classification: $y'_i=1$ for signal, $y'_i=0$ for background

The **general classification probability** including background events is

$$P(y'_i | \{k\}_i, \{\omega\}) = y_i^{y'_i} (1 - y_i)^{1 - y'_i}$$

Thus the **error function to be minimised during the training** is the **cross-entropy:**

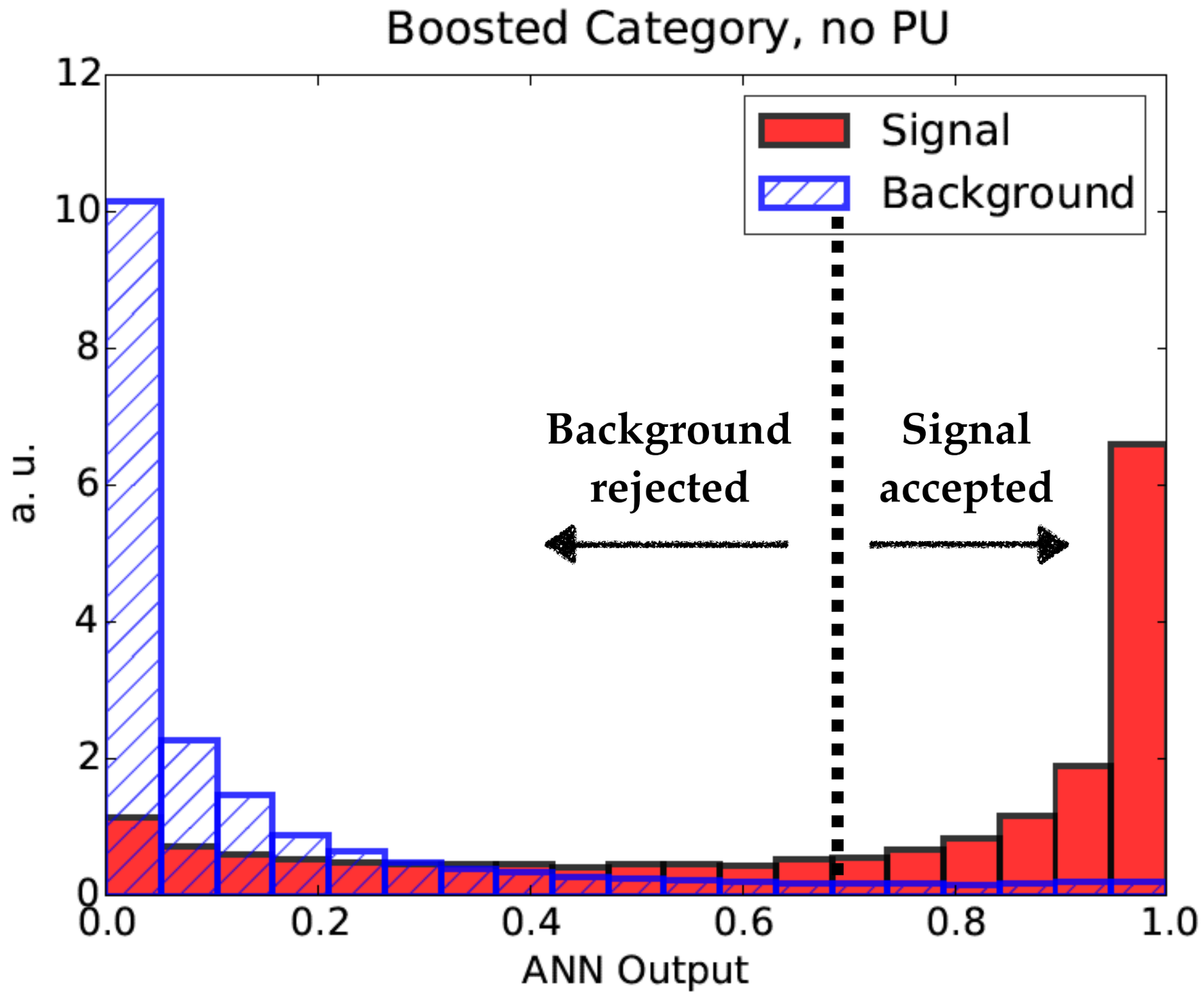
$$E(\{\omega\}) \equiv -\log \left(\prod_i^{N_{\mathrm{ev}}} P(y'_i | \{k\}_i, \{\omega\}) \right)$$

$$= \sum_i^{N_{\mathrm{ev}}} \left[y'_i \log y_i + (1 - y'_i) \log (1 - y_i) \right]$$

ANN training performed with **Genetic Algorithms** using **cross-validation stopping**

ML discriminator

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

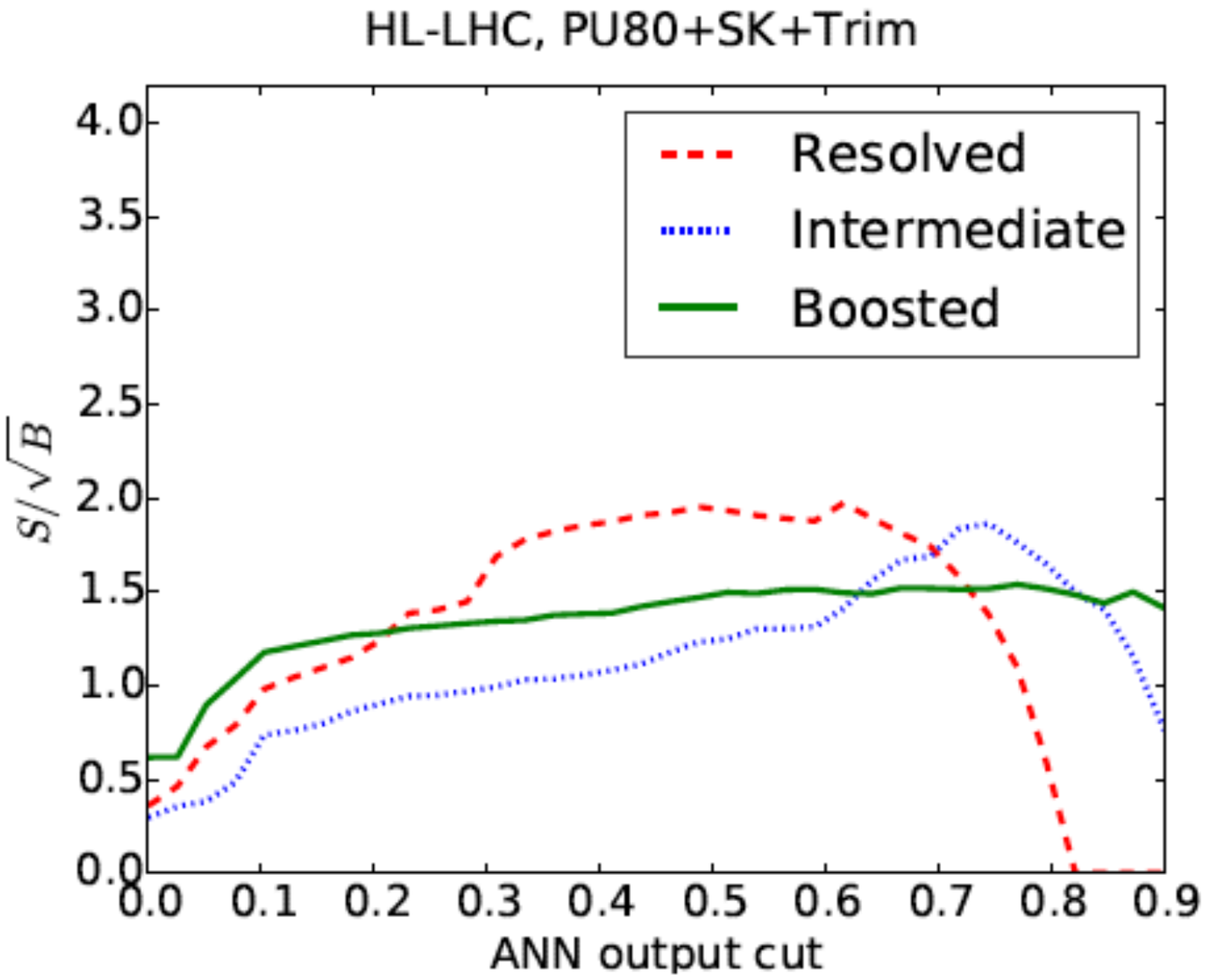


Results

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-MVA result

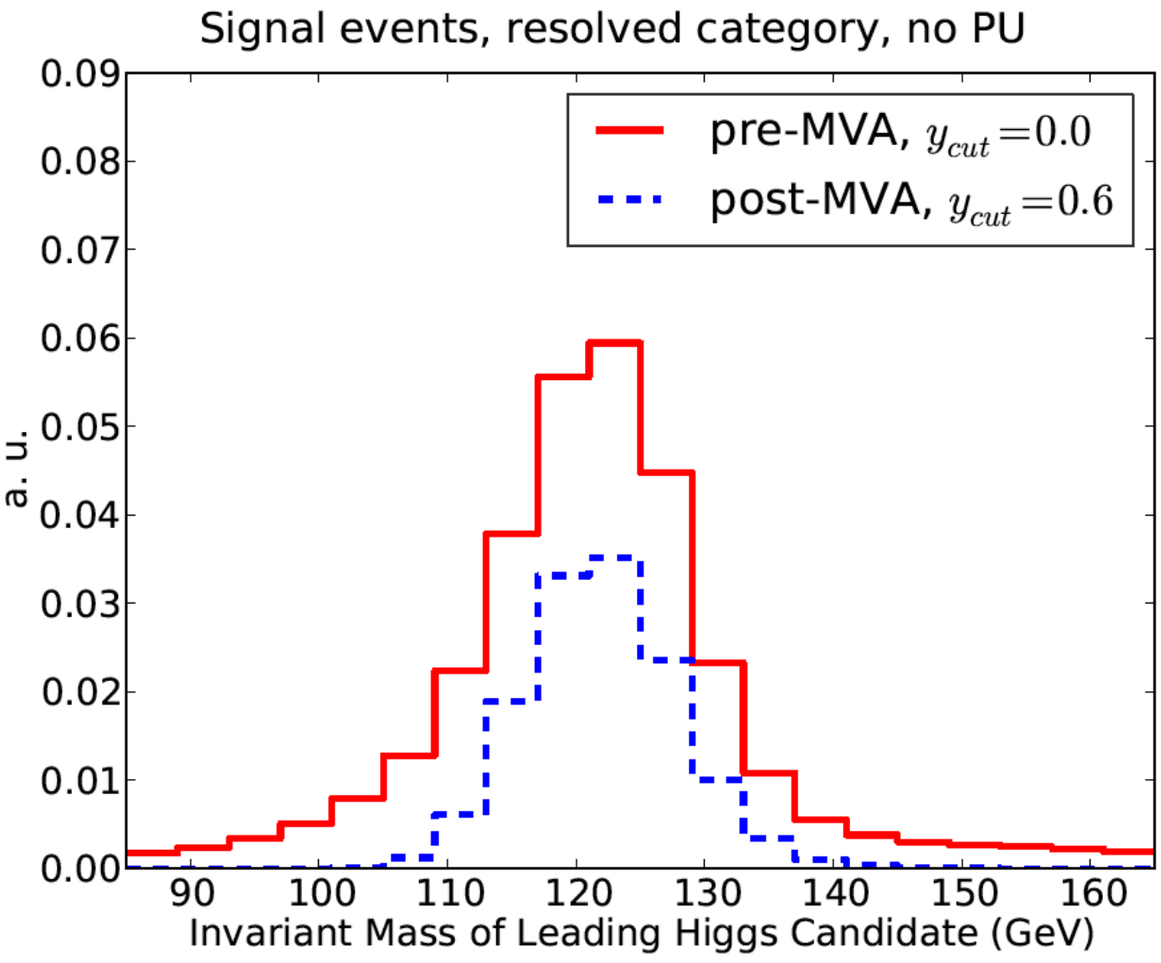


Opening the black box

- 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

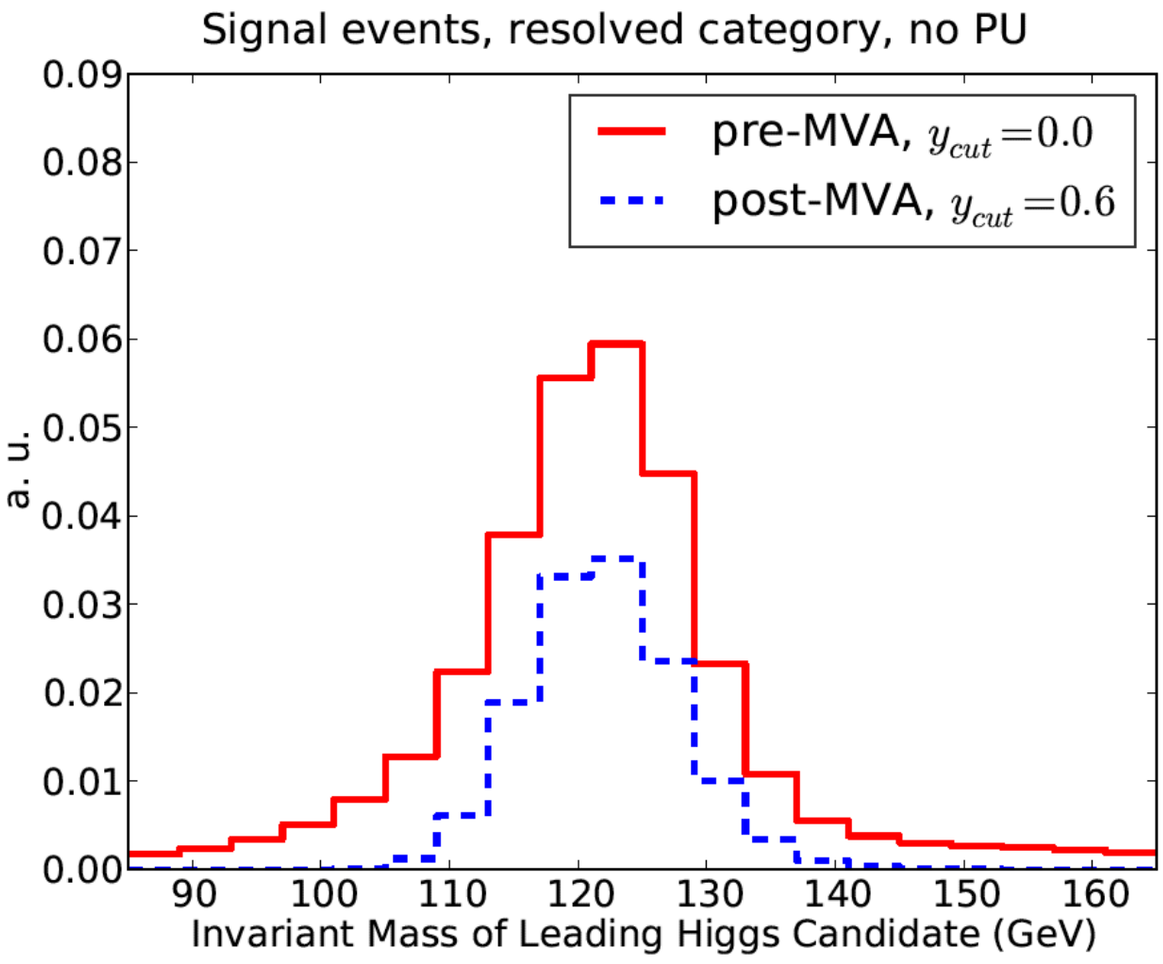
Opening the black box

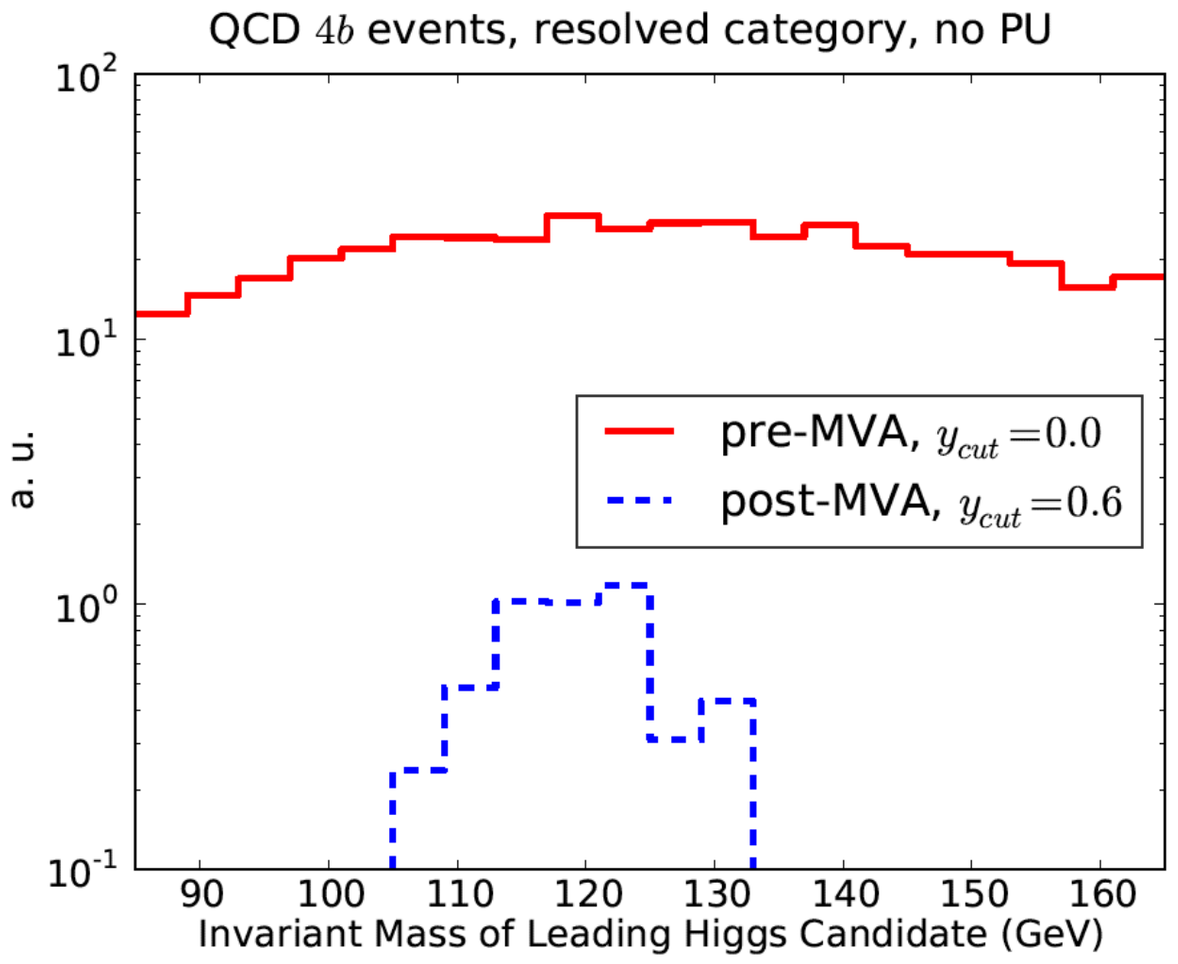
- ML tools often criticised as **black boxes**, with little understanding of inner working
- ANNs are simply a **set of combined kinematical cuts**, nothing mysterious in them!
- Plot kin distributions after and before the ANN cut to determine the **effective kinematic cuts**
- This info enough to **perform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