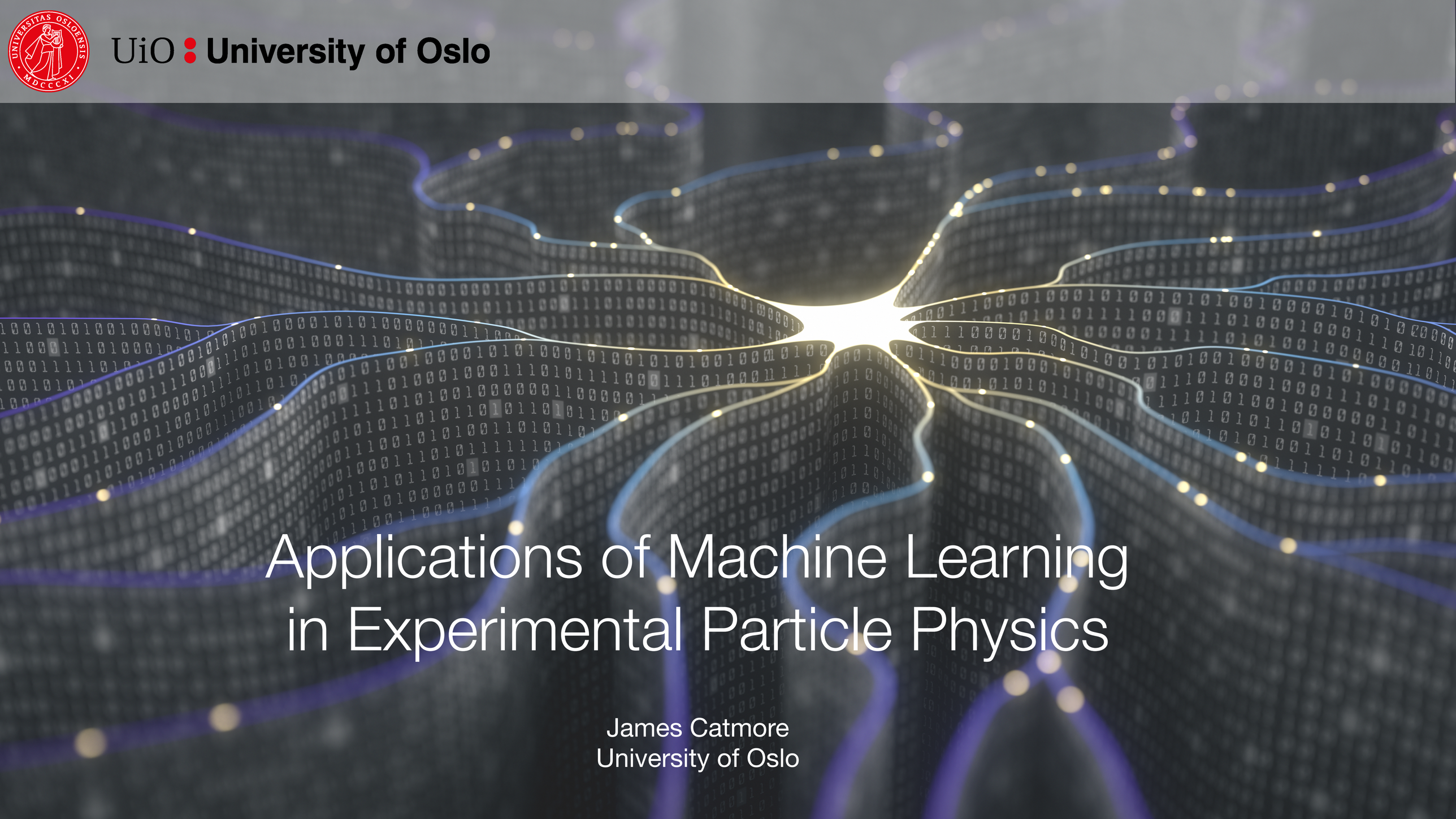
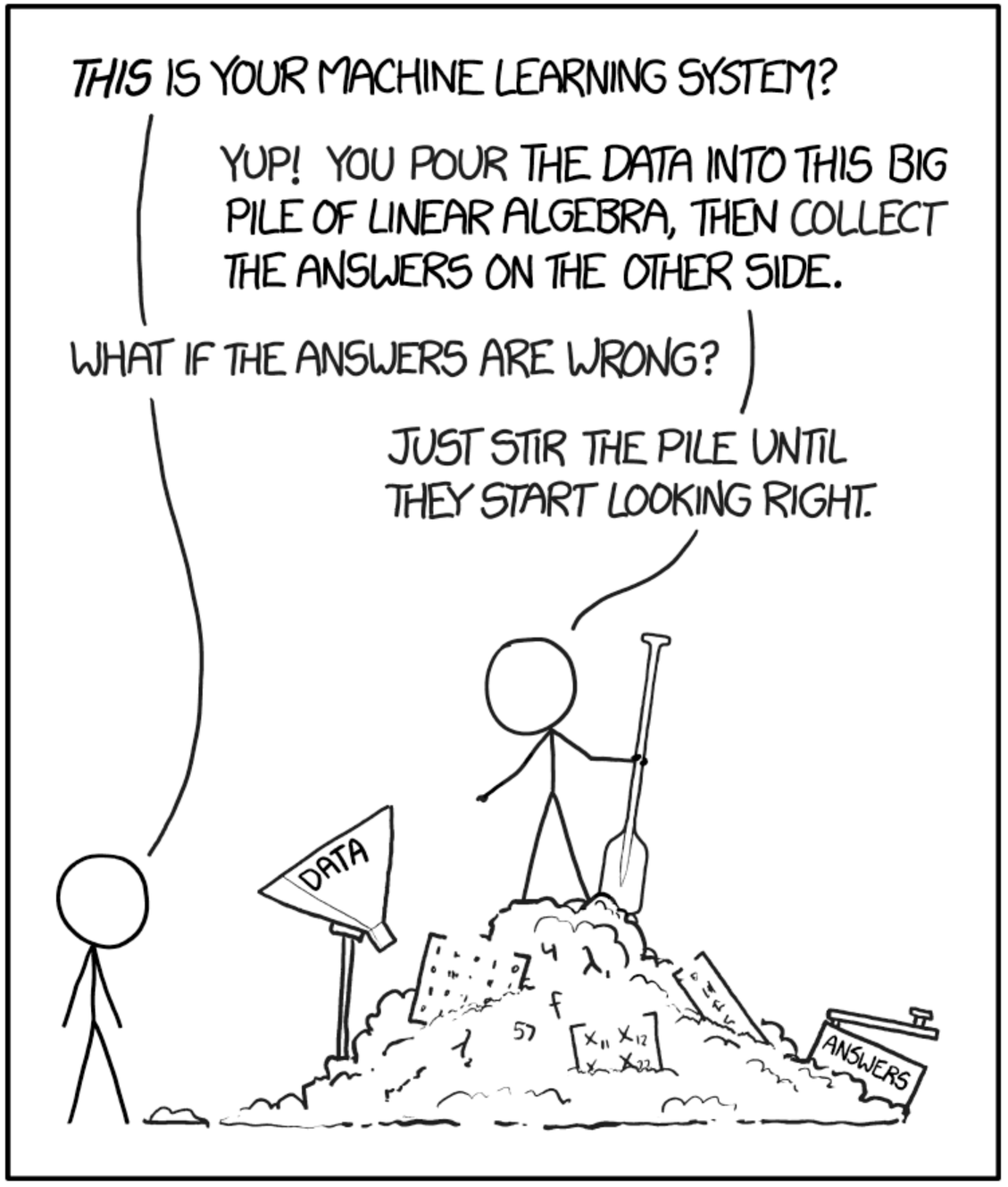


UiO : University of Oslo

Applications of Machine Learning in Experimental Particle Physics

James Catmore
University of Oslo


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

Reading material

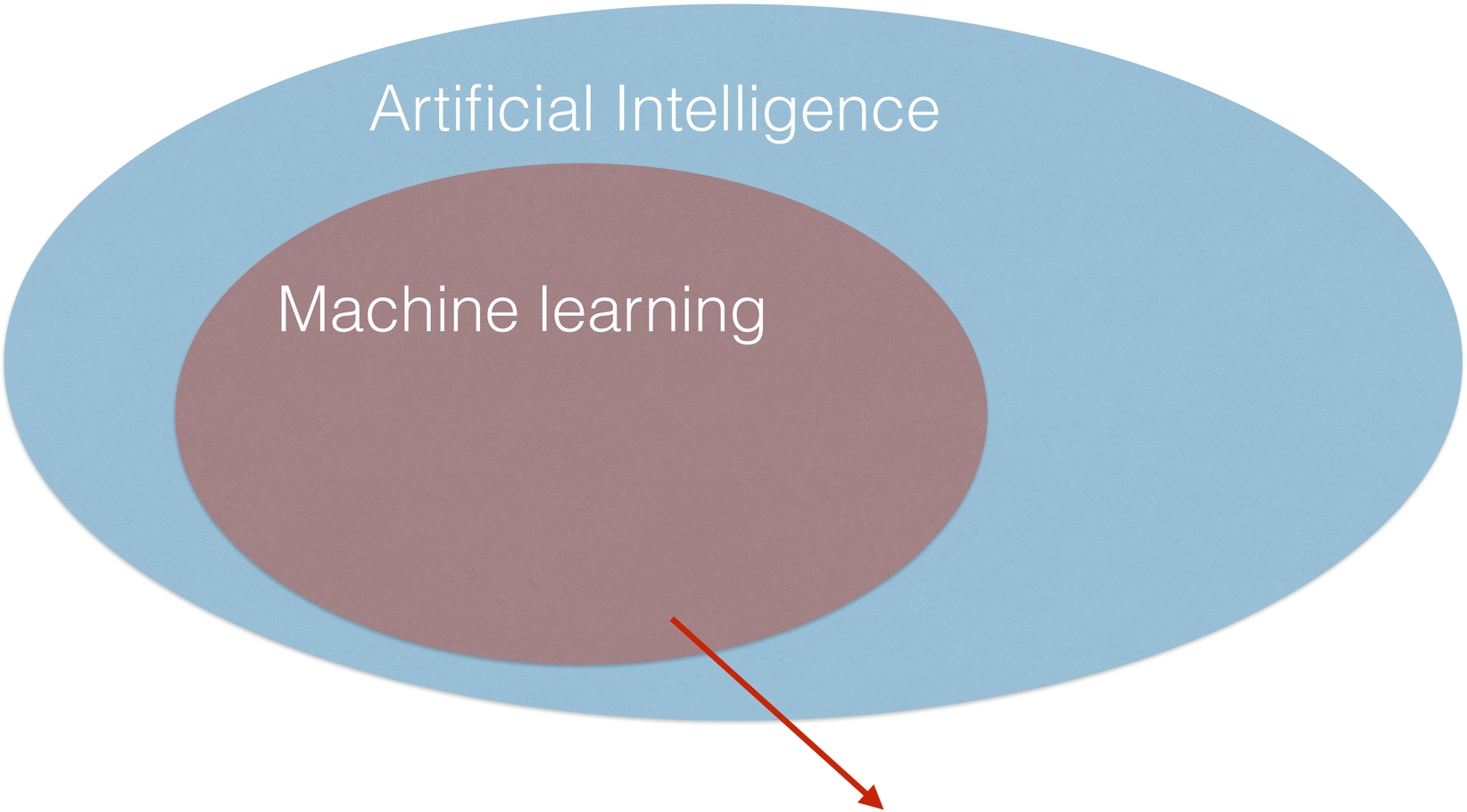
- There is a huge amount of material online, but here are a couple of highlights
- Famous Coursera course by Andrew Ng of Stanford University:
  - https://www.coursera.org/learn/machine-learning
- Deep learning book
  - https://www.deeplearningbook.org
- Many new developments in HEP ML end up in this journal:

https://www.springer.com/journal/41781

Overview

- Review of basic machine learning concepts
- ML techniques that are of particular interest in HEP
  - Boosted Decision Trees
  - Neural networks
- Software
- Current machine learning applications in High Energy Physics
- Future applications and new ideas
  - Generative models
  - Anomaly detection
- Thursday: chance to try some of this out!

What is "machine learning"



We used to call it
"multi-variate analysis"

uses statistical inference to extract generalities from "training" data

→ "learns" from the training data

→ when exposed to new data, demonstrates behaviours that have not been explicitly programmed

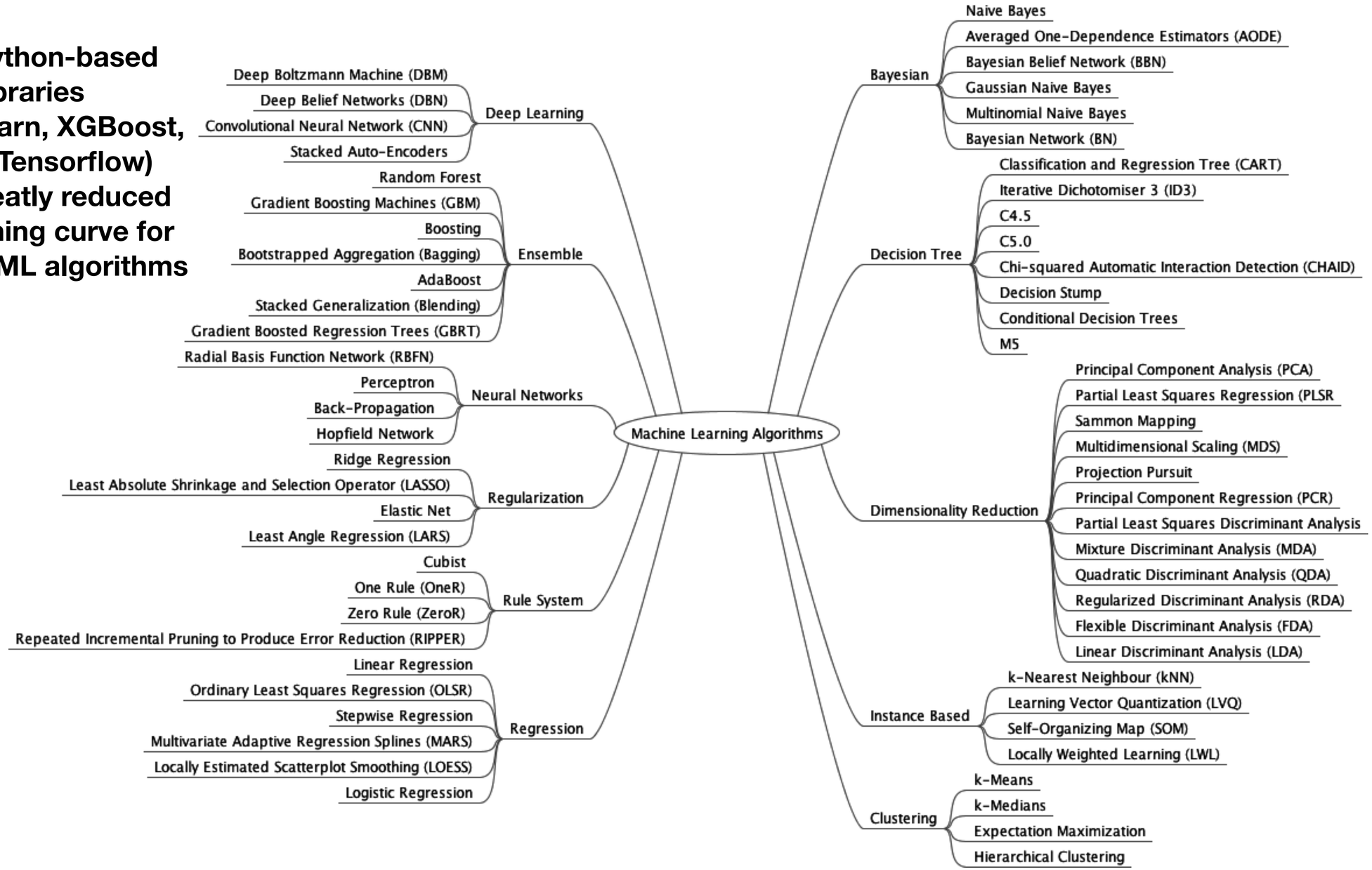
Supervised Learning

Unsupervised Learning

Reinforcement Learning

Types of ML algorithm

New Python-based libraries (SciKitLearn, XGBoost, Keras/Tensorflow) have greatly reduced the learning curve for applying ML algorithms



Objective function

- All machine learning algorithms are attempting to minimise an *objective function*:

$$J(\Theta) = L(\Theta) + \Omega(\Theta)$$

Loss function:
how far away from the target?

Regularisation: model complexity

$$L(\Theta) = L(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right)^2$$

Model parameters:
these are what we
change during training

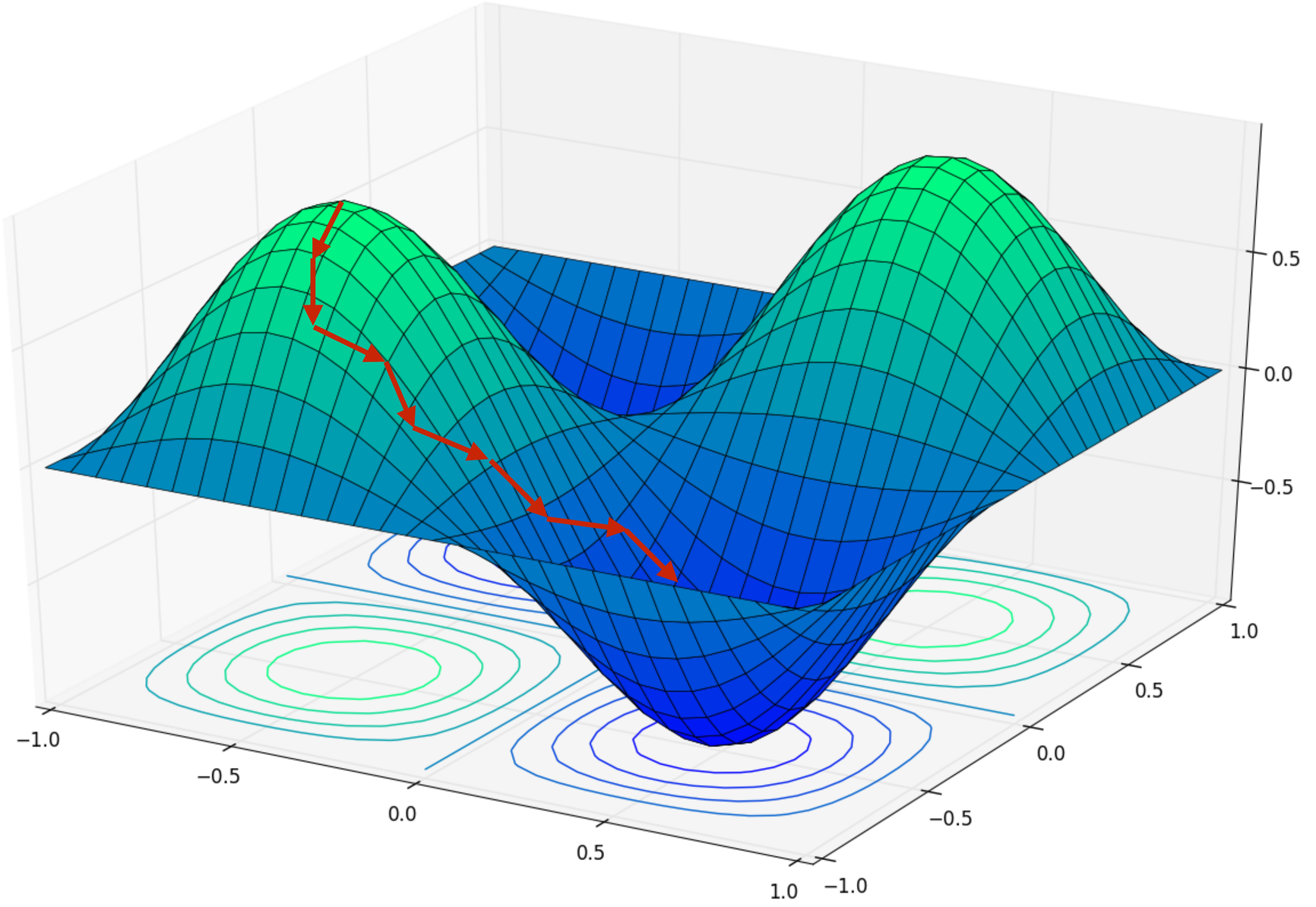
Mean squared error

Model

Training data

Target

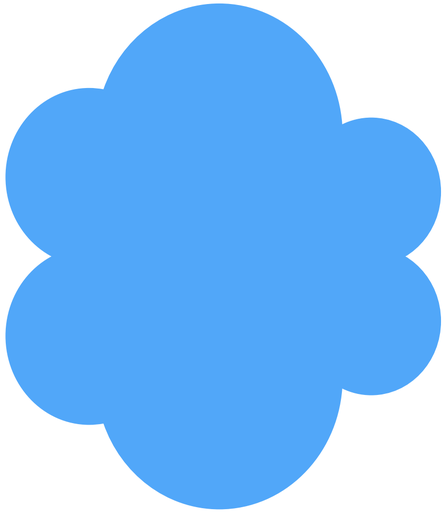
Method of Steepest Descent (a.k.a. Gradient Descent)



e.g. Minuit…

How supervised machine learning works

- Assume there is some data generating process that we wish to understand and/or whose behaviour we wish to be able to predict



PREPROCESSING

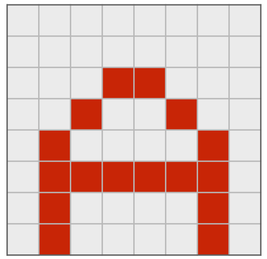
also… data preparation…
tabulation… feature extraction…

	var 1	var 2	…	var M
1	#	#	…	#
2	#	#	…	#
3	#	#	…	#
4	#	#	…	#
5	#	#	…	#
6	#	#	…	#
7	#	#	…	#
…	…	…	…	…
N	#	#	…	#

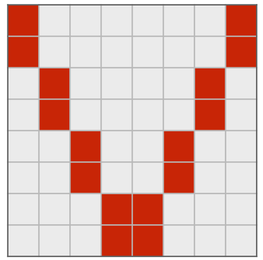
- Assume also that we are able to somehow **label** some of the data being produced…

Labelling examples

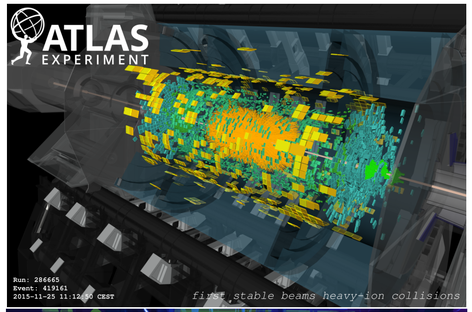
CLASSIFICATION



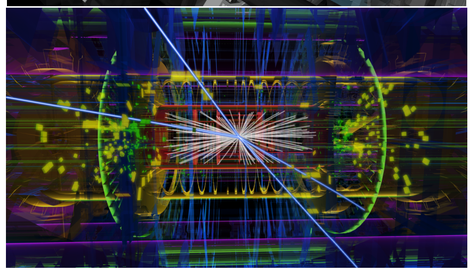
“A”



“V”



“Signal”

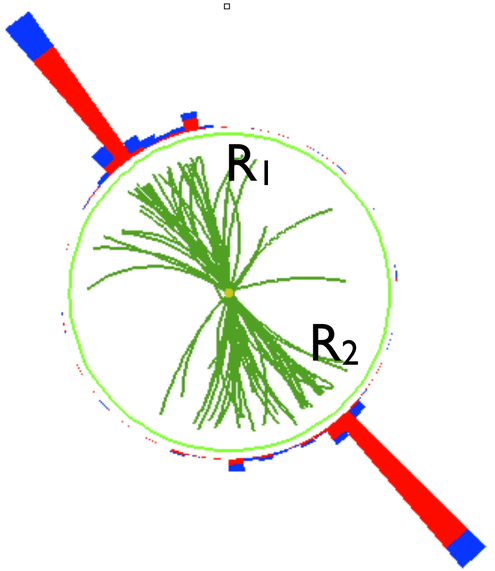


“Background”

REGRESSION

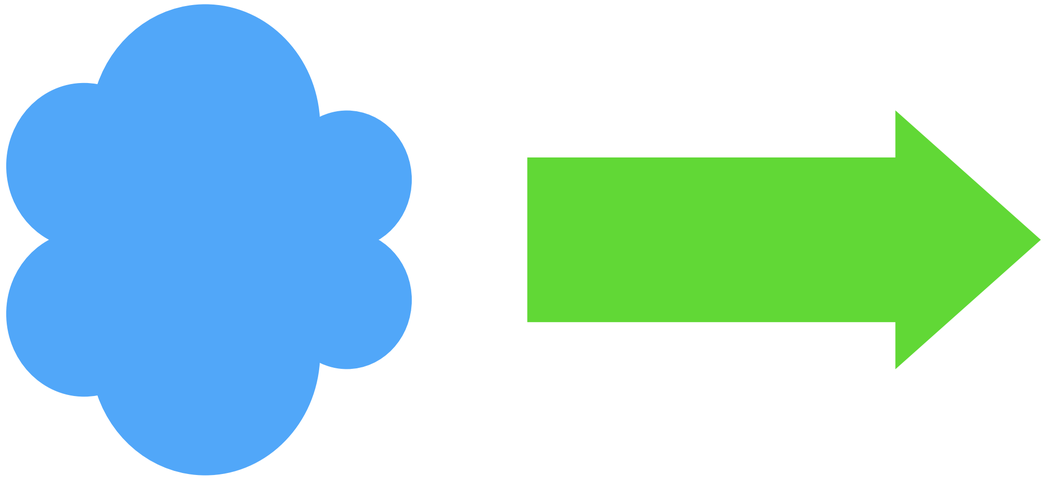
NOK 8 600 000

NOK 15 200 000



How supervised machine learning works

- Suppose that we collect and (somehow) label a batch of data from the generating process



	var 1	var 2	…	var M	**LABEL**
1	#	#	…	#	#
2	#	#	…	#	#
3	#	#	…	#	#
4	#	#	…	#	#
5	#	#	…	#	#
6	#	#	…	#	#
7	#	#	…	#	#
…	…	…	…	…	…
N	#	#	…	#	#

- Let us split this sample into three pieces

Training sample

Validation sample

Evaluation sample

How supervised machine learning works

Training

var 1	var 2	…	var M	**LABEL**
#	#	…	#	#
#	#	…	#	#
#	#	…	#	#
#	#	…	#	#
#	#	…	#	#
#	#	…	#	#
#	#	…	#	#
…	…	…	…	…
#	#	…	#	#

ALGORITHM

Hyperparameters

How supervised machine learning works

Validation

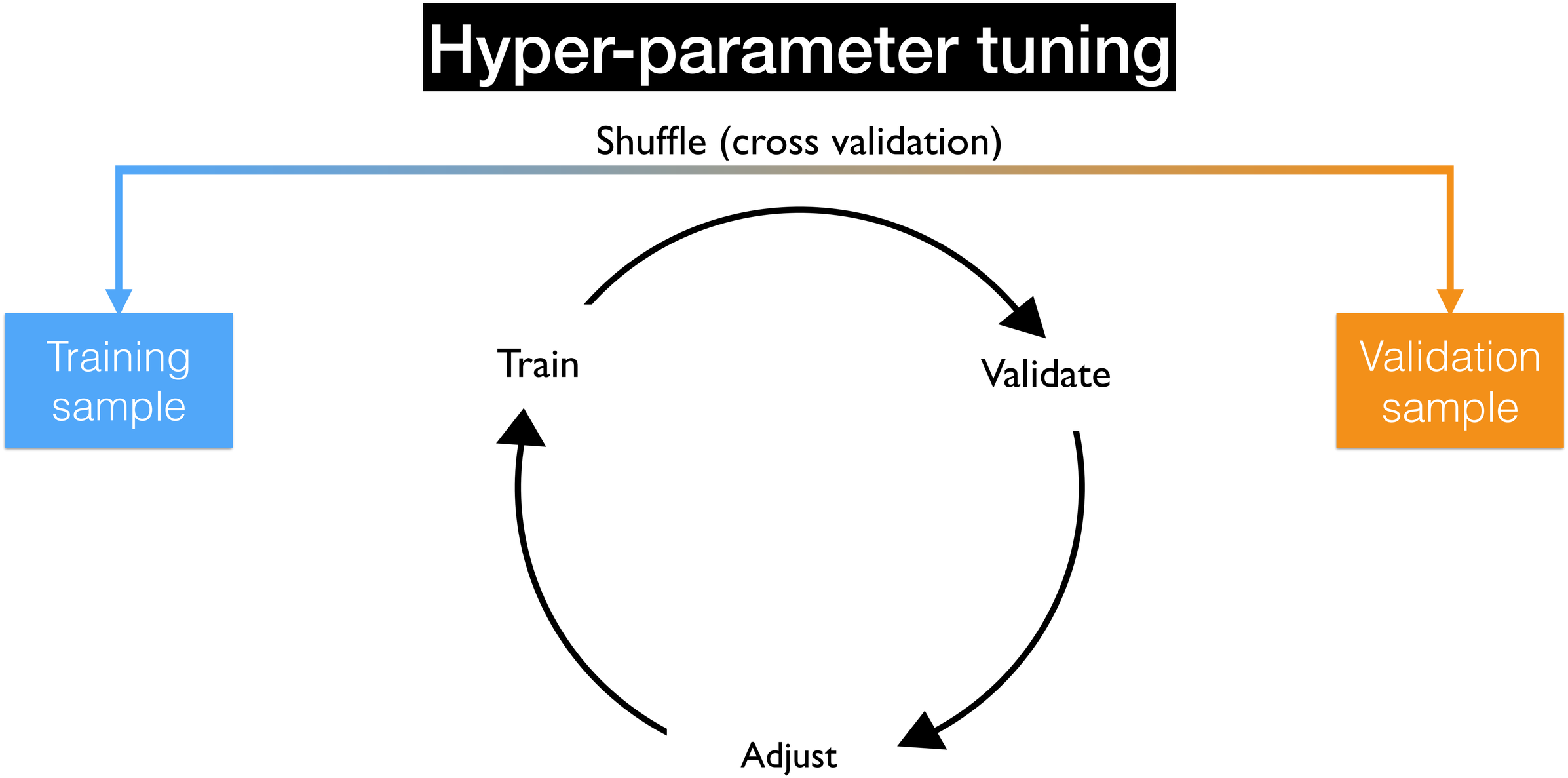
Comparison between the prediction and labels allows a determination of the algorithm performance

var 1	var 2	…	var M
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
…	…	…	…
#	#	…	#

PREDICTION
#
#
#
#
#
#
#
…
#

LABEL
#
#
#
#
#
#
#
…
#

Hyperparameters can be adjusted and the training repeated


Hyper-parameter tuning
Shuffle (cross validation)
Training sample
Validation sample
Train
Validate
Adjust

Performance evaluation

var 1	var 2	…	var M
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
…	…	…	…
#	#	…	#

PREDICTION
#
#
#
#
#
#
#
…
#

LABEL
#
#
#
#
#
#
#
…
#

No further adjustments permitted without fresh training data.

The performance as evaluated in this step is assumed to be the working performance of the algorithm

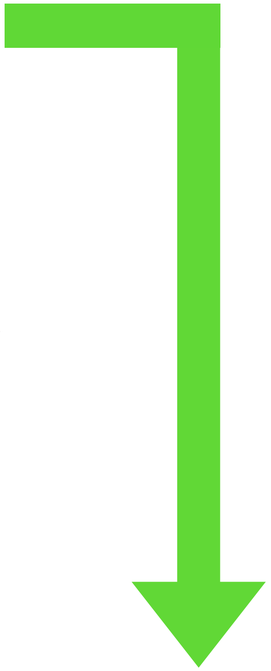
How supervised machine learning works

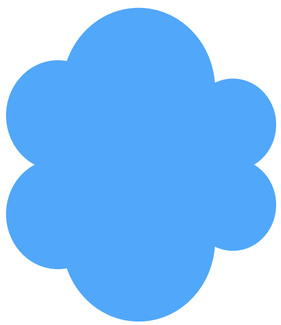
Deployment

var 1	var 2	…	var M
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
…	…	…	…
#	#	…	#

TRAINED ALGORITHM

PREDICTION
#
#
#
#
#
#
#
#
…
#





The predictions are now used for practical purposes

Projection to a single variable

var 1	var 2	…	var M
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
…	…	…	…
#	#	…	#

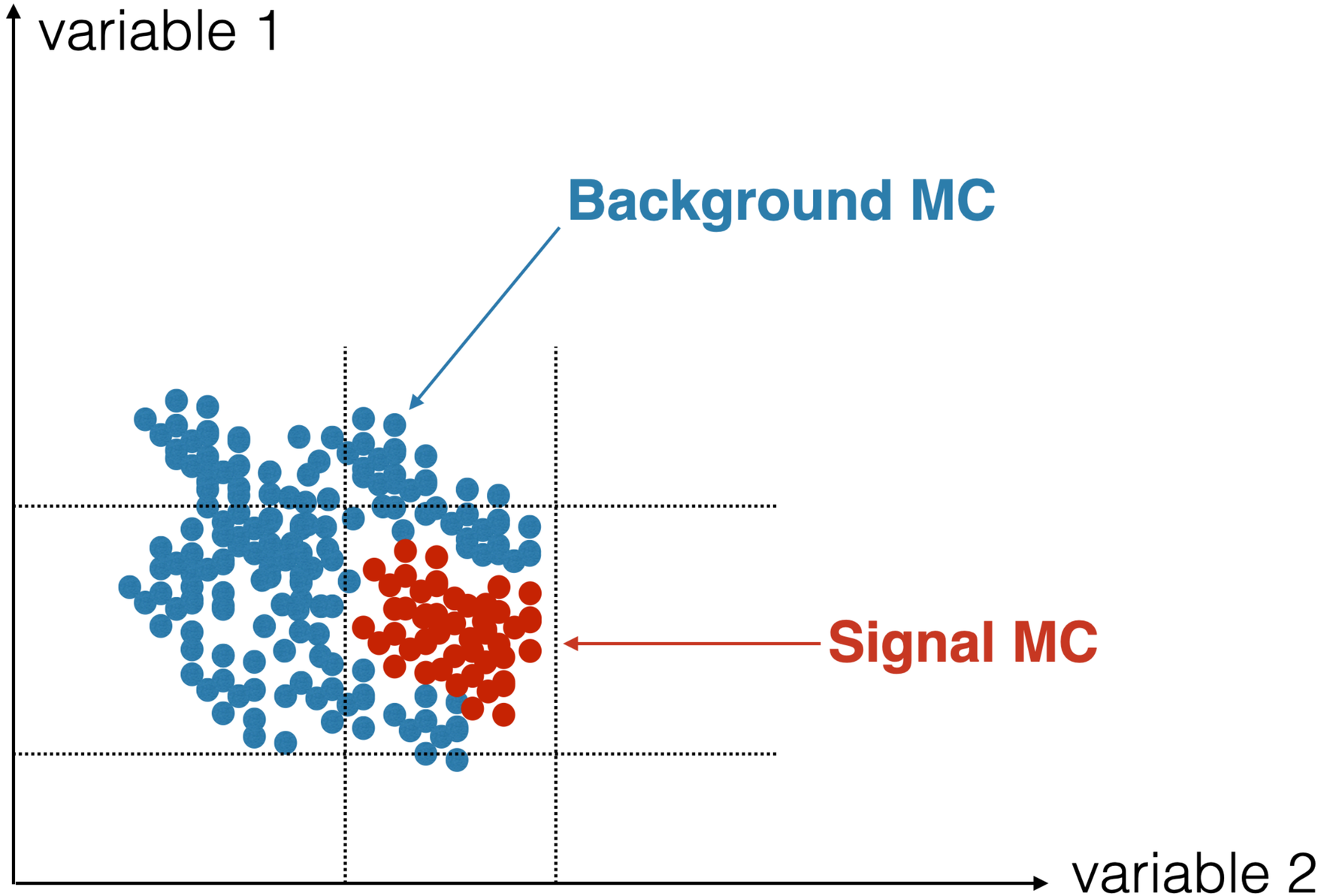
PREDICTION
#
#
#
#
#
#
#
#
…
#

M variables → **1 variable**

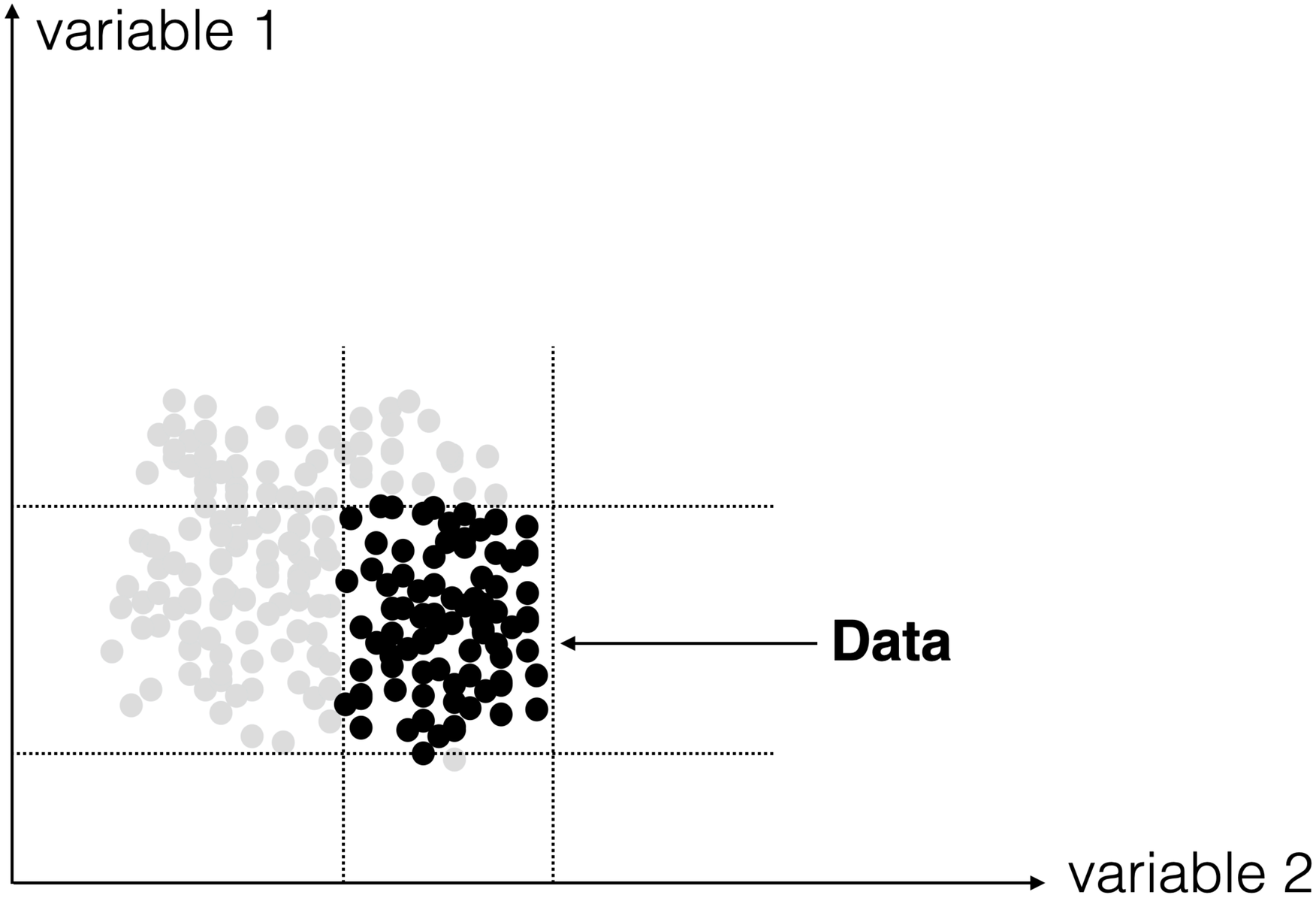
$$f(x_1, x_2, \ldots, x_M) = y$$

The algorithm learns how to project M variables into a single variable

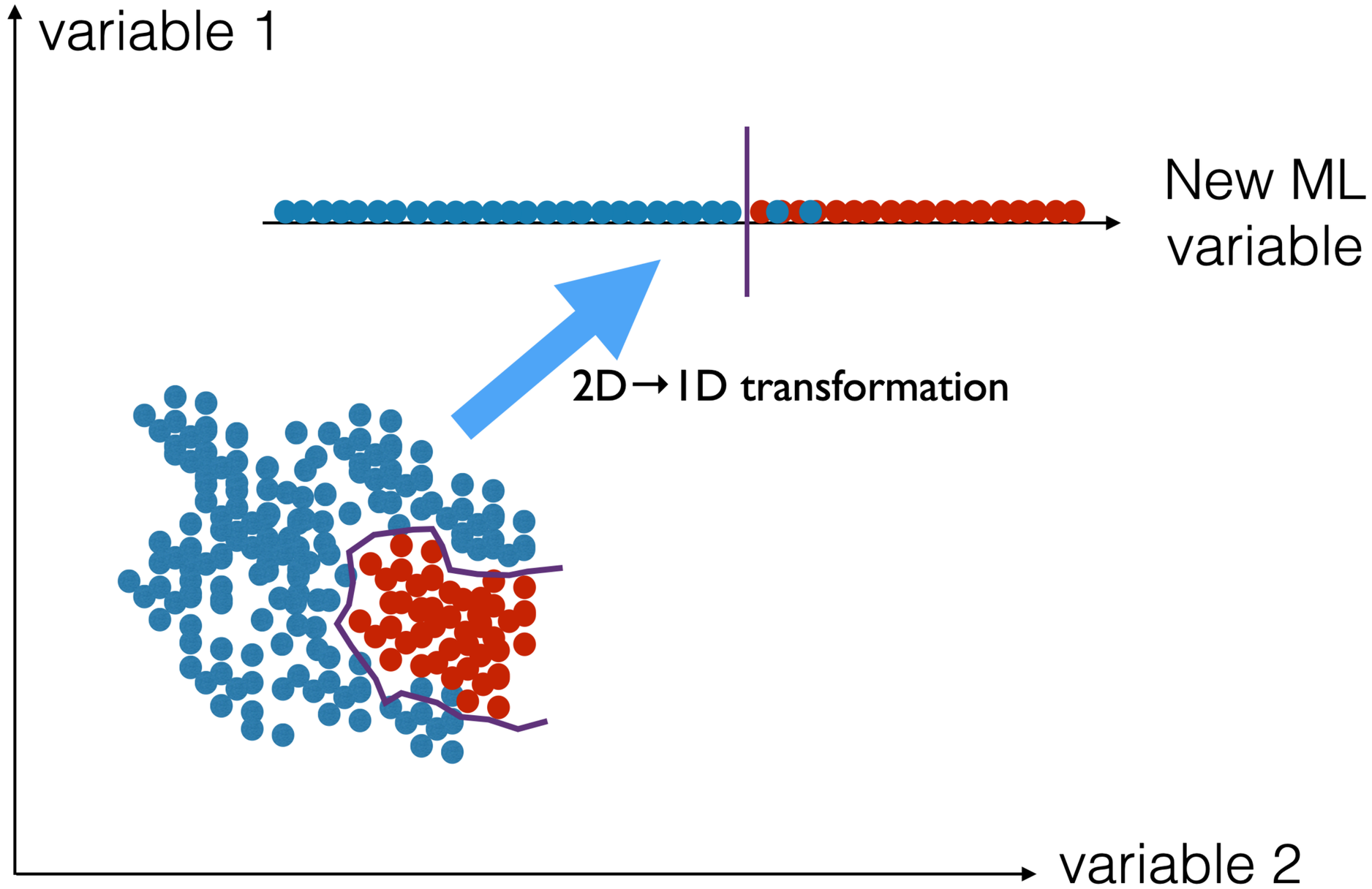
Separating signal from background: rectangular cuts



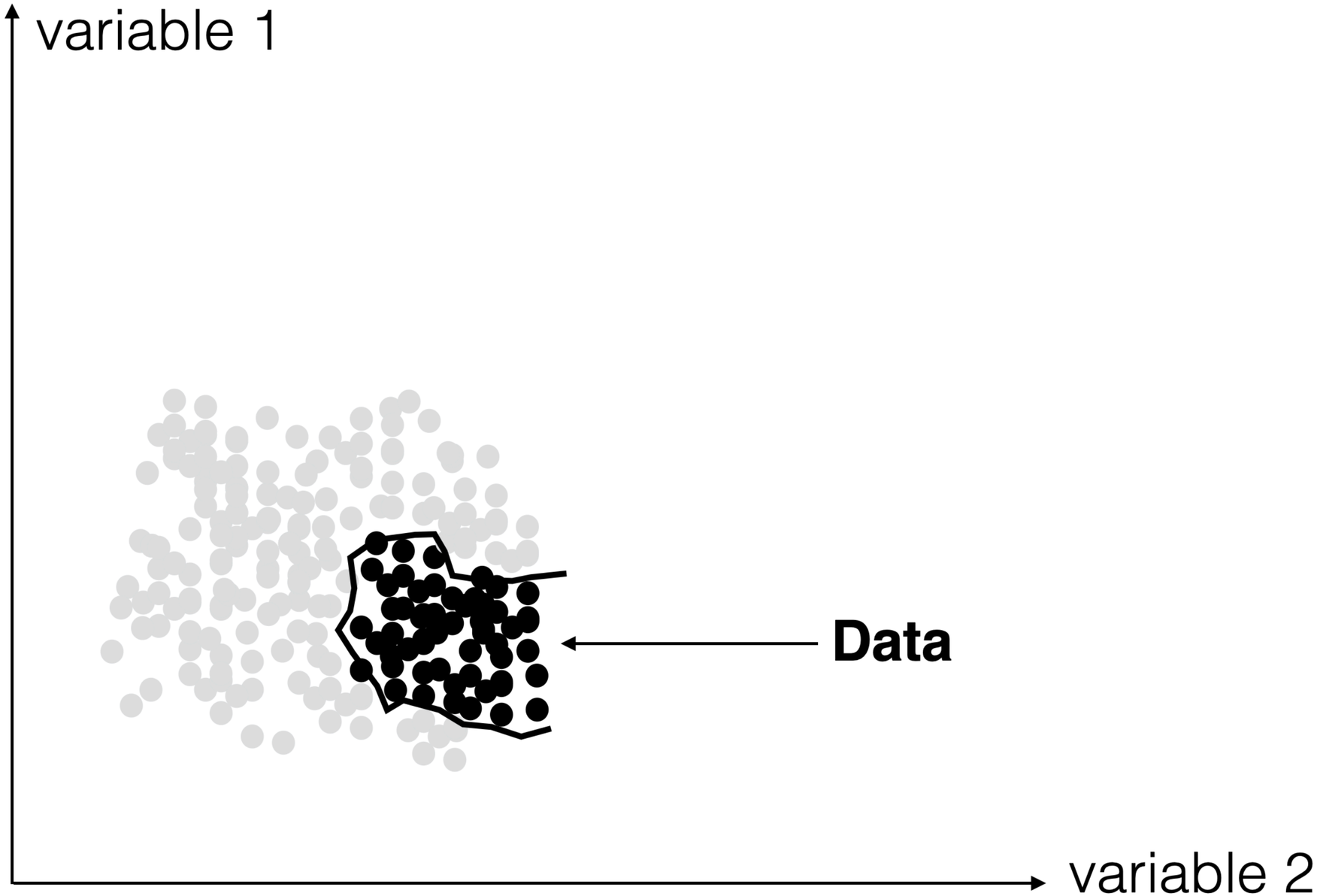
Separating signal from background: rectangular cuts



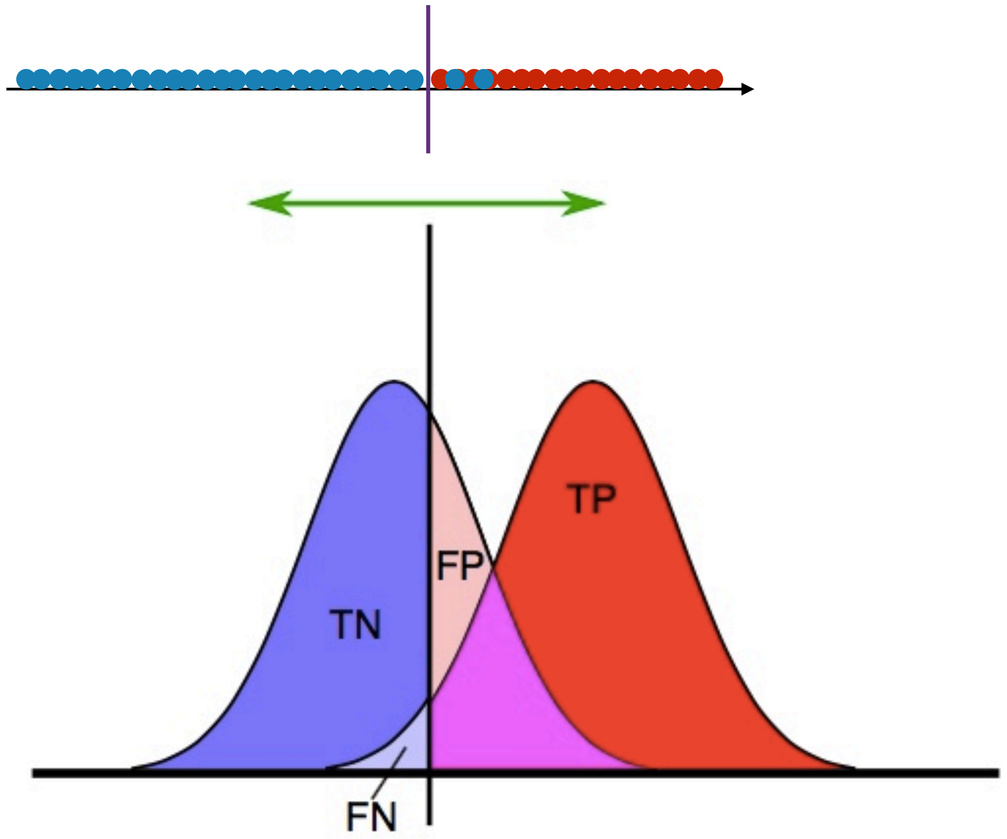
Separating signal from background: ML



Separating signal from background: ML

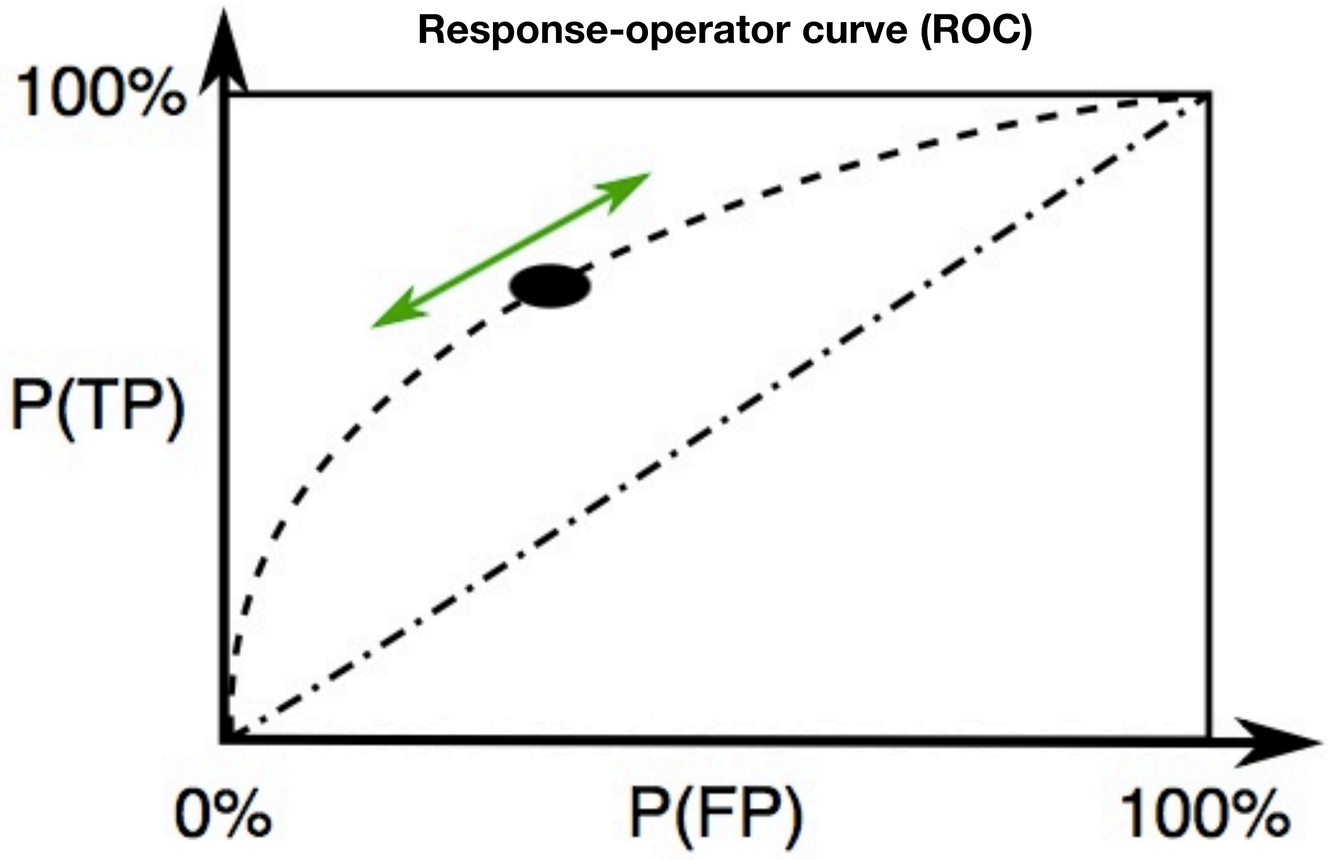


Confusion matrices and ROCs



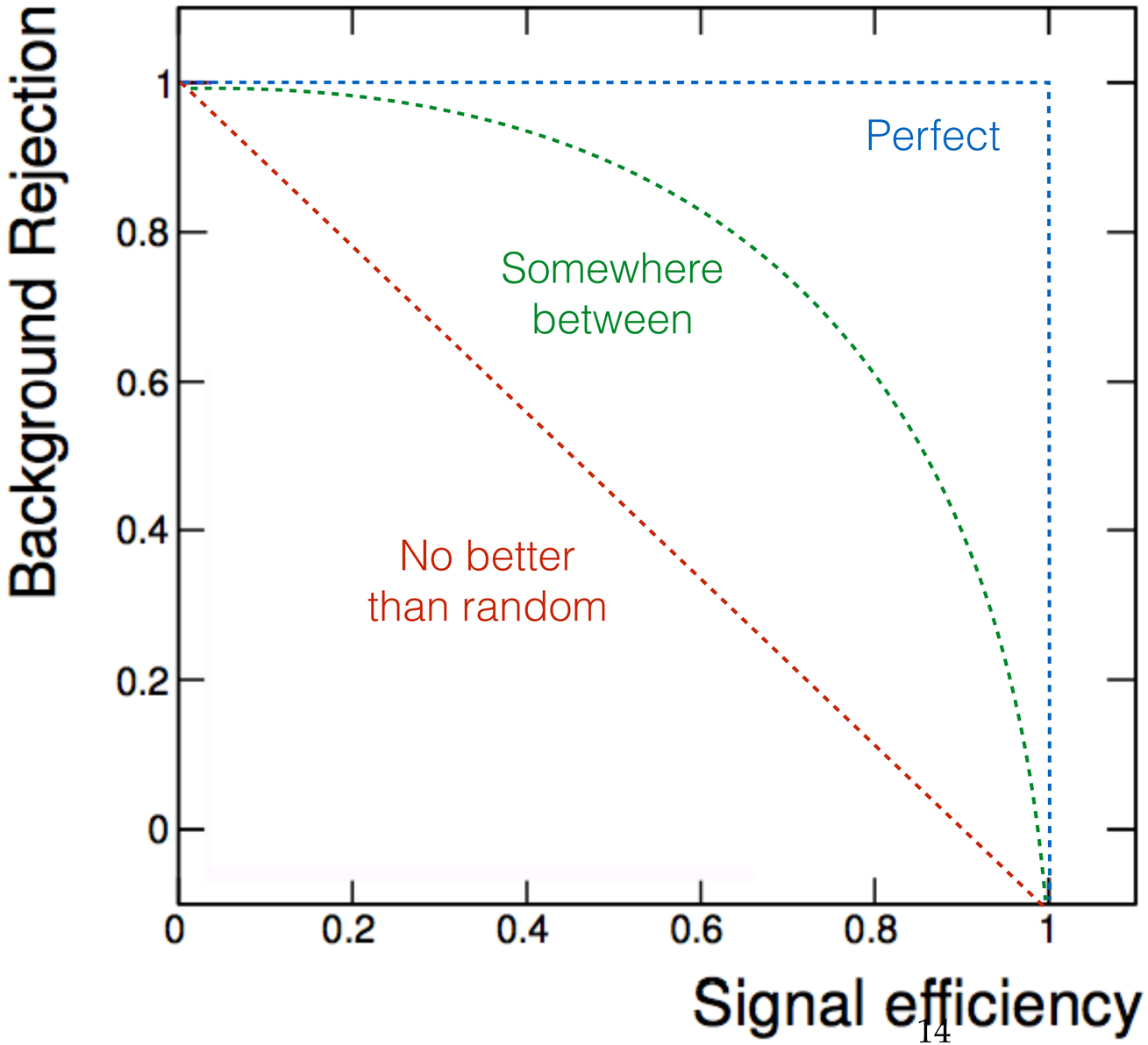
TP	FP
FN	TN

Confusion matrix

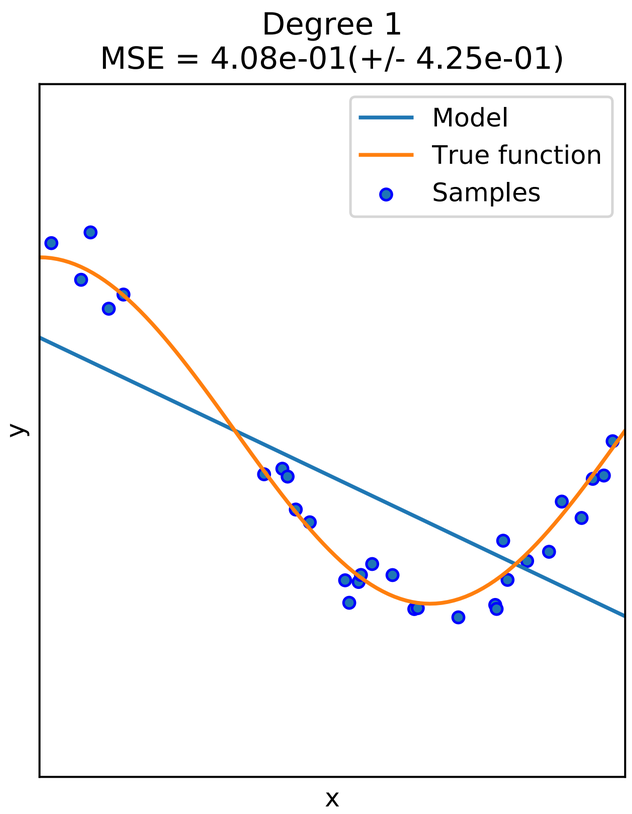


ROC as a measure of performance

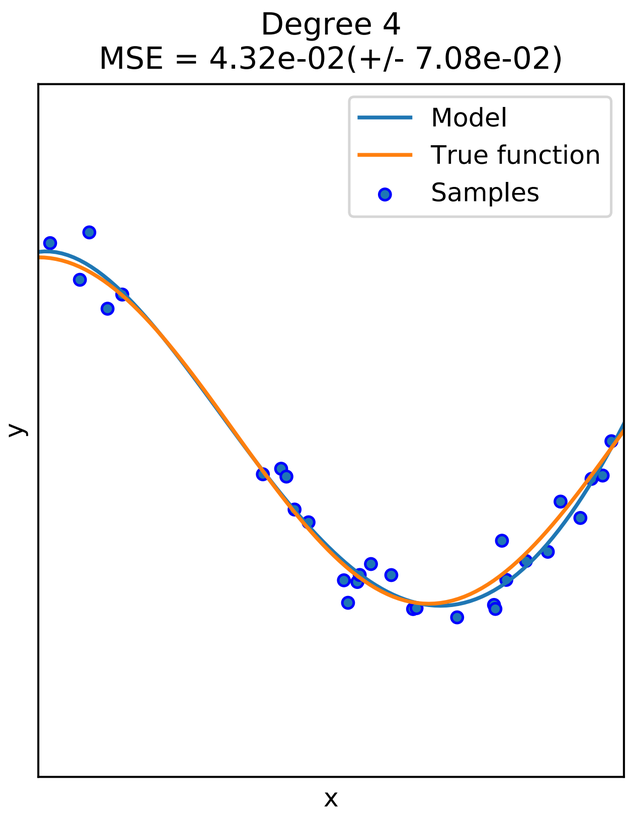
Area under the ROC:
0.5 = no better than random
1 = perfect



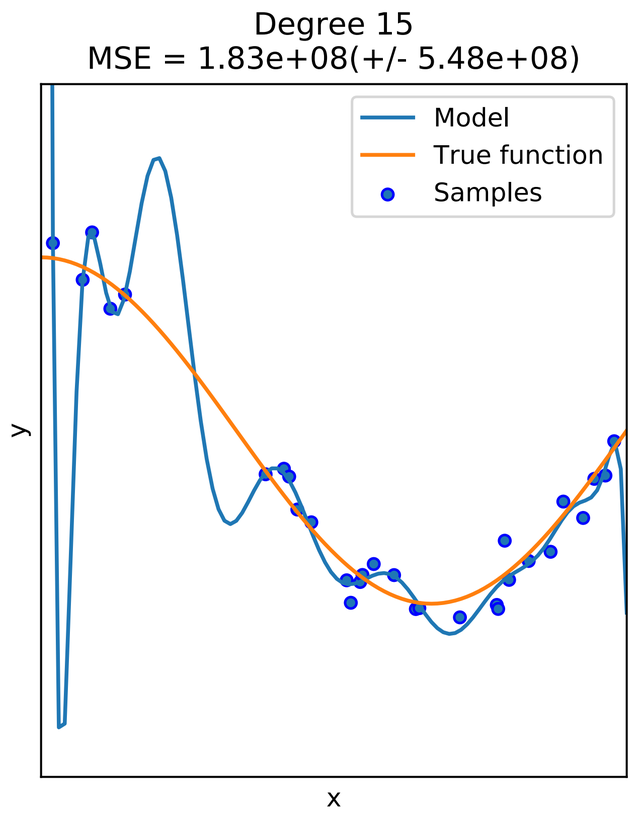
Overfitting and bias versus variance



Underfitted:
model is too simple
(high bias, low variance)



Appropriate

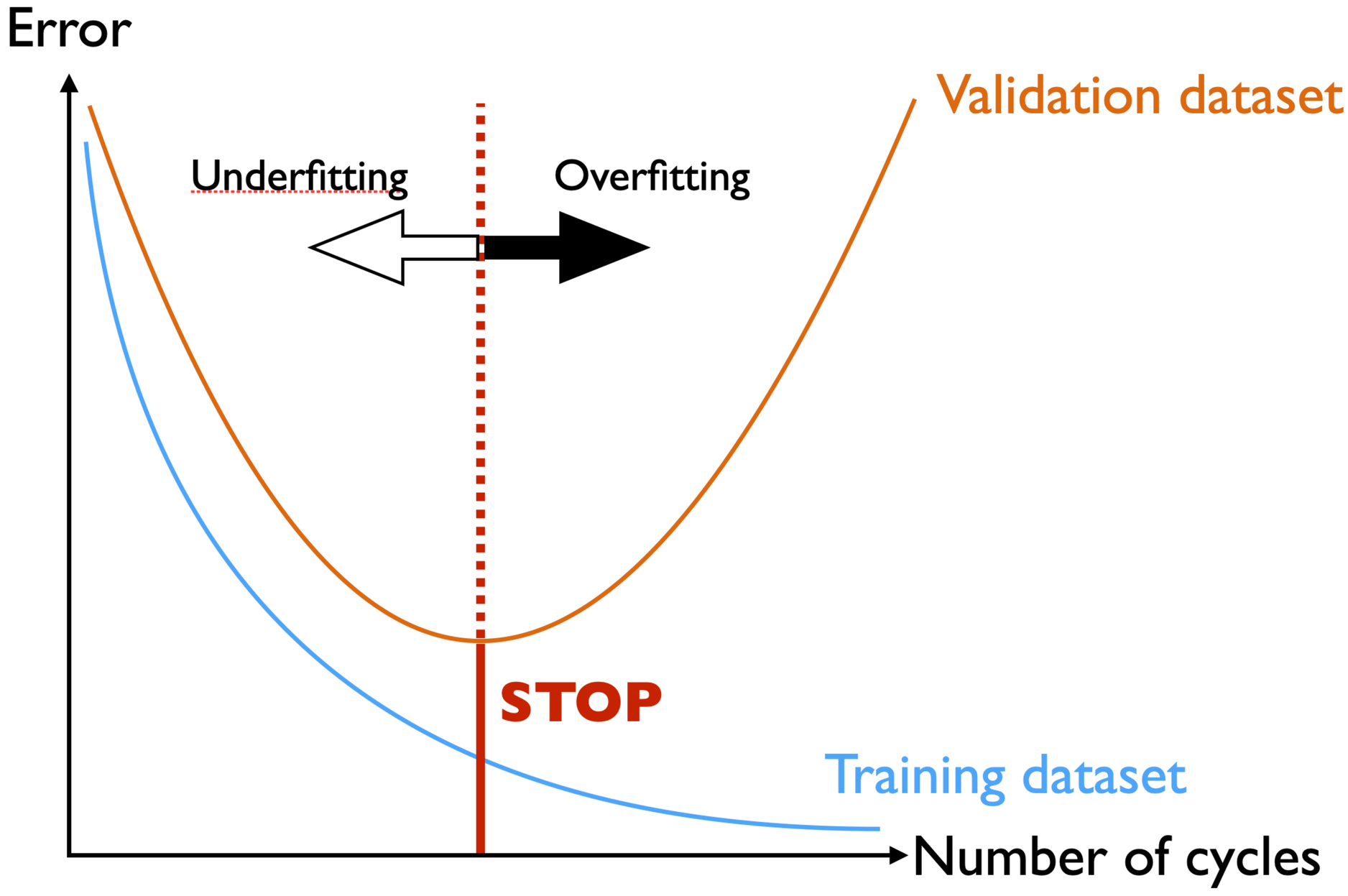


Overfitted:
model is too complex
(low bias, high variance)

$$J(\Theta) = L(\Theta) + \Omega(\Theta)$$

Regularisation is the means of avoiding overtraining: imposes a penalty for complexity

Overfitting and bias versus variance



Boosted Decision Trees

Simple decision trees

Consider some labelled multi-variate data in two categories, signal and background

To build a decision tree:

	var 1	var 2	…	var M	**LABEL**
1	#	#	…	#	S
2	#	#	…	#	S
3	#	#	…	#	B
4	#	#	…	#	B
5	#	#	…	#	B
6	#	#	…	#	S
7	#	#	…	#	B
…	…	…	…	…	…
N	#	#	…	#	B

1. Cut the data using the variable that *best separates* the signal from background

	var 17	**LABEL**
1	#	S
2	#	S
3	#	B
4	#	B
5	#	B
6	#	S
7	#	B
…	…	…
N	#	B

var17 > X

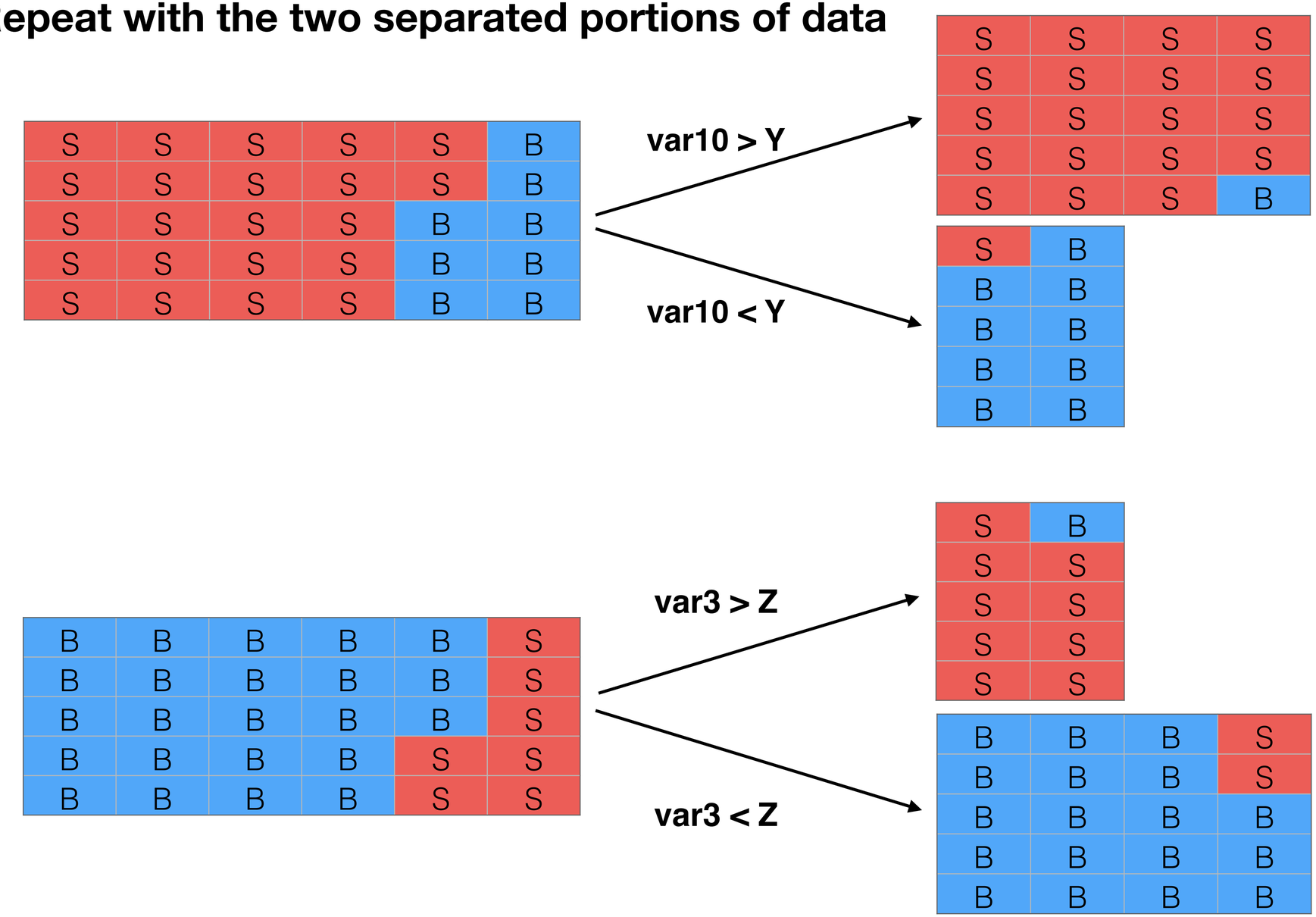
S	S	S	S	S	B
S	S	S	S	S	B
S	S	S	S	B	B
S	S	S	S	B	B
S	S	S	S	B	B

var17 < X

B	B	B	B	B	S
B	B	B	B	B	S
B	B	B	B	B	S
B	B	B	B	S	S
B	B	B	B	S	S

Simple decision trees

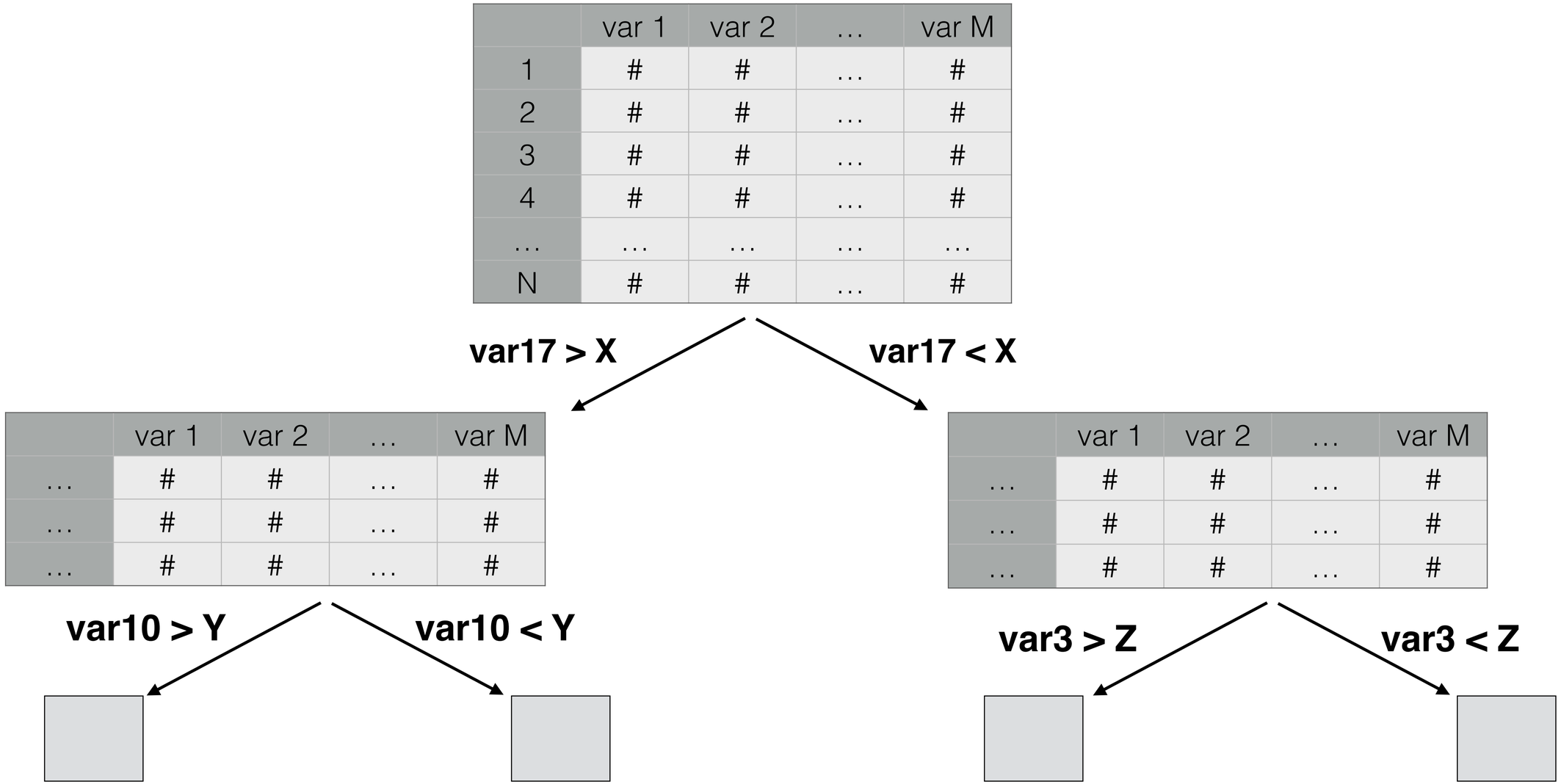
2. Repeat with the two separated portions of data



3. Continue until the separation between S and B reaches the required level

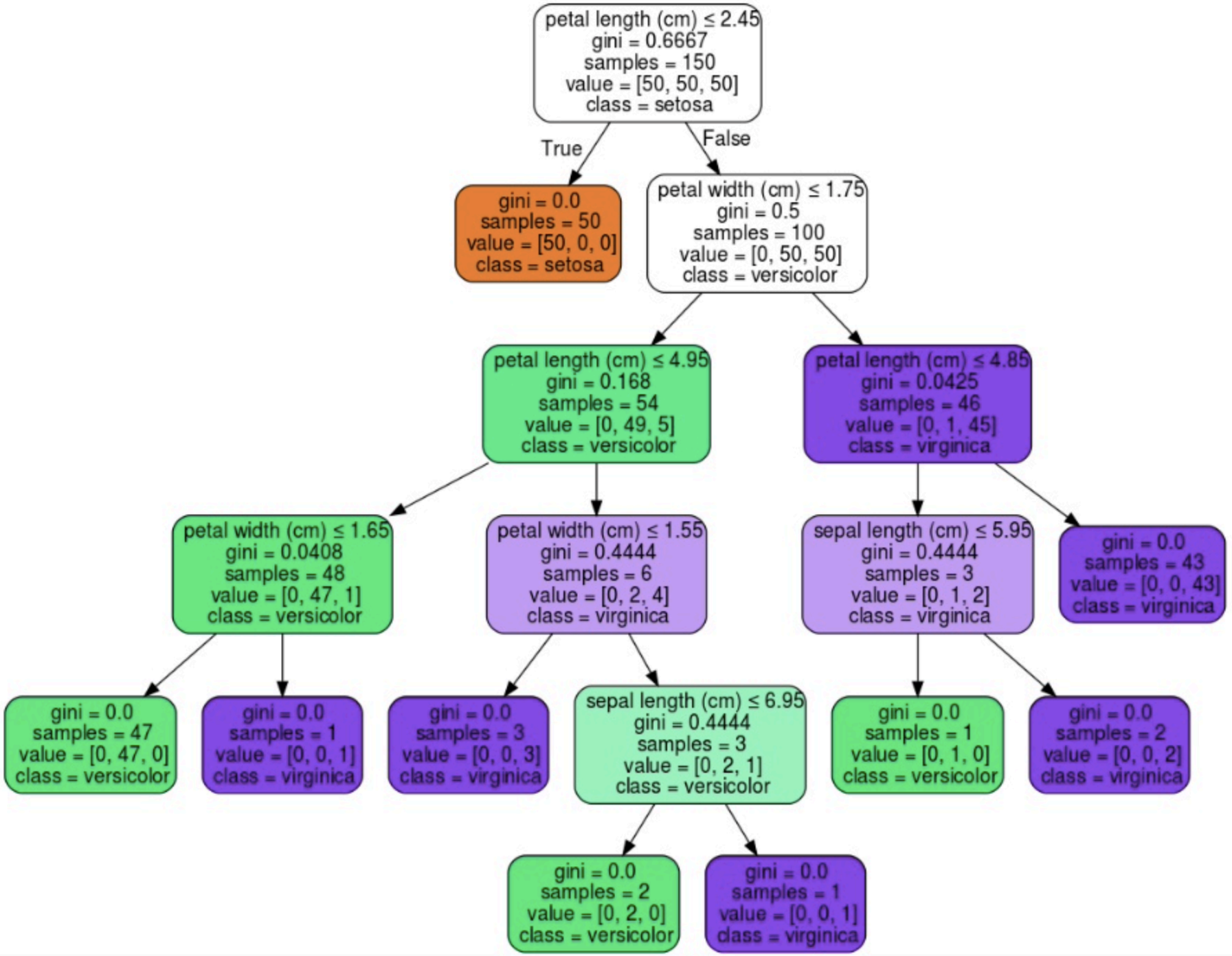
Simple decision trees

4. Use the trained tree (= cuts set, e.g. X, Y and Z) to separate unlabelled data



Simple decision trees

- Advantages
  - Variable scaling/normalisation not required
  - Missing and/or unimportant variables do not impact the algorithm
  - Very straightforward to use (and explain)
  - Can directly visualise how the training has proceeded (“transparent box”)
  - Able to handle numerical and categorical variables, multi-classes and run classification/regression without much change to the algorithm
- Disadvantages
  - Susceptible to learn from random fluctuations (over-training)
  - Unstable and fragile: small variations in the training data can lead to completely different trees being formed
  - Unbalanced classes may lead to skewed trees

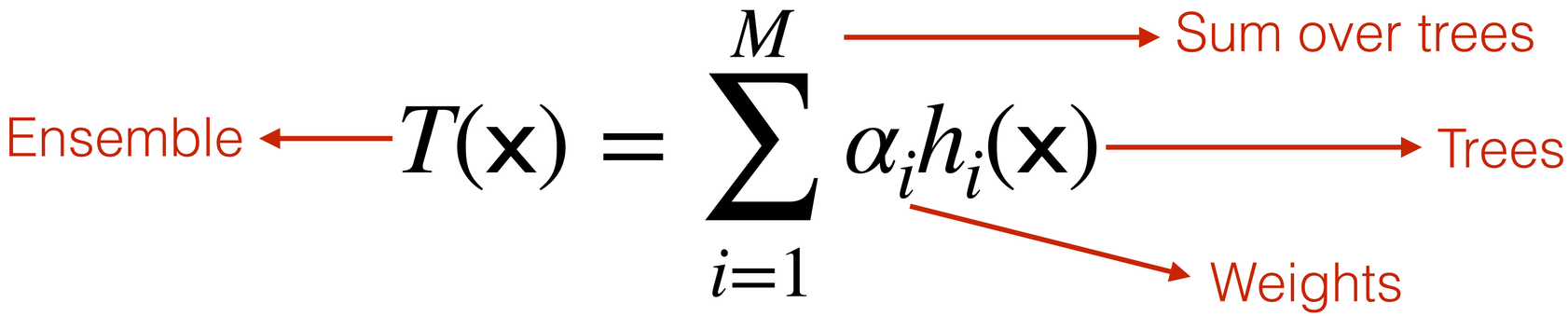


Boosting

- **Boosting** is based on the concept that an *ensemble of weak learners* can be statistically combined to produce much stronger predictions
  - Weak = only slightly better than 50:50 guess
  - Not restricted to decision trees
- Enables us to preserve many of the advantages of decision trees whilst avoiding some of their pitfalls
- Many different ways of boosting - we'll look at two that are heavily used in HEP
  - Adaptive boosting - AdaBoost
  - Gradient boosting - XGBoost

Boosting

- In all cases we seek an ensemble of trees T that minimises the value of some objective function on the training data, **with each new tree trying to correct the mistakes of its predecessors**



Boosting

- In all cases we seek an ensemble of trees T that minimises the value of some objective function on the training data, **with each new tree trying to correct the mistakes of its predecessors**

Ensemble $T(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i h_i(\mathbf{x})$ Trees

Sum over trees

Weights

First tree $T_0(\mathbf{x}) = 0$

Updated ensemble $T_m(\mathbf{x}) = T_{m-1}(\mathbf{x}) + \arg\min_{h_m \in H} \sum_{i=1}^{n} J\left(y_i, T_{m-1}(x_i) + h_m(x_i)\right)$

Current tree

Sum over events

Add the tree h_m from the total space of trees H that minimises the objective J

The means by which the new tree is added distinguishes the different boosting methods

Adaptive boosting (AdaBoost)

Basic idea: when building the ensemble of decision trees, previously misclassified events are **upweighted** for the next tree

$$\mathbf{x} = \{w_1\mathbf{x}_1, \ldots, w_N\mathbf{x}_N\}; \mathbf{y} = \{y_1, \ldots, y_N\} \qquad y \in \{-1, 1\}$$

Weighted data

Target

$$\{w_{1,1} \ldots, w_{N,1}\} = \{\frac{1}{N}, \ldots, \frac{1}{N}\}$$

Initial event weights

AdaBoost

Let us build an ensemble T of *M* decision trees:

For *m* in 1….. *M*:

$h_m(\mathbf{x})$ *m*th tree built as per recipe on previous slides

$$\epsilon_m = \left(\sum_{i=0}^{n} w_i^{(m)} : h_m(x_i) \neq y_i \right) / \sum_{i=0}^{n} w_i^{(m)}$$

Error: sum of weights for misclassified events / total

Append the *m*th tree to the ensemble:

$$T(\mathbf{x}) = T(\mathbf{x}) + \alpha_m h_m(\mathbf{x})$$ where $$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - \epsilon_m}{\epsilon_m} \right)$$

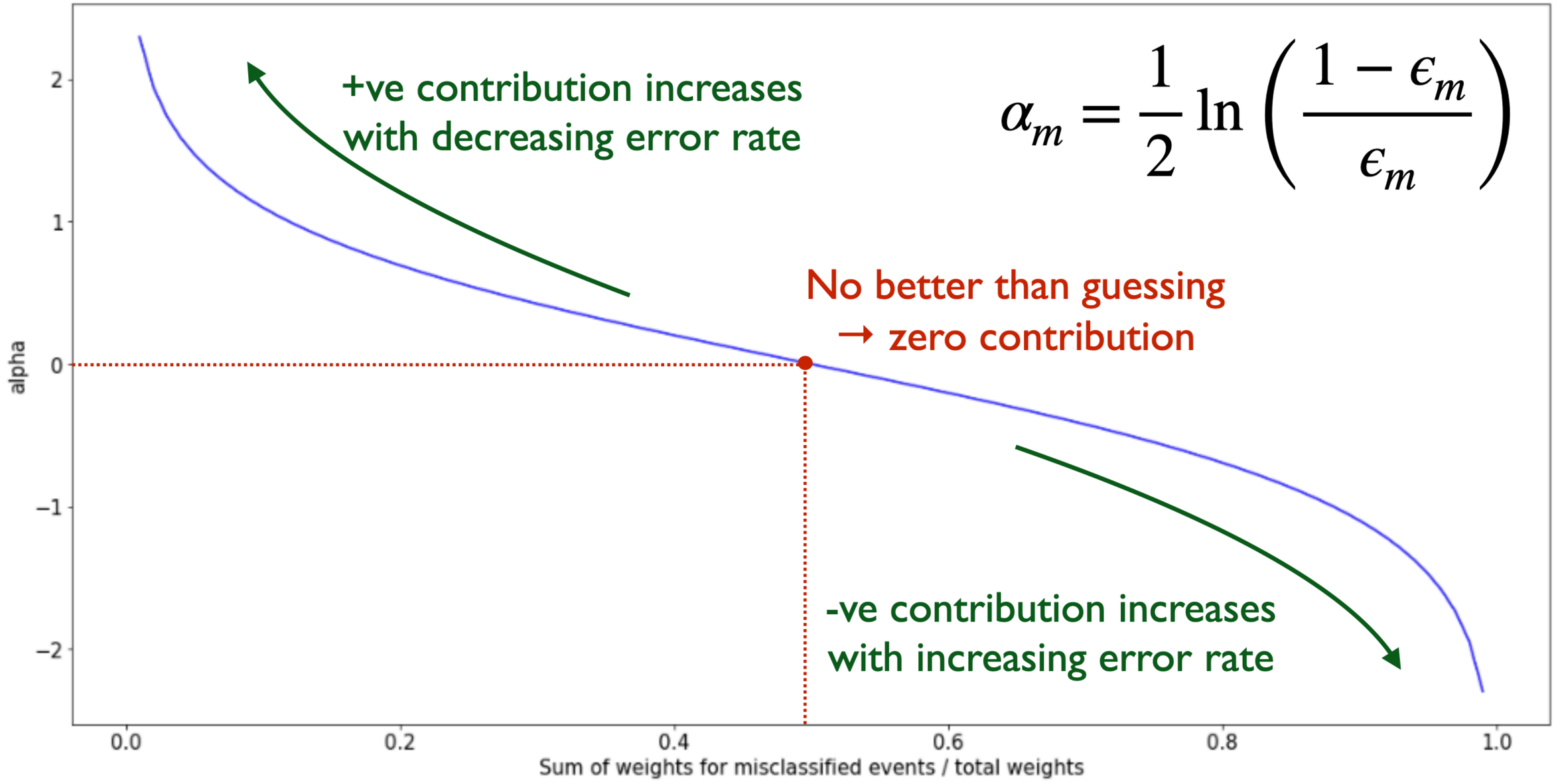
Update the weights:

$$w_{i,m+1} = w_{i,m} e^{-y_i \alpha_m h_m(x_i)}$$

T is used to evaluate new data

Observe that wrongly classified events are **upweighted** and correctly classified events are **downweighted**

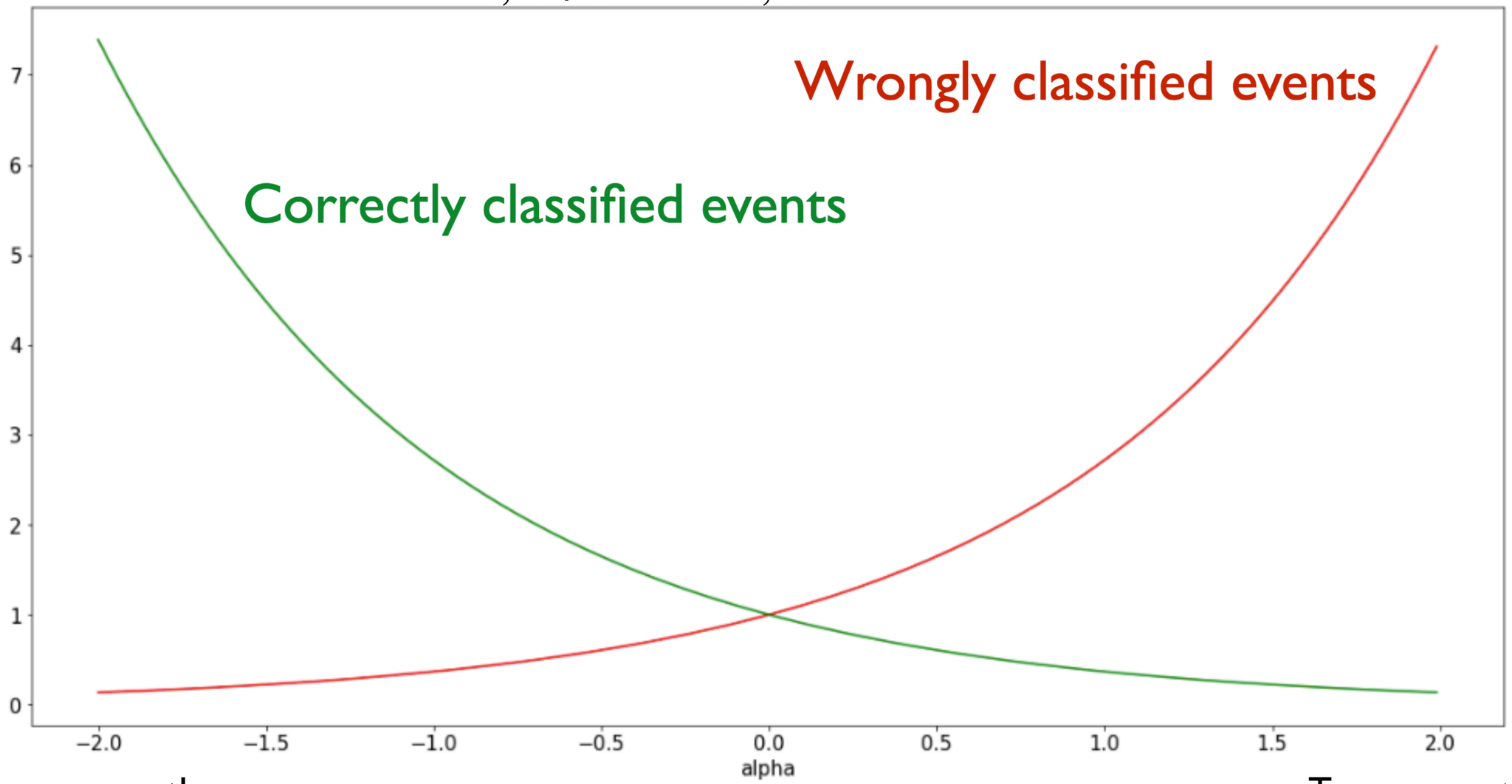
α: contribution per tree



$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - \epsilon_m}{\epsilon_m} \right)$$

Weight update

$$w_{i,m+1} = w_{i,m} e^{-y_i \alpha_m h_m(x_i)}$$



Tree *m* mostly wrong

Tree *m* useless

Tree *m* mostly right

Gradient boosting: basic idea

- Once a BDT has reached perfection:

$$T_{m+1}(\mathbf{x}) = T_m(\mathbf{x}) + h(\mathbf{x}) = \mathbf{y}$$

such that $$h(\mathbf{x}) = \mathbf{y} - T_m(\mathbf{x})$$

- These residuals are the negative gradients w.r.t. T(x) of the objective function → fit each new h to the current residuals

- Equivalent to gradient descent on the loss function

Gradient boosting: algorithm

For m in 1,…, M:

$$r_{i,m} = -\left[\frac{\partial J(y_i, T_{m-1}(\mathbf{x}_i))}{\partial T_{m-1}(\mathbf{x}_i)}\right] \text{ for } i = 0,\ldots,n$$ Gradients

Train h_m using $r_{i,m}$ as the event weights

Find: $$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} J\left(y_i, T_{m-1}(\mathbf{x}_i) + \gamma h_m(\mathbf{x}_i)\right)$$

Update T: $$T_m(\mathbf{x}) = T_{m-1}(\mathbf{x}) + \gamma_m h_m(\mathbf{x})$$

Trained BDT: T_M

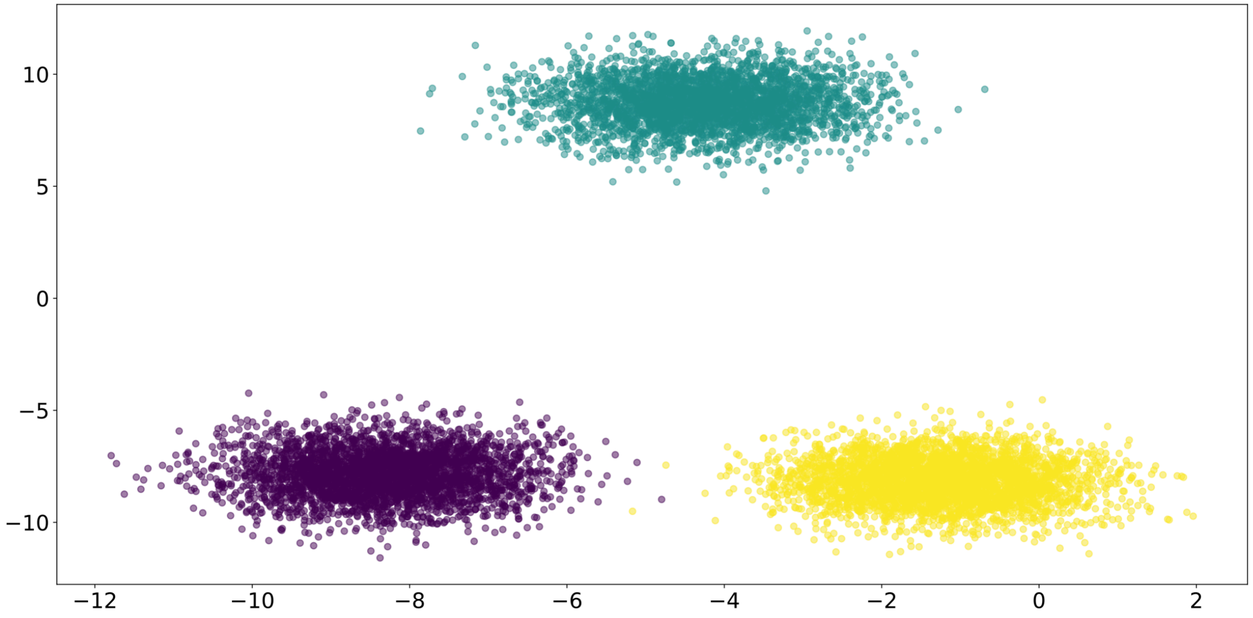
Typical BDT hyperparameters

- Total number of decision trees in the ensemble: usually in the low 100s
- Maximum depth of the individual decision trees: usually around 3-6, can even be 1 (e.g. one partition per cycle)
- Learning rate (if less than 1, shrinks the contribution of each new classifier): means of regularisation - avoiding over-fitting
- Minimum number of events in a leaf (e.g. threshold at which a split is made): often no minimum, e.g. it is 1
- Criterion by which nodes are split
- Objective function
- Evaluation method

XGBoost

- E**x**treme **G**radient **Boost**ing - documentation
- Implementation of gradient boosted decision trees
- Well-known for winning the HiggsML challenge
- Versions exist for Python (incl. Anaconda), R, C++ and TMVA (via the R interface)
- Key features
  - Heavily optimised, leading to fast performance
  - Built-in support for multi-threaded training and distributed training
  - Automated handling of missing variables (very useful, e.g. jet pT when there are no jets…)
  - Feature importance analysis and tree visualisation

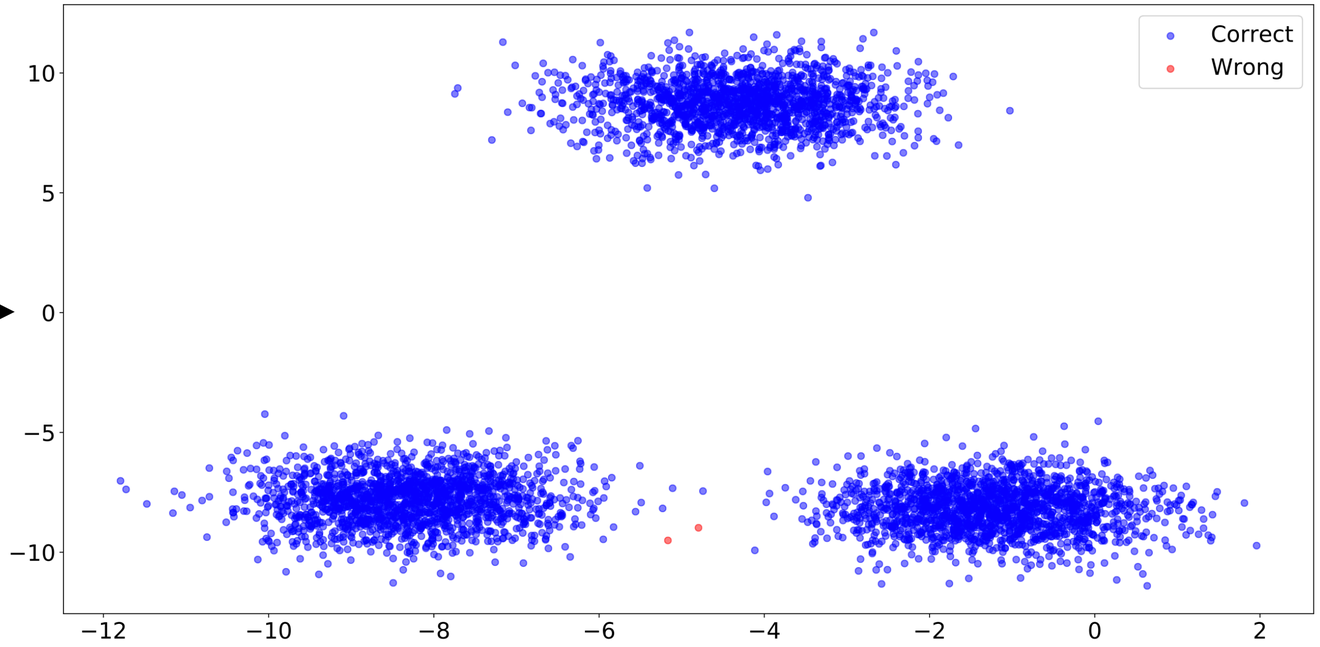
Linear discriminant: linearly separable clusters



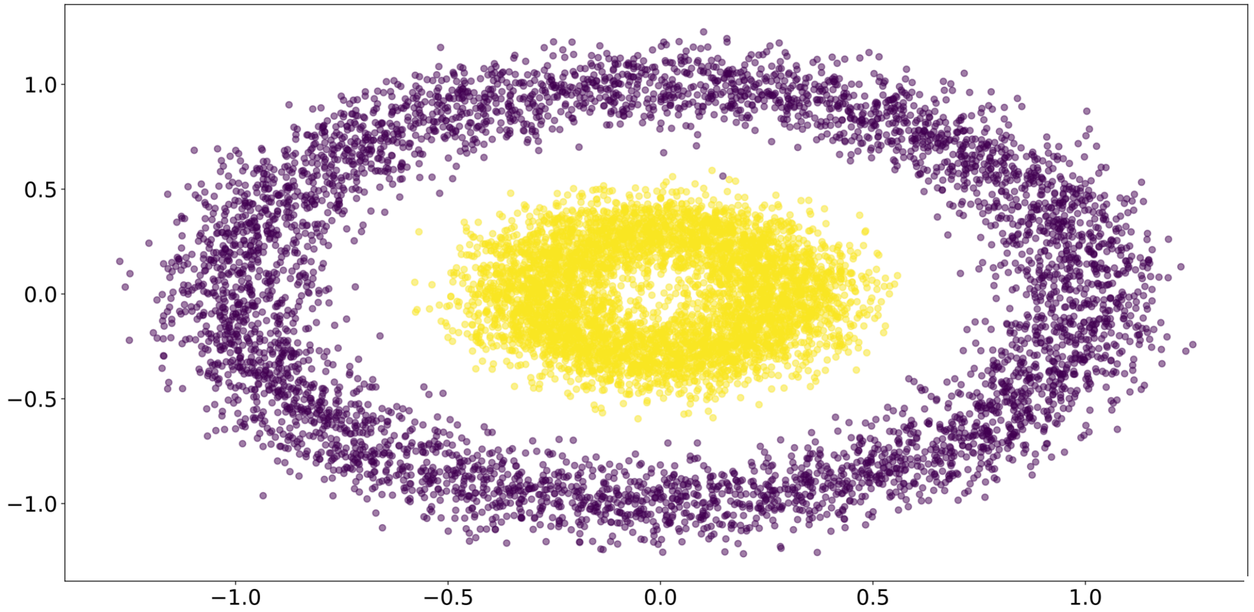
Try it on Thursday!

Linear discriminant (Fisher)

LD adequate to separate classes



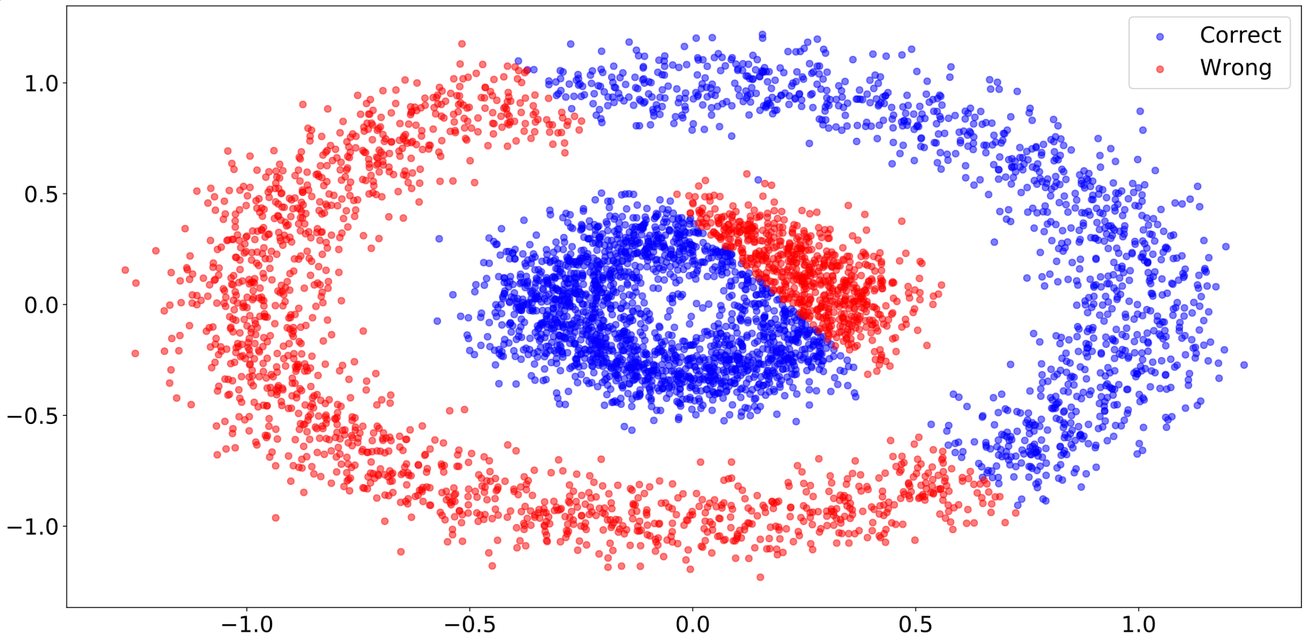
Linear discriminant: concentric rings



Try it on Thursday!

Linear discriminant (Fisher)

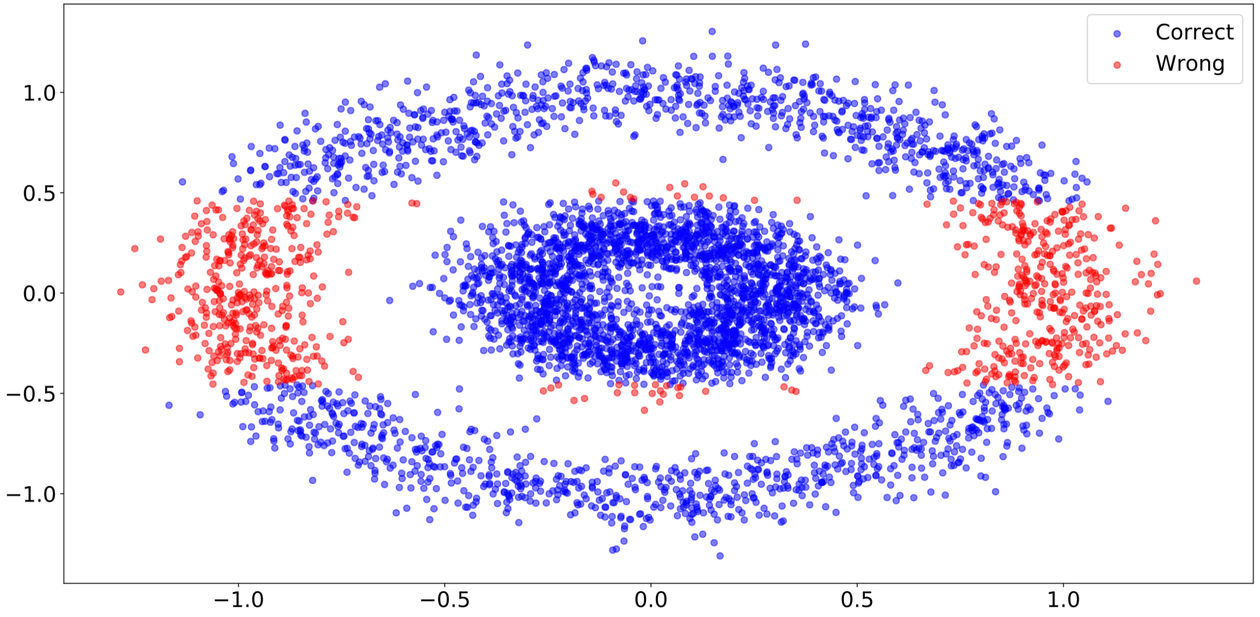
LD not adequate to separate classes



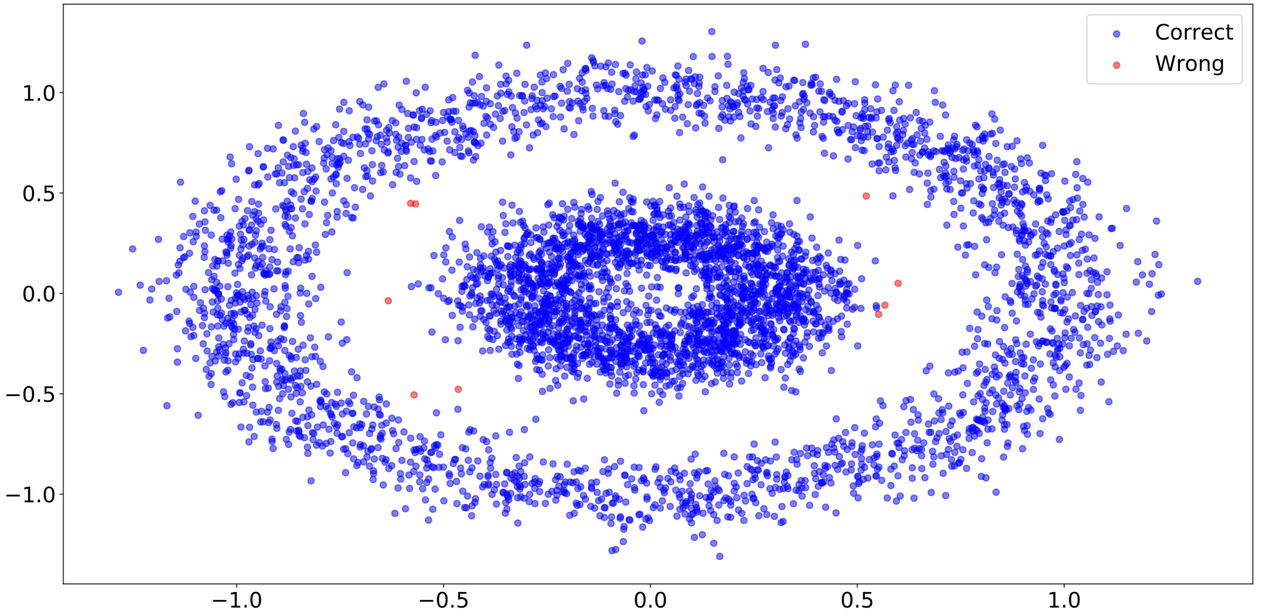
BDT on concentric circles

Try it on Thursday!

1 tree, depth=2



50 trees, depth=2



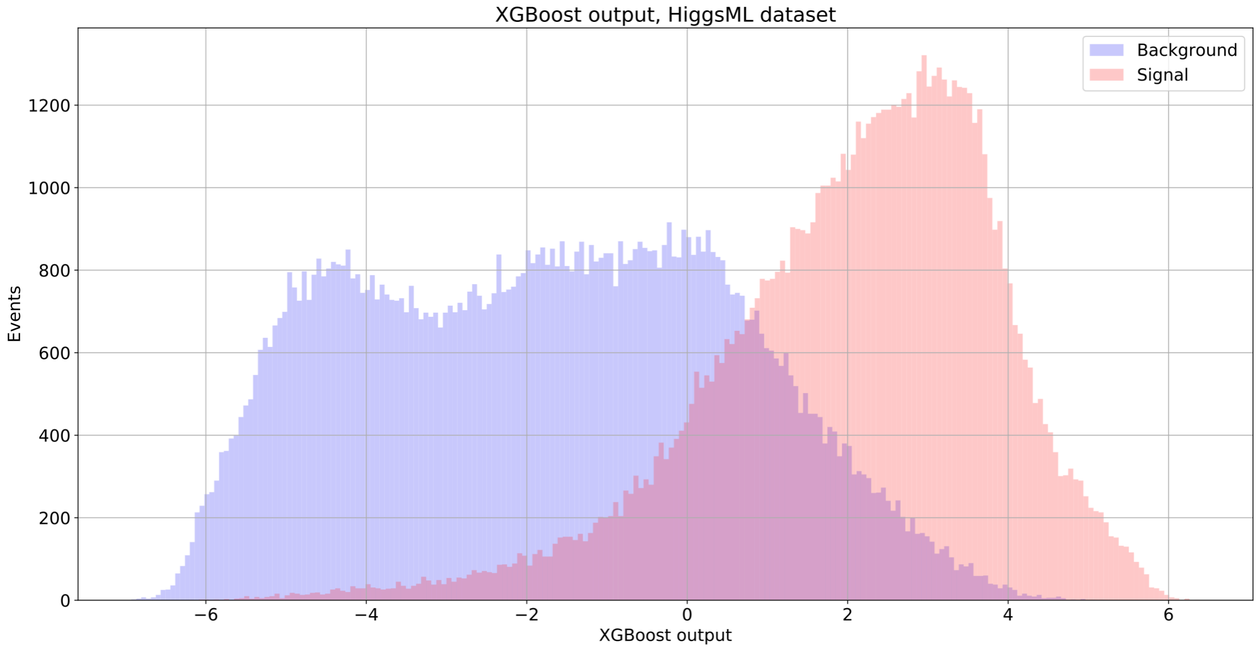
(Note: single tree with depth 6 can achieve the same without boosting)

BDT on example HEP data: HiggsML

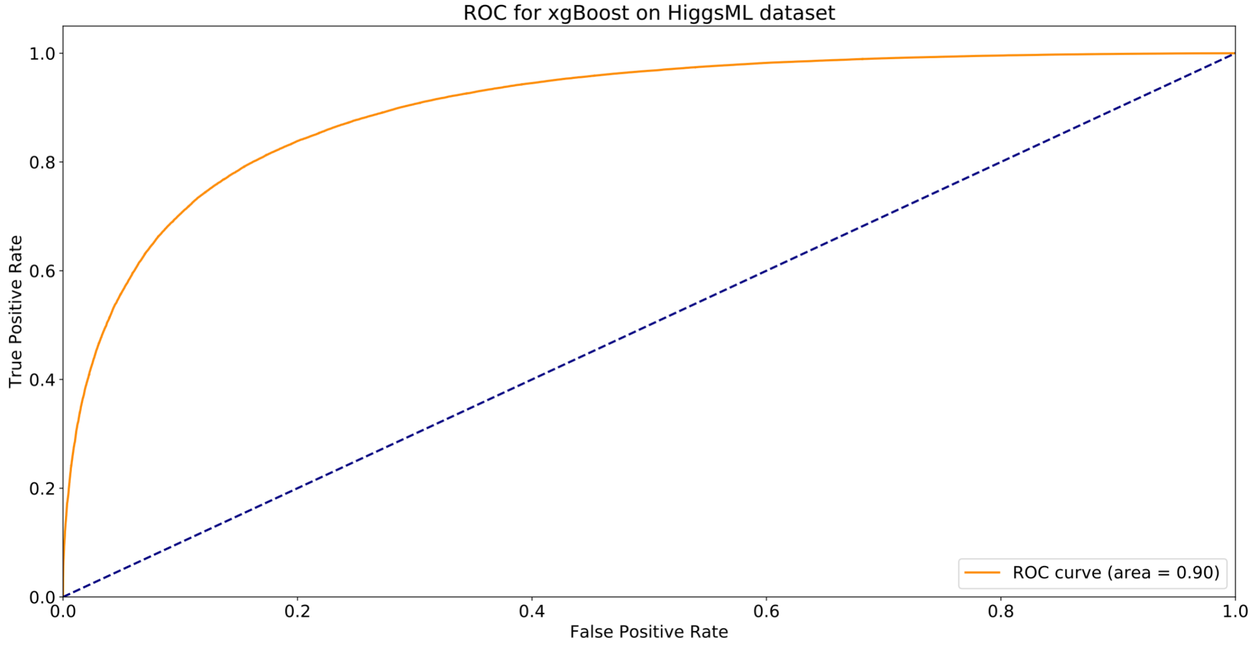
- Data sample released as part of a competition (XGBoost won in fact)
- Signal: H→ττ; background: Z→ττ, ttbar, W→leptons
  - ~30 variables + labels and weights
  - 818238 events available in total
  - http://opendata.cern.ch/record/328
- One of the examples to try on Thursday

DER_mass_MMC: The estimated mass m_H of the Higgs boson candidate, obtained through a probabilistic phase space integration.
DER_mass_transverse_met_lep: The transverse mass between the missing transverse energy and the lepton.
DER_mass_vis: The invariant mass of the hadronic tau and the lepton.
DER_pt_h: The modulus of the vector sum of the transverse momentum of the hadronic tau, the lepton and the missing transverse energy vector.
DER_deltaeta_jet_jet: The absolute value of the pseudorapidity separation between the two jets (undefined if PRI_jet_num $\leq$ 1).
DER_mass_jet_jet: The invariant mass of the two jets (undefined if PRI_jet_num $\leq$ 1).
DER_prodeta_jet_jet: The product of the pseudorapidities of the two jets (undefined if PRI_jet_num $\leq$ 1).
DER_deltar_tau_lep: The R separation between the hadronic tau and the lepton.
DER_pt_tot: The modulus of the vector sum of the missing transverse momenta and the transverse momenta of the hadronic tau, the lepton, the leading jet (if PRI_jet_num $\geq$) and the subleading jet (if PRI jet num = 2) (but not of any additional jets).
DER_sum_pt: The sum of the moduli of the transverse momenta of the hadronic tau, the lepton, the leading jet (if PRI jet num $\geq$ 1) and the subleading jet (if PRI jet num = 2) and the other jets (if PRI jet num = 3).
DER_pt_ratio_lep_tau: The ratio of the transverse momenta of the lepton and the hadronic tau.
DER_met_phi_centrality: The centrality of the azimuthal angle of the missing transverse energy vector w.r.t. the hadronic tau and the lepton.
DER_lep_eta_centrality: The centrality of the pseudorapidity of the lepton w.r.t. the two jets (undefined if PRI_jet_num $\leq$ 1).
PRI_tau_pt: The transverse momentum $\sqrt{p_x^2 + p_y^2}$ of the hadronic tau.
PRI_tau_eta: The pseudorapidity η of the hadronic tau.
PRI_tau_phi: The azimuth angle ϕ of the hadronic tau.
PRI_lep_pt: The transverse momentum $\sqrt{p_x^2 + p_y^2}$ of the lepton (electron or muon).
PRI_lep_eta: The pseudorapidity η of the lepton.
PRI_lep_phi: The azimuth angle ϕ of the lepton.
PRI_met: The missing transverse energy $\overrightarrow{E}_T^{miss}$
PRI_met_phi: The azimuth angle ϕ of the mssing transverse energy
PRI_met_sumet: The total transverse energy in the detector.
PRI_jet_num: The number of jets (integer with value of 0, 1, 2 or 3; possible larger values have been capped at 3).
PRI_jet_leading_pt: The transverse momentum $\sqrt{p_x^2 + p_y^2}$ of the leading jet, that is the jet with largest transverse momentum (undefined if PRI_jet_num = 0).
PRI_jet_leading_eta: The pseudorapidity η of the leading jet (undefined if PRI jet num = 0).
PRI_jet_leading_phi: The azimuth angle ϕ of the leading jet (undefined if PRI jet num = 0).
PRI_jet_subleading_pt: The transverse momentum $\sqrt{p_x^2 + p_y^2}$ of the leading jet, that is, the jet with second largest transverse momentum (undefined if PRI_jet_num $\leq$ 1).
PRI_jet_subleading_eta: The pseudorapidity η of the subleading jet (undefined if PRI_jet_num $\leq$ 1).
PRI_jet_subleading_phi: The azimuth angle ϕ of the subleading jet (undefined if PRI_jet_num $\leq$ 1).
PRI_jet_all_pt: The scalar sum of the transverse momentum of all the jets of the events.
Weight: The event weight w_i
Label: The event label (string) $y_i \in \{s, b\}$ (s for signal, b for background).

HiggsML output and ROC



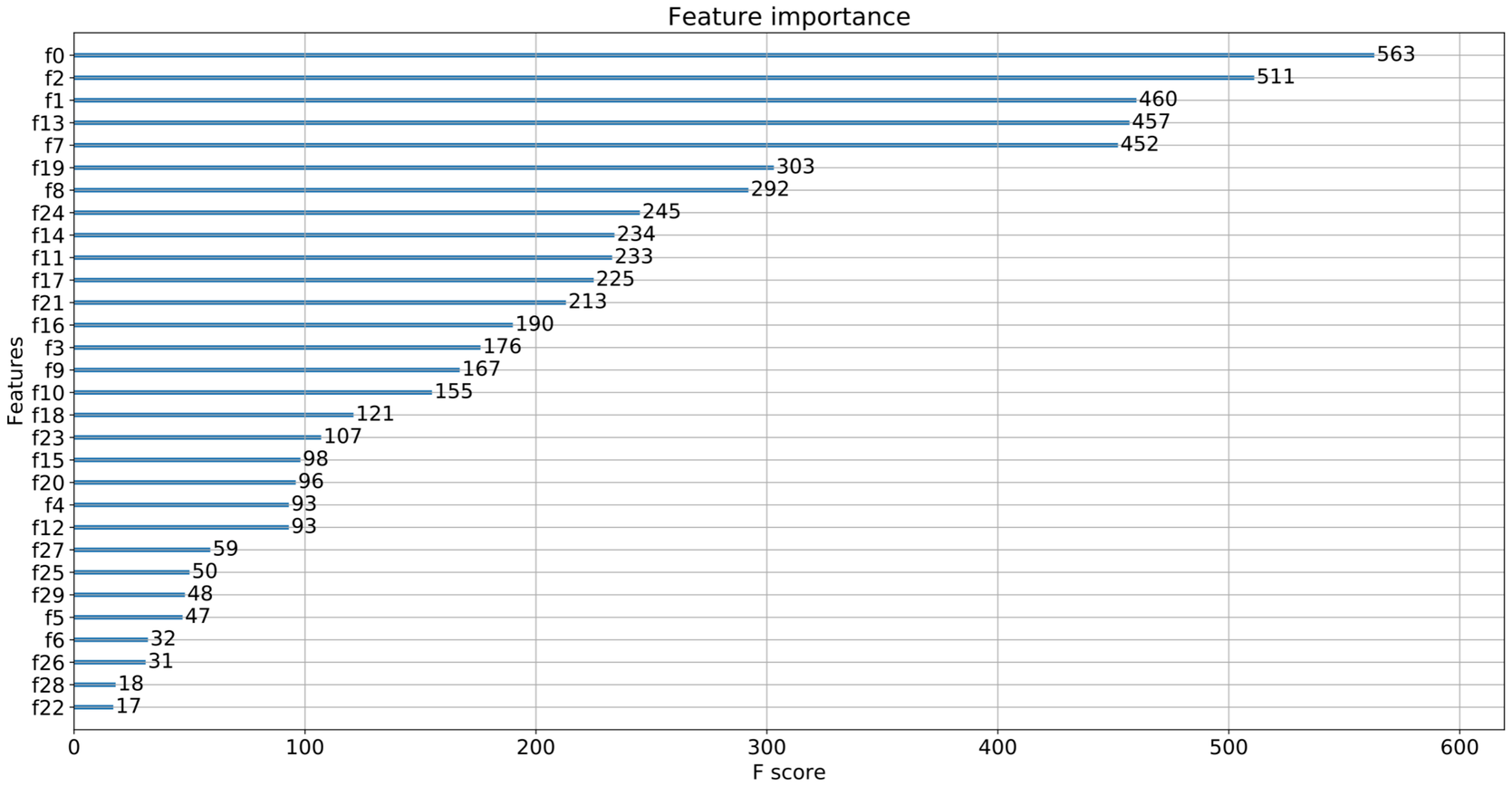
Try it on Thursday!



HiggsML variable importance

- The more often a variable is used to split the data, the more important (discriminating) it is
- Enables creation of plots like this:

Try it on Thursday!



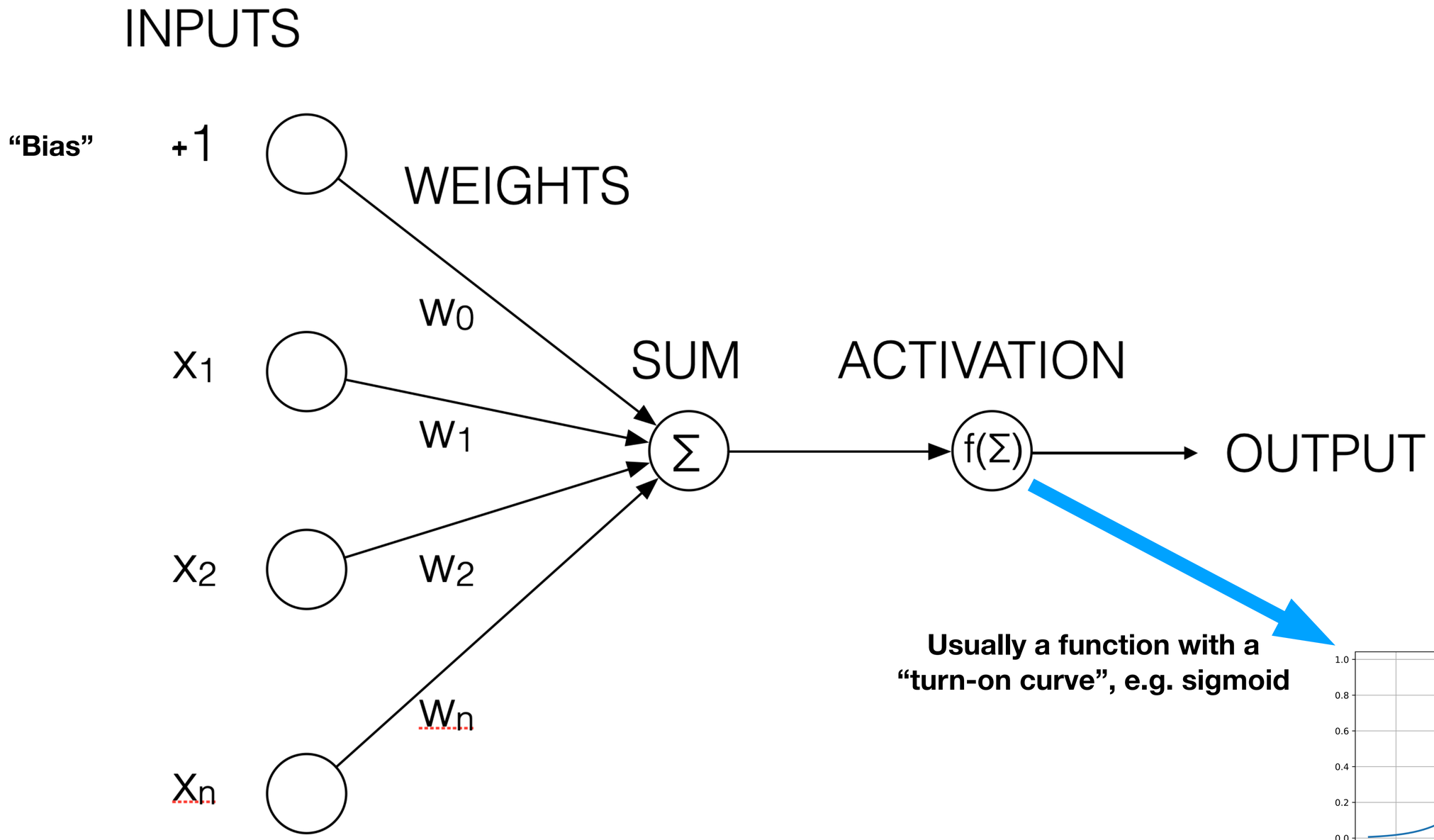
Note that BDTs aren't perturbed by under-utilised variables - they just don't use them to cut the data

Artificial neural networks

Artificial neural networks

- First conceived of in the 1940s-1950s, inspired by research into biological neural processes
- Key developments:
  - Perceptron (1958)
  - Back-propagation (1975)
- Initial development was slow due to limited computing power and insufficient training data; many machine learning researchers lost interest and focused on other techniques such as support vector machines and linear methods
- In the past 15 years several factors combined to completely change the situation
  - Growth of the internet and smart phones = massively more training data and network capacity
  - Huge increase in computing power and memory
  - New ideas from the academic community, spread via open source software
- This led to the astonishing capabilities of **deep learning** that we see today

The perceptron model



The perceptron model: how it learns

$\mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}; \mathbf{y} = \{y_1, \ldots, y_M\}$ Training data

M events with N variables

At each step t: evaluate for each event *j:*

$$h_j(t) = f\left[w_0(t) + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \ldots + w_N(t)x_{j,N}\right]$$

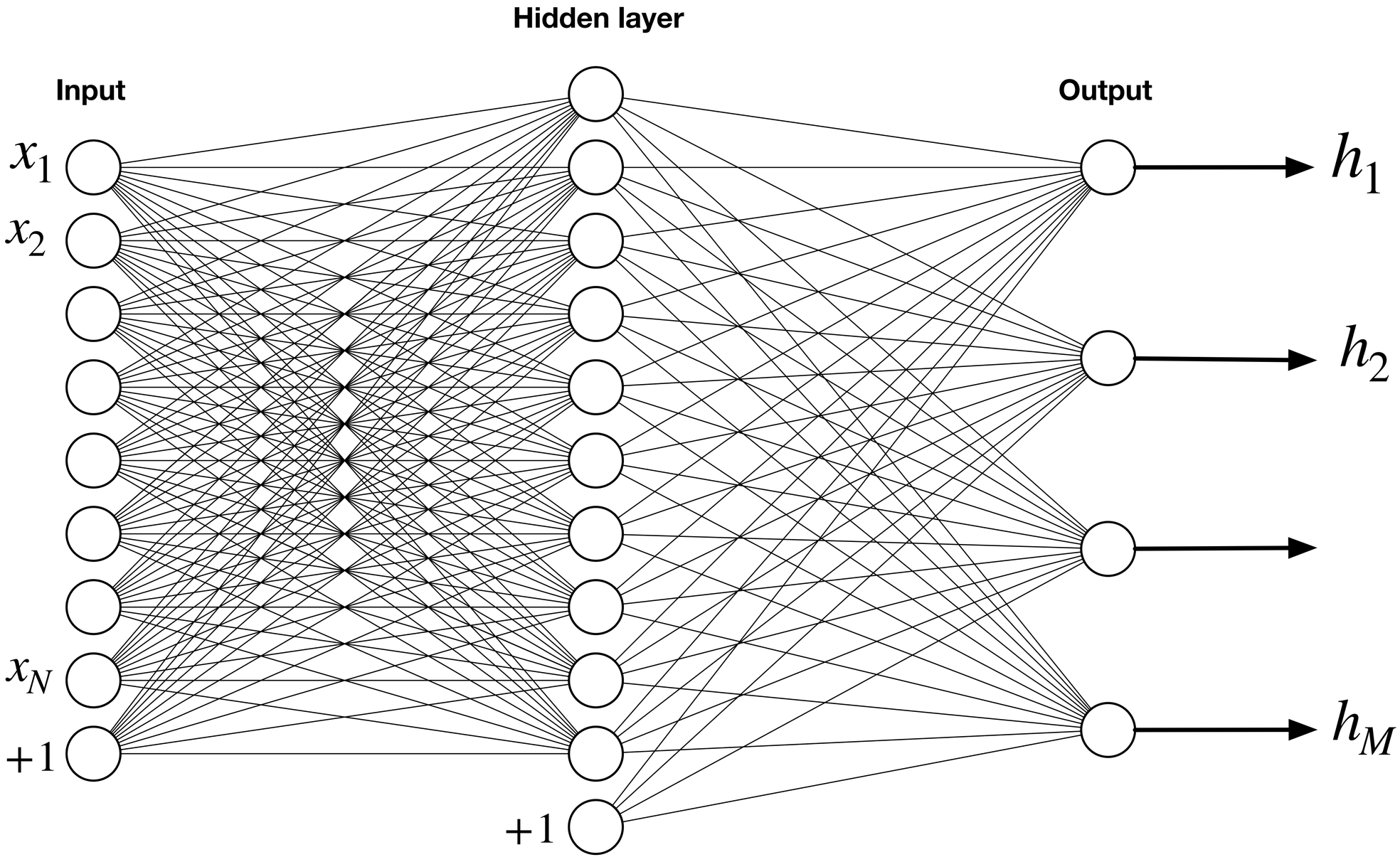
For each variable *i* from 1 to N, update the weights:

$$w_i(t+1) = w_i(t) + r \cdot \left(y_j - h_j(t)\right) x_{j,i}$$

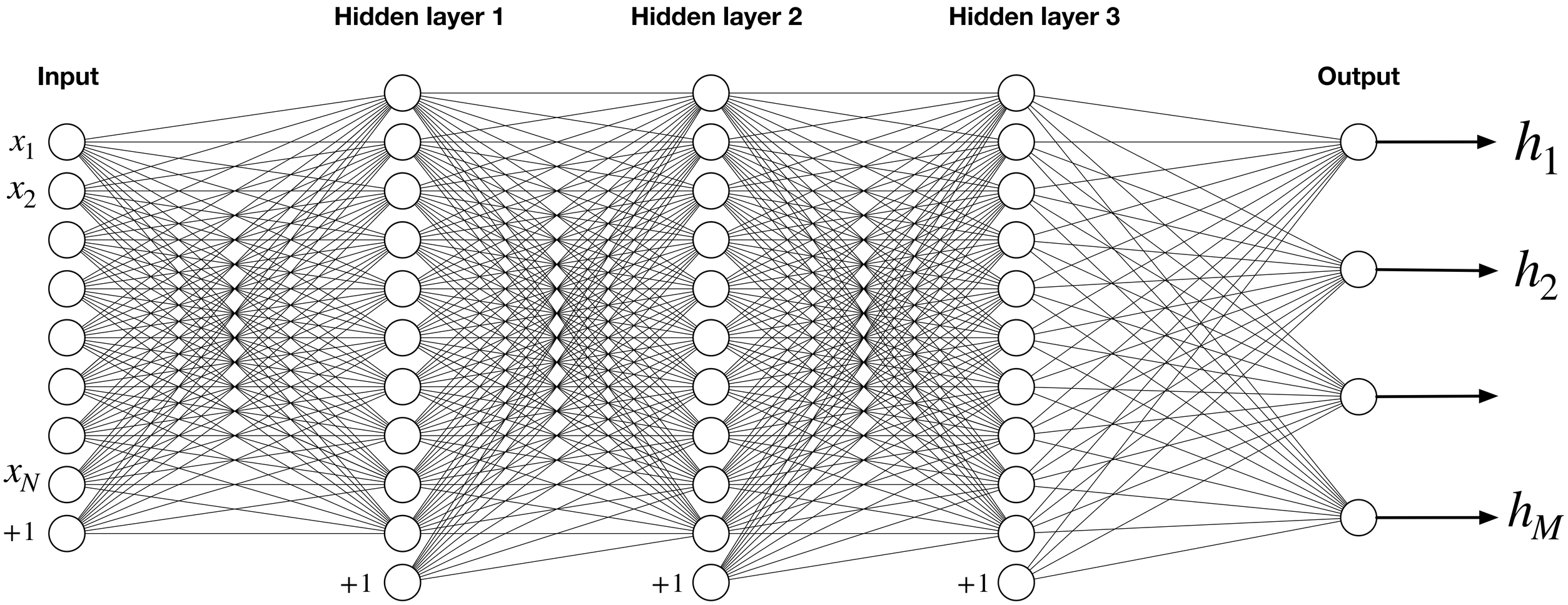
Learning rate

Continue until $\frac{1}{M}\sum_{j=0}^{M} |y_j - h_j(t)|$ reaches some appropriate level, or the number of steps t exceeds some value

Shallow multi-layer perceptron



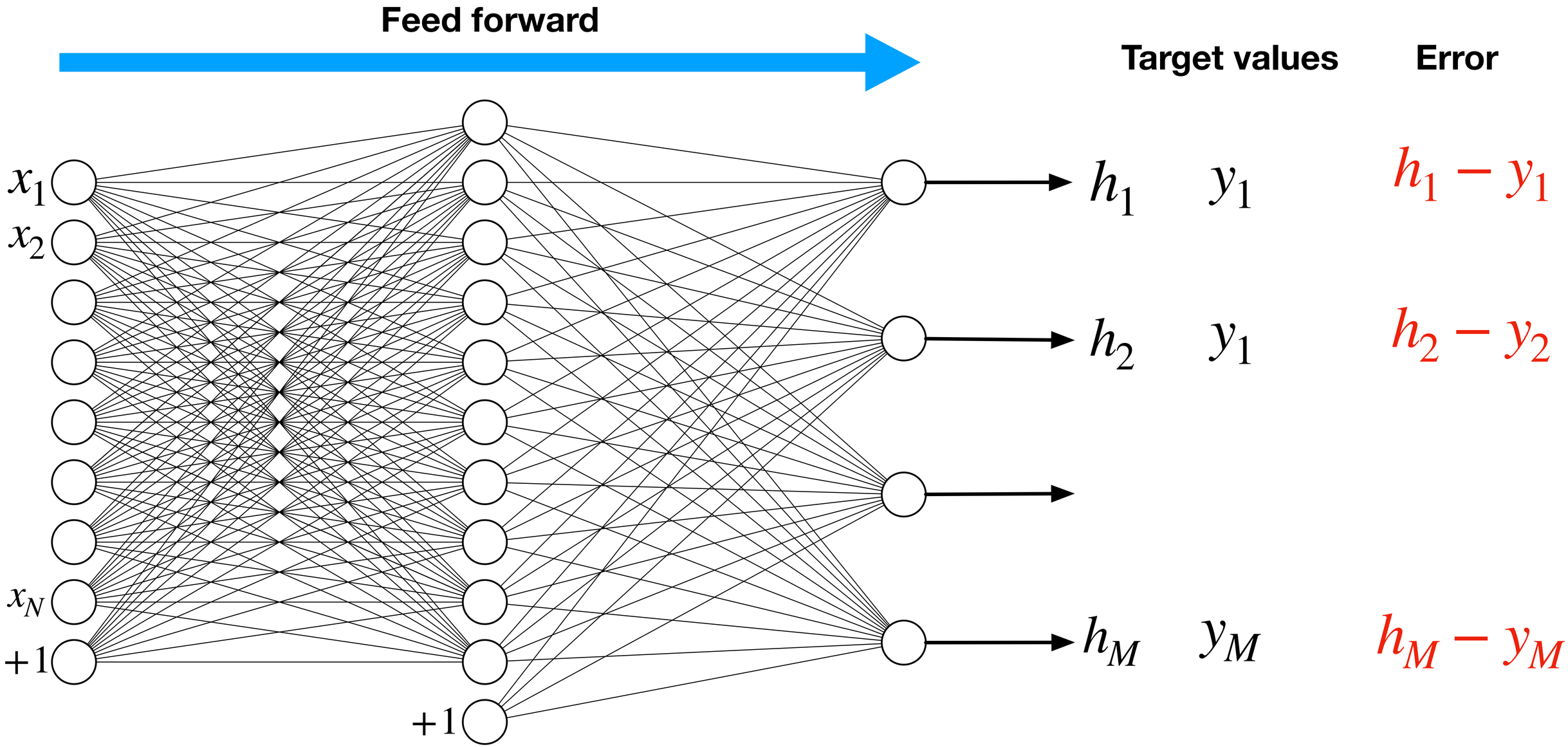
Fully connected deep multi-layer perceptron



How a neural network is trained

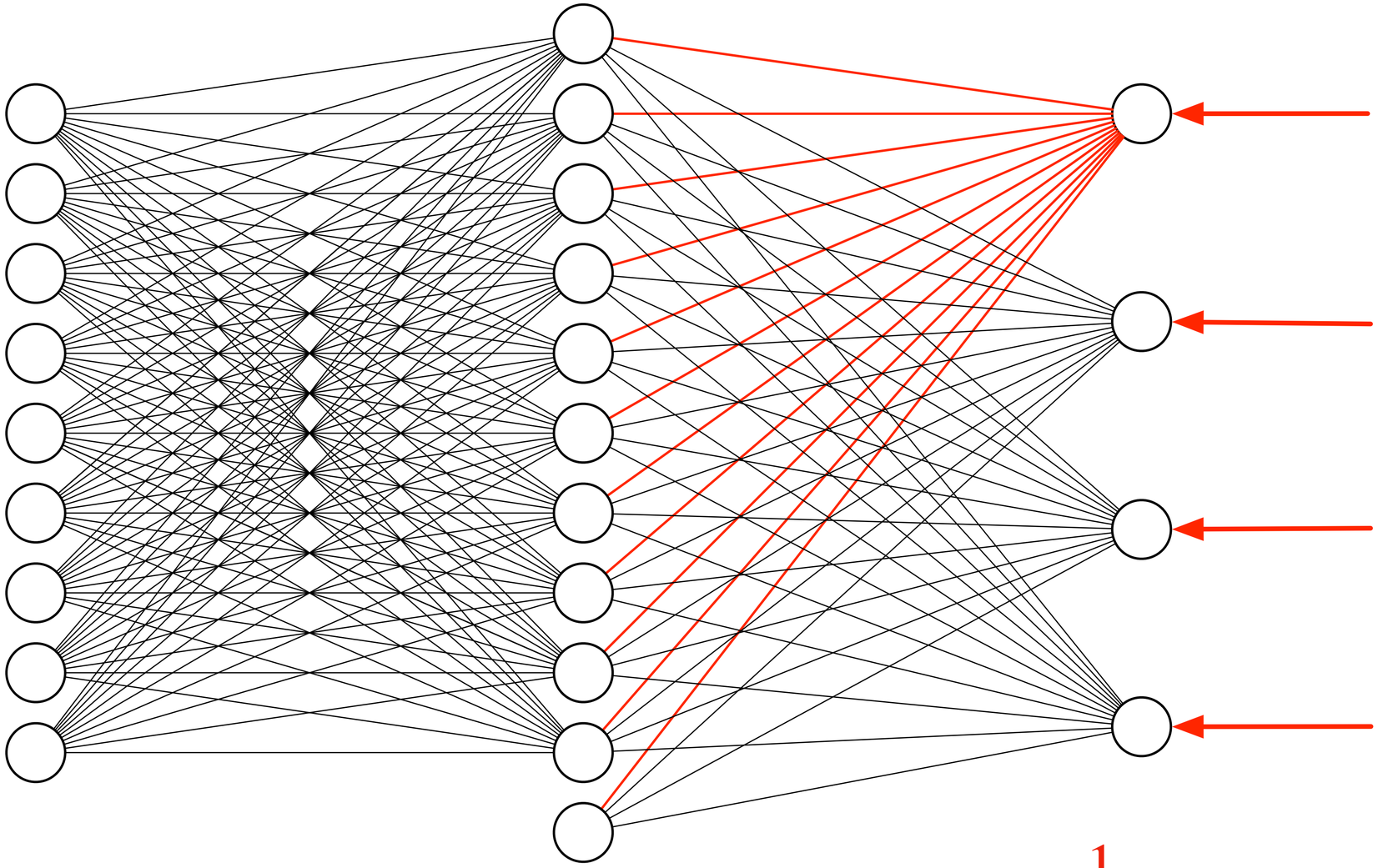
- The perceptron learning mechanism doesn't extend to multiple layers
- To produce a set of gradients which a minimiser can work on to adjust the weights, the *back propagation* method must be used
- How it works:
  - Pass each event through the network (feed forward) arriving at a result
  - Compare with the target using some loss function, arriving at some error
  - Propagate this error backwards through the network, node by node and layer by layer, until each node has its own contribution to the overall error
  - Use these errors to calculate the partial derivatives of the loss function w.r.t. the weight at each neuron (gradients), by recursively applying the chain rule for derivatives
  - Use gradient descent to minimise the loss, which it does by updating the weights
  - New examples are then passed through the trained network (fixed weights)

How a neural network is trained



Weights are randomised at the start

How a neural network is trained



$h_1 - y_1$

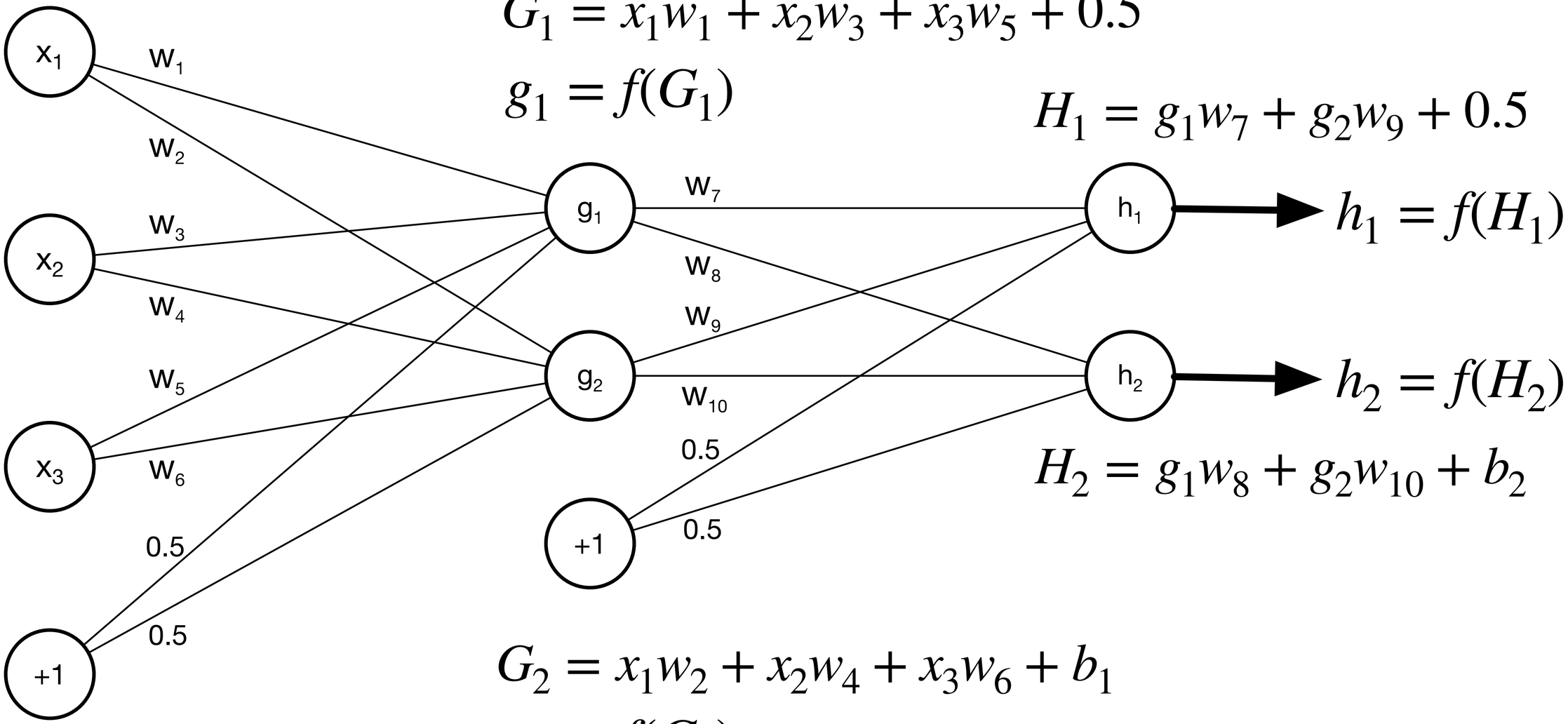
$h_2 - y_2$

$h_M - y_M$

$$E = \frac{1}{2}\left((h_1 - y_1)^2 + (h_2 - y_2)^2 \dots\right)$$

Back propagation of errors

Feed forward



$$G_1 = x_1 w_1 + x_2 w_3 + x_3 w_5 + 0.5$$

$$g_1 = f(G_1)$$

$$H_1 = g_1 w_7 + g_2 w_9 + 0.5$$

$$h_1 = f(H_1)$$

$$h_2 = f(H_2)$$

$$H_2 = g_1 w_8 + g_2 w_{10} + b_2$$

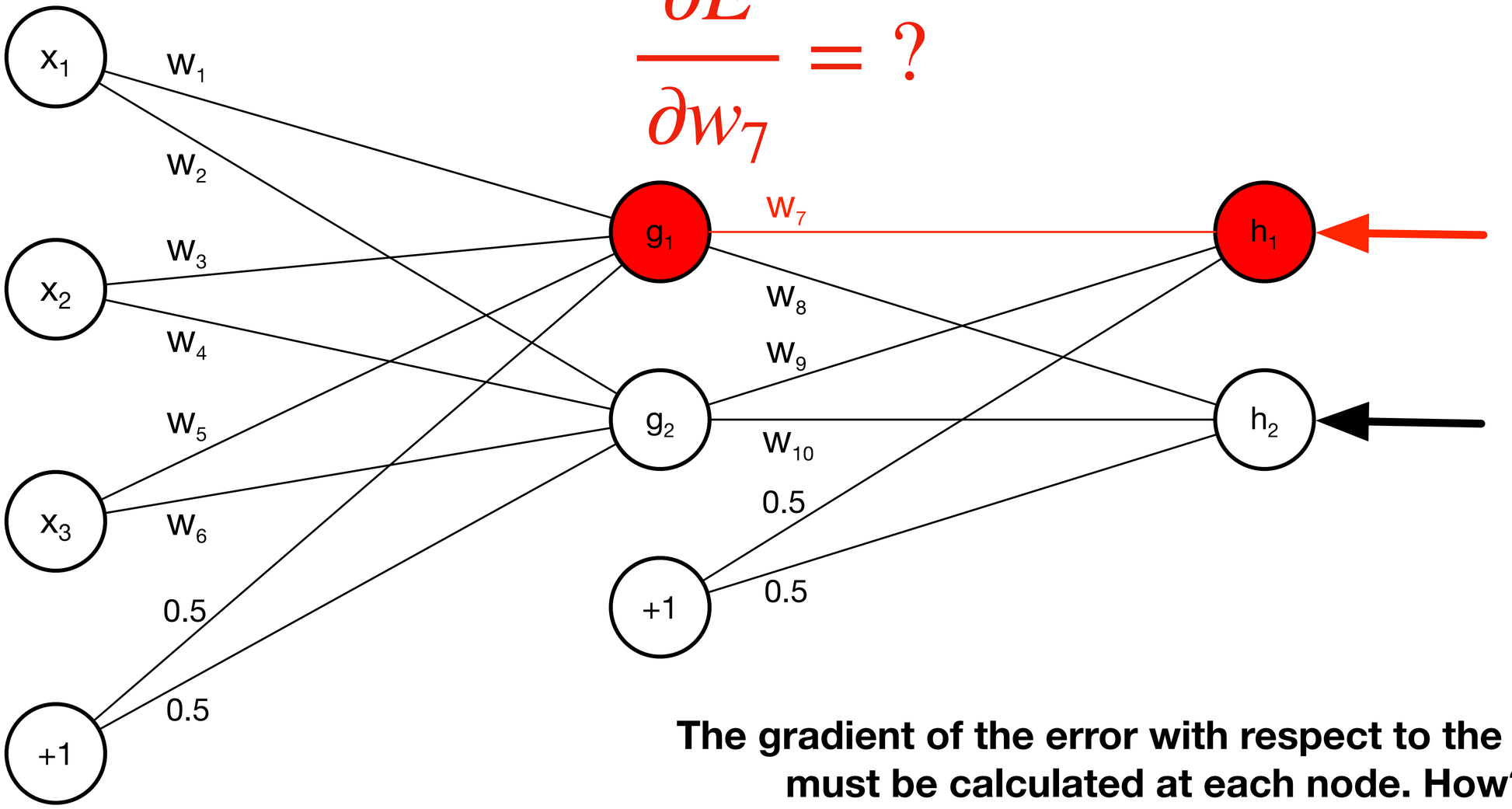
$$G_2 = x_1 w_2 + x_2 w_4 + x_3 w_6 + b_1$$

$$g_2 = f(G_2)$$

Weights are randomised at the start

Back propagation

$$\frac{\partial E}{\partial w_7} = ?$$



The gradient of the error with respect to the weights must be calculated at each node. How??

→ Recursive application of the chain rule for derivatives

Back propagation via the chain rule

$$\frac{\partial E}{\partial w_7} = \frac{\partial E}{\partial h_1} \cdot \frac{\partial h_1}{\partial H_1} \cdot \frac{\partial H_1}{\partial w_7} = (h_1 - y_1) \cdot h_1(1 - h_1) \cdot g_1$$

$$\frac{\partial E}{\partial h_1} = \frac{1}{2}\frac{\partial}{\partial h_1}\left((h_1 - y_1)^2 + (h_2 - y_2)^2\right) = h_1 - y_1$$

$$\frac{\partial h_1}{\partial H_1} = \frac{\partial f(H_1)}{\partial H_1} = f(H_1)\left(1 - f(H_1)\right) = h_1(1 - h_1)$$

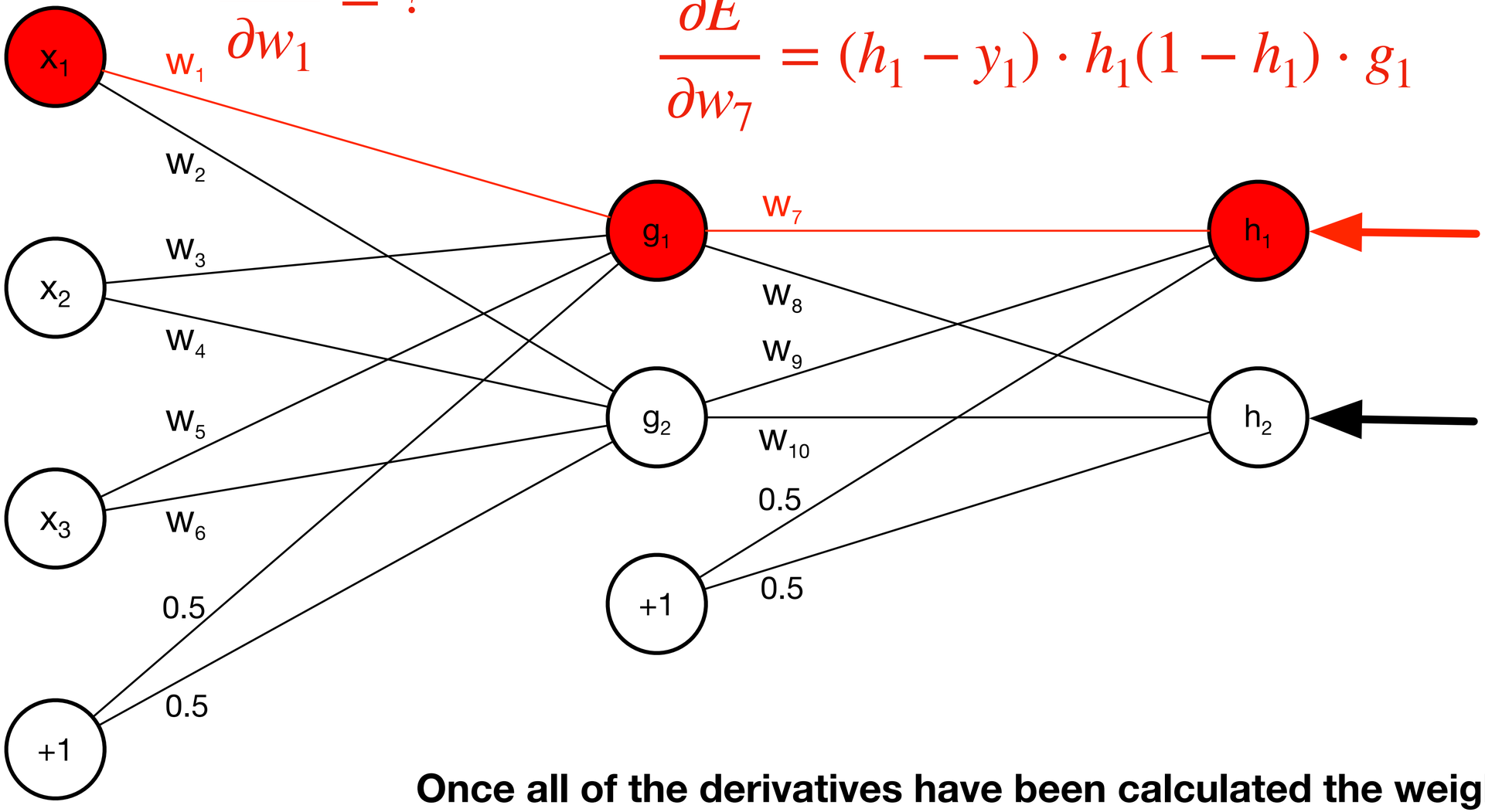
Assuming the activation function f(x) is sigmoid… Others will have different expressions

$$\frac{\partial H_1}{\partial w_7} = \frac{\partial}{\partial w_7}(g_1 w_7 + g_2 w_9 + 0.5) = g_1$$

Back propagation via the chain rule

$$\frac{\partial E}{\partial w_1} = ?$$

$$\frac{\partial E}{\partial w_7} = (h_1 - y_1) \cdot h_1(1 - h_1) \cdot g_1$$



Once all of the derivatives have been calculated the weights can be adjusted via gradient descent to minimise the overall loss function

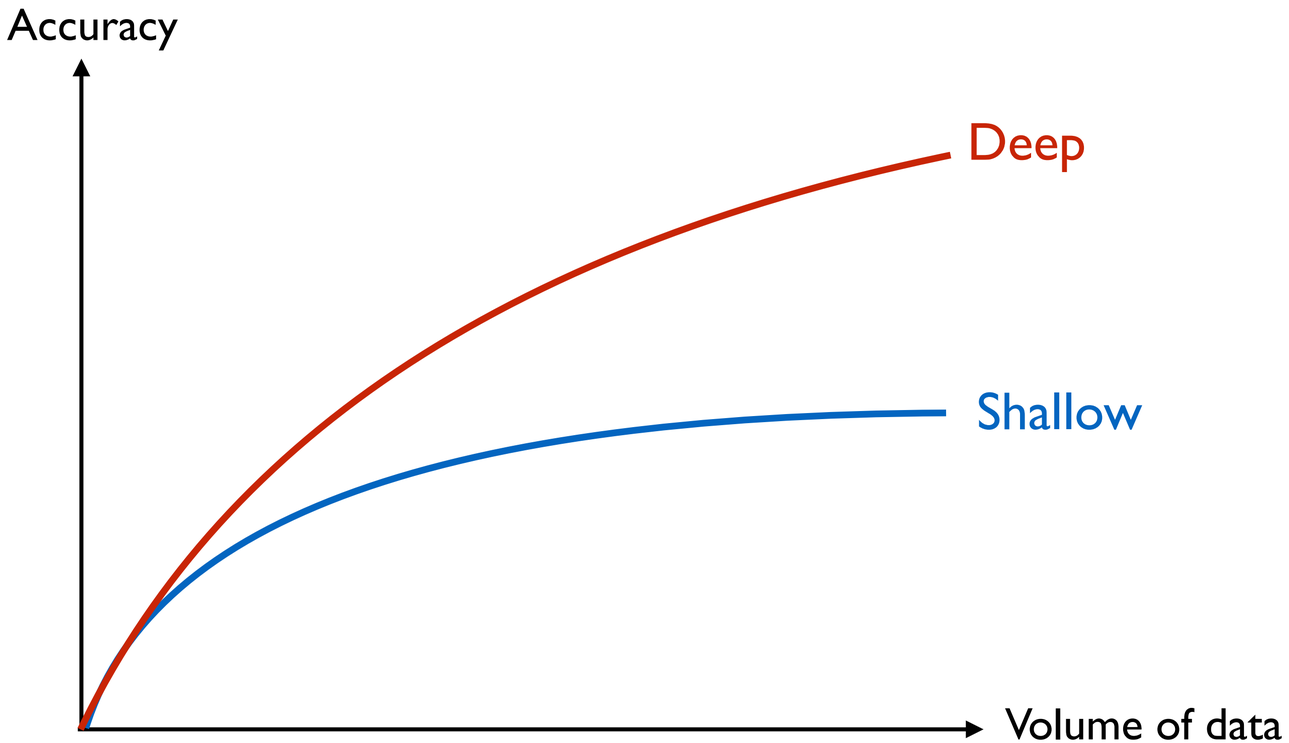
Choreography of neural network training

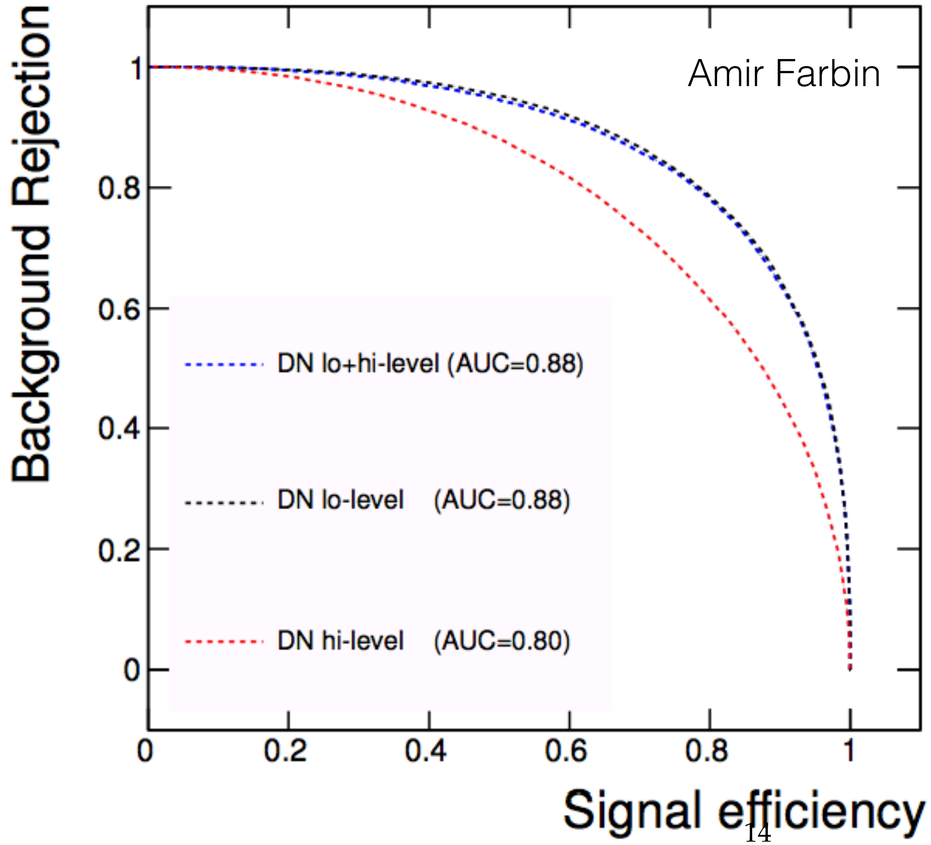
- Back-propagation is **not** the entire training procedure for a neural network!
  - It sets up the gradient through calculation of the derivatives but the training is completed by the minimisation of this function via weight adjustment
- An *epoch* is a *single pass through all of the training data*
  - Training over 100 epochs means the networks sees the training data, in full, 100 times
- In *stochastic training* each new event leads to a weight update → noisier so less likely to fall into local minima during minimization
- In *batch training* weights are only updated after a large number of events have been fed forwards, with the errors accumulating → faster but uses more memory
- *Mini-batch training* is a compromise between the two, with small batches being selected at random from the full sample

Deep learning

- Deep neural networks have a much larger parameter space and, **given enough training data**, can model more complex behaviour than a shallow network or a BDT

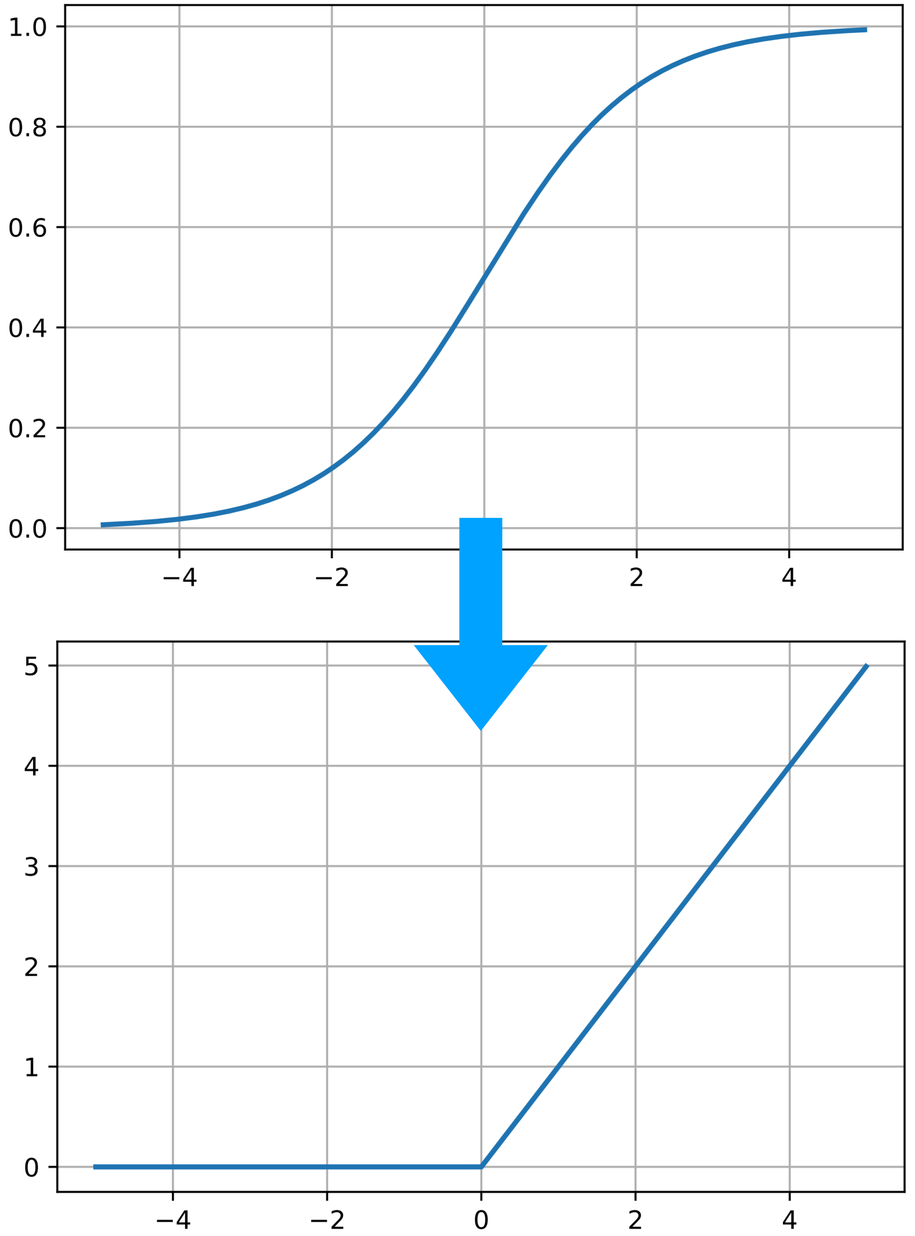
- Deep neural networks training is inherently suited to vectorisation and co-processors → allows use of GPUs or dedicated hardware





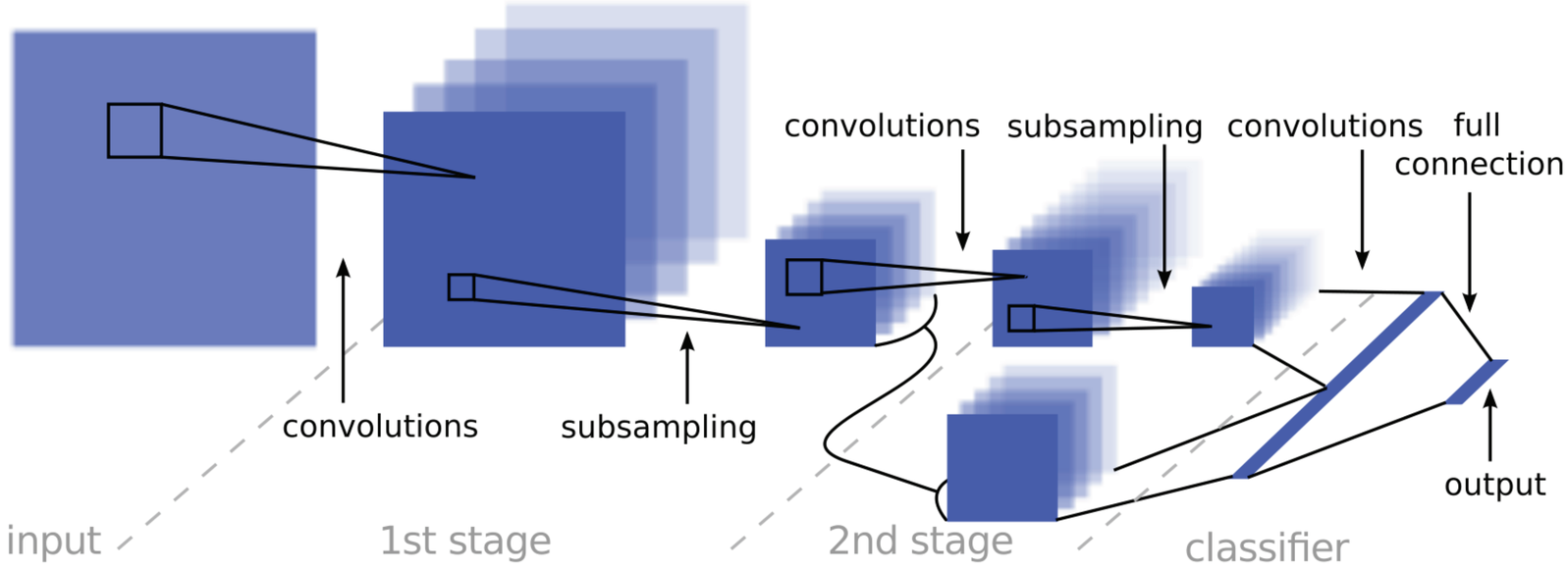
Three particular deep learning challenges

- Vanishing gradient problem
  - With so many nodes the gradients can tend to zero leading to a breakdown in the training
  - Partly solved by use of rectifier activation functions, e.g. ReLU, rather than sigmoid or tanh (etc)
  - Use of sign of the gradients only, etc
- Overtraining
  - Very easy for deep NNs to train on random fluctuations due to their high capacity
  - Solving this has required many innovations in regularisation for neural networks
  - e.g. dropout, weight decay, early stopping, data augmentation, momentum…
- Complexity: infinite variety in structure, hyper parameters, etc.



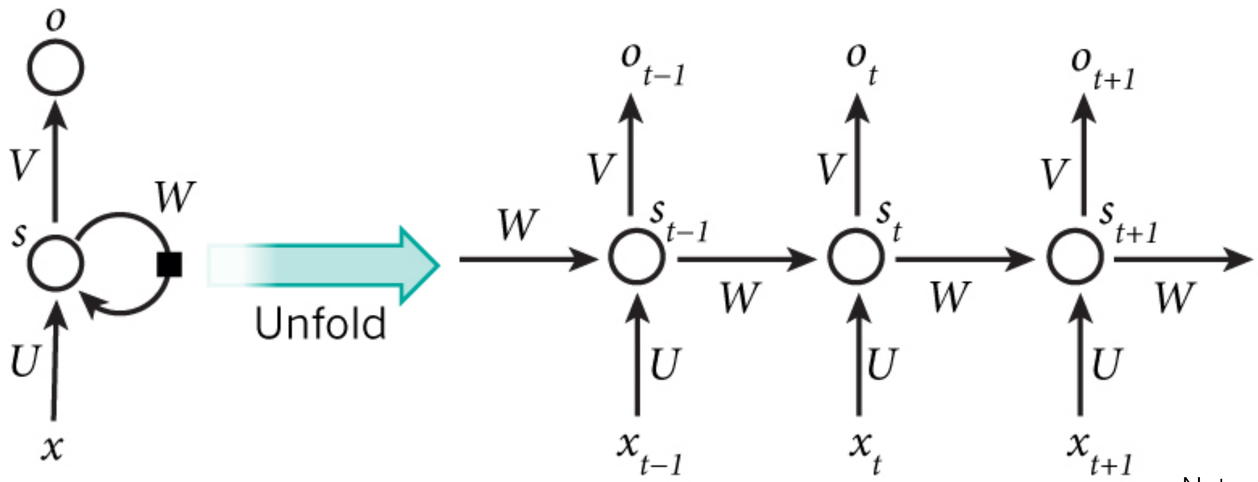
Neural networks come in different shapes and sizes

Convolutional - image recognition



P. Sermanet & Y. LeCun, DOI: 10.1109/IJCNN.2011.6033589.

Recurrent - time dependent sequences
(speech recognition, flavour tagging)



Nature

HEP applications

Software commonly used in HEP

Package	Description	Data structure	Main language	Download from…
TMVA	ML framework + algorithms	ROOT TTrees	C++/Python	Ships with ROOT (root.cern)
SciKitLearn	ML framework + algorithms	NumPy arrays	Python	http://scikit-learn.org or via Anaconda
XGBoost	Gradient boosted decision trees	NumPy arrays ++	Python/R/Julia/Scala/C++	https://xgboost.readthedocs.io or via Anaconda
TensorFlow/Keras	Deep learning	NumPy arrays	Python	www.tensorflow.org or via Anaconda

- The frameworks offer tools for making performance plots, organising data, cross validation etc.
  - They have built-in algs but these may not be the best for a given task - be prepared to plug in external applications from outside our community
- Divide between the ROOT and Python ecosystems becomes less important as Python-driven ROOT becomes more advanced
  - Uproot: convert ROOT files to Pandas dataframes without a ROOT installation
- If you are using the Python ecosystem, use Anaconda rather than installing each package independently
- Most ML software supports GPUs… and we have three very nice ones :-) Think about using them.

Exemplary achievements of ML in HEP

- Many HEP analyses use ML these days. But these are the particular highlights…

Top quark mass measurement @ Tevatron	Shallow NNs, BDTs
Single top quark discovery @ Tevatron	Shallow NNs, BDTs
Higgs discovery ($H \to \gamma\gamma$) @ CMS	BDT
Observation of $H \to bb$ @ ATLAS, CMS	BDT
Observation of $B_s \to \mu\mu$ @ ATLAS, CMS, LHCb	BDT
Observation of associated Higgs and top quark pair production (“ttH”) @ ATLAS, CMS	BDT (XGBoost @ ATLAS)
Jet flavour tagging	BDT, shallow NN, recurrent NN
Calorimeter response modelling	Generative-adversarial networks

Discovery with ML: single top quark production

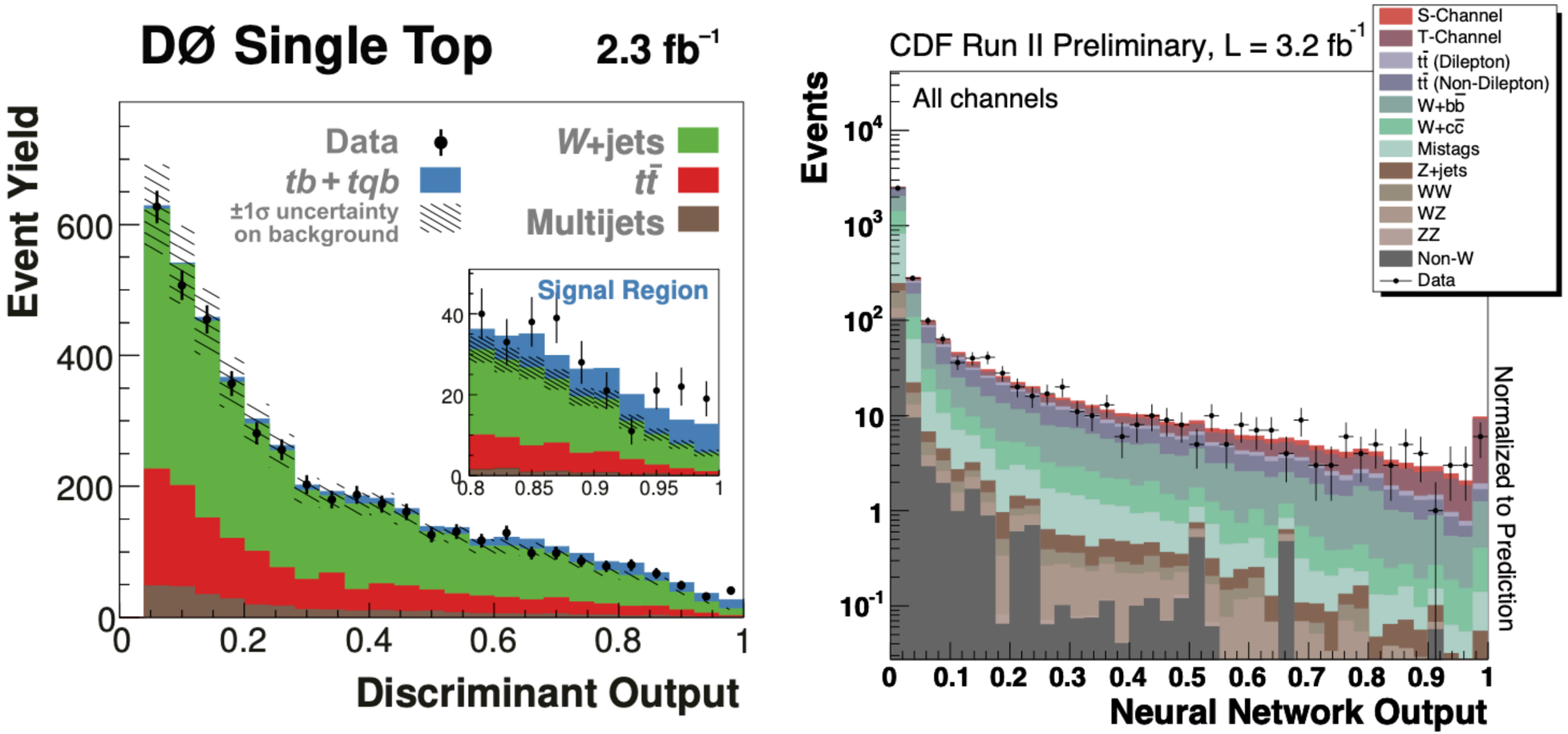
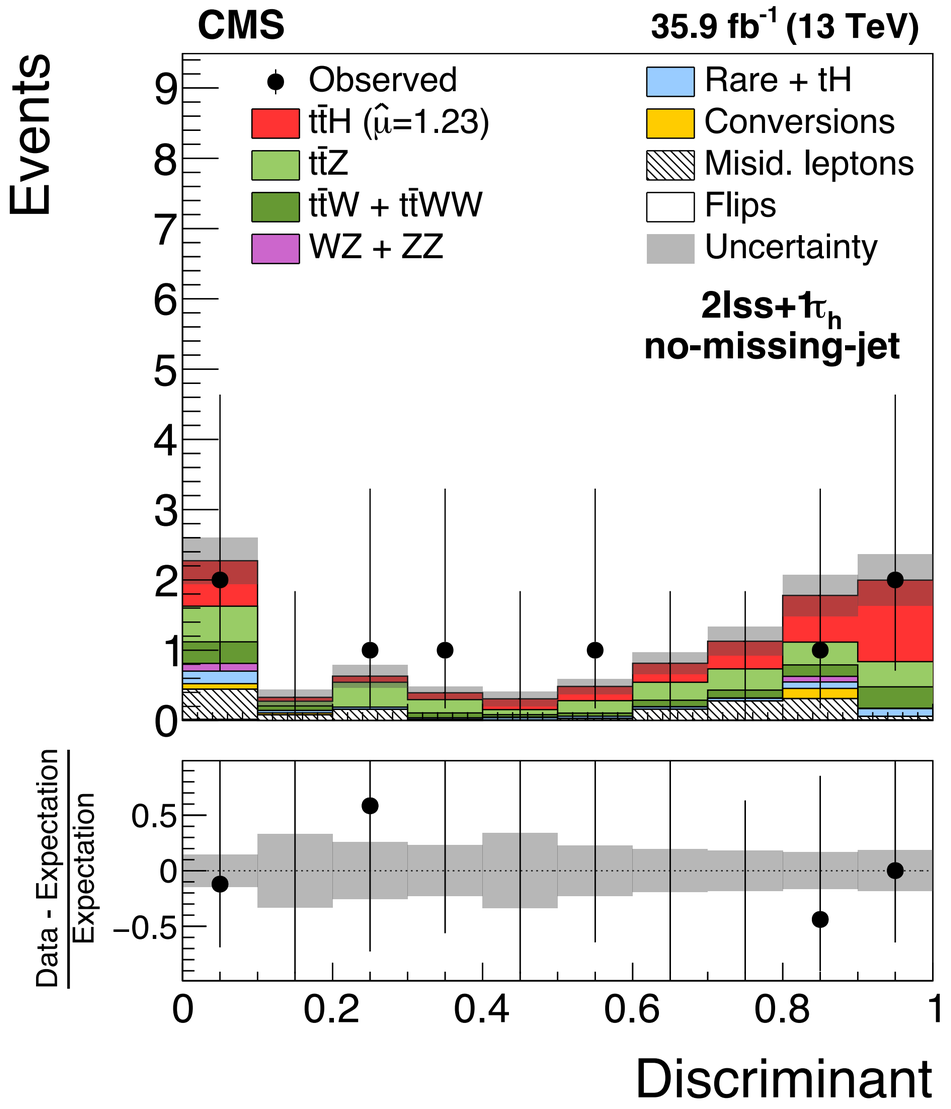
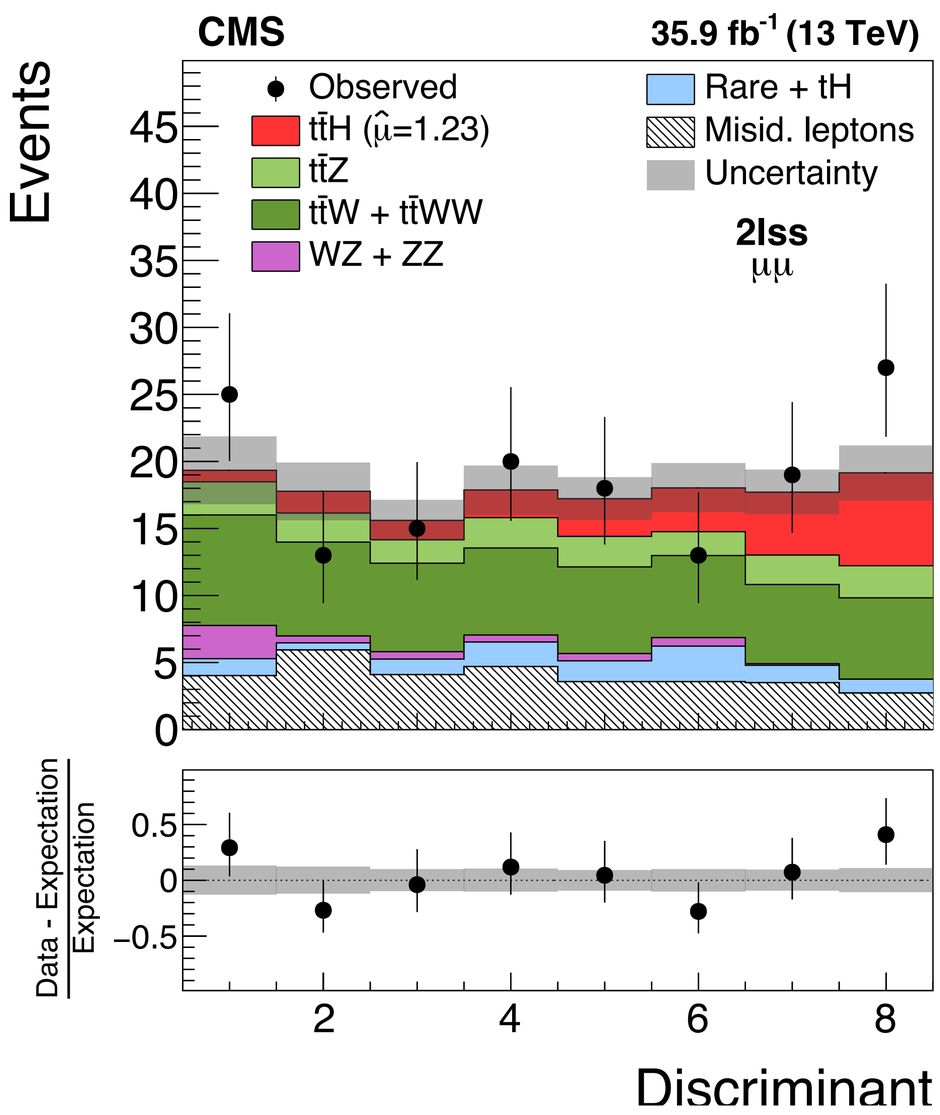


Figure 2: Combined super discriminants for DØ (left) and CDF (right).

https://arxiv.org/pdf/0906.0523.pdf

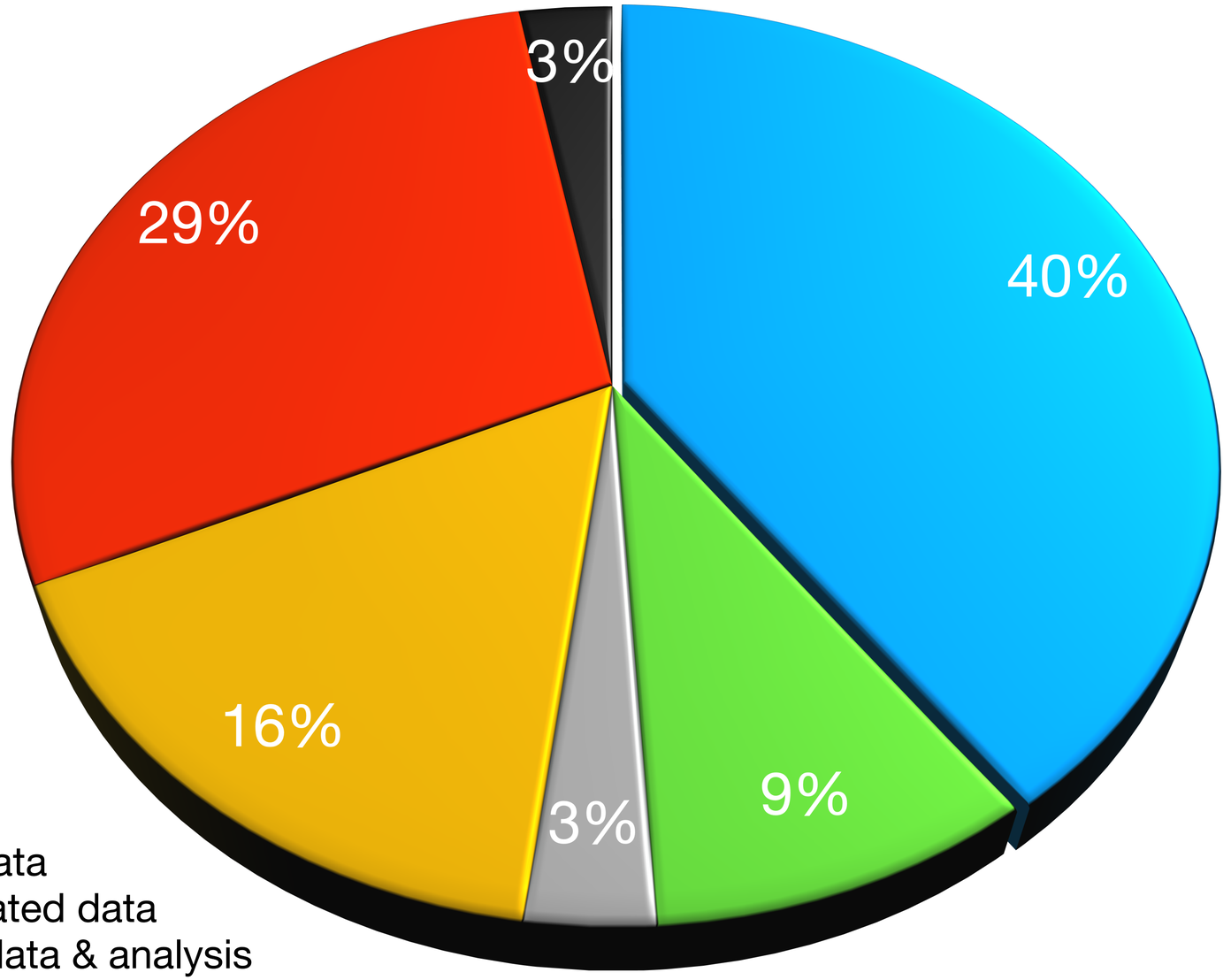
Discovery with ML: ttH observation at CMS

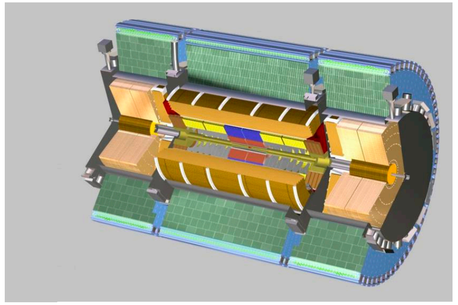




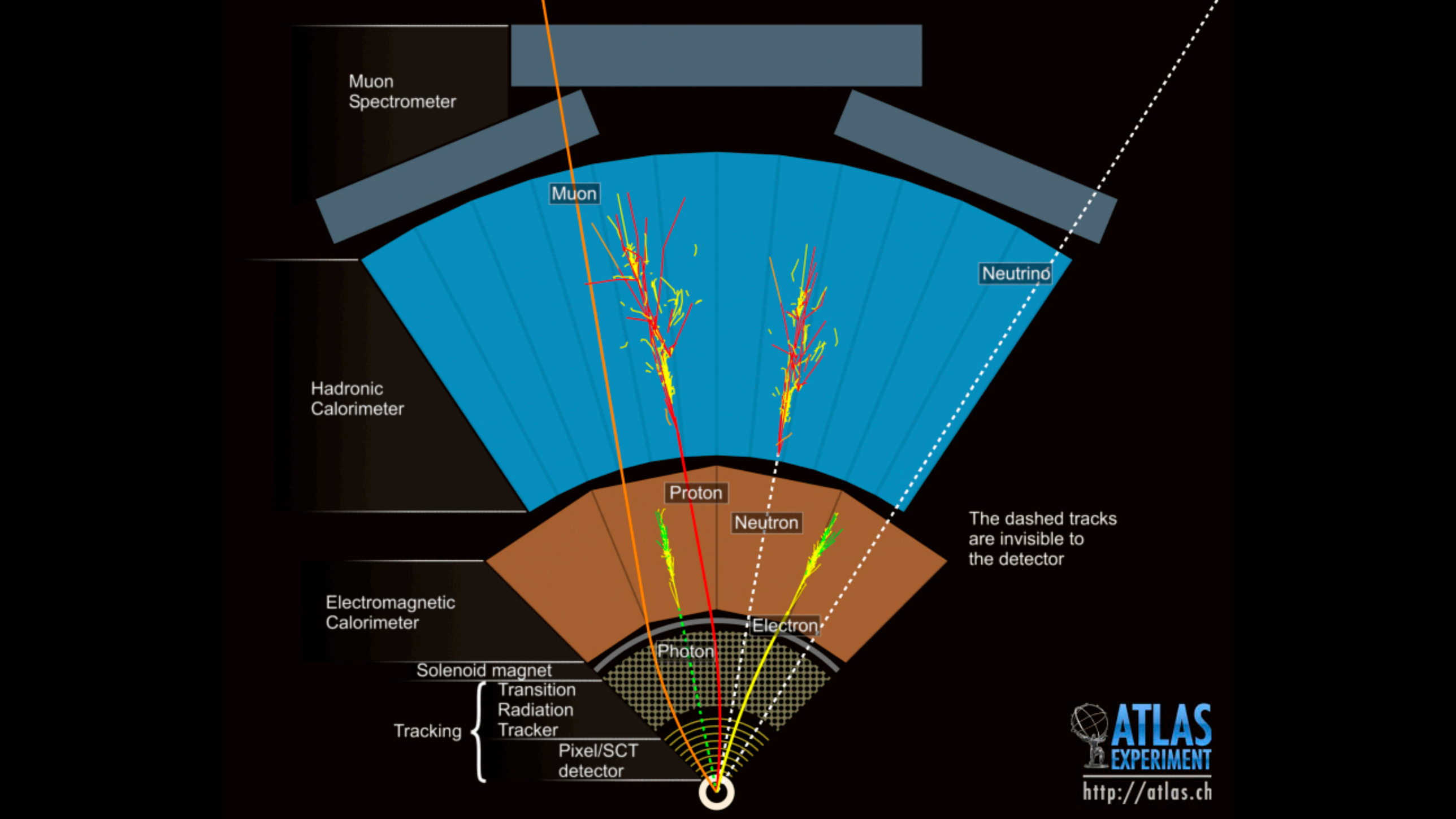
ATLAS CPU consumption for the main processing steps

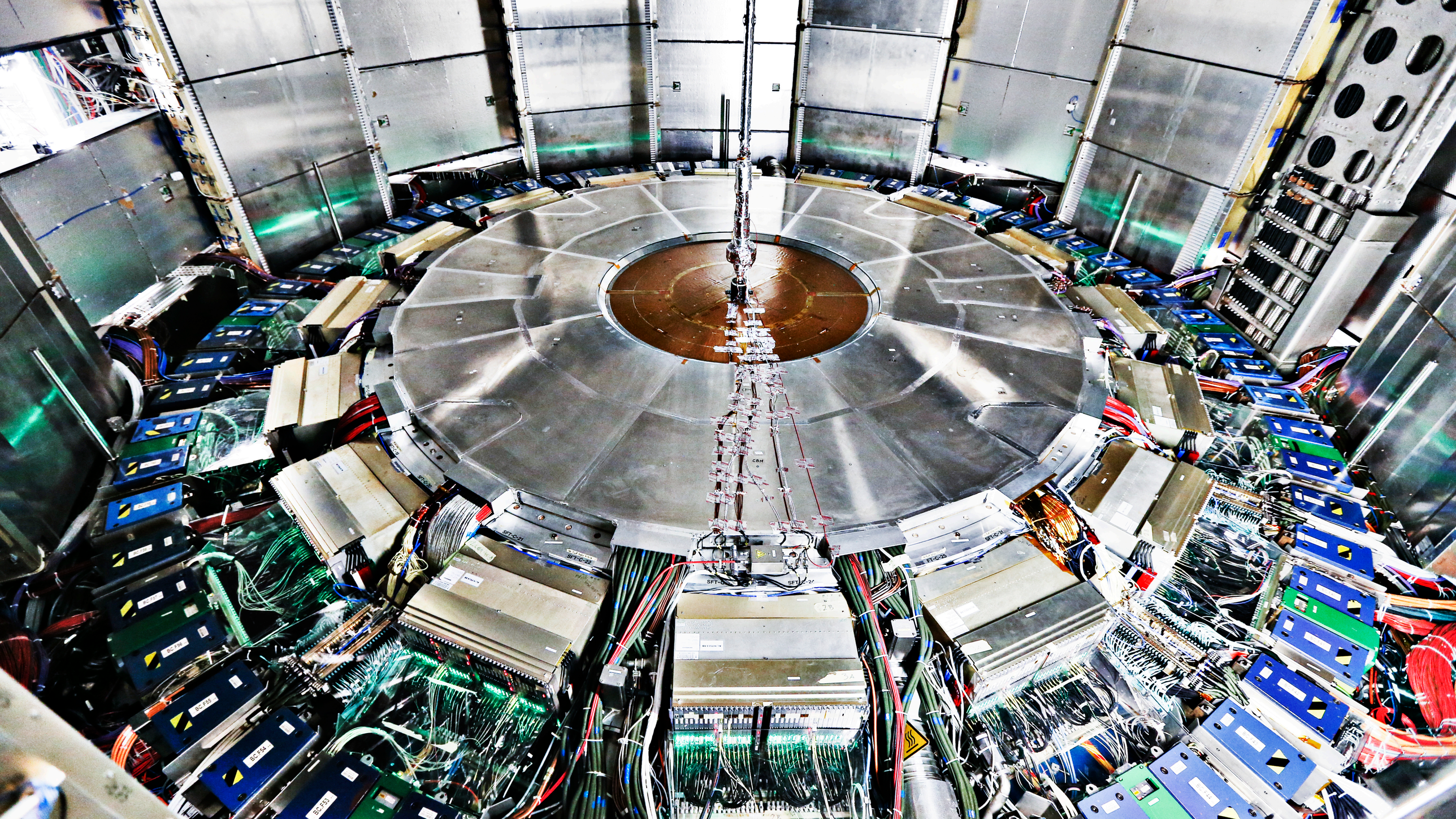
Running job slots corrected to processor performance
12 months to 4th May 2021





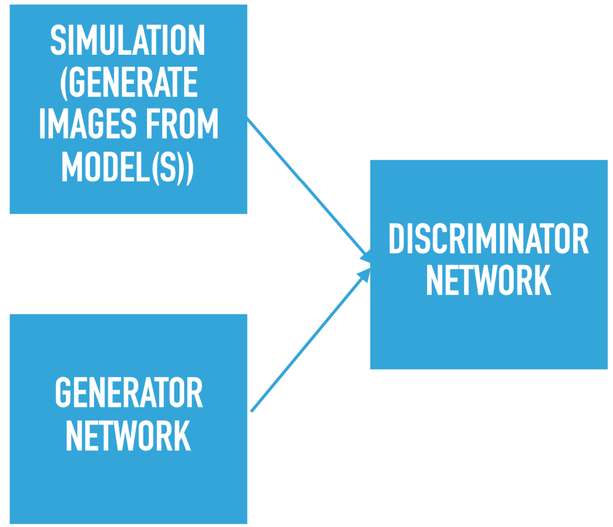
Simulating the calorimeters is very compute-intensive.

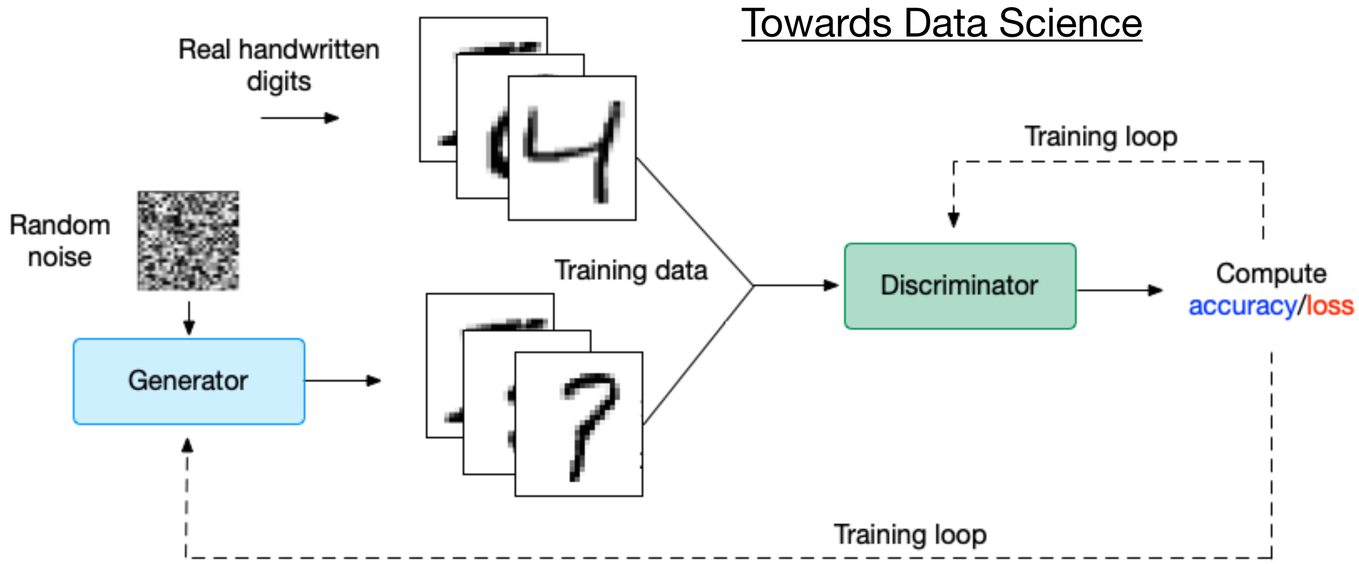

Muon
Spectrometer
Muon
Neutrino
Hadronic
Calorimeter
Proton
Neutron
The dashed tracks
are invisible to
the detector
Electromagnetic
Calorimeter
Electron
Photon
Solenoid magnet
Tracking
Transition
Radiation
Tracker
Pixel/SCT
detector
ATLAS
EXPERIMENT
http://atlas.ch


BC-F53
BC-F54
BC-F55
BC-F56
SFT-C-21
SFT-C-22
SFT-C-25
SFT-C-26

Generative-adversarial network (GAN) training

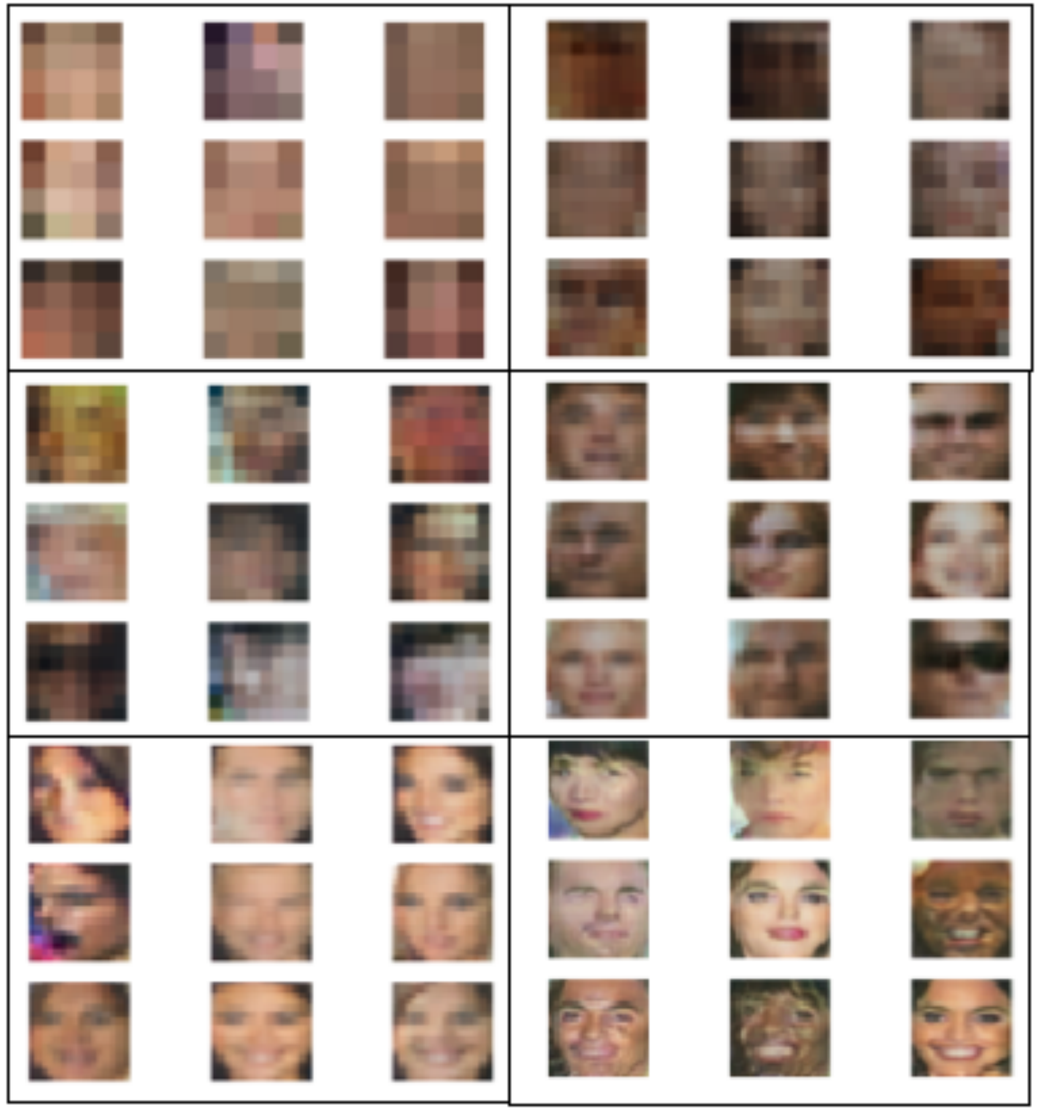
- Adversarial training: two competing algorithms
  - *Generator* tries to produce fake data using random noise as input
  - *Discriminator* tries to distinguish fakes from real examples
  - Each gets a chance to train while the other is kept fixed
  - Generator eventually learns to fool the discriminator
  - Once they reach an optimum, the generator can be used to create new examples, much more quickly than the training data can be produced
- Clear application in mimicking fully simulated events





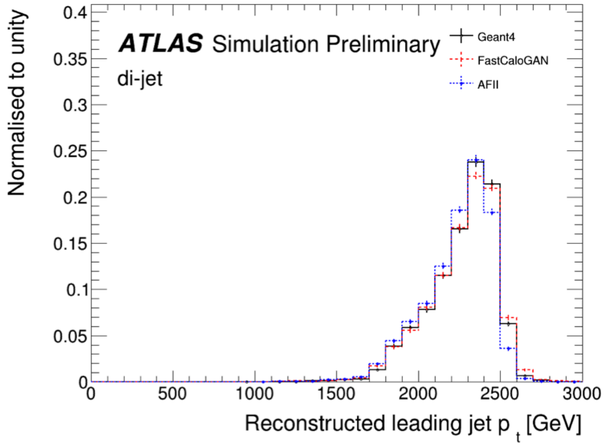
Generative-adversarial training

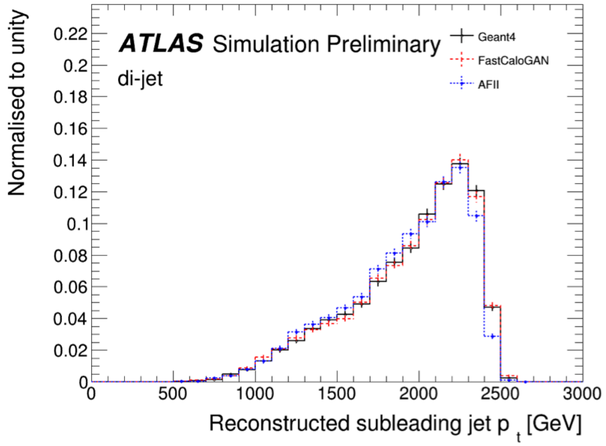
Towards Data Science

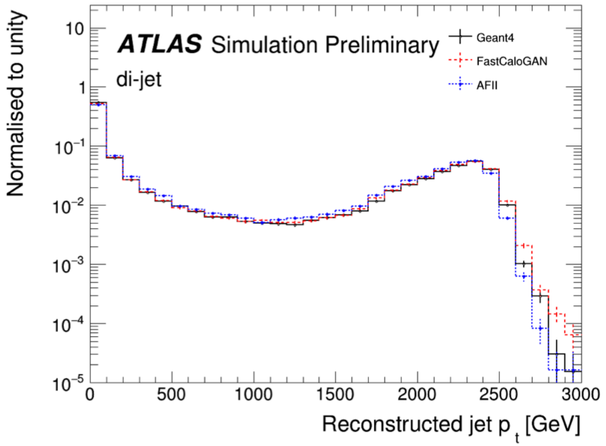


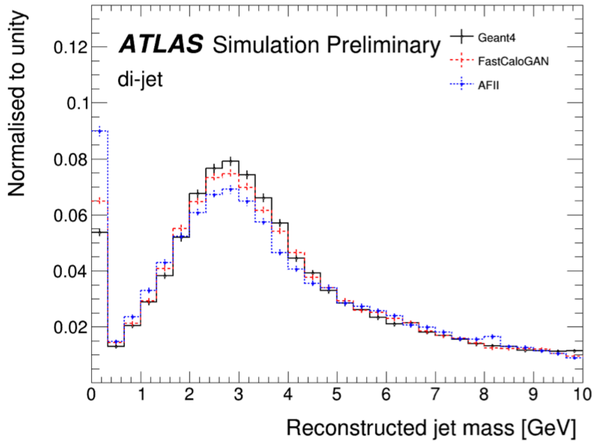
Wikipedia

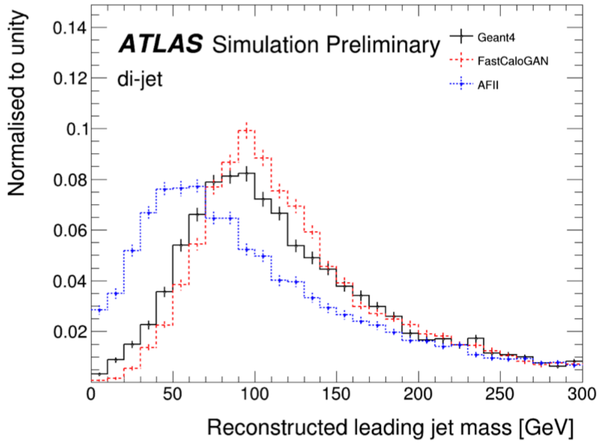
ATLAS FastCaloGAN

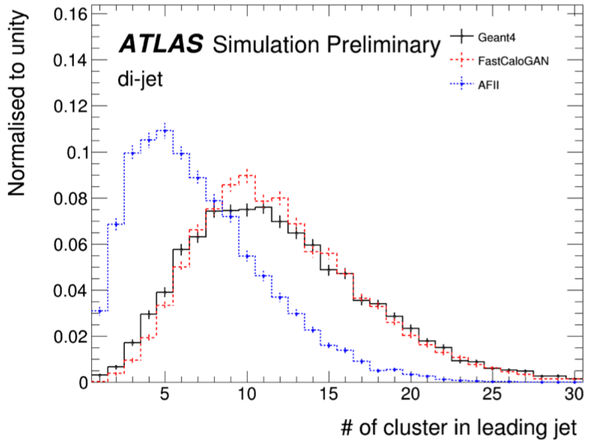


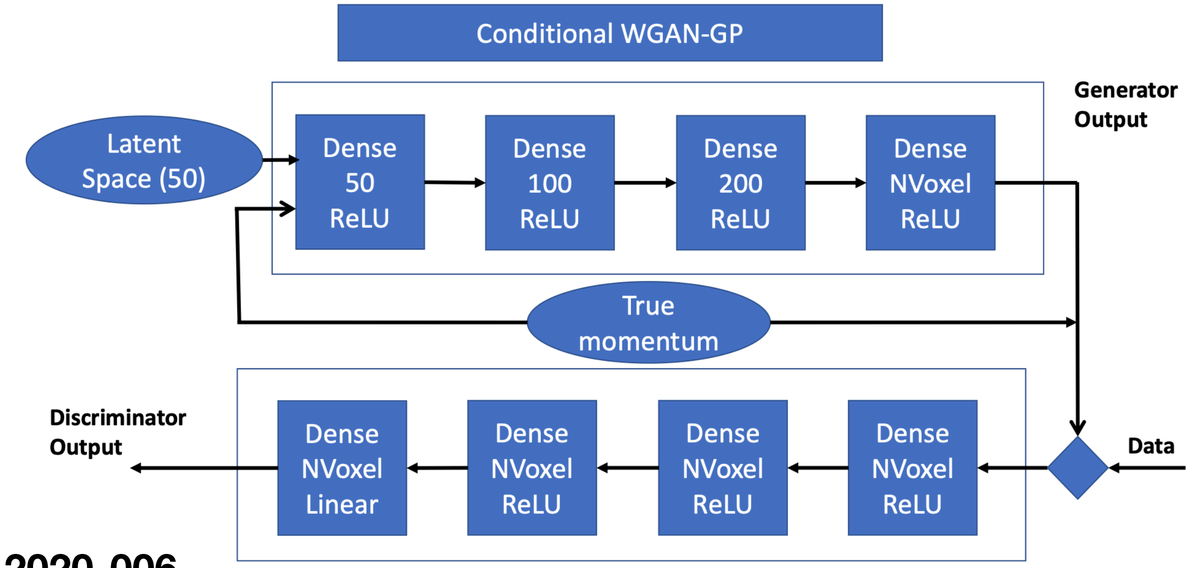










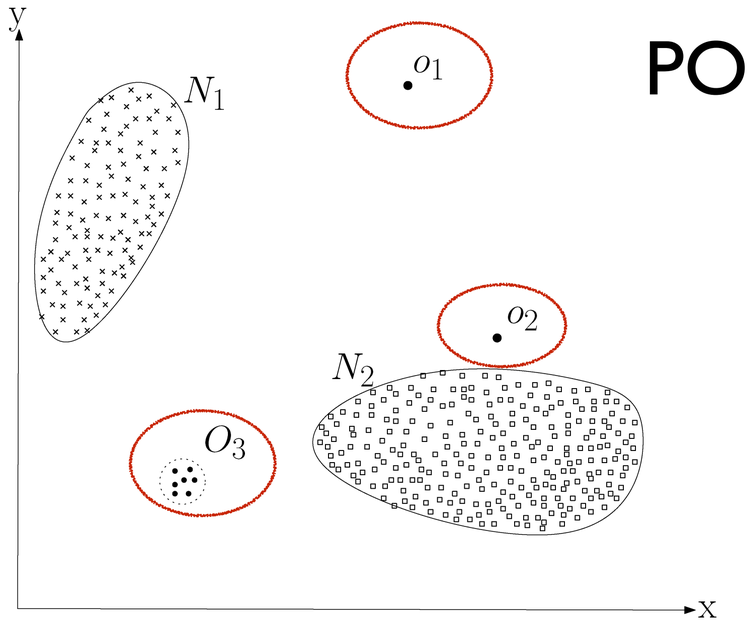


ATL-SOFT-PUB-2020-006

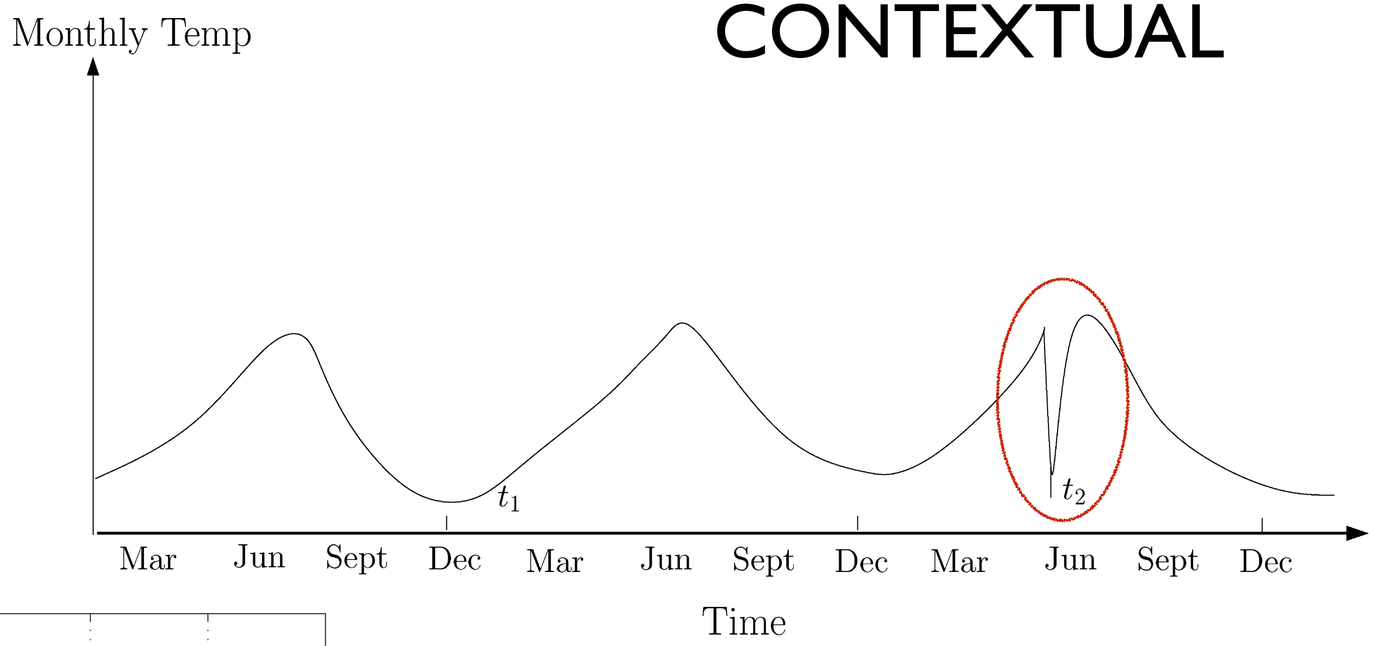
Anomaly detection

- Automatic identification of data instances (events) that are in some way different from the bulk of the data and which need detailed scrutiny by experts. Usually implied:
  - produced by a different mechanism than the bulk of the events
  - small number of anomalies w.r.t. the main part of the data
- Can be
  - supervised: train to recognise specific anomalous cases
  - semi-supervised: train only on the bulk of the data without anomalies → strong relation to one-class classification
  - unsupervised: algorithm automatically identifies the bulk by some means and thence the anomalies
- Difficult problem because in general we don't know what the anomalies look like, and there may be very few of them
  - Testing is particularly challenging: how do we evaluate the performance of an algorithm on a type of event that we have never seen before?
- **The "holy grail" of ML in HEP is a generalised anomaly detection system that could perceive new physics in the data, having only been trained on background MC or control data**

Types of anomaly

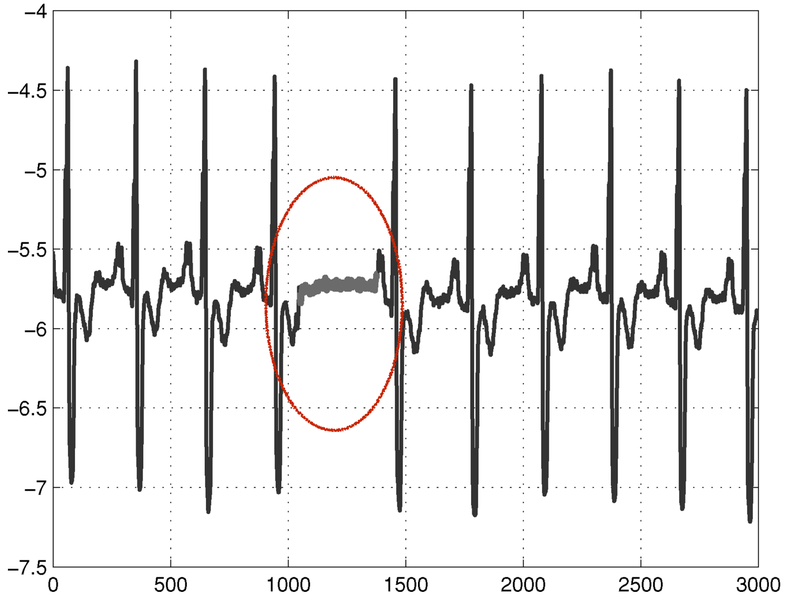


POINT



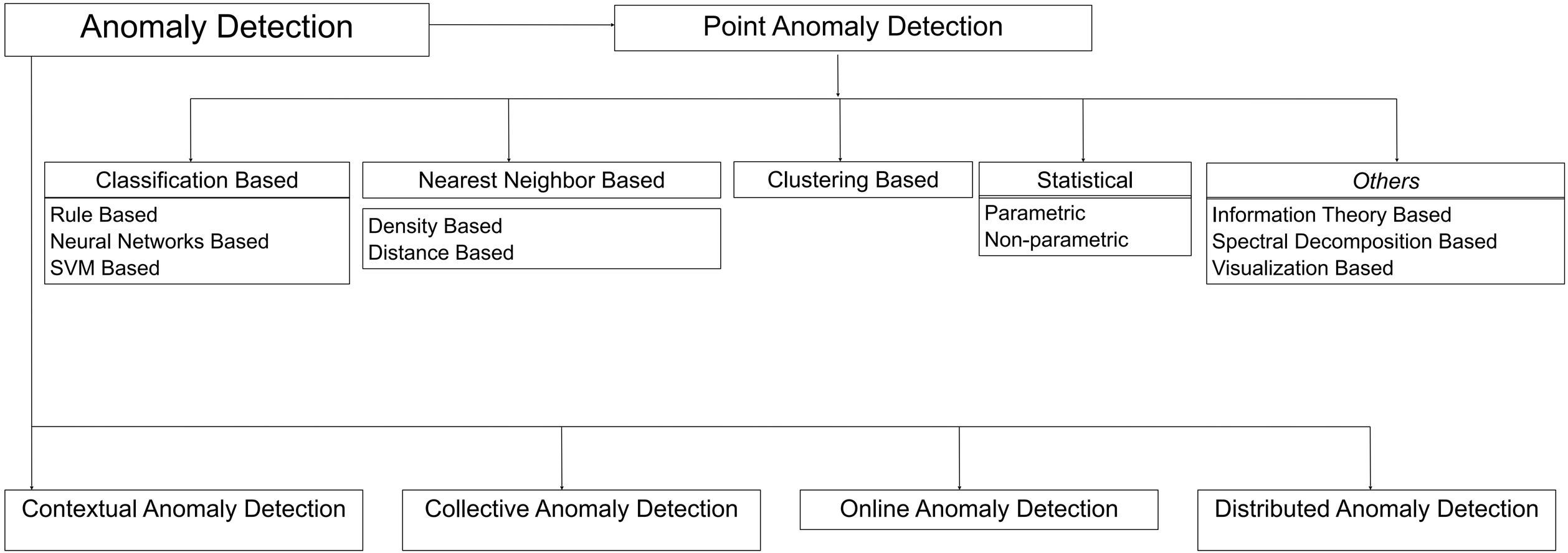
CONTEXTUAL

COLLECTIVE
(population-level)



Images:
ACM Computing Surveys,
Vol 41, No. 3, Article 15

Taxonomy



* Outlier Detection – A Survey, Varun Chandola, Arindam Banerjee, and Vipin Kumar, Technical Report TR07-17, University of Minnesota (Under Review)

Auto-encoders

- Neural network which is trained on its *own input*

Try it on Thursday!

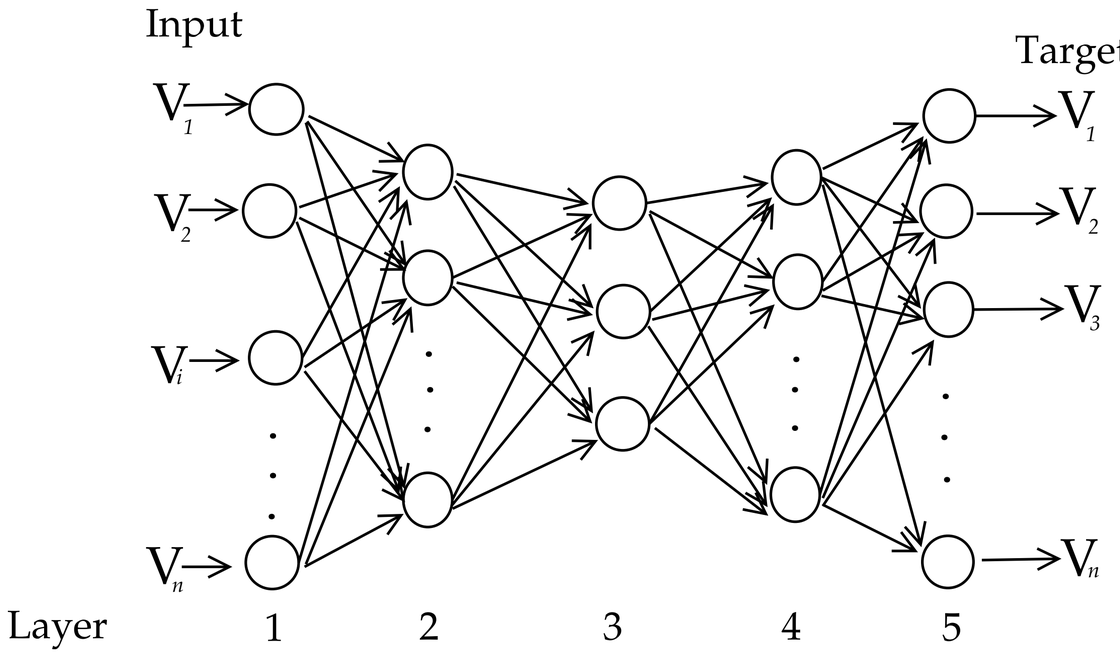
- Learns how to regenerate data which "looks like" the input, on the assumption that the input is sampled from the same distribution that the training data came from

- Usually includes a bottleneck to compress the data into fewer dimensions

- Normally used for dimension-reduction and compression but proposed as a means of anomaly detection

  - If a sample comes from a different distribution (e.g. it is an anomaly) the auto-encode will do a poor job of regenerating it, leading to a high *reconstruction error*

  - Provides a natural measure of abnormality: the reconstruction error (difference between the input and the output)



$$E_{rec} = \sum_{i=1}^{N} (x_i^{in} - x_i^{out})^2$$

N = number of features

Open data

- Normally you have to be a member of a Collaboration in order to access its data for analysis
  - Those who are not members of a big experiment may struggle to get enough data to try out their new ideas
  - For those of us in an experiment there is no shortage of data and simulation to play with, but getting it into a reasonable shape can be a lot of work
- Solution: Open Data Portal - http://opendata.cern.ch
- Ideal for educational purposes
- Examples in this talk prepared with the HiggsML challenge dataset:
- This is what we'll use on Thursday

Plan for Thursday

- Use Jupyter notebooks running on Galaxy
- Try BDTs and neural networks
  - Simple toy data
  - Higgs ML dataset
  - General open data (new physics including SUSY)
- Anomaly detection techniques with auto-encoders