An introduction to Machine Learning Methods in High Energy Physics

Arun Nayak

Institute of Physics, Bhubaneswar

(with help from Sanu Varghese)

Online workshop on
Software Tools and Techniques used in EHEP and its Applications
MNIT, Jaipur
12th – 19th July 2021.

Introduction

What is Machine Learning?

- A method of data analysis that automates analytical model building
  - Based on the idea that computers can learn from data, recognize patterns, and make decisions with minimal human intervention.
  - Has made big advancement recently because of new computing technologies.

Example Applications you are familiar with:

- Online recommendation offers, such as from google, Amazon, Netflix
- Fraud Detection
- Spam detection in email
- Recognizing hand-written letters and digits

Popular Learning Methods

- Supervised Learning
  - Algorithms are trained using labeled examples, i.e. with desired outputs known
  - Learns by comparing actual output to correct/known outputs to find errors → Modifies the model accordingly
  - Use patterns to predict values of the output for an unknown data.
  - commonly used in applications where historical data predicts likely future events
  - Classification, Regression, Gradient Boosting etc..
- Unsupervised Learning
  - Used against data that has no historical labels – unknown desired outputs
  - Algorithm must figure out what is being shown
  - Goal is to explore the data and find some structure within
  - e.g. Detecting Anomalies
  - Popular techniques like nearest-neighbor mapping, k-means clustering

In this talk we will discuss only supervised learning methods

Popular Learning Methods

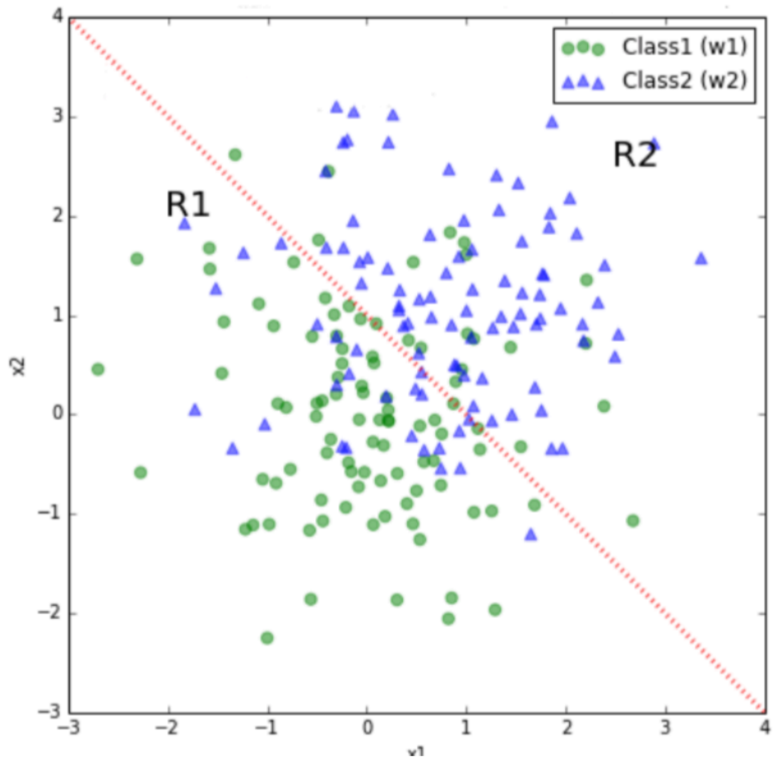
- Semisupervised Learning
  - Similar application as supervised learning
  - Uses both labeled and unlabeled data for training
  - Small amount of labeled, large amount of unlabeled data

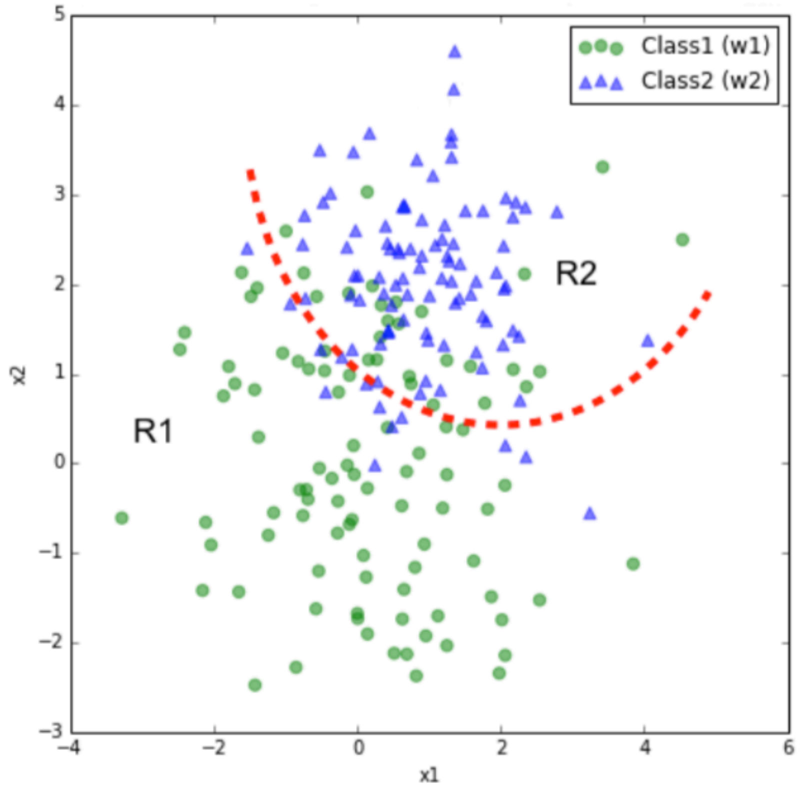
- Reinforcement Learning
  - Often used for robotics, gaming and navigation
  - the algorithm discovers through trial and error which actions yield the greatest rewards

In this talk we will discuss only supervised learning methods

Multivariate Analysis Methods

- Any statistical analysis technique that analyzes many variables at once
- Normally, cut-based methods, that apply selections on one variables at a time, are robust, but result in a low signal efficiency
- MVA techniques belong to the family of "supervised learning" algorithms
- MVA methods make use of training events, for which the desired output is known, to determine the mapping function



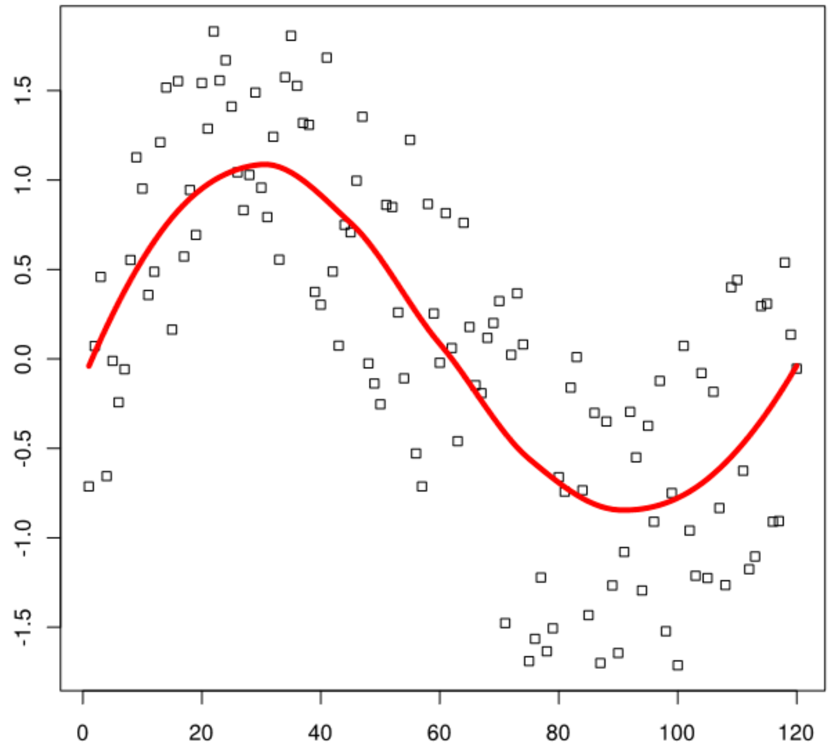


(images from google)

Multivariate Analysis Methods

- MVA methods are used for both classification and regression:
  - Classification: The mapping function describes a decision boundary
  - Regression: The mapping function describes an approximation of the underlying functional behaviour defining the target value
- Example MVA techniques:
  - Artificial Neural Network, Boosted Decision Trees

e.g. learn to classify birds and animals



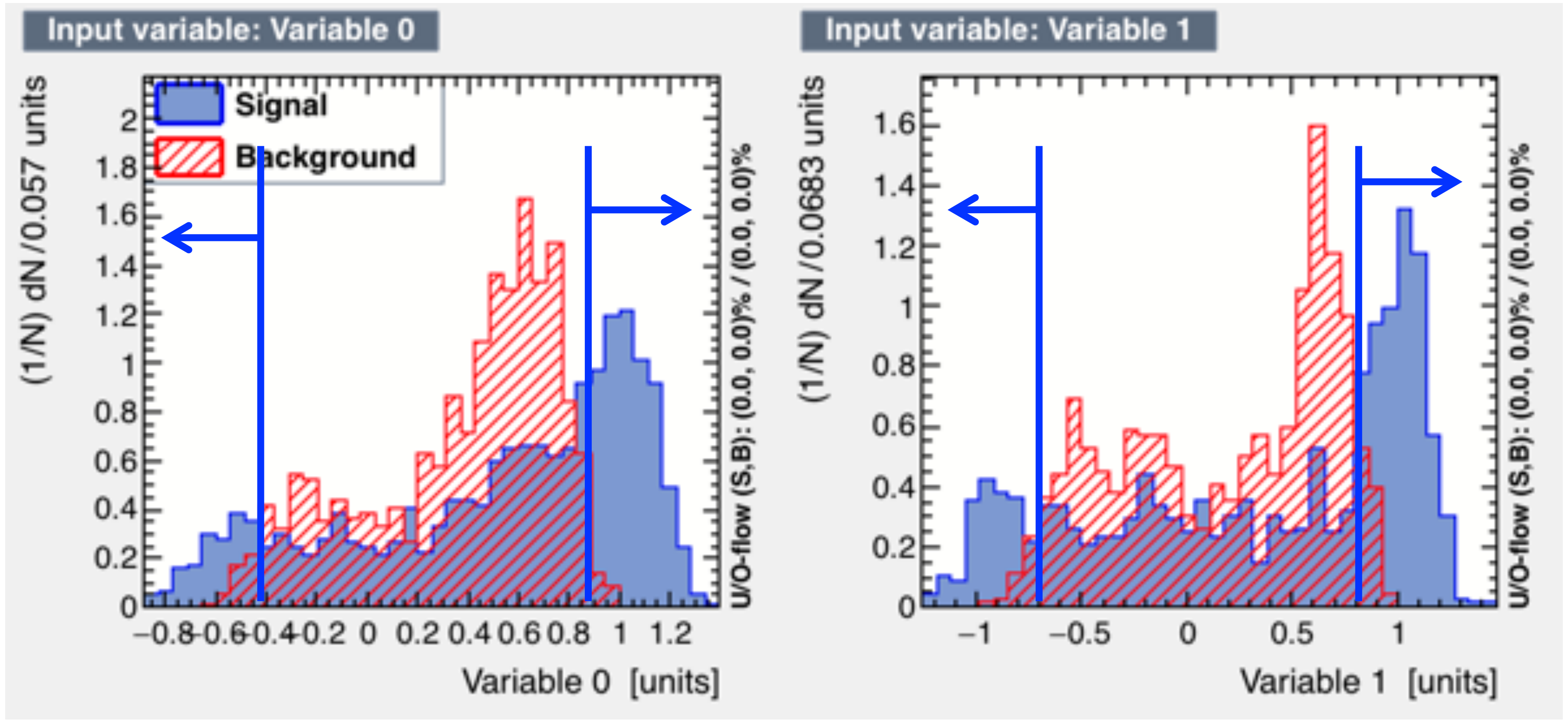
(images from google)

Why use MVA Analysis?

In High Energy Physics Experiments, we often perform data analysis to search for some signals which are produced at much smaller rate than that of the backgrounds.

Signal: Some event/object that we are interested in

Backgrounds: Events/Objects that we are not interested, but they look very much similar to that of our signal.



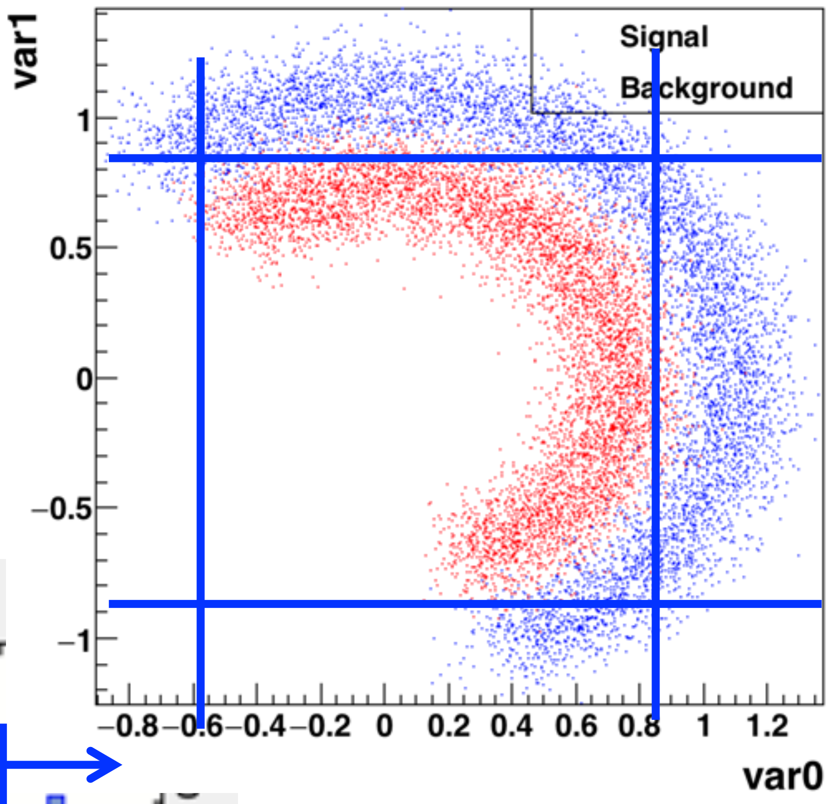
Cuts are not optimal

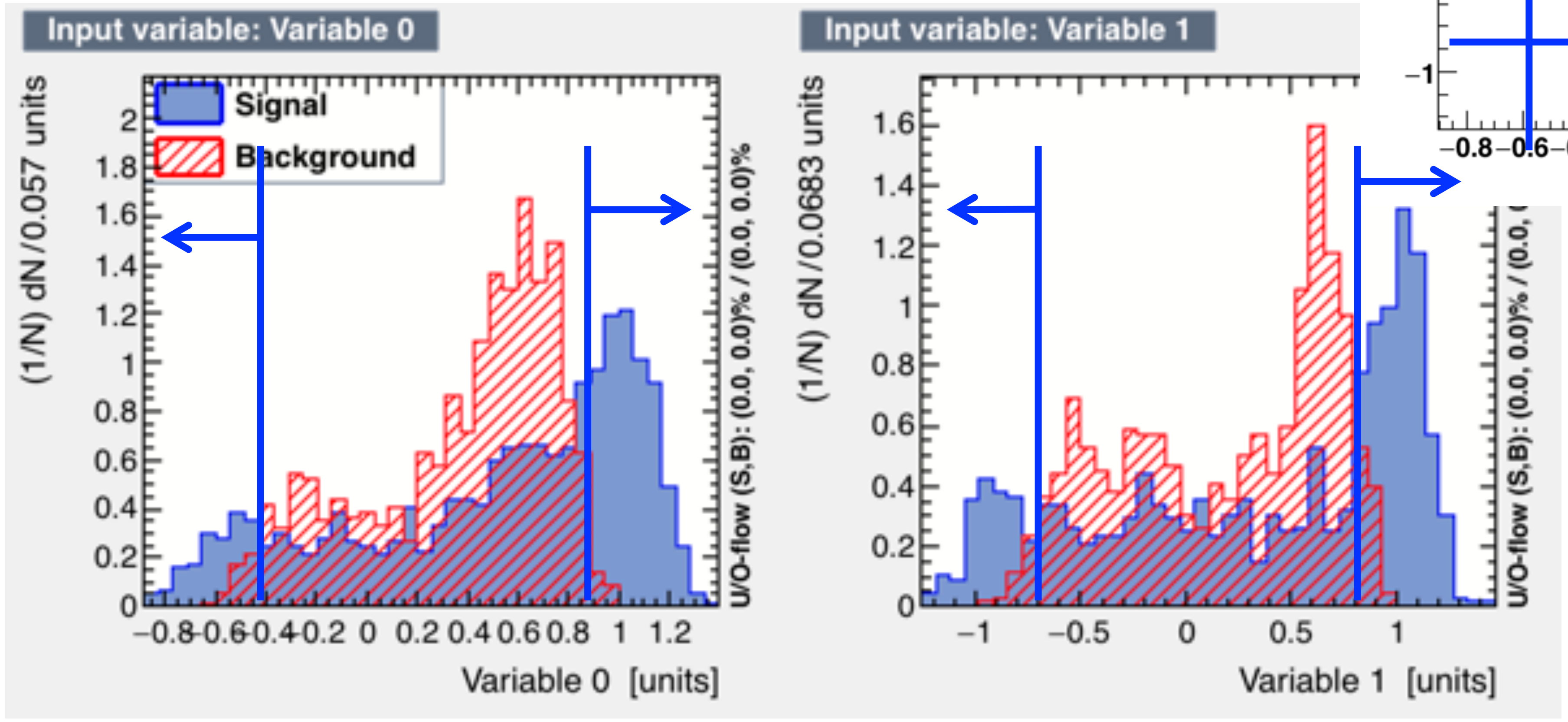
- Can not take correlations in to account
- Can lead to low signal efficiency and high background rate

Why use MVA Analysis?

Cuts are not optimal

- Can not take correlations in to account
- Can lead to low signal efficiency and high background rate

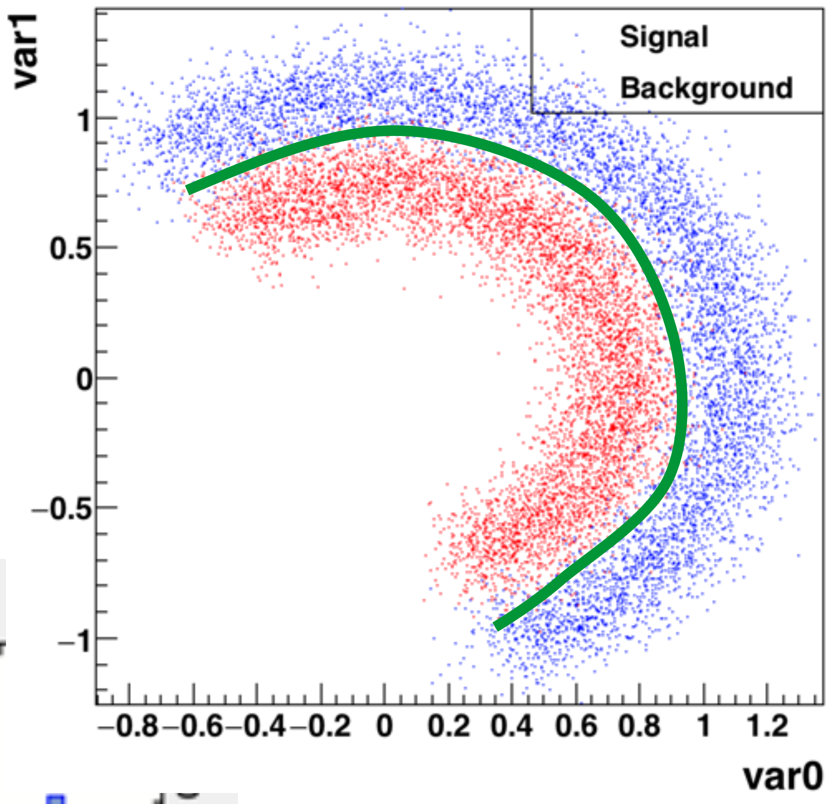


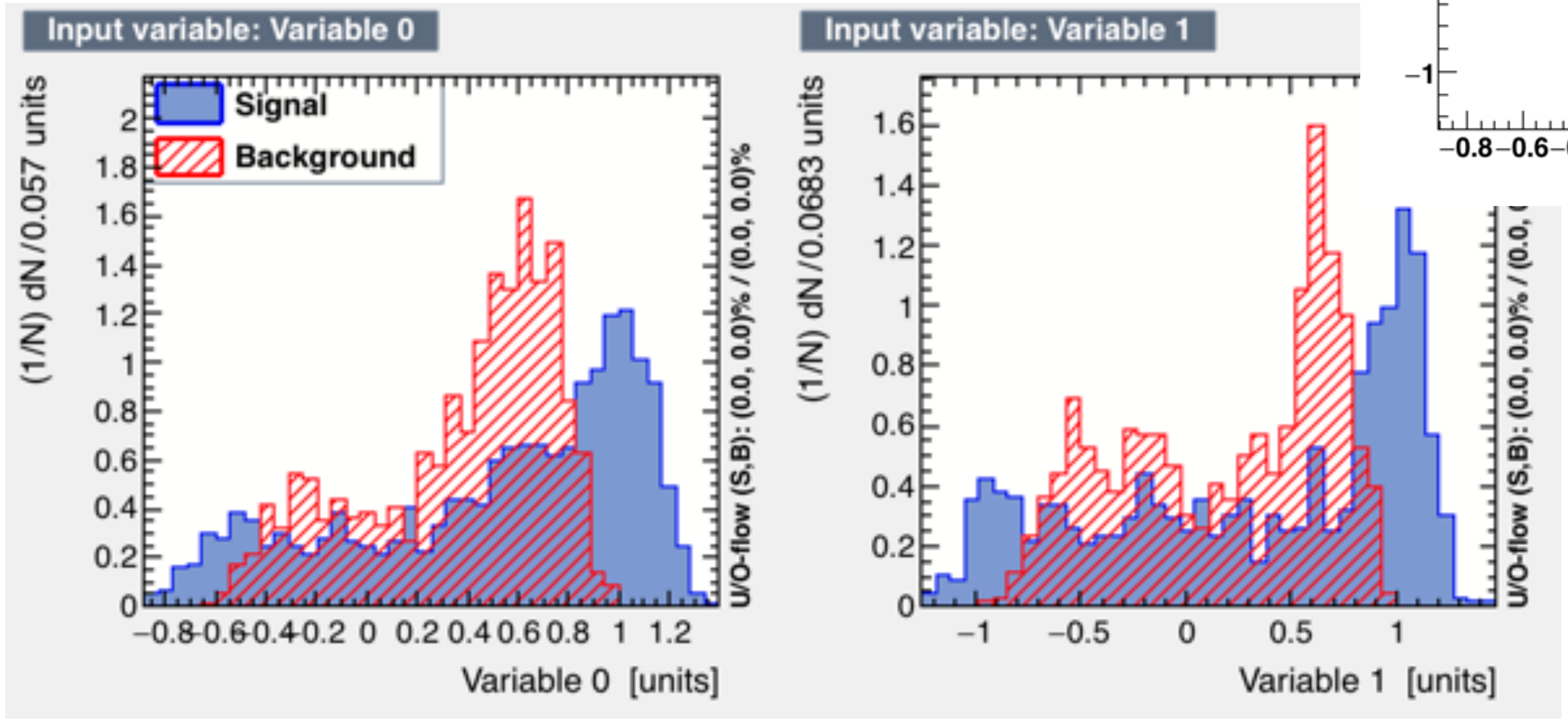


Data generated using Root/tutorials/tmva/ createData.C

Why use MVA Analysis?

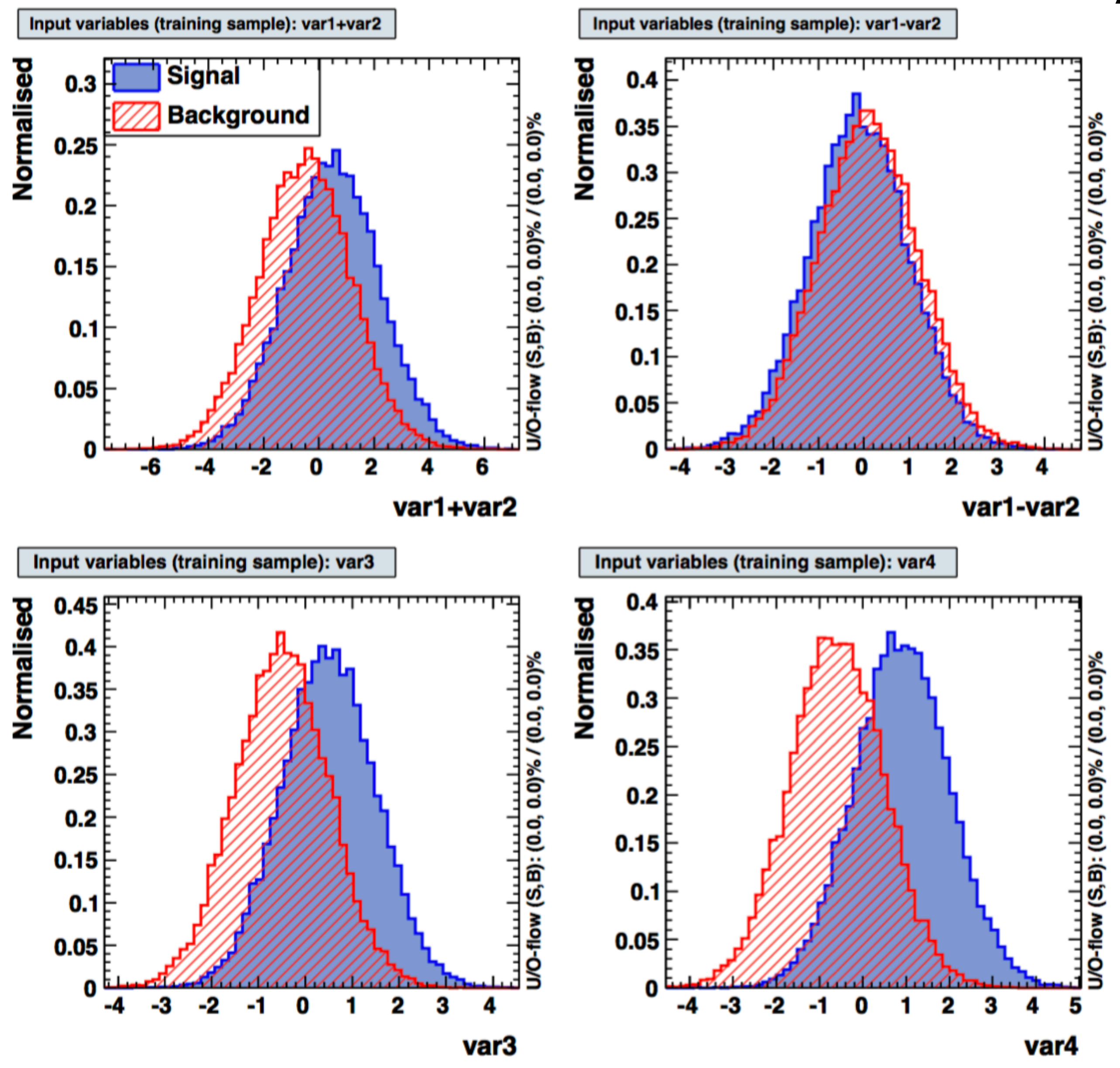
The actual boundary should be →



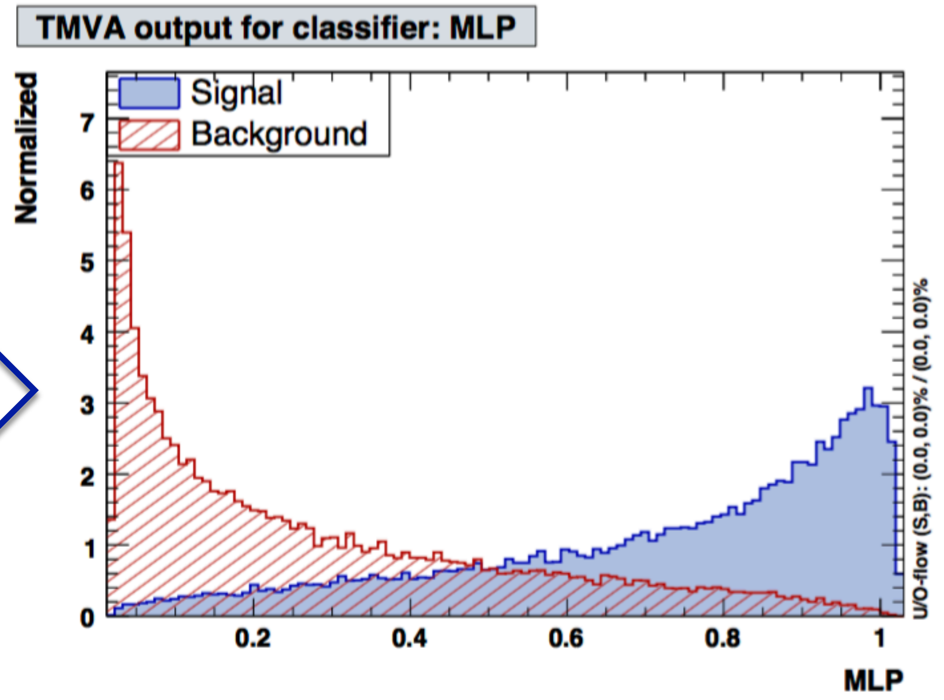


Why use MVA Analysis?

An Example



MVA

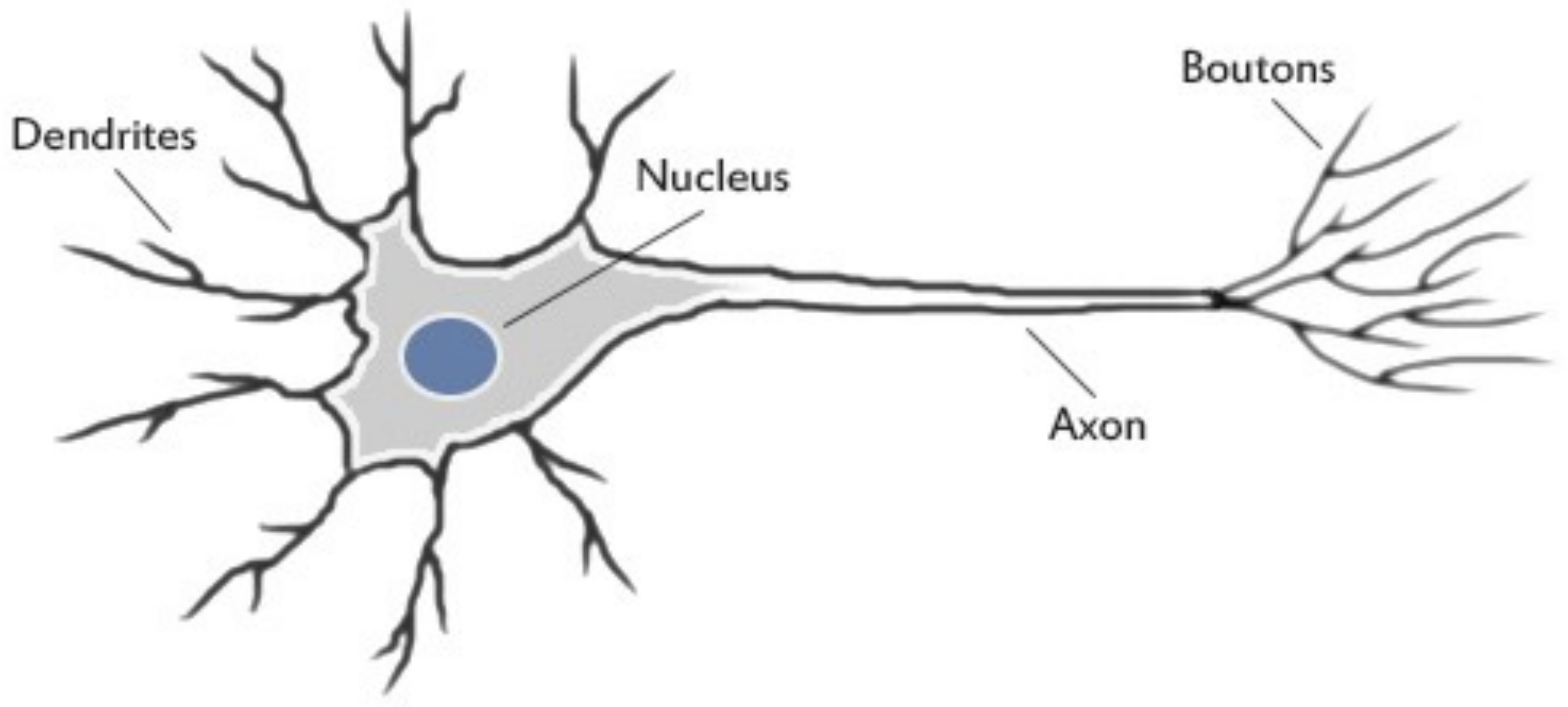


Combines many variables to a single variable, on which a cut can be applied depending on the required signal efficiency and purity

(figures from TMVA userguide)

Neural Network

Neural connections in human brain

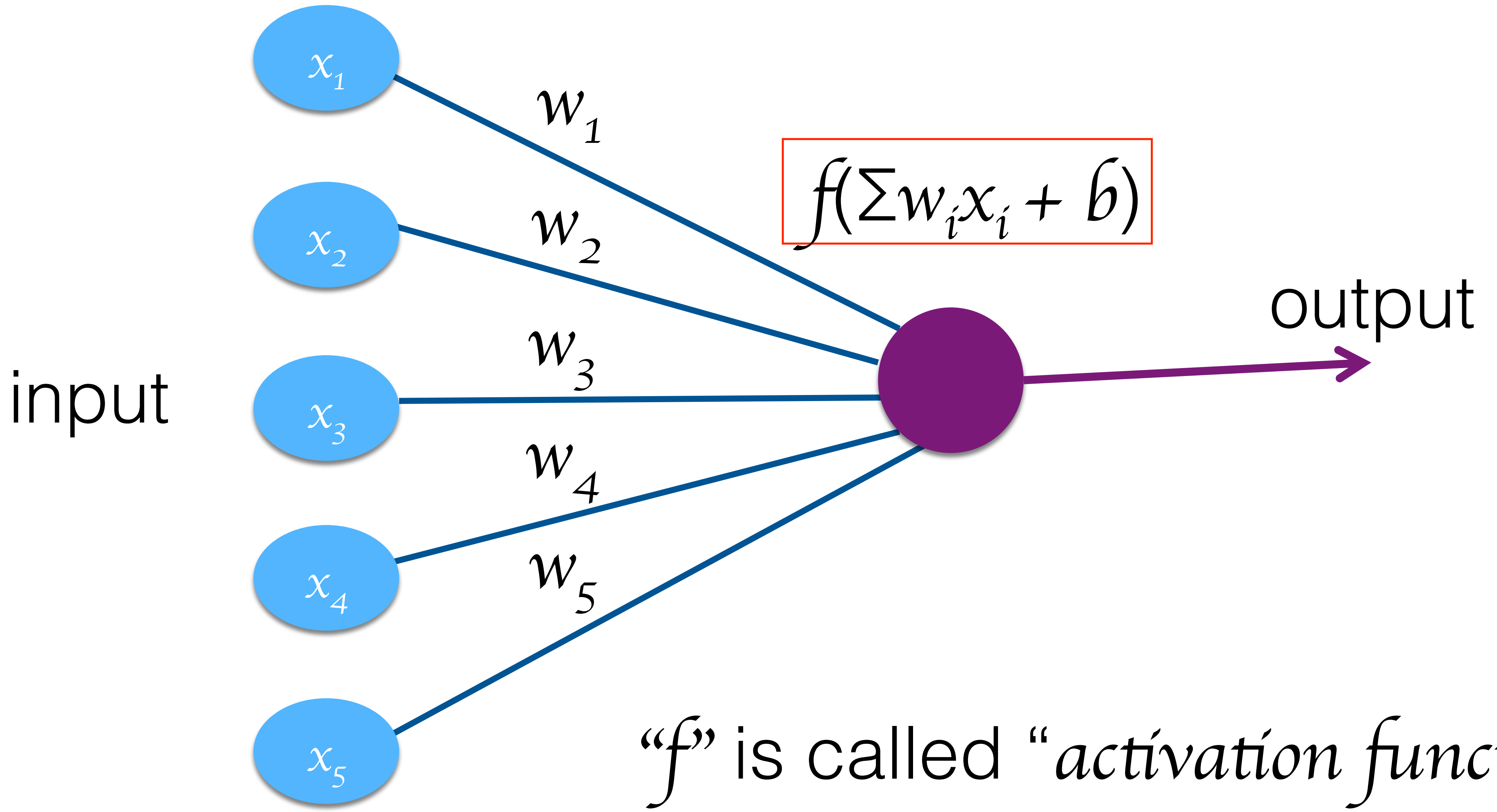


- Collects inputs from other neurons using dendrites
- Sums all the inputs, and fires, if the value is greater than a threshold
- The fired signal is then sent to other neurons through the Axon

Our brain uses the extremely large interconnected network of neurons for information processing and to model the world around us

Artificial Neuron

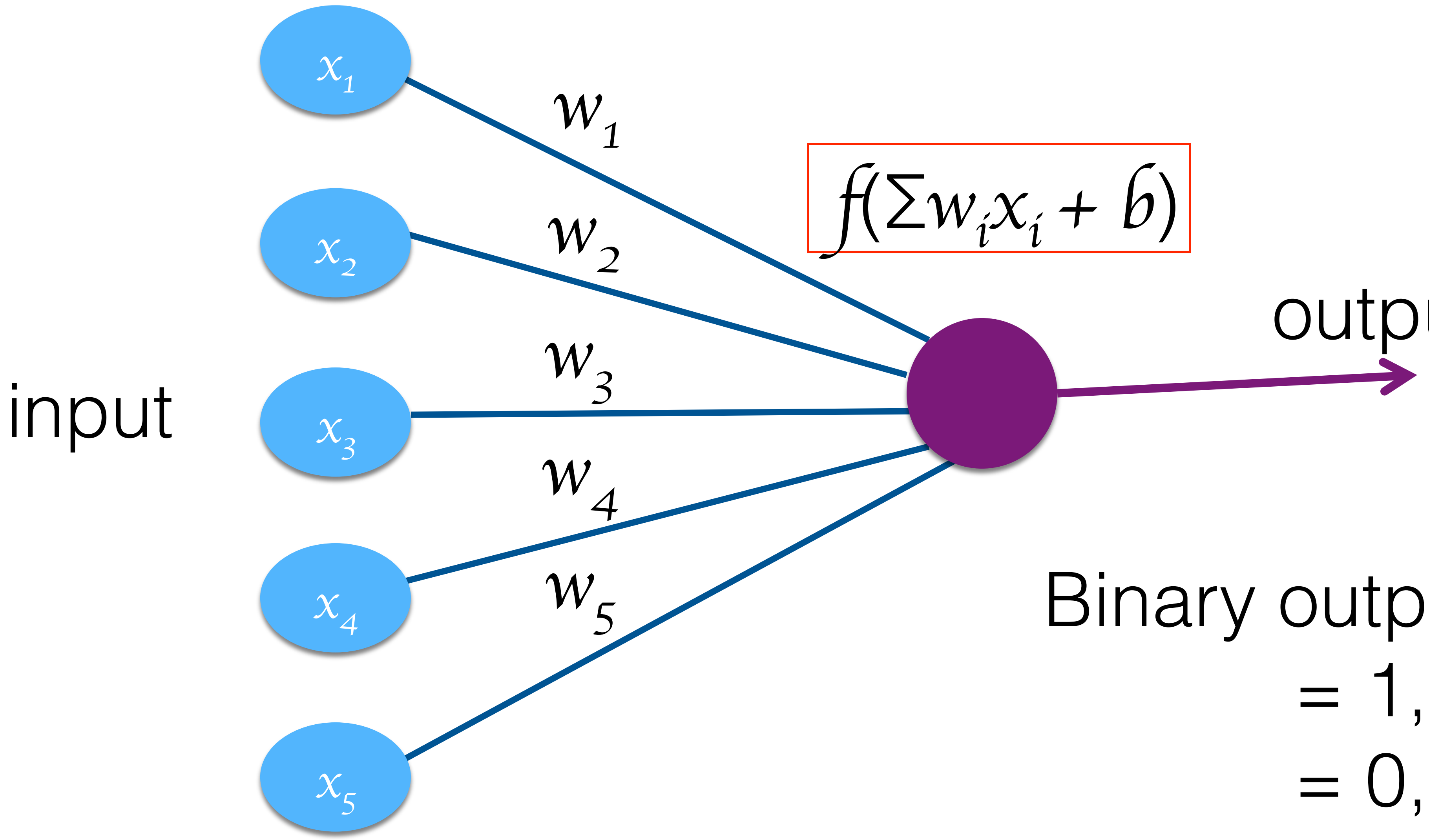
Model of an artificial neuron

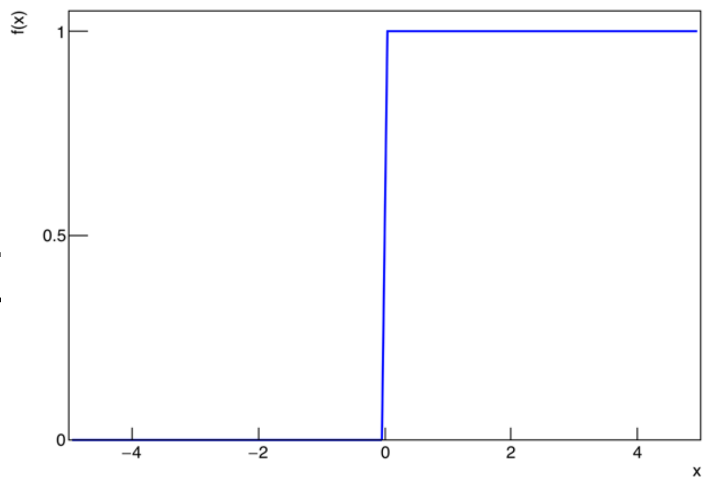


"f" is called "*activation function*"

b is a bias term → Can be represented by a node with input "1".

Perceptron



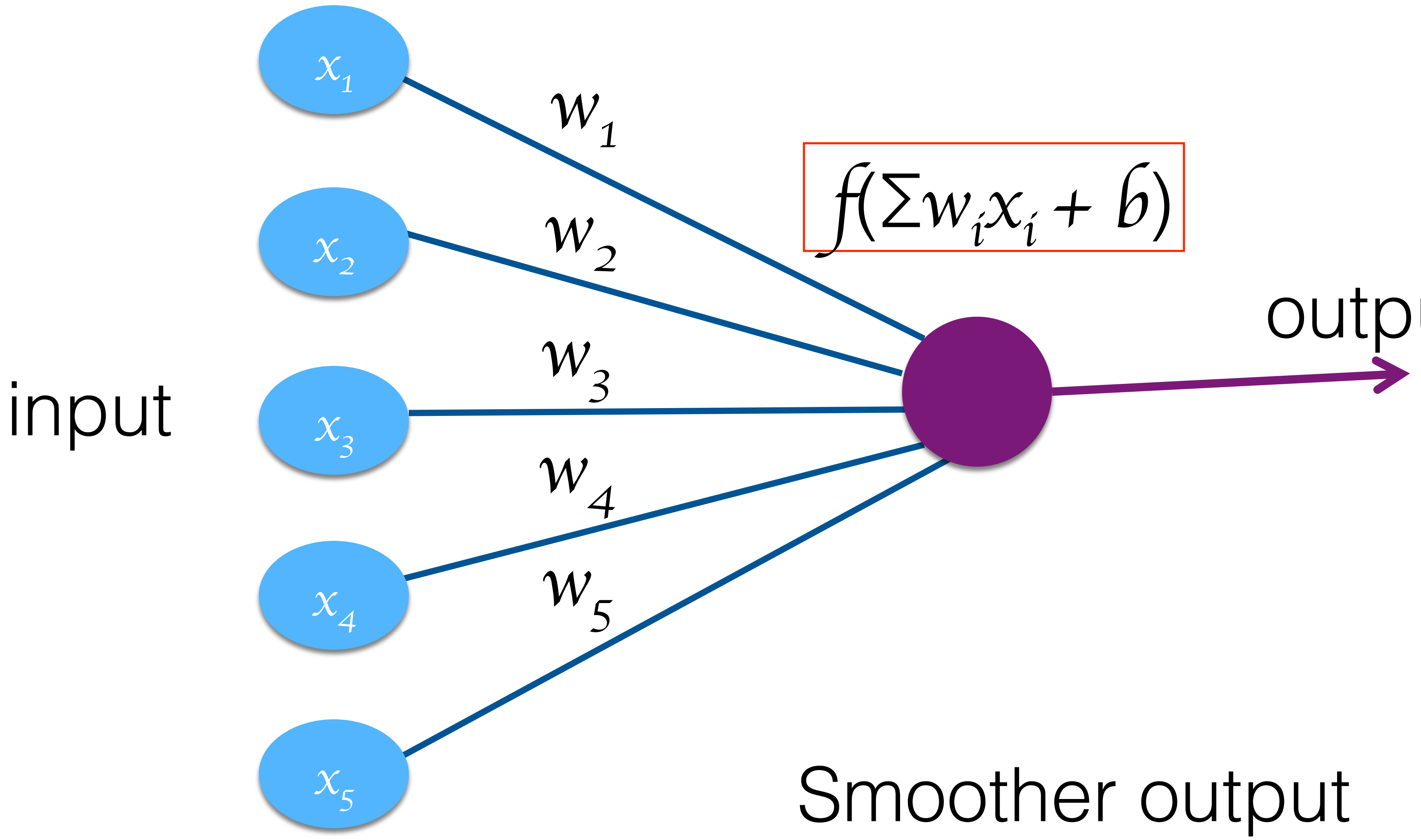


Binary output:

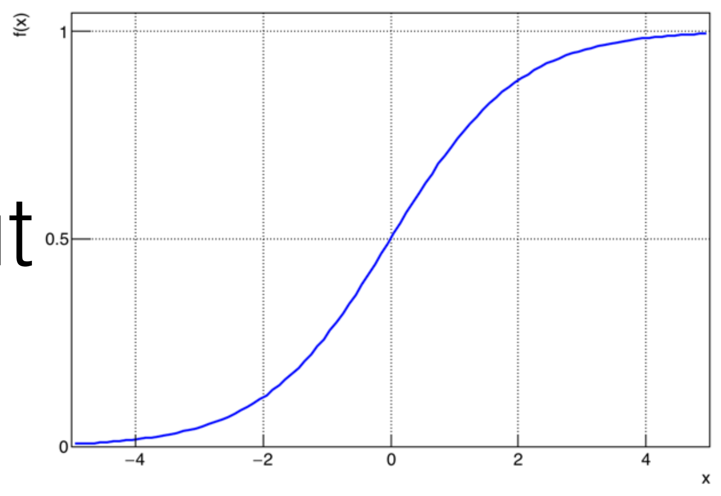
$= 1$, if $\Sigma w_i x_i + b > 0$

$= 0$, if $\Sigma w_i x_i + b < 0$

Sigmoid Neuron



$$f(x) = \frac{1}{1+e^{-x}}$$



Smoother output

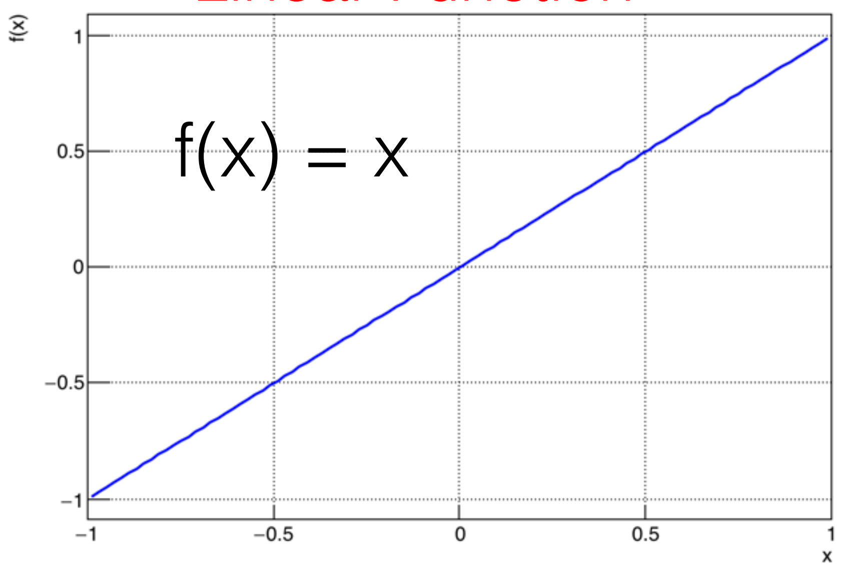
Small change in weight => small corresponding change in output

This property makes the learning possible

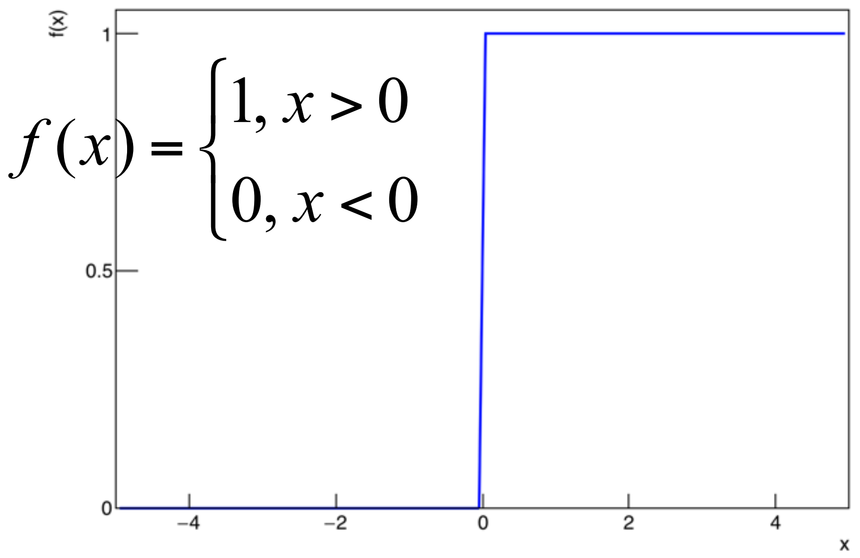
Activation Functions

It determines at what threshold the neuron will fire
OR the frequency at which a neuron fires

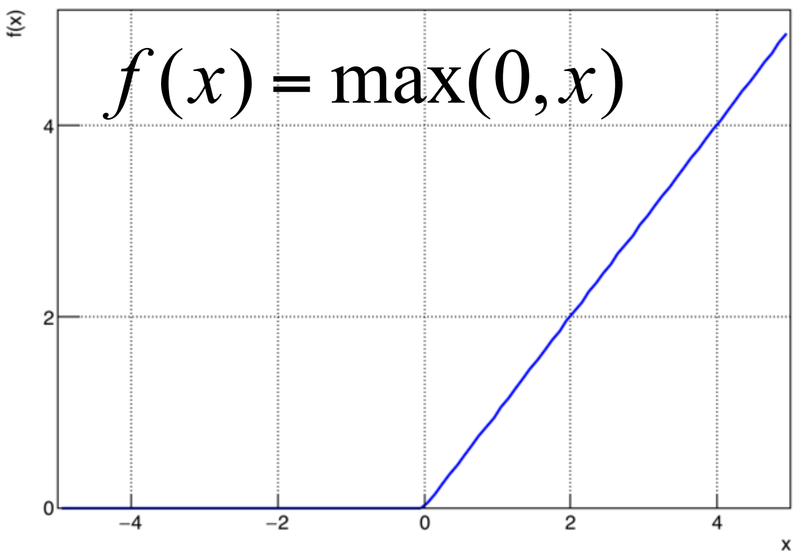
Linear Function



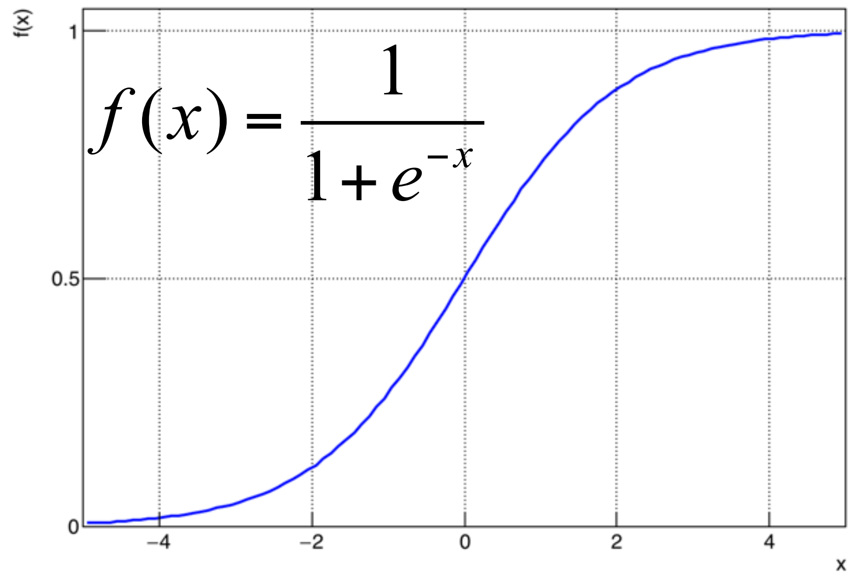
Step Function



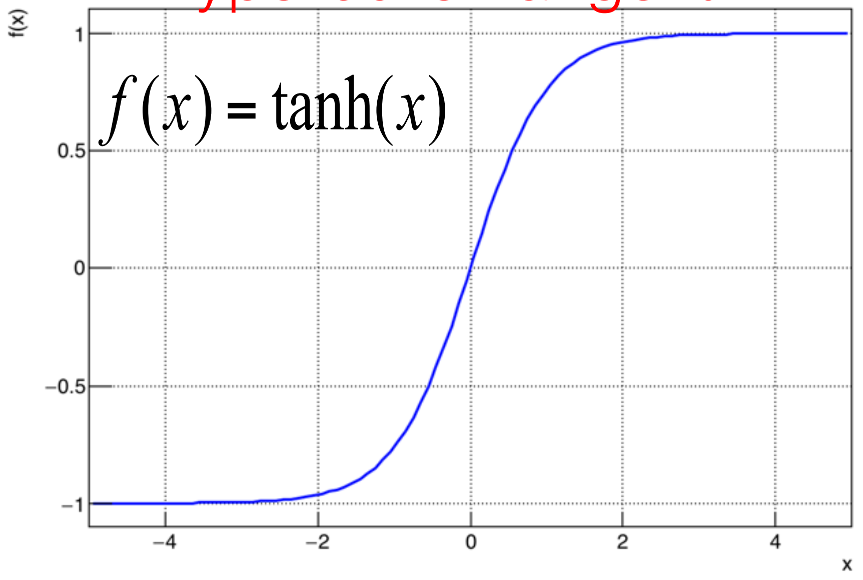
Rectified Linear Units (ReLU)



Sigmoid Function



Hyperbolic Tangent

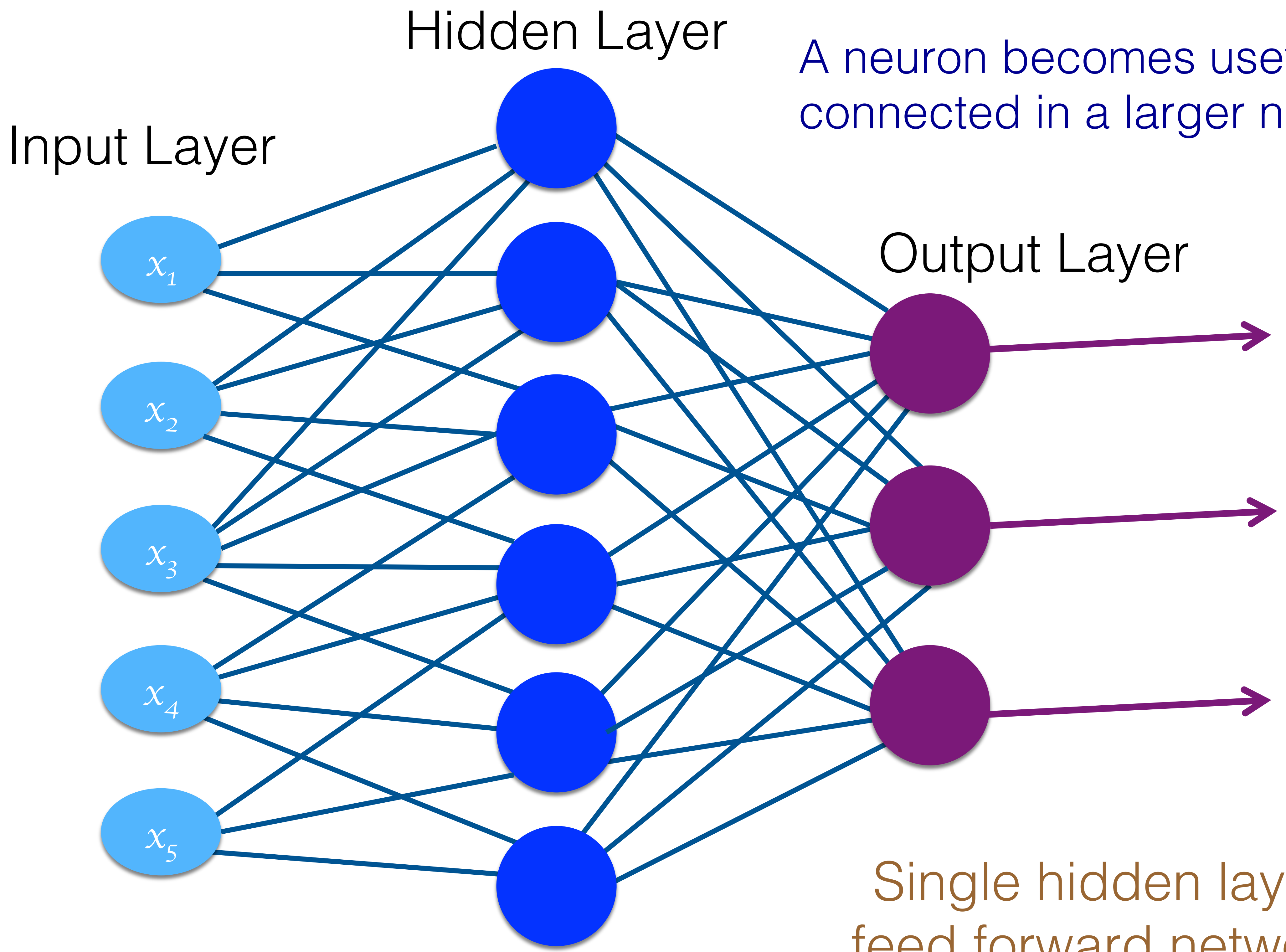


Softmax

$$f(x_i) = \frac{e^{x_i}}{\sum e^{x_i}}$$

(used in output layer of a multiclassification network)

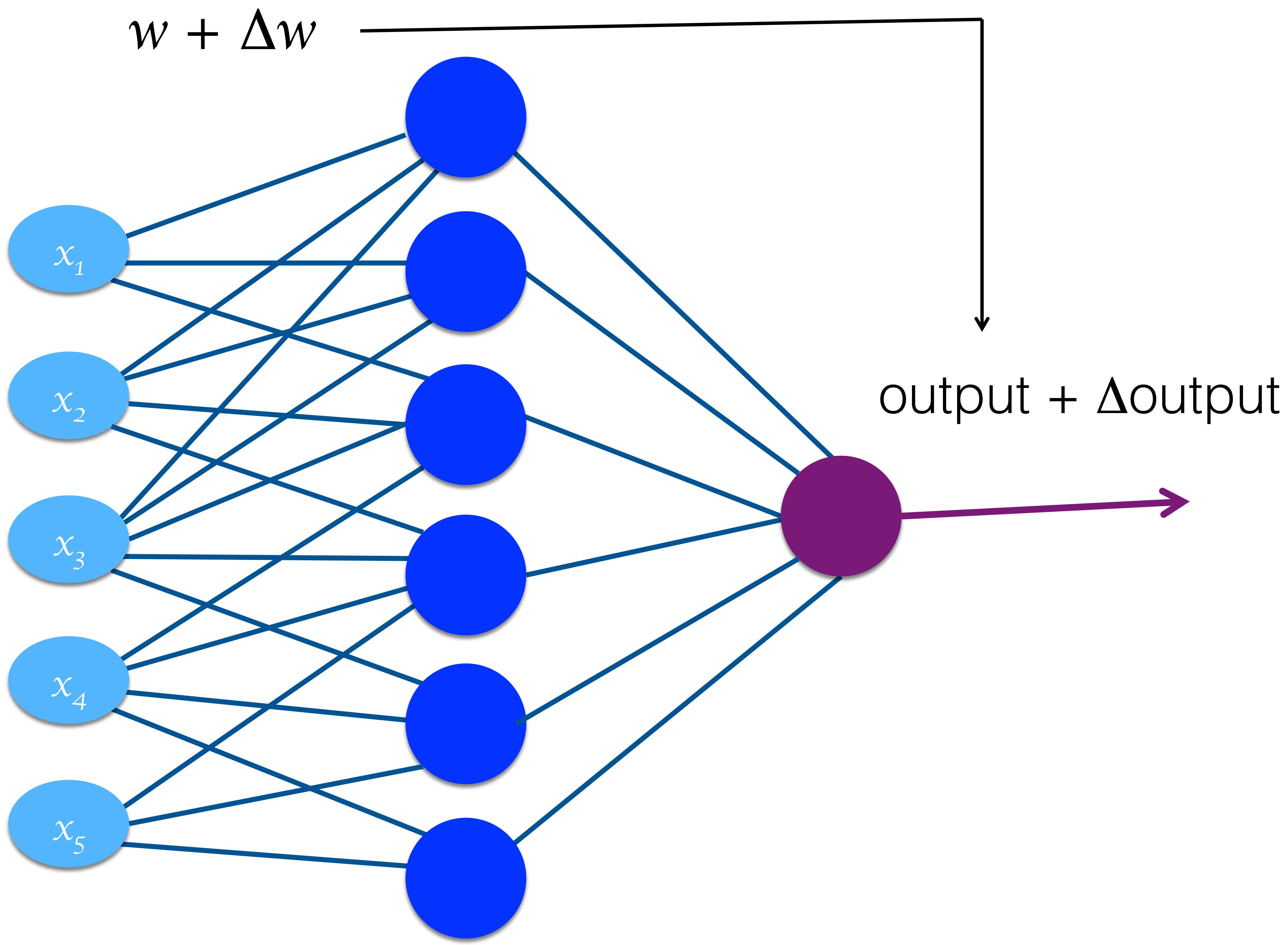
Artificial Neural Network



A neuron becomes useful when connected in a larger network

Single hidden layer feed forward network

How Does a Network Learn?



Loss Function

Loss : measure of misclassification

1. Mean Squared Error: $$L(w,b) = \frac{1}{2}\sum_{k=1}^{K}\sum_{i=1}^{N}\left(y_{ik} - \hat{y}_{ik}\right)^2$$

2. Cross Entropy: $$L(w,b) = -\sum_{k=1}^{K}\sum_{i=1}^{N} y_{ik} \log \hat{y}_{ik}$$

Where, index "i" is for events and "k" is for output nodes

Network Training => Minimizing Loss in an iterative way

Backpropagation

Propagating in the backward direction to update the weights

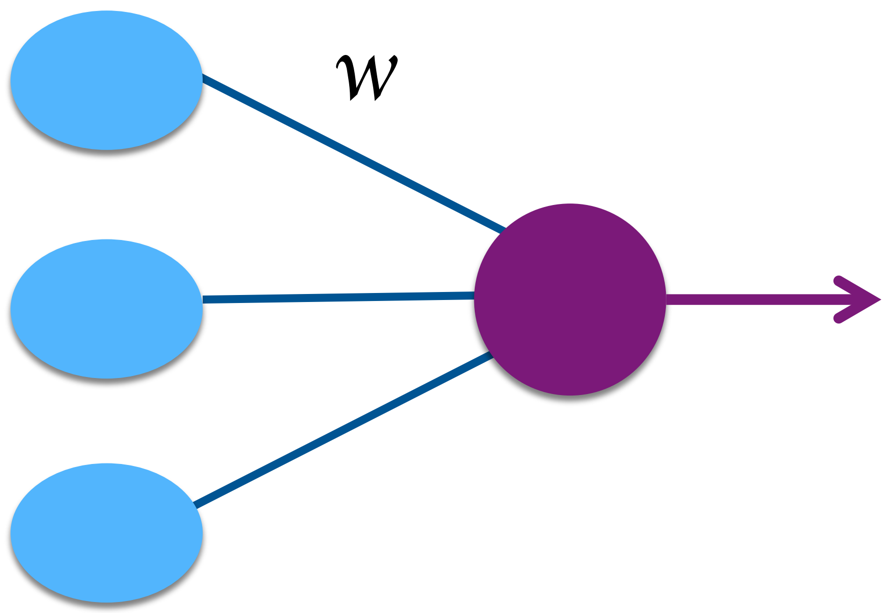
Steps:

1. Compare the computed output to actual output and determine loss
2. Determine in which direction to change each weight to reduce the loss
3. Determine the amount by which to change the weights
4. Apply correction to the weights
5. Repeat the procedure in each iteration till the loss is reduced to an accepted value

Backpropagation

Let's consider only one output node in the network and MSE loss function

$$L(w,b) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2$$



L is a function of weights and biases
→ A hypothetical surface in weight space

In every iteration, weights should be changed in a direction such that *L* is reduced (i.e. Δ*L* is –ve)

$$\Delta L \approx \sum \frac{\partial L}{\partial w_j}\Delta w_j = \nabla L \bullet \Delta w$$

Gradient descent

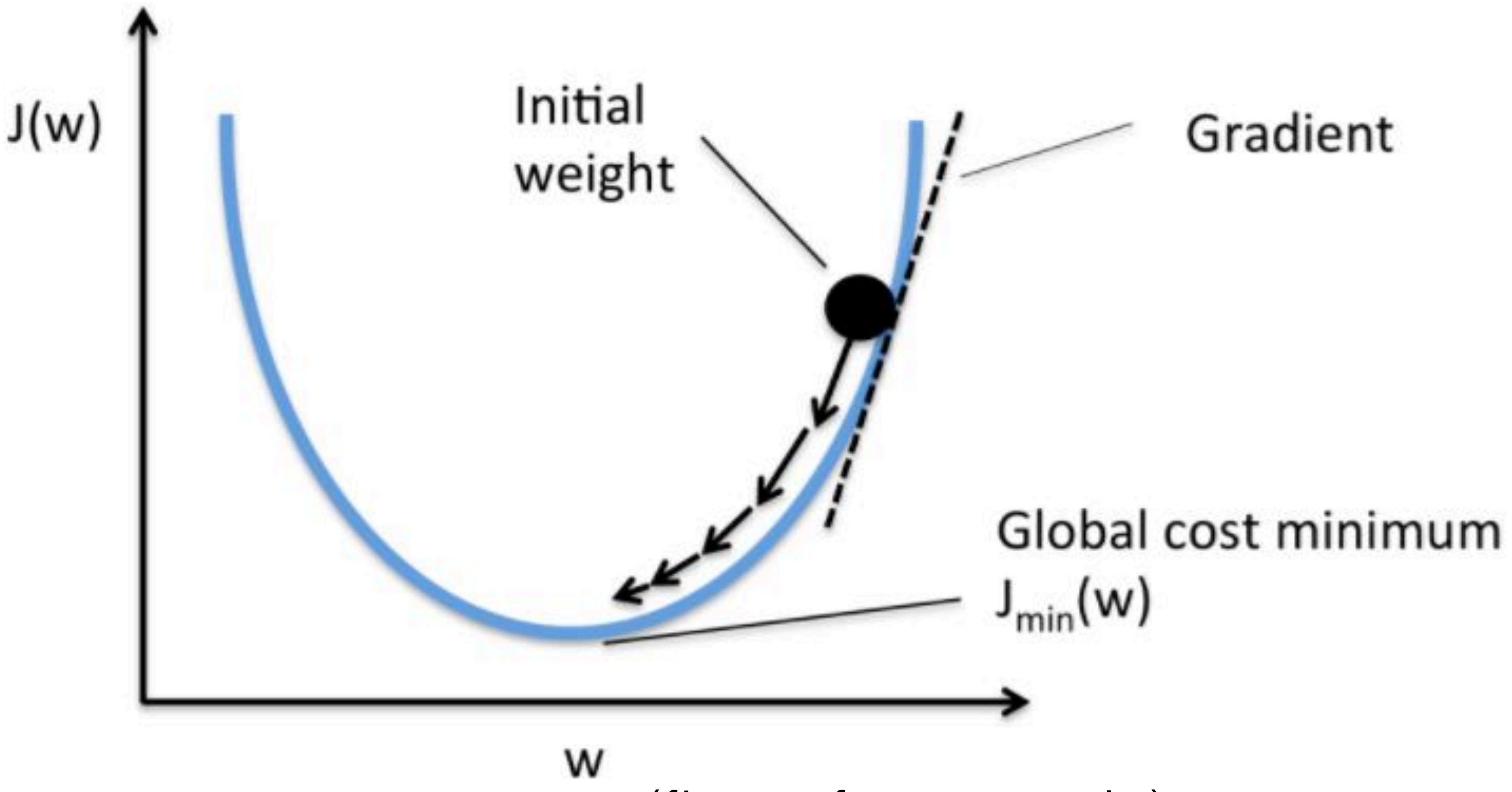
If we choose $\Delta w = -\eta \nabla L, \Rightarrow \Delta L \approx -\eta \,||\, \nabla L \,||^2 \Rightarrow \Delta L < 0$

Gradient descent

Starting from an arbitrary weight vector, the weights are updated after every iteration "t",

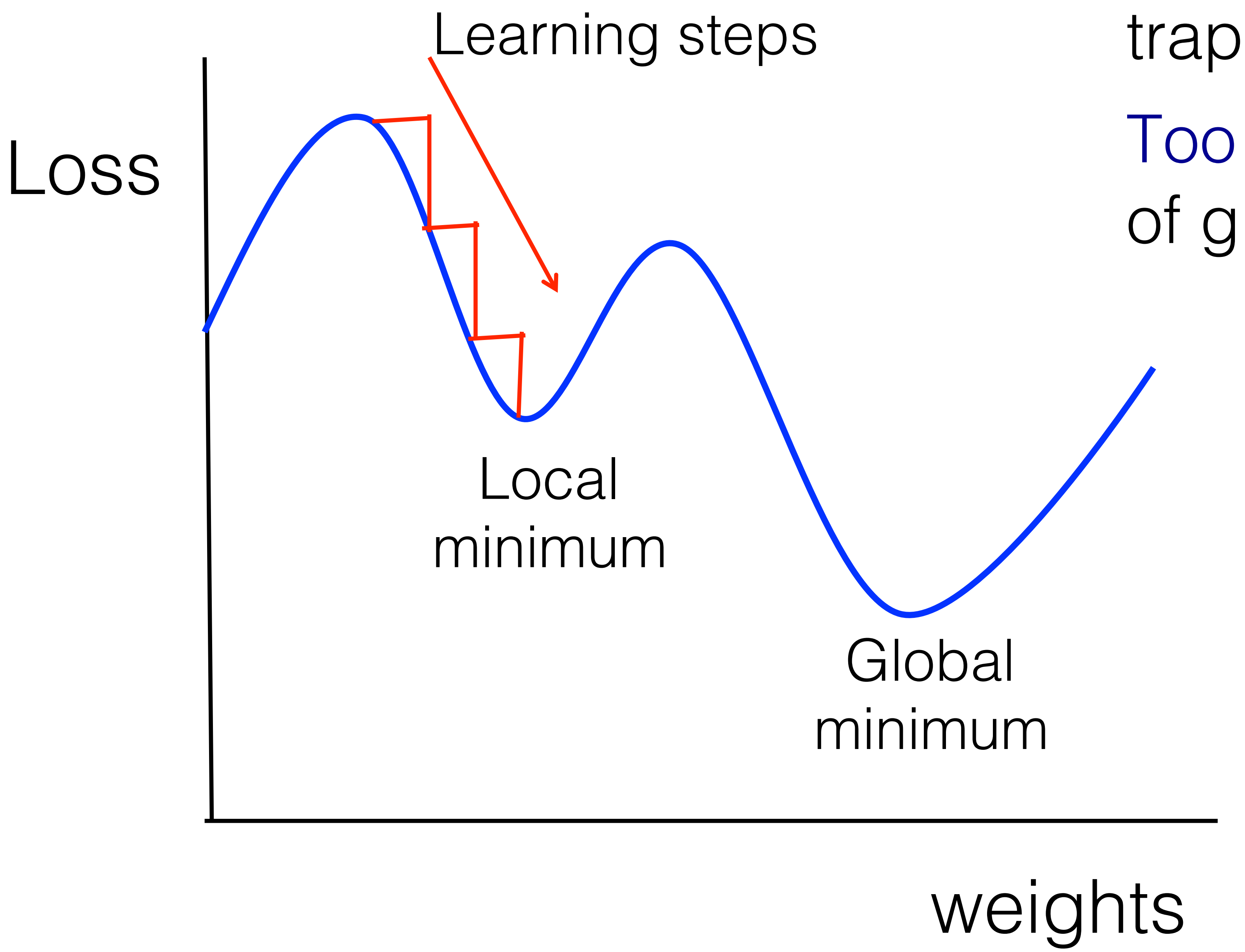
$$w_j(t+1) = w_j(t) - \eta \frac{\partial L}{\partial w_j}(t)$$

η is called the learning rate



(figure from google)

Learning rate



Too small → Can be trapped in a local minimum

Too Large → Can jump out of global minimum

Possible to update lr as learning proceeds

Weights of hidden layer

- The output values of hidden layer is not known, so we don’t know what should be the correct outputs
- But, the total error is related to the output values on the hidden layer.
- Thus, weights of the hidden layer are also updated in the same way as that of the output layers

Derivatives of Loss function

For a network with only one output node and one hidden layer, and considering MSE loss function

The net input to the jth hidden unit is $S_j = \sum_l w_{jl} x_l$ (ignoring bias)

ℓ runs over input connections

The output of this node is $I_j = f(S_j)$

The equations for output node are: $S = \sum_j w_j I_j$ and $y = f(S)$

j runs over hidden layer nodes

Thus, for the weights of the output node:

$$\frac{\partial L}{\partial w_j} = \sum_{i=1}^{N} (y - \hat{y}) \frac{\partial f}{\partial S} \frac{\partial S}{\partial w_j} = \sum_{i=1}^{N} (y - \hat{y}) \frac{\partial f}{\partial S} I_j$$

Derivatives of Loss function

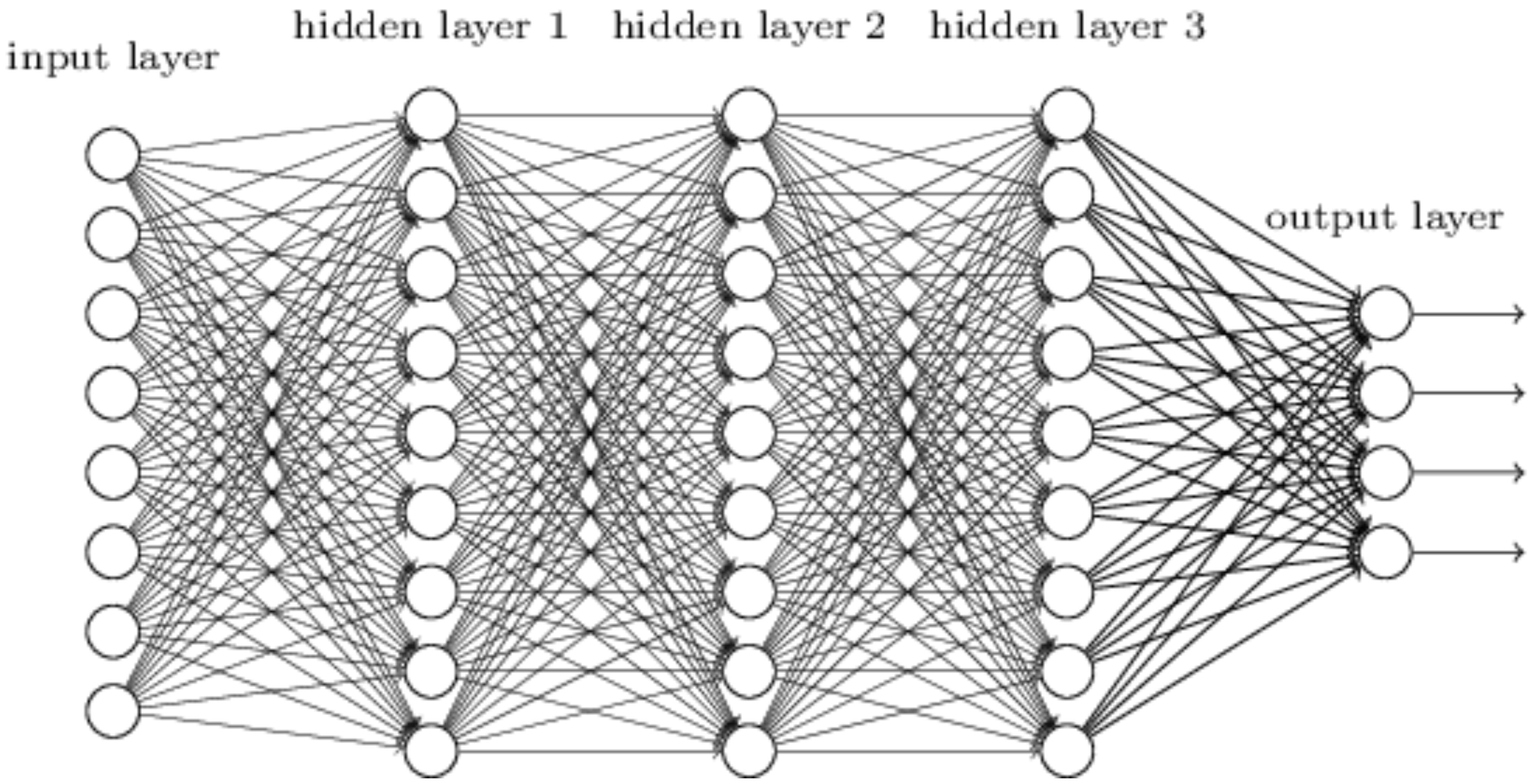
For the weights of the hidden layer node:

$$\frac{\partial L}{\partial w_{jl}} = \sum_{i=1}^{N}(y-\hat{y})\frac{\partial f(S)}{\partial S}\frac{\partial S}{\partial I_j}\frac{\partial f(S_j)}{\partial S_j}\frac{\partial S_j}{\partial w_{jl}} = \sum_{i=1}^{N}(y-\hat{y})\frac{\partial f(S)}{\partial S}w_j\frac{\partial f(S_j)}{\partial S_j}x_l$$

The activation functions "*f*" need to be differentiable

Deep Neural Network

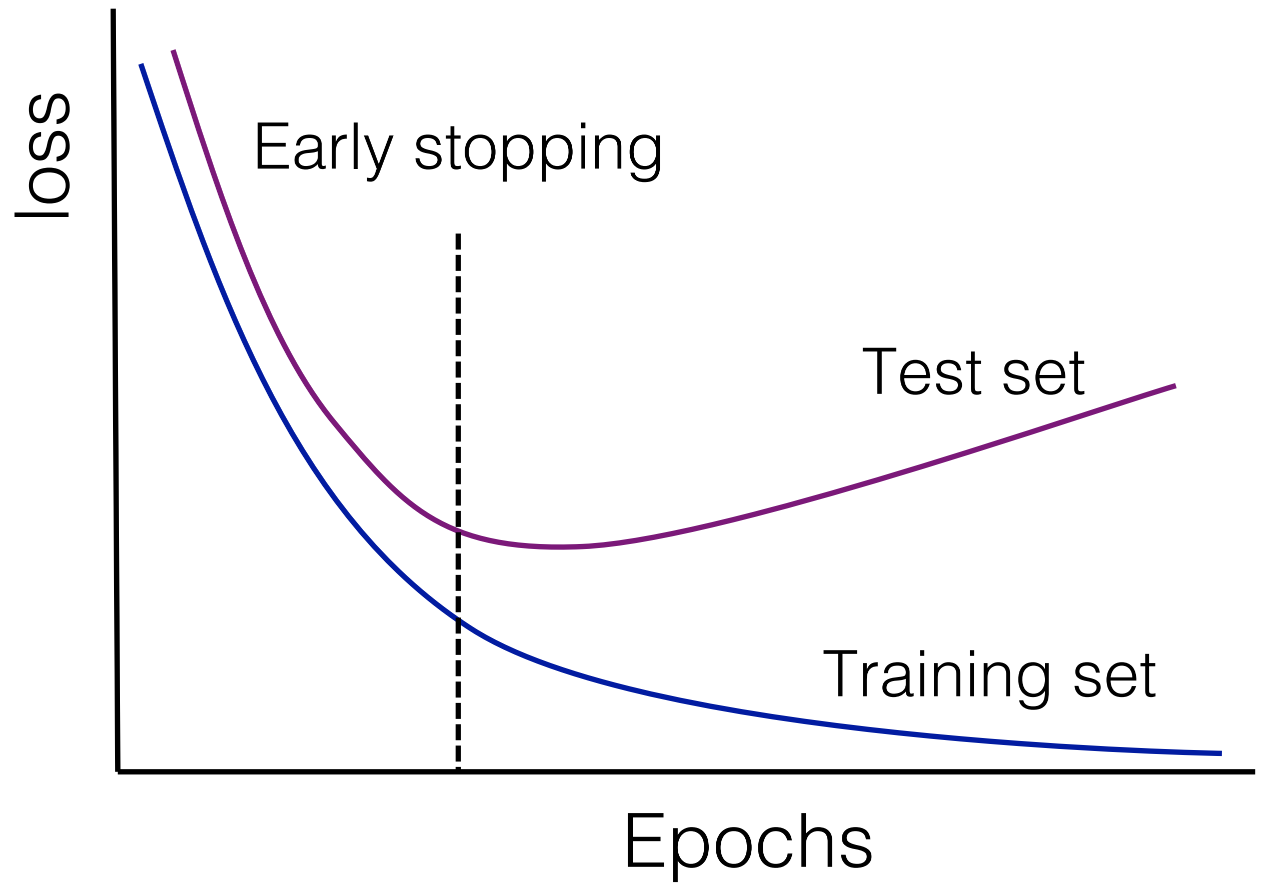
- Networks with many-layer structure - two or more hidden layers
- Deep learning techniques are based on stochastic gradient descent and backpropagation, but also introduce new ideas
- Deep nets have ability to build up a complex hierarchy of concepts



Overtraining

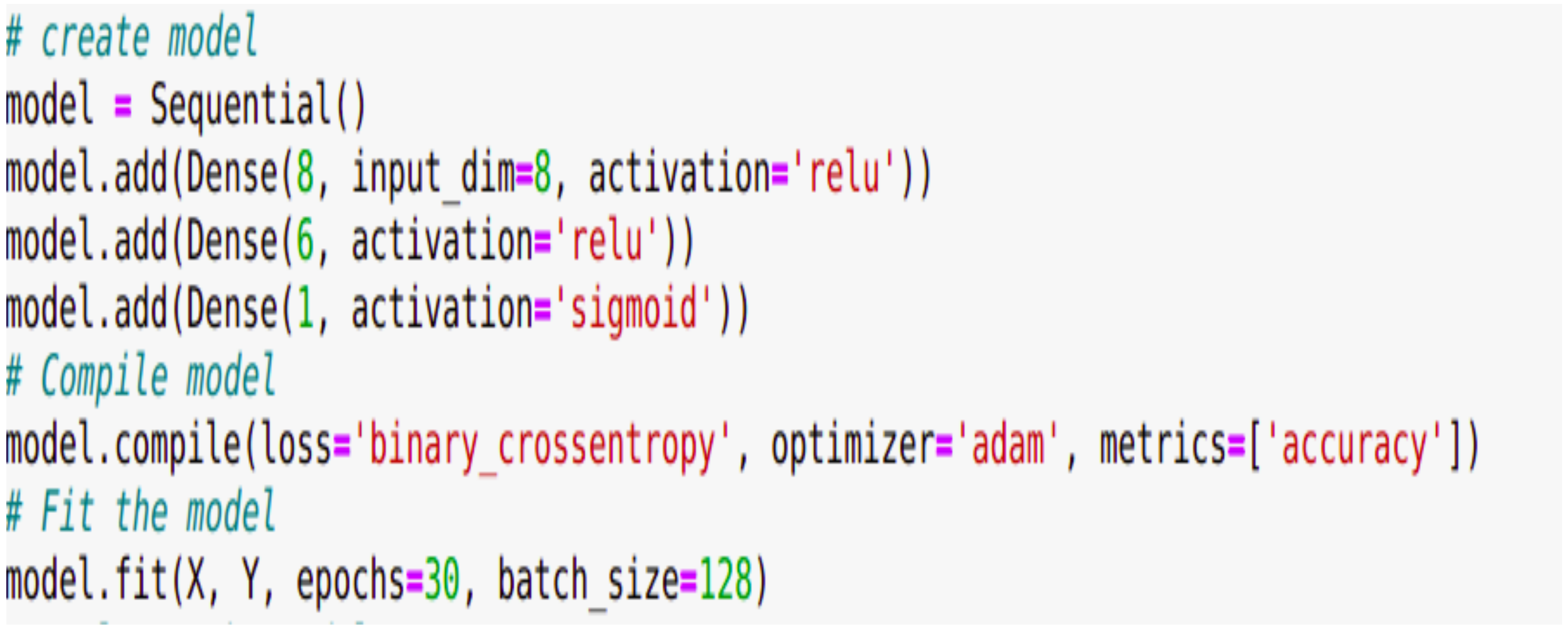
Overtraining is a situation where a network learns to predict the training examples with very high accuracy but cannot generalize to new data.

- Leads to poor performance in other samples
- Mostly due to small training sample size, or data that is too homogenous
- Over sensitive to some features of the training data



Practical information

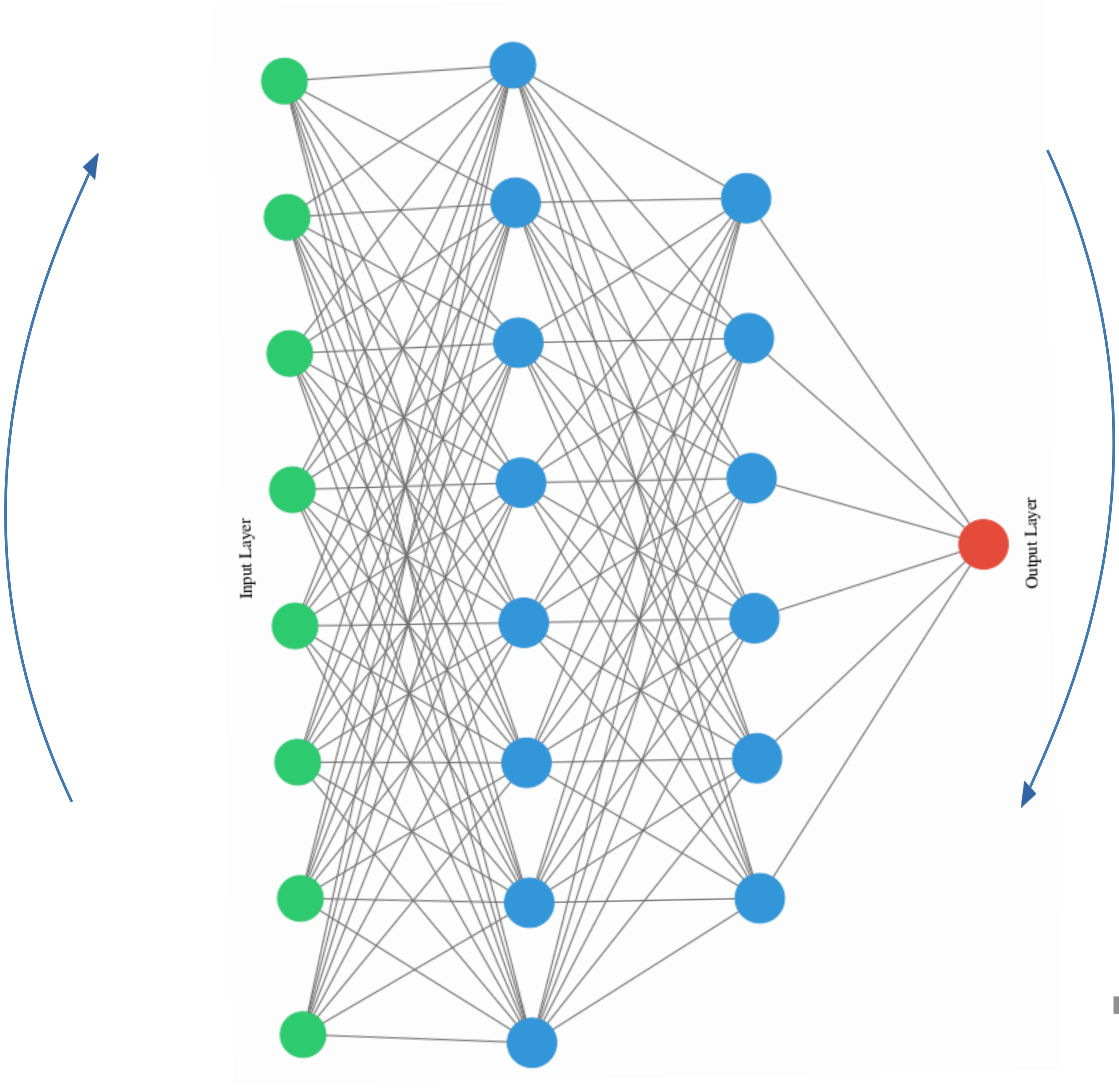
- To construct a neural network model you need to specify the following:
- The number of hidden layers and neurons in each layer.
- Activation function(eg Relu or tanh),Cost function(here cross entropy)
- Batch size,learning rate(here 'adam' optimises the learning rate for Gradient decent)
- Number of epocs, i.e. the number of training cycles.



```python
# create model
model = Sequential()
model.add(Dense(8, input_dim=8, activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=30, batch_size=128)
```

Lets see what this model looks like(next slide)

How this model looks like

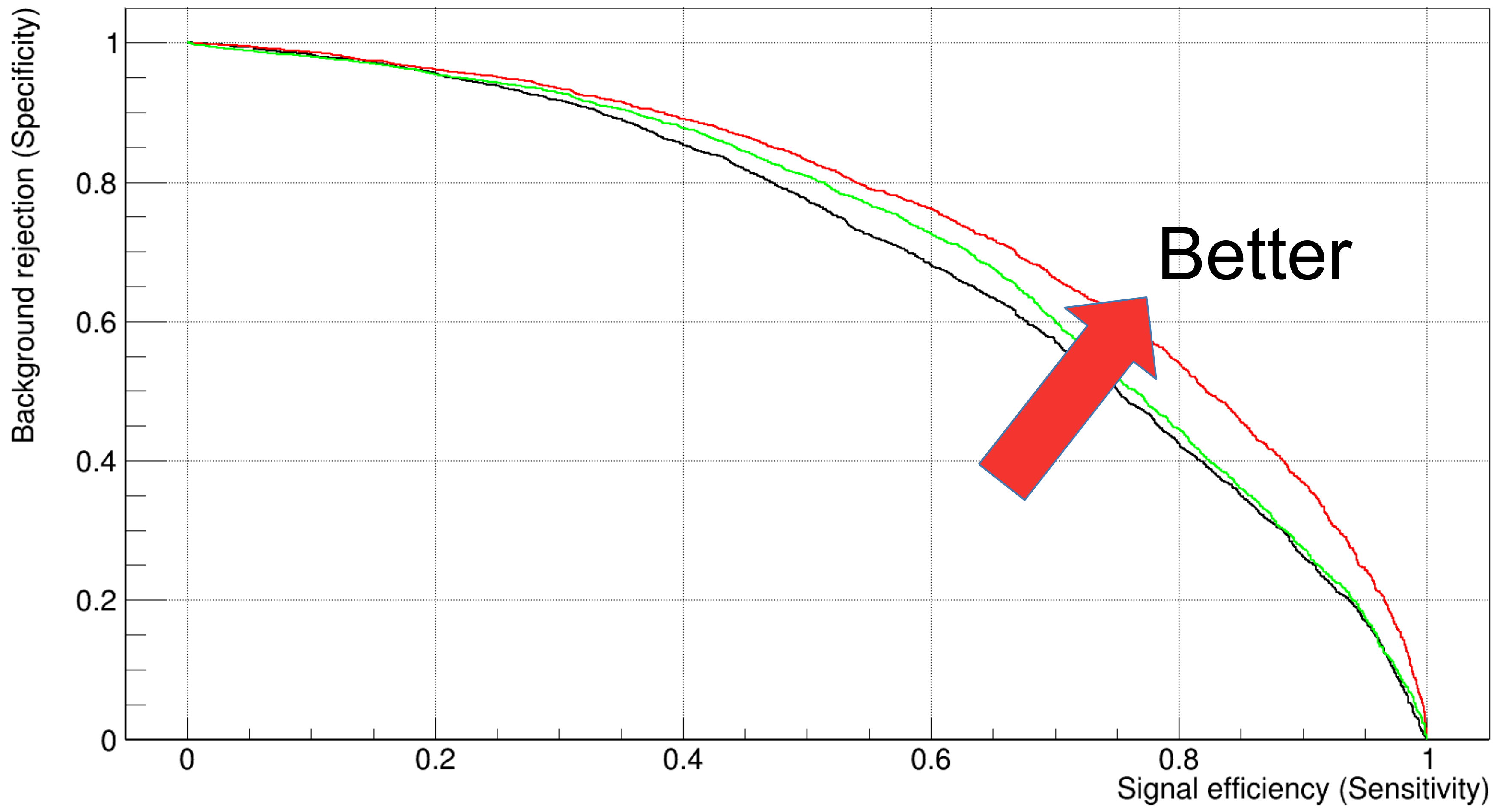


Calculate the output and error

Repeat for N=30 epoch cycles

ROC Curves

- ROC (**R**eceiver **O**perating **C**haracteristic) Curves are a good way to illustrate the performance of given classifier
- Shows the background rejection over the signal efficiency of the remaining sample
- Best classifier can be identified by the largest AUC (Area under curve)

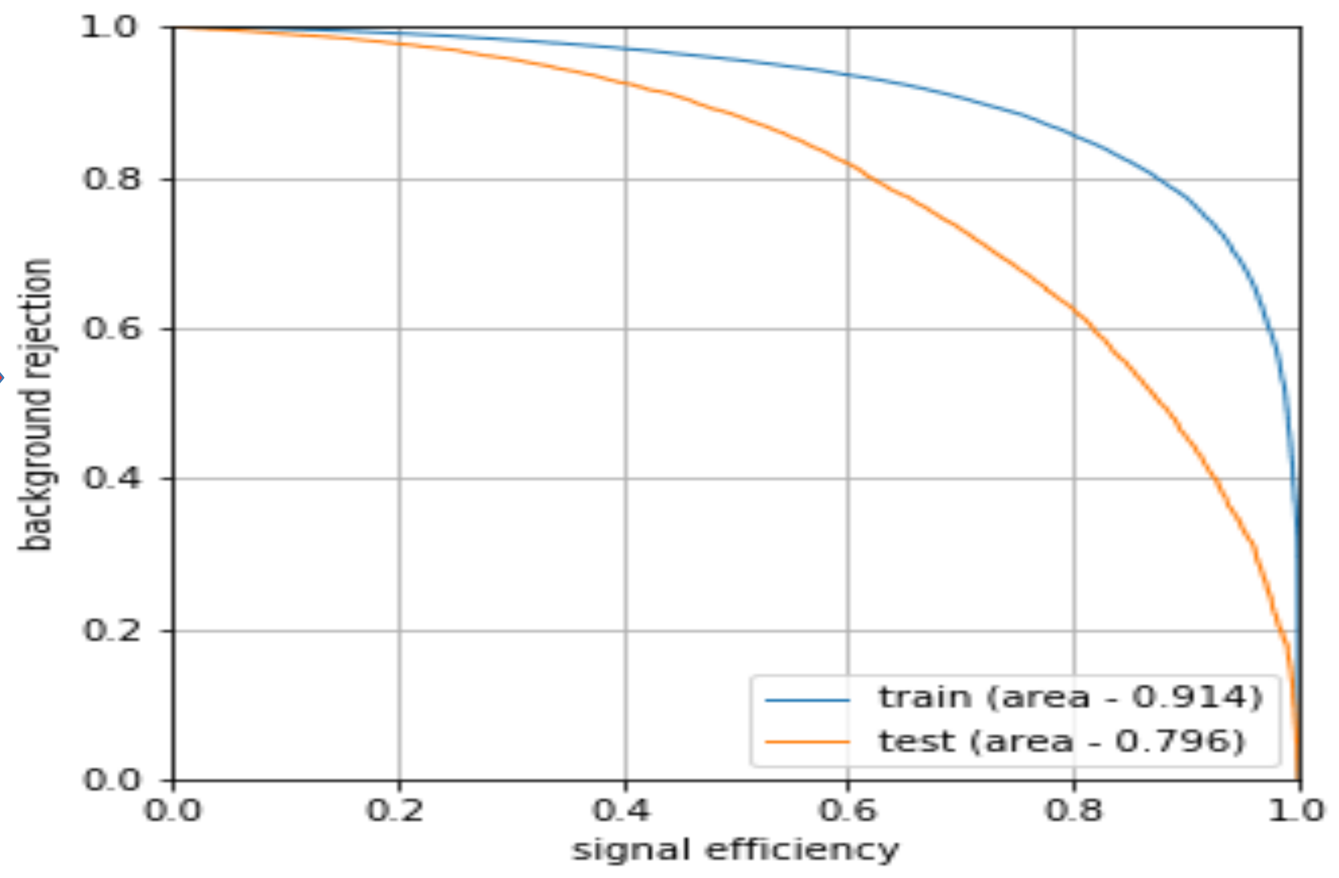


Overtraining

Overtraining is a situation where a network learns to predict the training examples with very high accuracy but cannot generalize to new data.

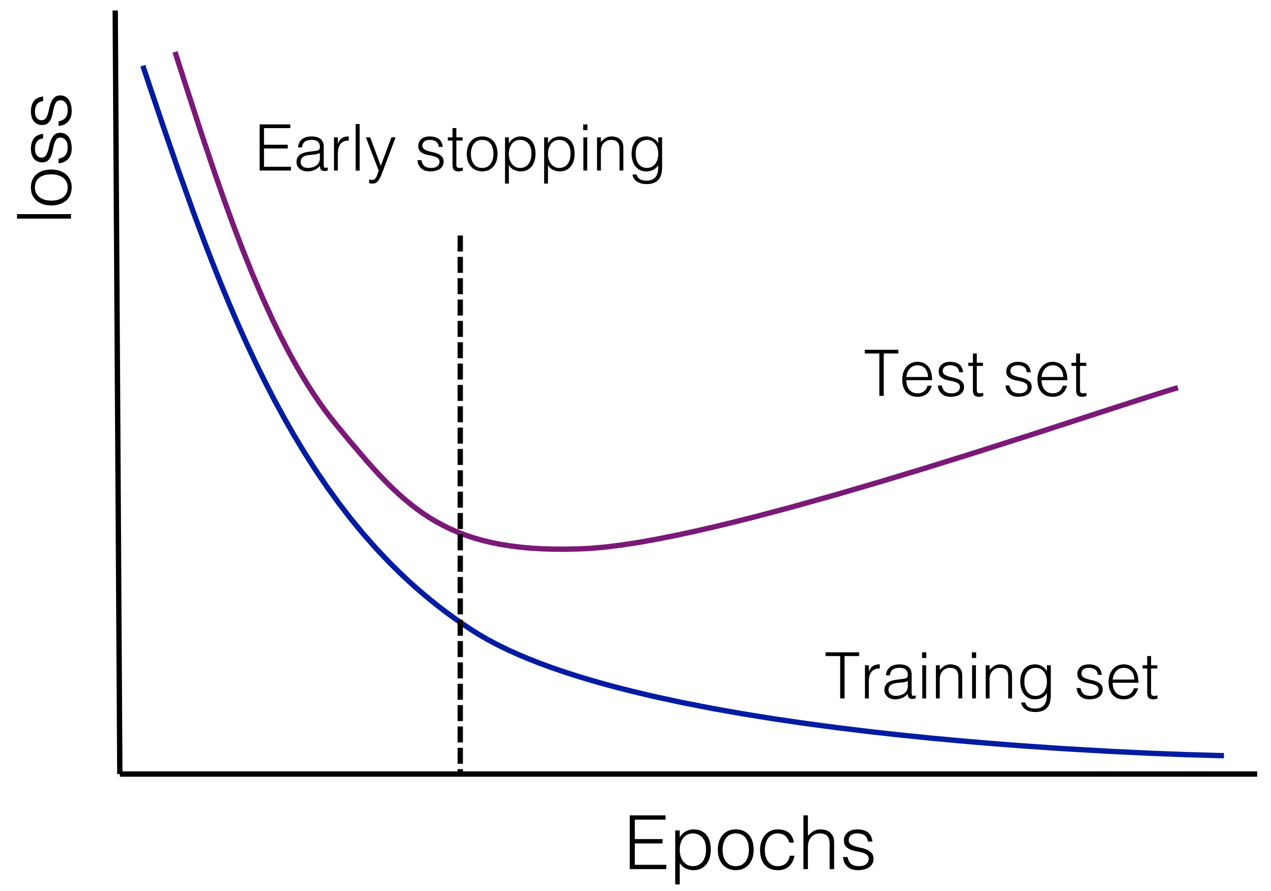
- Leads to poor performance in other samples
- Mostly due to small training sample size, or data that is too homogenous
- Over sensitive to some features of the training data

Large difference in performance of training and testing dataset usually is an indicator of overtraining



How to deal with Overtraining

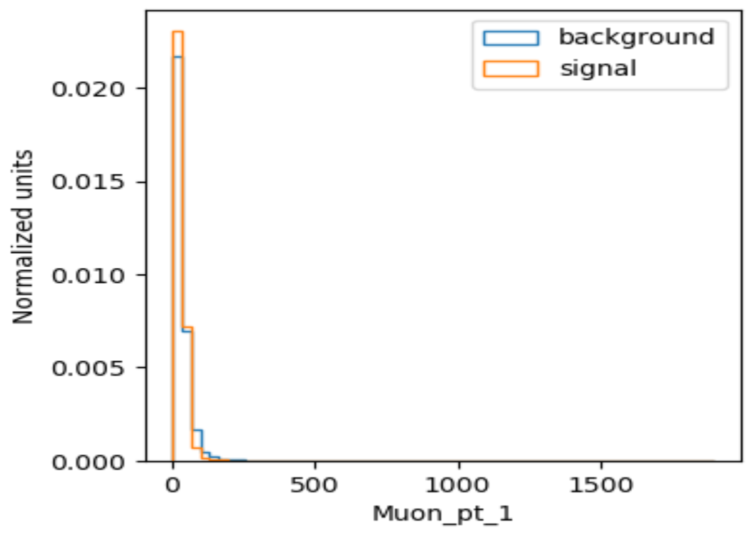
- If you think the model is overtrained,you can reduce the complexity of the the model ie reduce the number of hidden layers/neurons.
- Increase the training dataset.This will give the classifier more data to learn the correct model.
- You can use a feature called soft drop that randomly drops a predefined set of neurons during each epoch.
- In most neural network packages like Keras or DNN in TMVA, the training set is further divided into a training and validation set. The training and validation are done over many epocs and the model with the least validation error is selected.
- This procedure usually automatically takes care of overtraining.

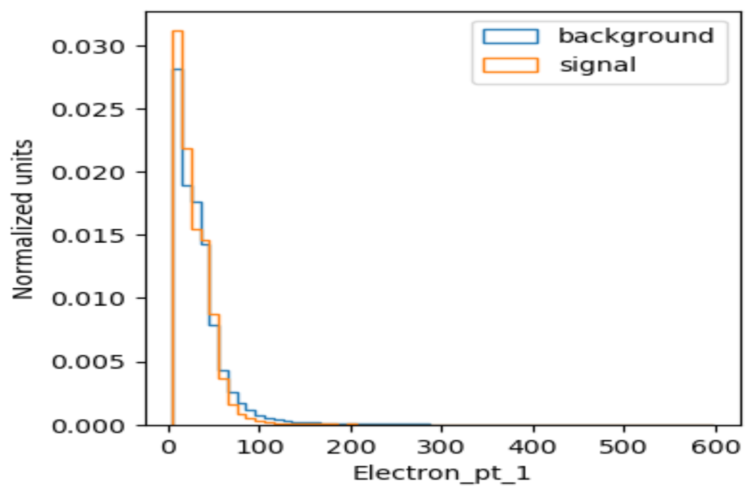


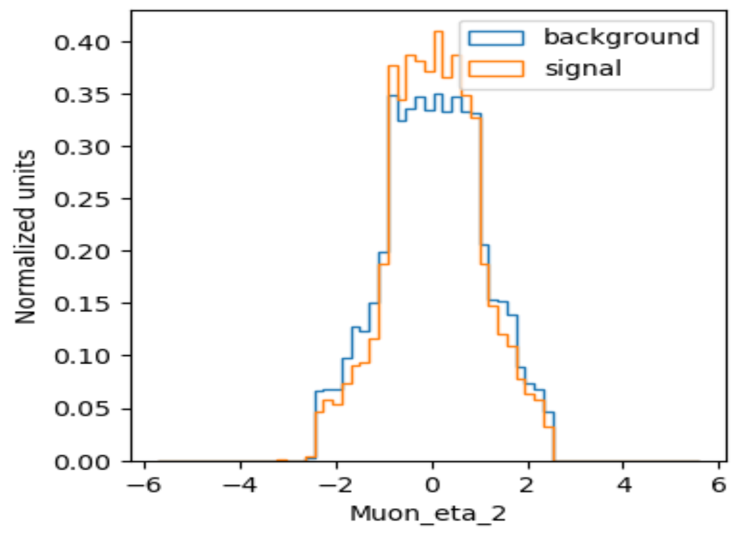
Example from Physics

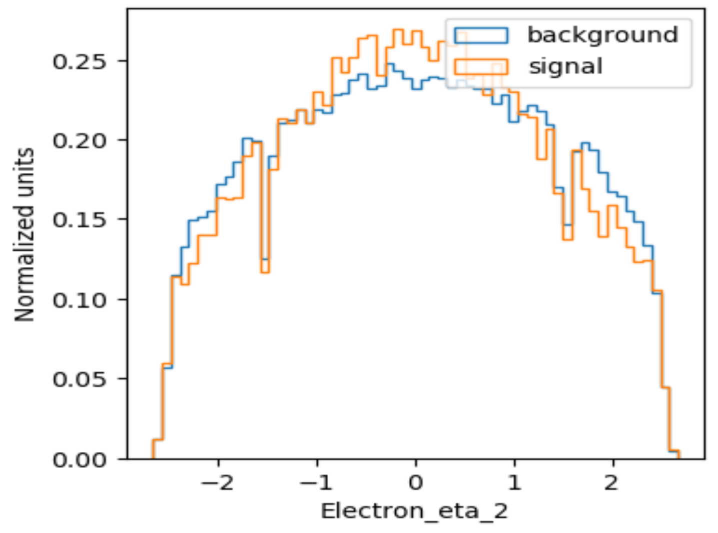
Separate $H \rightarrow ZZ^* \rightarrow 2e2\mu$ (signal*) from $ZZ \rightarrow 2e2\mu$ (background*)

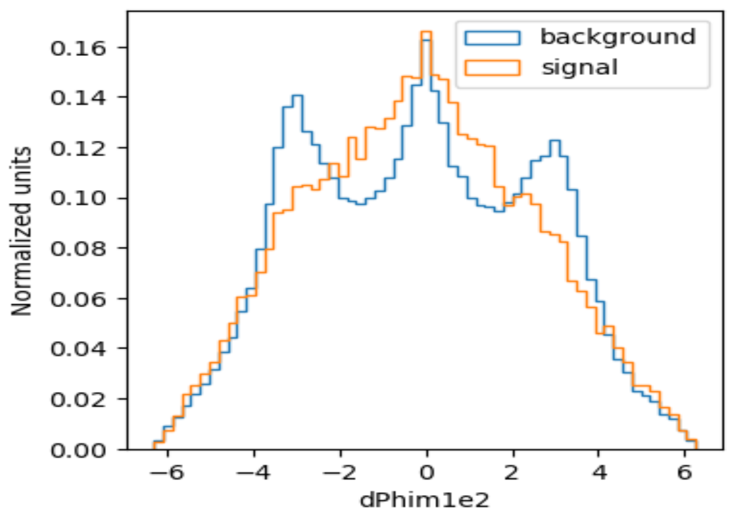
Input Variables used : p_T of individual leptons, Delta Phi combinations of each lepton pair, eta of each lepton(total 14 variables)

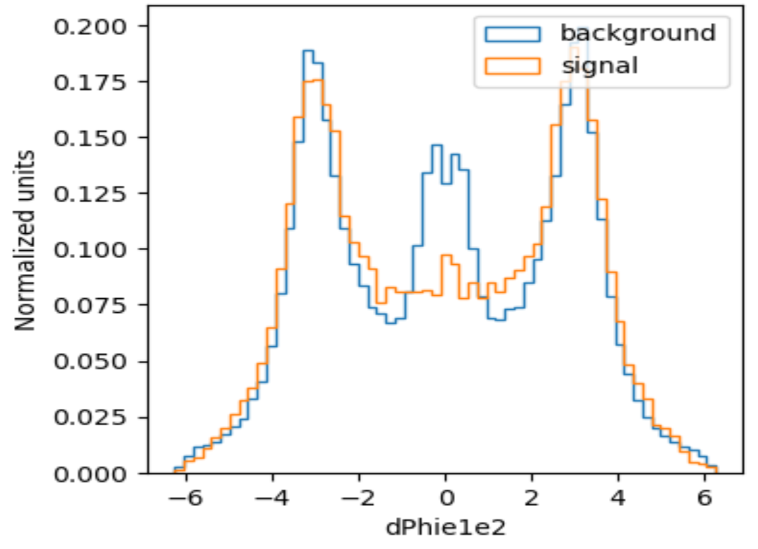








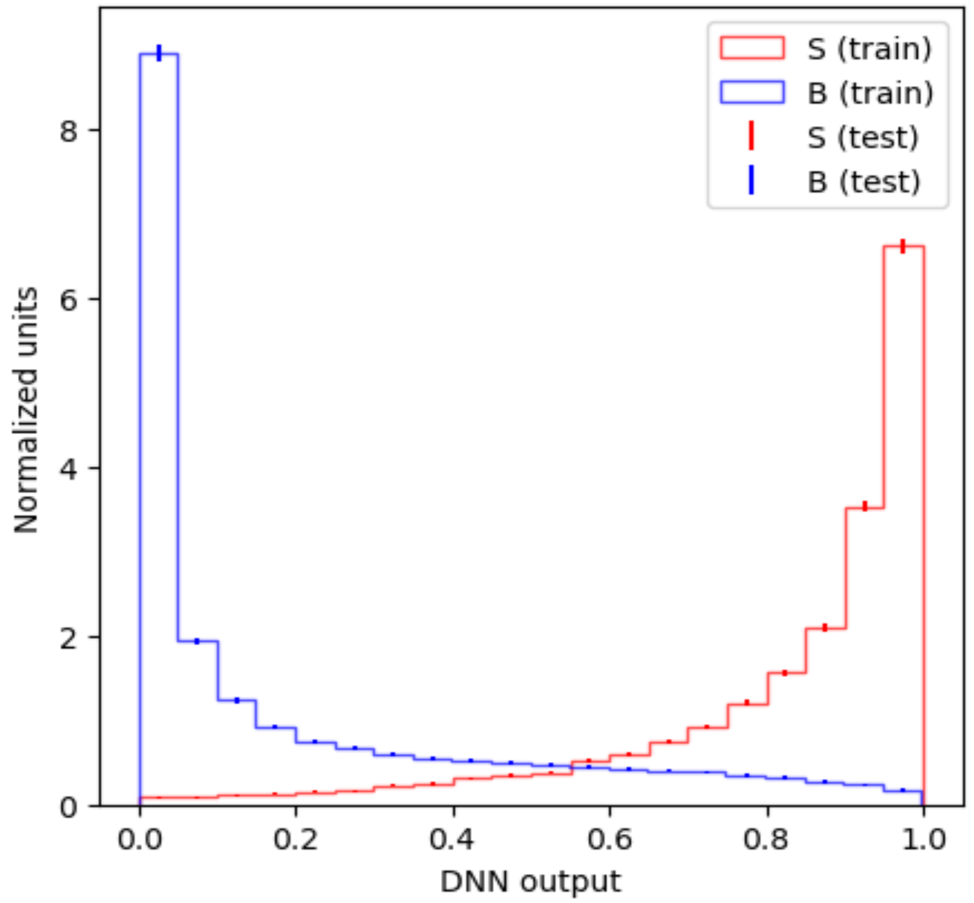


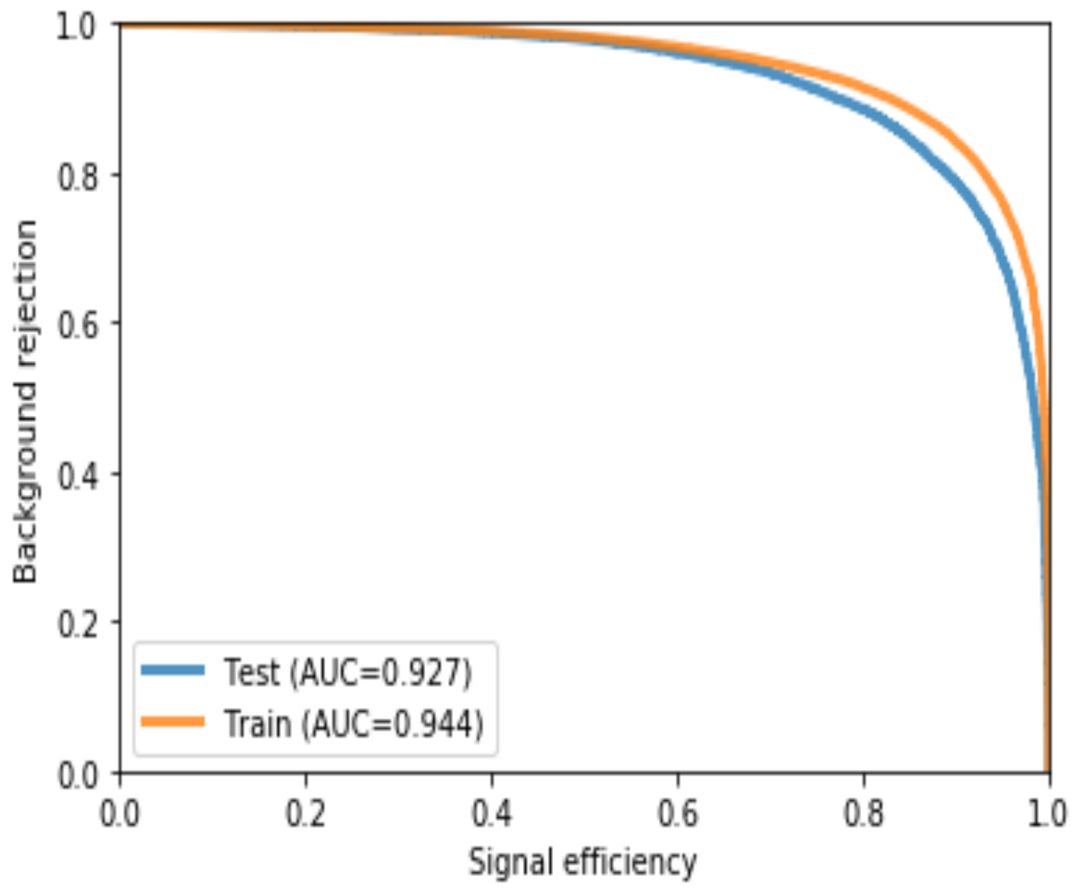


* Both signal and background mc samples taken from cms open data

Performance

- 5 Hidden layers with 100(relu) x 100 x 100 x 32 x 32 neurons
- Relu activation function
- Batch size 256 ,epochs = 30
- Early stopping with patience = 5



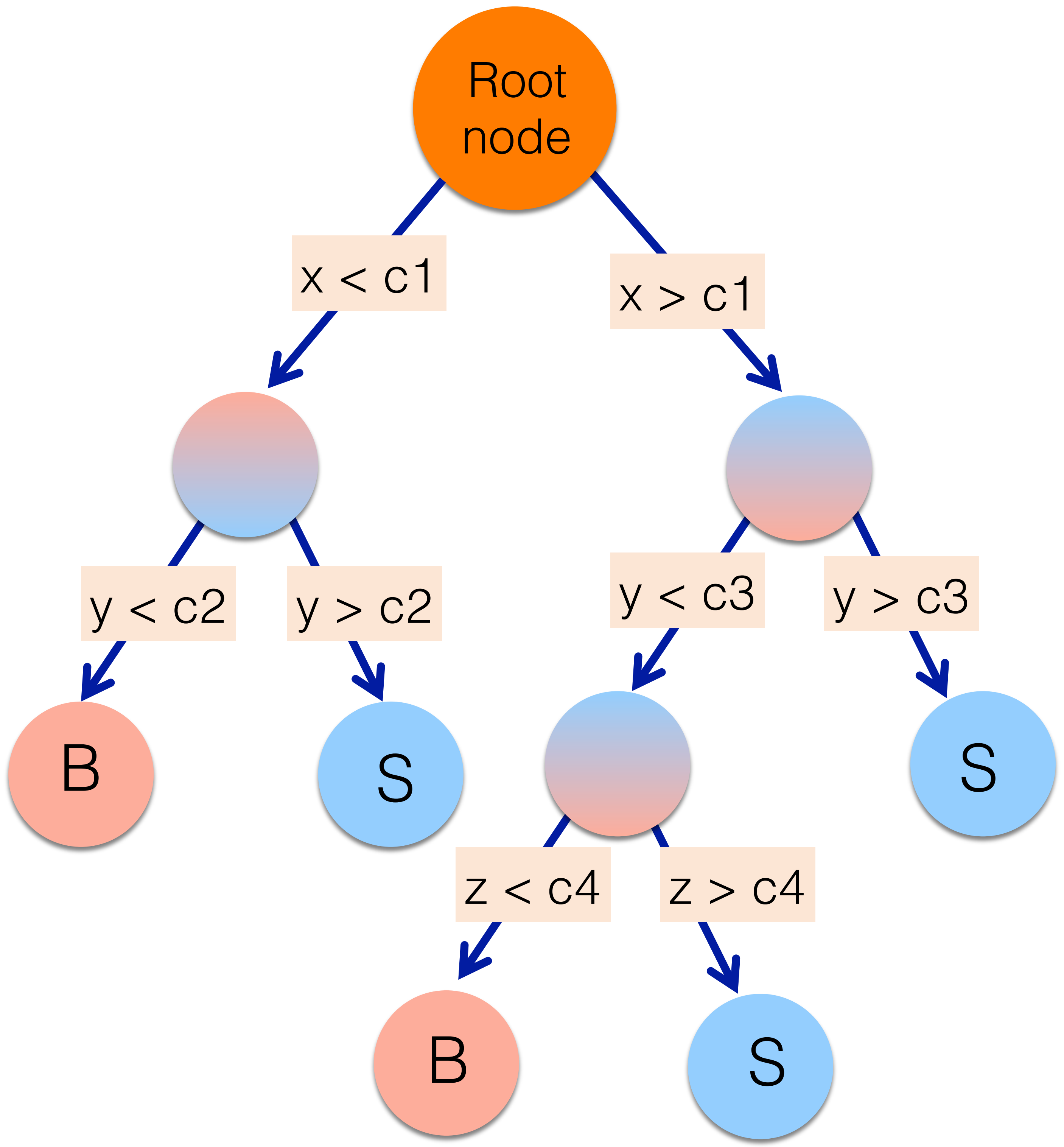


Will be discussed in more detail at the tutorial tomorrow

Boosted Decision Tree

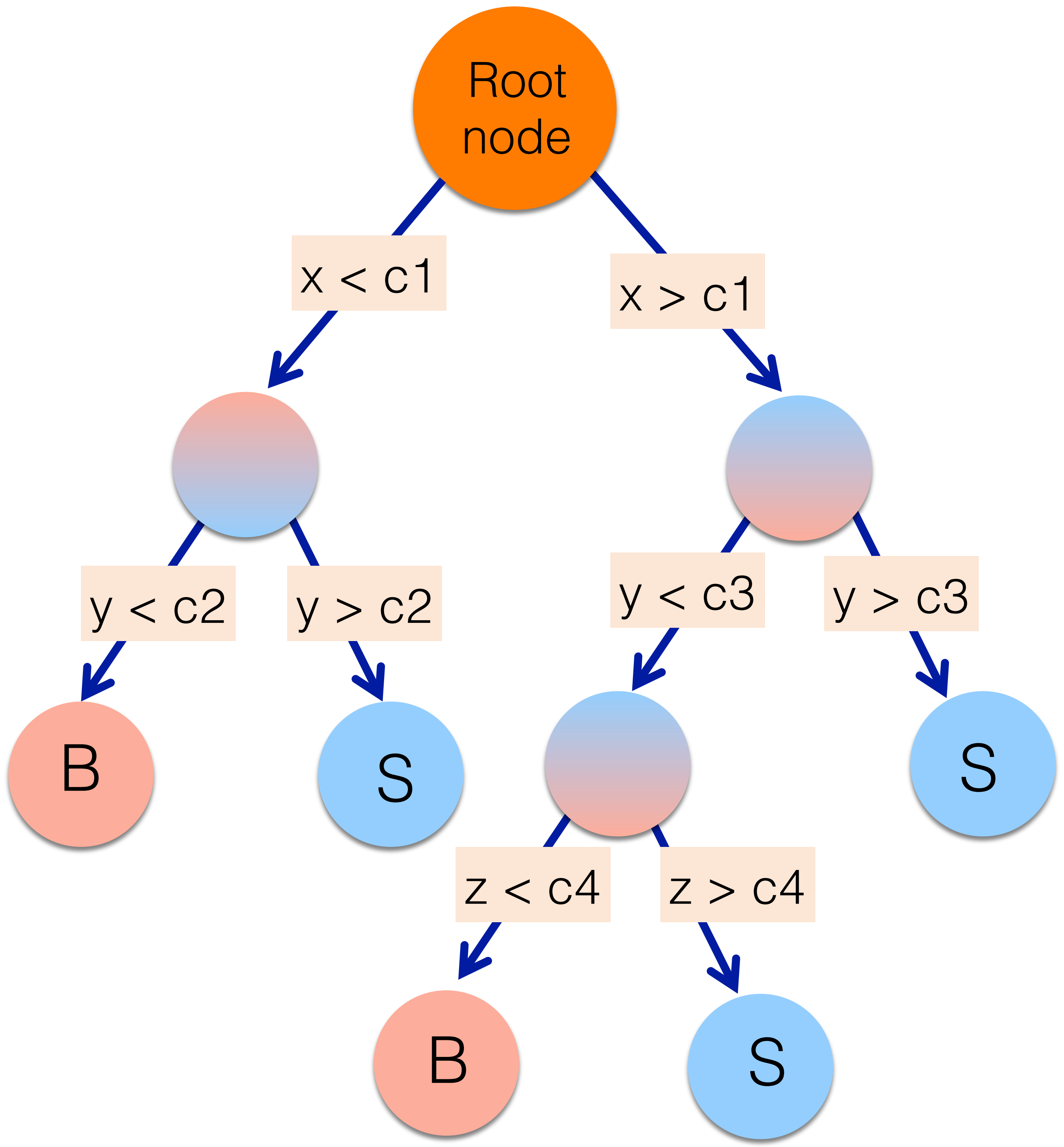
Decision Trees

- Sequential application of cuts splits the data into nodes
- Each cut depends on cuts in previous nodes
- Final nodes (leafs) classify an event as signal or background
- Final leafs are reached after a given maximum number of nodes
- Advantage:
  - Easy to understand/interpret
  - Training is fast
  - Good performance with multivariate data



Decision Trees

- Nodes:
  - The data is split based on a value of one of the input features at each code
- Leaves:
  - Terminal nodes
  - Represent a class label or probability
  - When the outcome is a continuous variable it is considered a regression tree.
- The splits are created recursively
  - The process is repeated until some stop condition is met
  - Ex: depth of tree, no more information gain, etc...



Example Application in HEP

MiniBooNE experiment
arXiv: 0408124

- Go through all variables and find best variable and value to split events.
- For each of the two subsets repeat the process
- Proceeding in this way a tree is built.
- Ending nodes are called leaves.

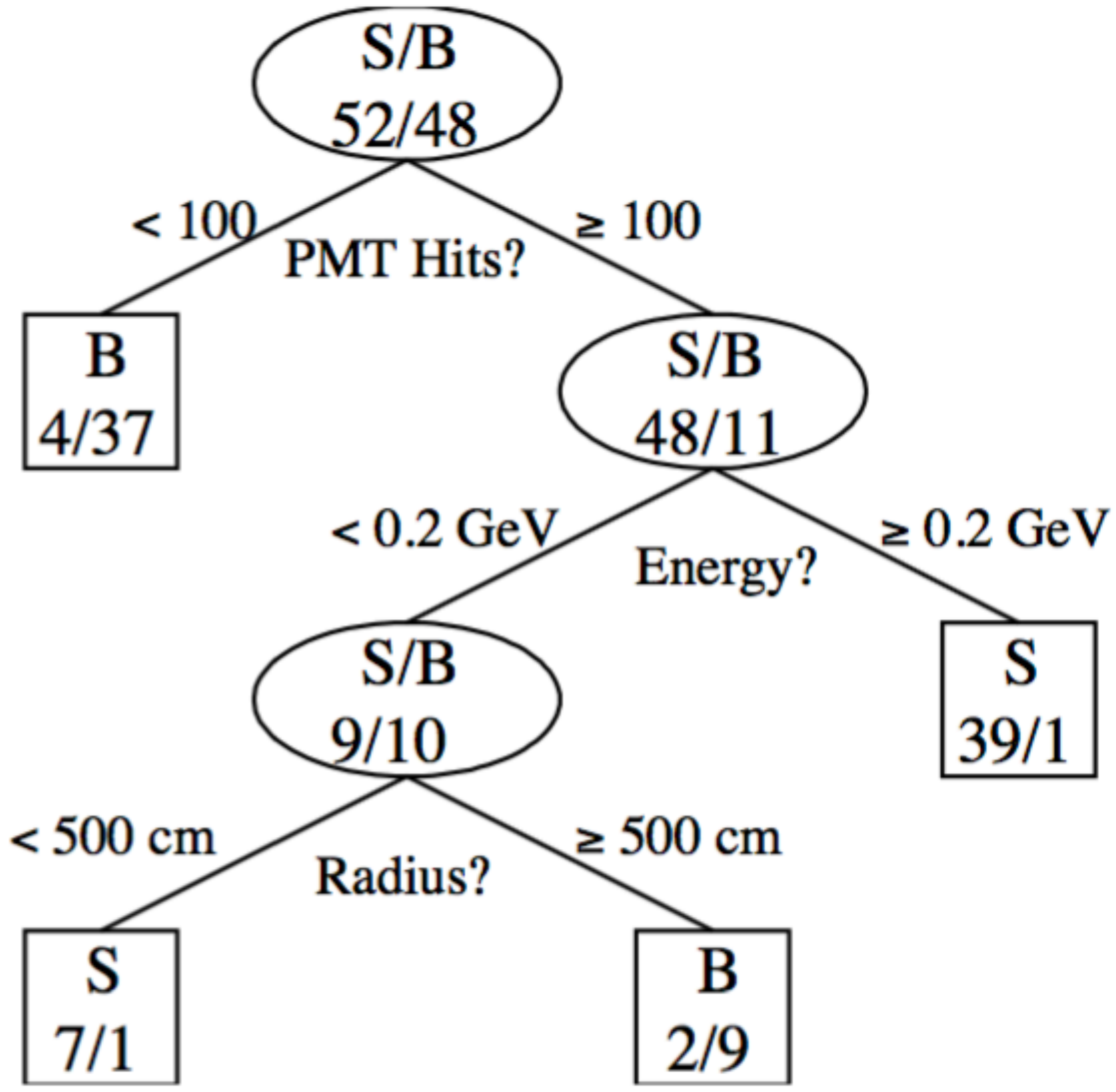
Criterion for "Best" Split

- SignalPurity, *P*, is the fraction of the weight of a leaf due to signal events.
- Gini: Note that Gini is 0 for all signal or all background. W_i is the weight of event "i".

$$G_{ini} = \left(\sum_{i=1}^{n} W_i\right) P(1-P)$$

- The criterion is to minimize Gini_left + Gini_right of the two children from a parent node

Signal/Background



Tree Boosting

Boosting:

- The general idea is to use the weak learning method several times to get a succession of hypotheses, each one refocused on the events that the previous ones found difficult and misclassified
  - producing a very accurate prediction rule by combining rough and moderately inaccurate

$l(\hat{y}_i, y_i)$

$\Omega(f_t)$

Tree Boosting

Procedure:

- Each tree is created iteratively
- The tree's output ($f(x)$) is given a weight (w) relative to its accuracy
- Events which are misclassified, increase their weights
- Build a new tree, repeat the procedure for several trees
- The final score of an event is the weighted average of scores from all trees

$$\hat{y} = \sum_k w_k f_k(x), \quad f_k \in F \longrightarrow \boxed{\text{Space of functions containing all regression tree}}$$

- *This means that samples that are difficult to classify receive increasing larger weights until the algorithm identifies a model that correctly classifies these samples*
- The goal is to minimize an objective function

$$obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

$l(\hat{y}_i, y_i)$ is the loss function (distance between truth and prediction value for i_{th} sample)

$\Omega(f_t)$ is the regularization function (it penalizes the complexity of the k_{th} tree)

Types of Boosting

There are many different ways of iteratively adding learners to minimize a loss function

Most common boosting algorithms are

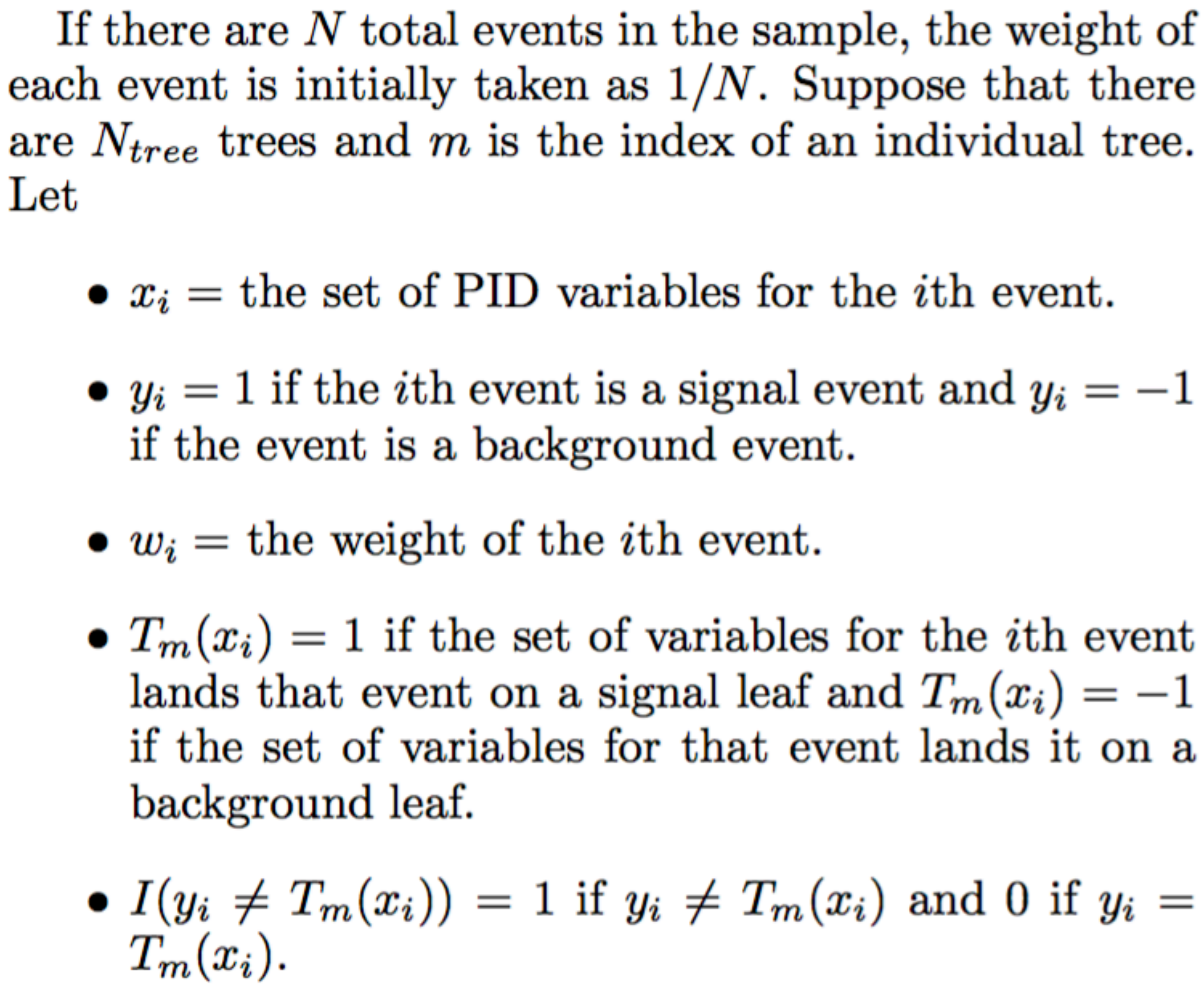
- AdaBoost
  - Adaptive Boosting
  - One of the early methods

- Gradient Boosting (GradBoost)
  - Uses gradient descent to create new learners
  - The loss function should be differentiable
  - http://statweb.stanford.edu/~jhf/ftp/trebst.pdf

- XGBoost (Xtreme gradient Boosting)
  - One type of gradient boosting
  - Very popular and widely used currently
  - Chen and Guestrin: https://arxiv.org/abs/1603.02754

AdaBoost

arXiv: 0408124



If there are N total events in the sample, the weight of each event is initially taken as $1/N$. Suppose that there are N_{tree} trees and m is the index of an individual tree. Let

- x_i = the set of PID variables for the ith event.
- $y_i = 1$ if the ith event is a signal event and $y_i = -1$ if the event is a background event.
- w_i = the weight of the ith event.
- $T_m(x_i) = 1$ if the set of variables for the ith event lands that event on a signal leaf and $T_m(x_i) = -1$ if the set of variables for that event lands it on a background leaf.
- $I(y_i \neq T_m(x_i)) = 1$ if $y_i \neq T_m(x_i)$ and 0 if $y_i = T_m(x_i)$.

AdaBoost

arXiv: 0408124

Define for m$_{th}$ tree

$$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq T_m(x_i))}{\sum_{i=1}^{N} w_i}.$$

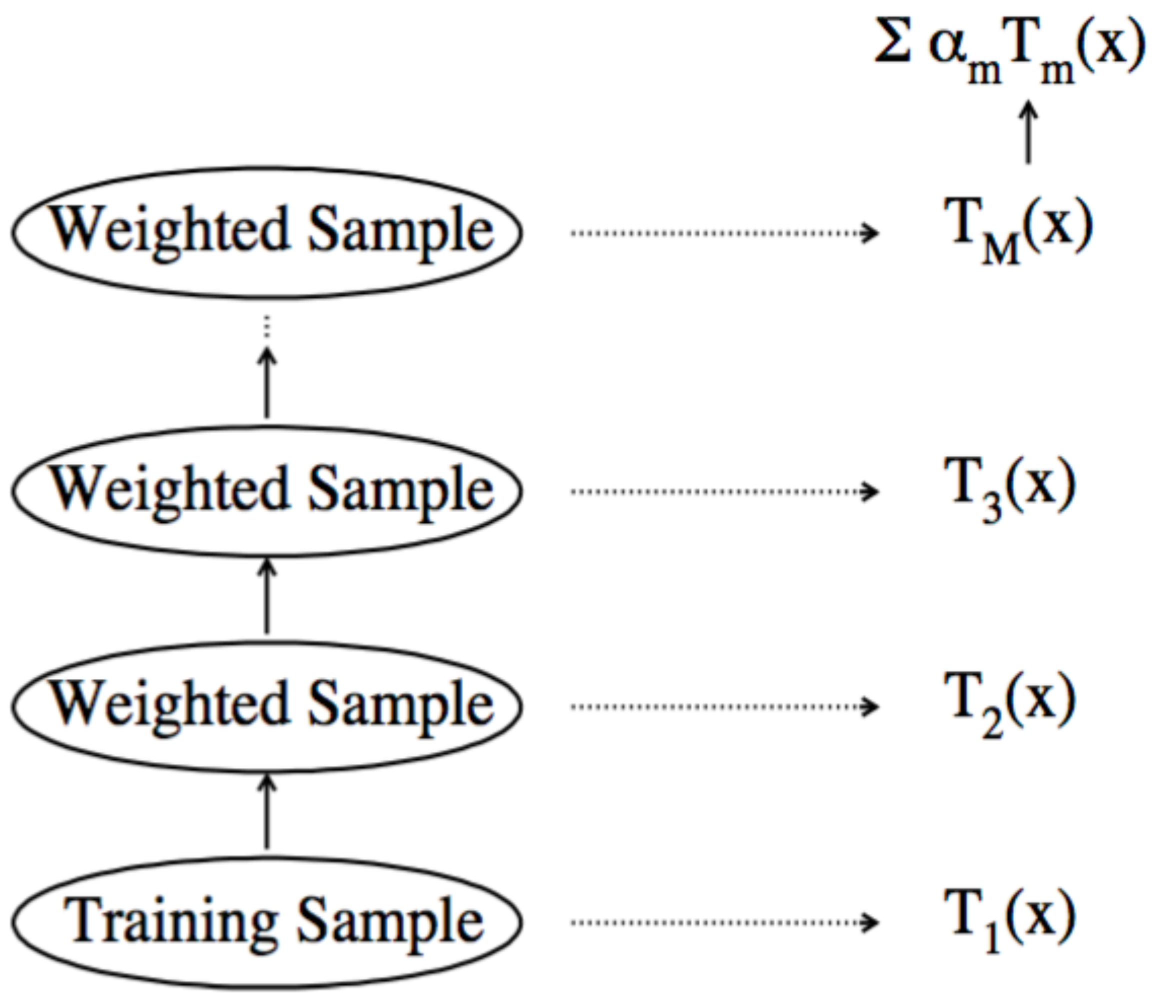
$$\alpha_m = \beta \times \ln((1 - err_m)/err_m).$$

Change the weight of each event by

$$w_i \rightarrow w_i \times e^{\alpha_m I(y_i \neq T_m(x_i))}.$$

Normalize weights

$$w_i \rightarrow w_i / \sum_{i=1}^{N} w_i.$$



Repeat training for N$_{trees}$
The score of a given event is:

$$T(x) = \sum_{m=1}^{N_{tree}} \alpha_m T_m(x),$$

ε-Boost (shrinkage)

arXiv: 0408124

Change the weight of the i_{th} event as

$$w_i \rightarrow w_i e^{2\epsilon I(y_i \neq T_m(x_i))}$$

ε is a constant of the order of 0.01

Normalize weights

$$w_i \rightarrow w_i / \sum_{i=1}^{N} w_i.$$

The score for a given event is

$$T(x) = \sum_{m=1}^{N_{tree}} \epsilon T_m(x)$$

renormalized, but unweighted, sum of the scores over individual trees.

Both the boosting algorithms minimize the expectation value of the loss function:

$$L(F, y) = e^{-yF(x)}$$

Where y = 1 for signal,
-1 for background

$$F(x) = \Sigma_{i=1}^{N_{tree}} f_i(x)$$

$f_i(x) = 1$ If event lands on signal leaf

$f_i(x) = -1$ If event lands on background leaf

Gradient Boosting

- Exponential loss has the shortcoming that it lacks robustness in presence of outliers or mislabelled data points
  - The performance of AdaBoost therefore is expected to degrade in noisy settings
- The GradientBoost algorithm attempts to cure this weakness by allowing for other, potentially more robust, loss functions without giving up on the good out-of-the-box performance of AdaBoost

(ref: TMVA userguide)

Gradient Boosting

- Current TMVA implementation uses the binomial log-likelihood loss:

$$L(F, y) = \ln \left(1 + e^{-2F(\mathbf{x})y}\right)$$

- Minimization is performed using steepest-descent approach
- Implementation in TMVA: Calculate the current gradient of the loss function and then grow a regression tree whose leaf values are adjusted to match the mean value of the gradient in each region defined by the tree structure – www.jstor.org/stable/2699986
- Iterating this procedure yields the desired set of decision trees which minimizes the loss function
- Robustness can be enhanced by reducing the learning rate of the algorithm (shrinkage), which controls the weight of the individual trees

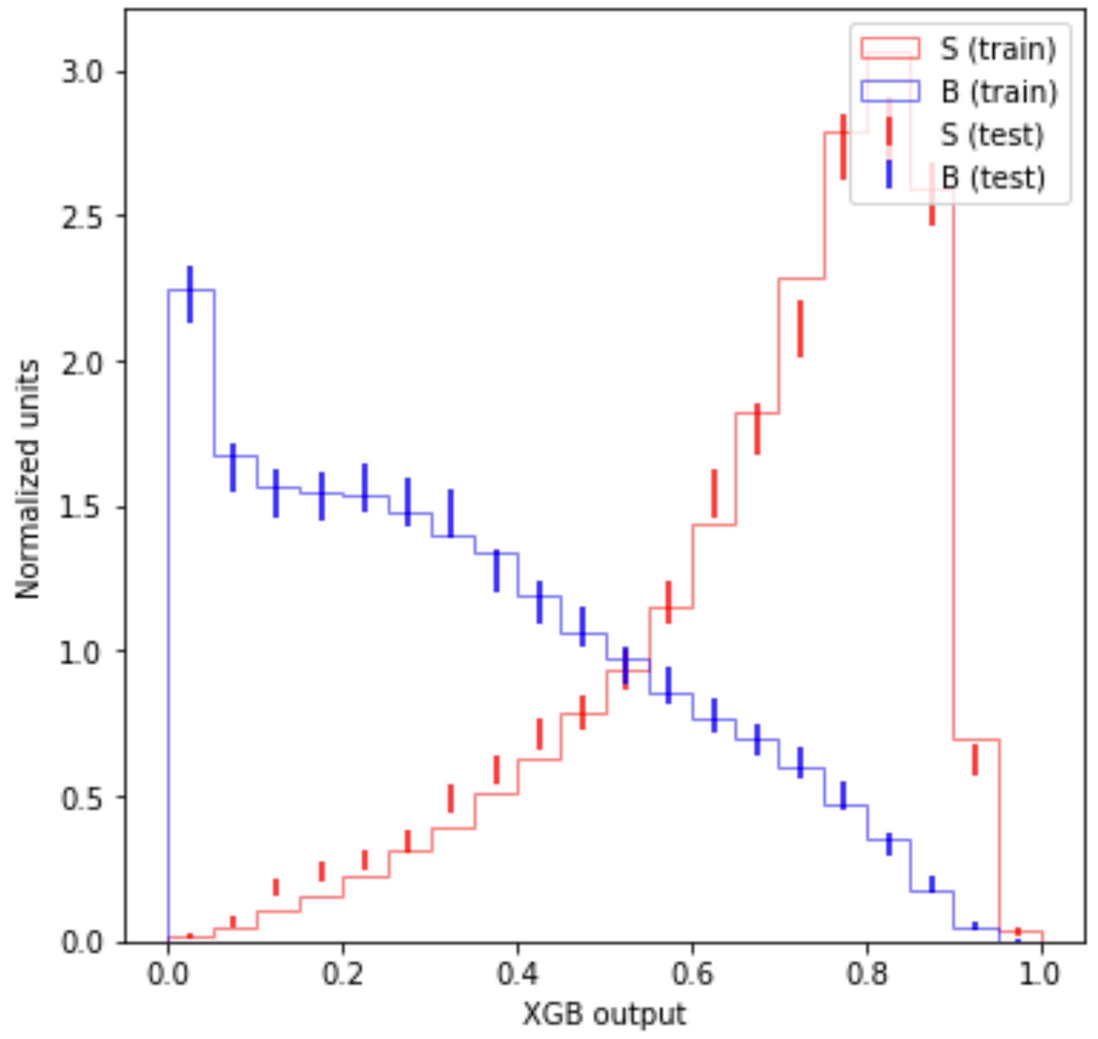
(ref: TMVA userguide)

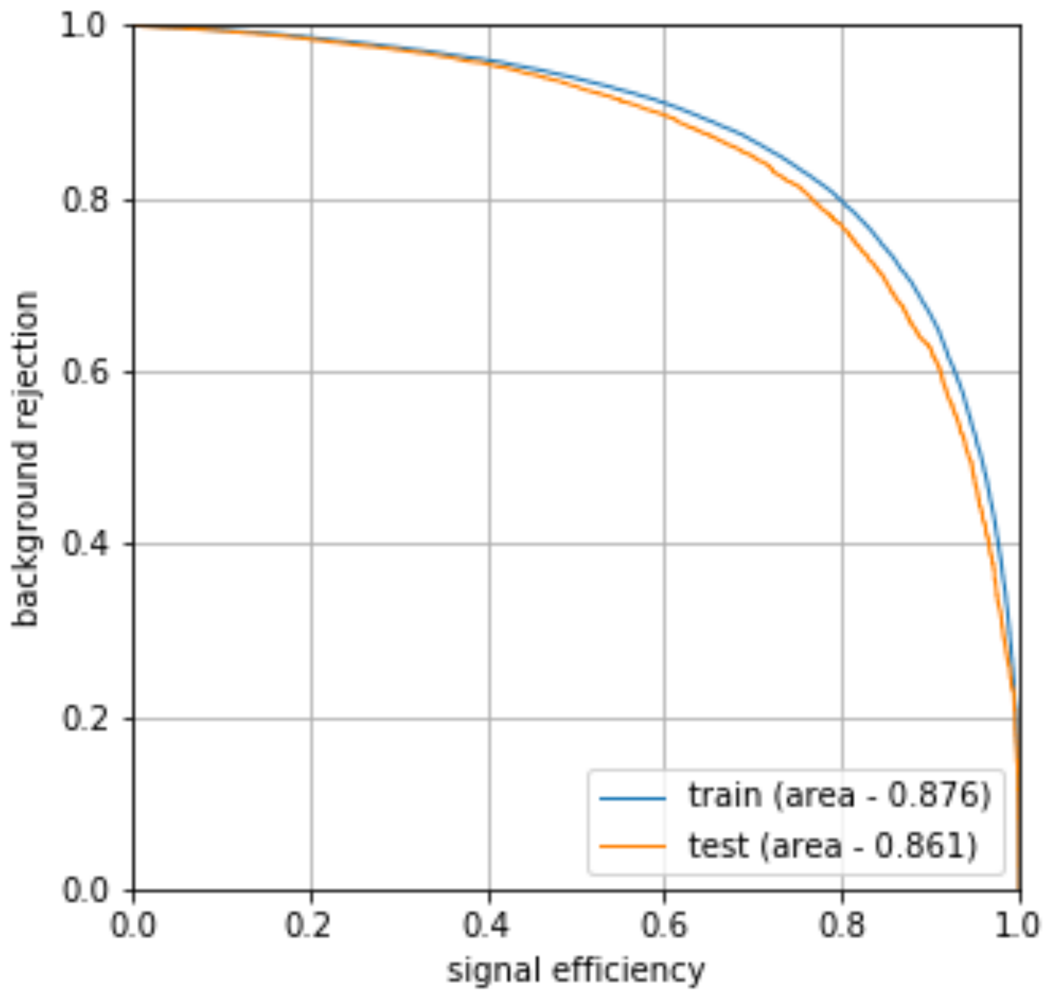
Stochastic Gradient Boosting

- Introduce a resampling procedure using random subsamples of the training events for growing trees – called “Bagging”
- The sample fraction used in each iteration can be controlled through a parameter, typical values to get best results are 0.5—0.8.
- Stability against statistical fluctuations

Example

- Same signal,background and input variables as the DNN example.
- XGBoost Algorithm trained on 600 trees, Depth=3 ,learning rate = 0.1, with bagged boost.





Details will be discussed in the tutorial tomorrow

Summary

- Machine learning methods, such as neural network and boosted decision trees are heavily used in high energy physics data analysis
  - In particular, in the experiments at LHC
  - Achieves much better performance than the cut based analyses with same amount of data
  - Not only the signal to background separation, but also better object identification
- This lecture discussed only basic concepts about Artificial Neural Network and Boosted Decision Trees, with their example applications HEP problems.
  - Could not cover many more advanced tools, such as CNN, RNN, graph neural network etc.., that are currently being used.
  - There will be a hands-on session tomorrow with some practical examples

References

1. The elements of statistical learning, Hastie, Tibshirani, Friedman
2. http://neuralnetworksanddeeplearning.com/
3. https://machine-learning-for-physicists.org/
4. TMVA userguide
5. XGBoost userguide
6. And may more from the web….

Thanks

Supervised Learning

Supervised learning is a method where we use the training data (with multiple features) x_i to predict a target variable y_i.

Model & Parameters:

Model usually refers to the mathematical structure by which the prediction y_i is being made from the input x_i.
A common example is the liner model: $y_i = \Sigma\omega_{ij}x_j$
The prediction value can have different interpretations, depending on the task, i.e. regression or classification
ω are the undetermined part that we need to learn from data

Objective Function:

A function that measures how well the model fit the training data
Consists two parts, Training loss and Regularization terms:

$$\text{obj}(\theta) = L(\theta) + \Omega(\theta)$$

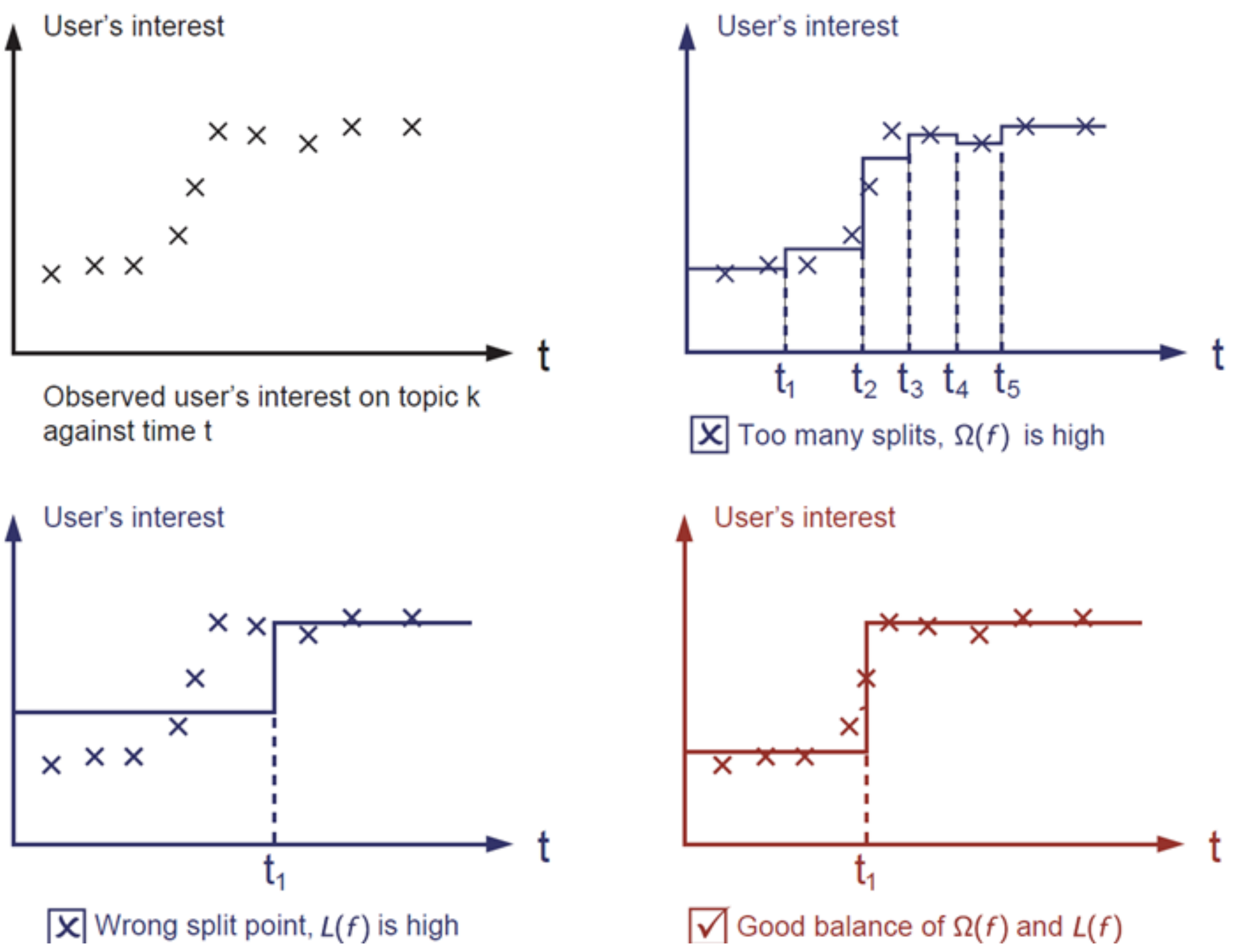
L measures how predictive our model is with respect to our training data
Common choices for L are "mean squared error, logistic loss etc.."

$$L(\theta) = \Sigma(y_i - \hat{y}_i)^2$$

$$L(\theta) = \Sigma[y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i)\ln(1 + e^{\hat{y}_i})]$$

Regularization term

The regularization term controls the complexity of the model, which helps us to avoid over-fitting



(ref: XGBoost userguide)