An introduction to Artificial Neural Network

Arun Nayak
(Institute of Physics, Bhubaneswar)

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

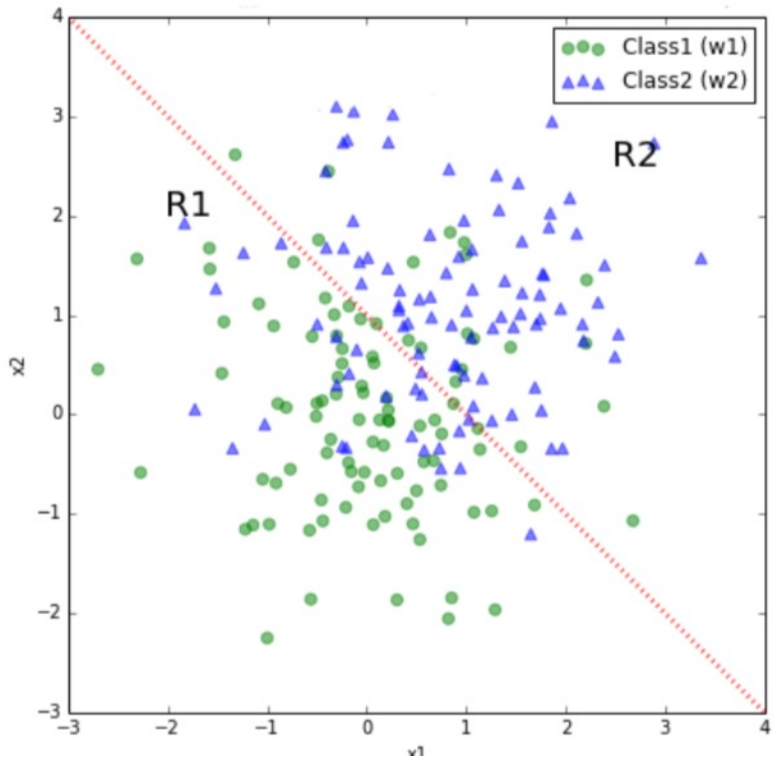
Popular Learning Methods

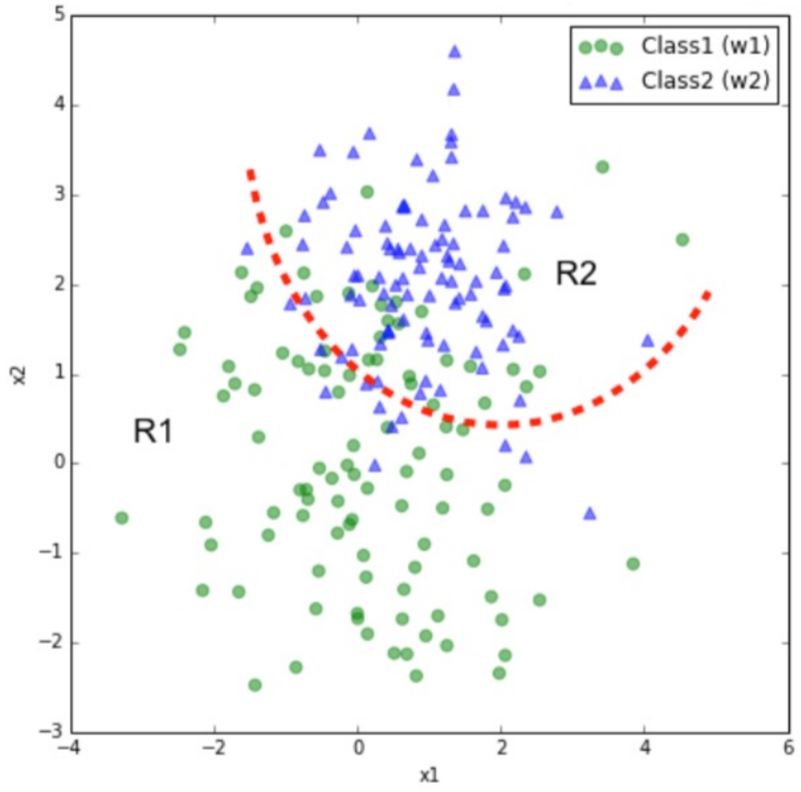
- Semisupervised Learning
  - Similar application as supervised learning
  - Uses both labeled and unlabeled data for training
  - Small amount of labeled, large amount of unlabeled data
- Reinforcement Learning
  - Often used for robotics, gaming and navigation
  - the algorithm discovers through trial 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



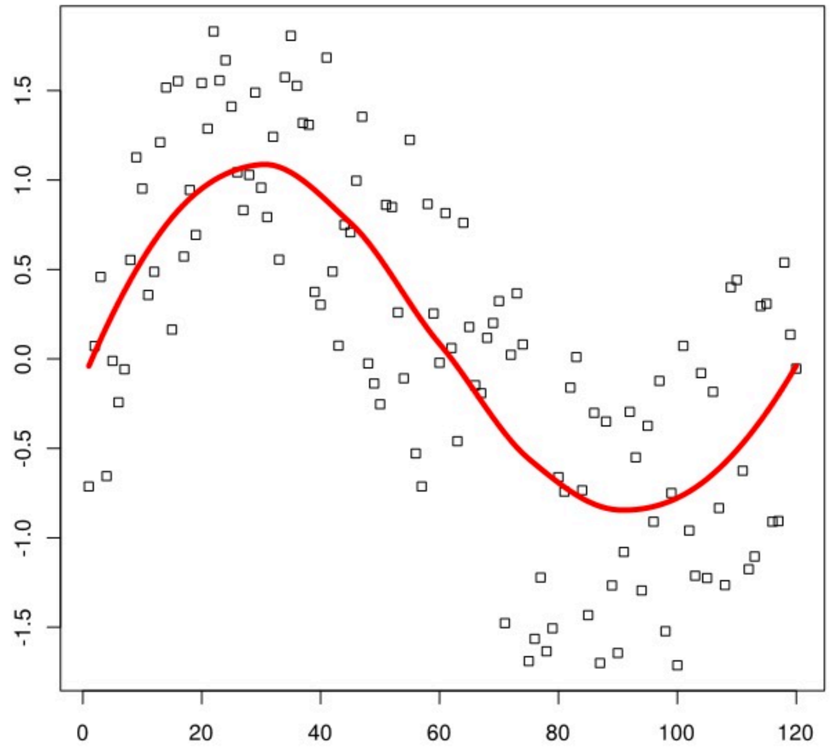


(images from google)

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



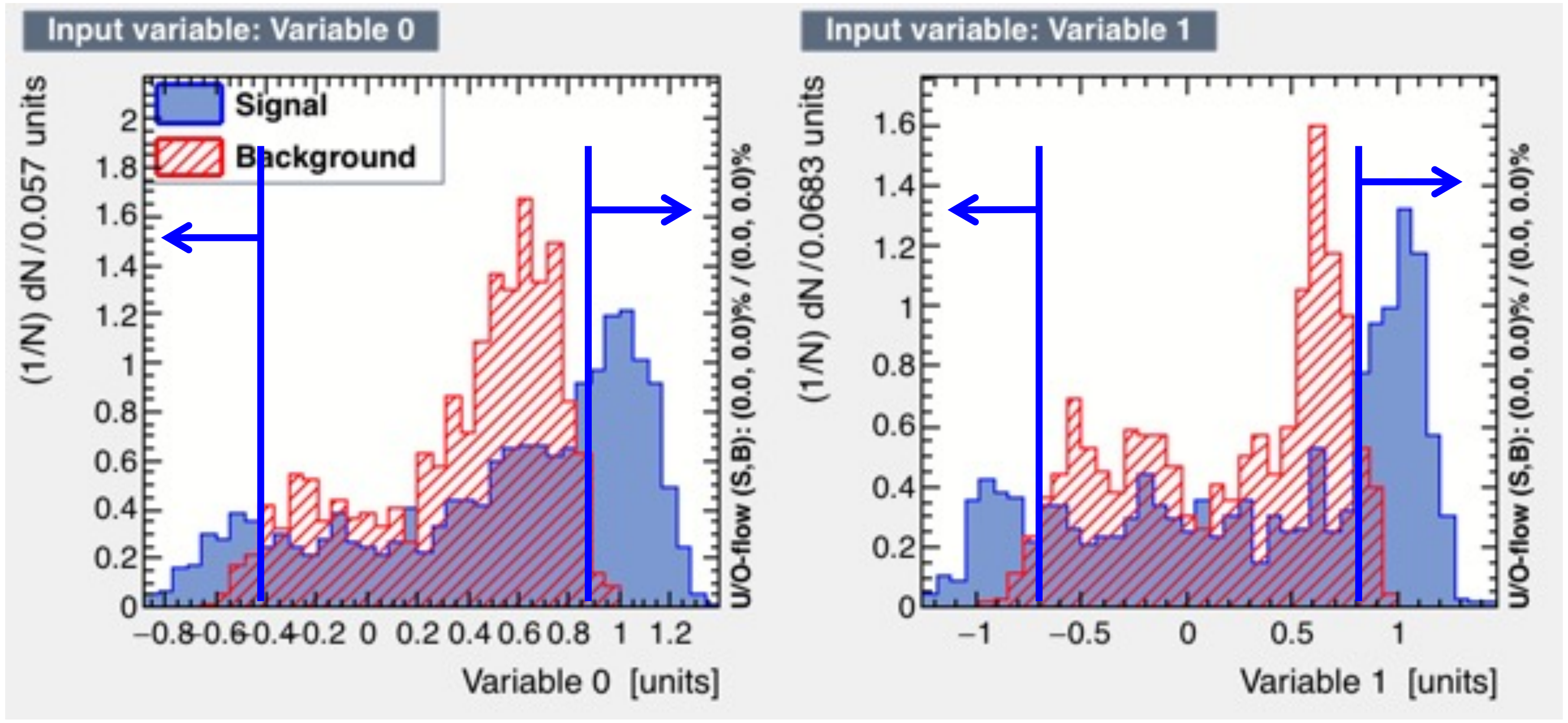
(images from google)

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.



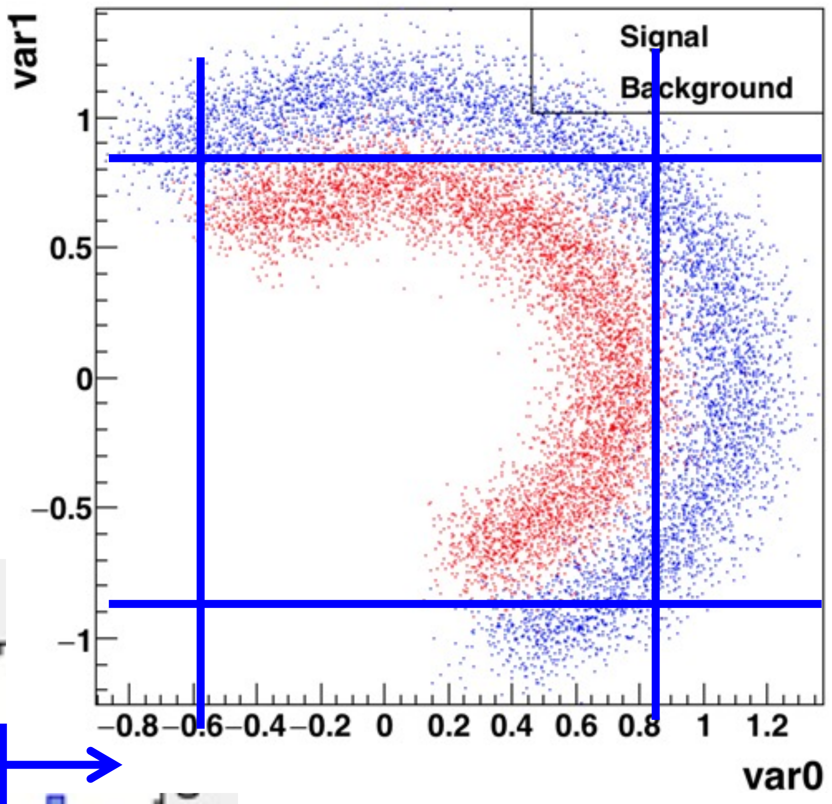
Cuts are not optimal

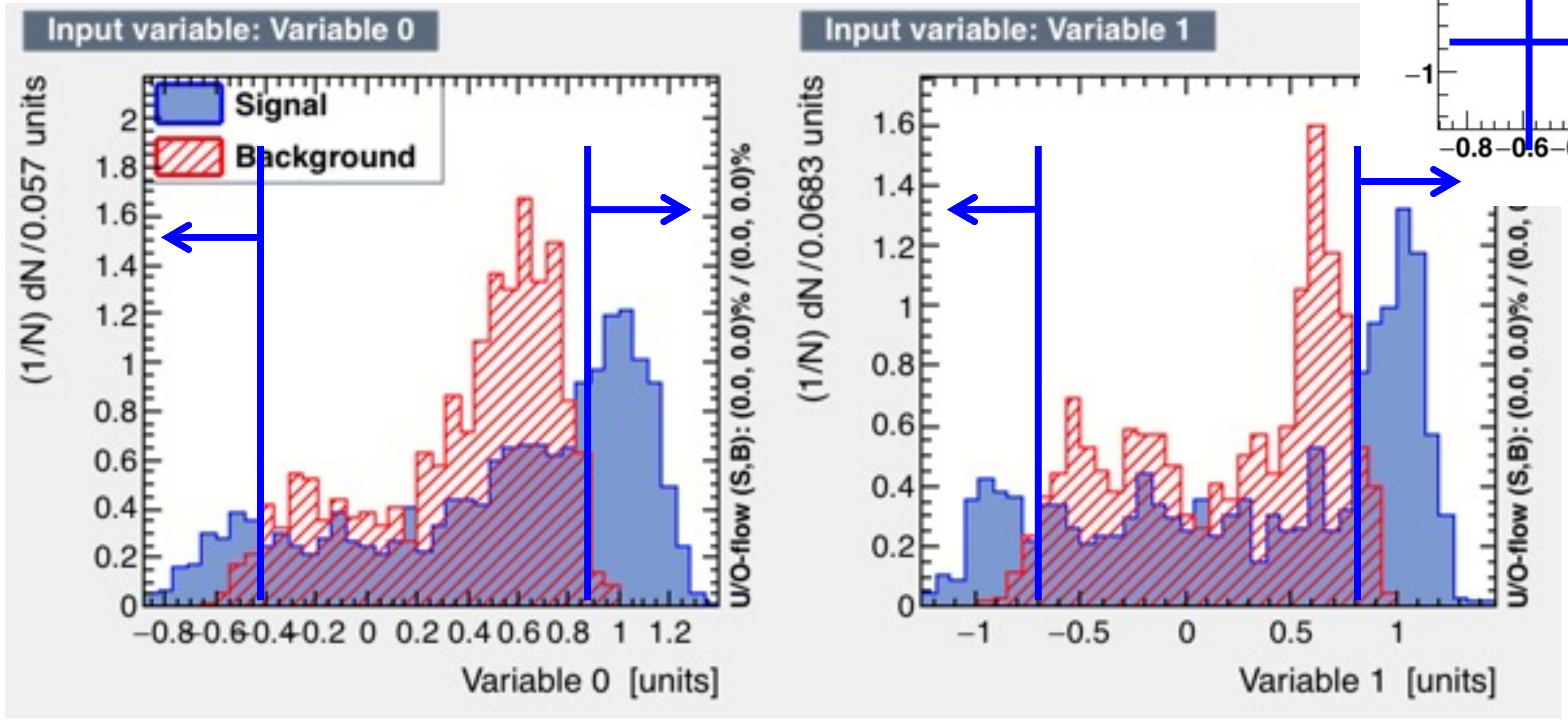
- Can not take correlations in to account
- Can lead to low signal efficiency and high background rate

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

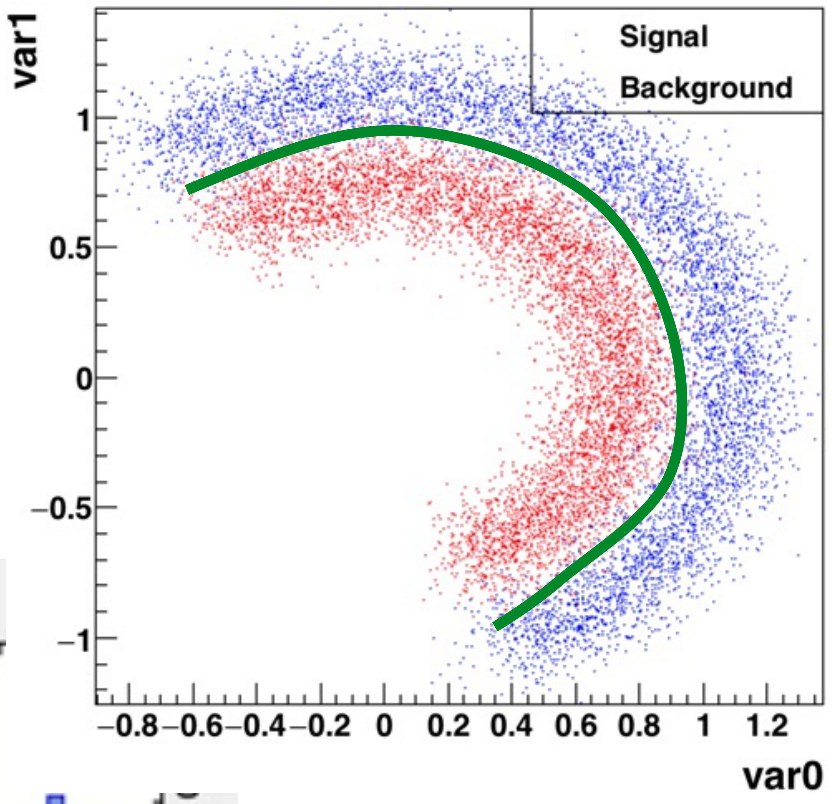


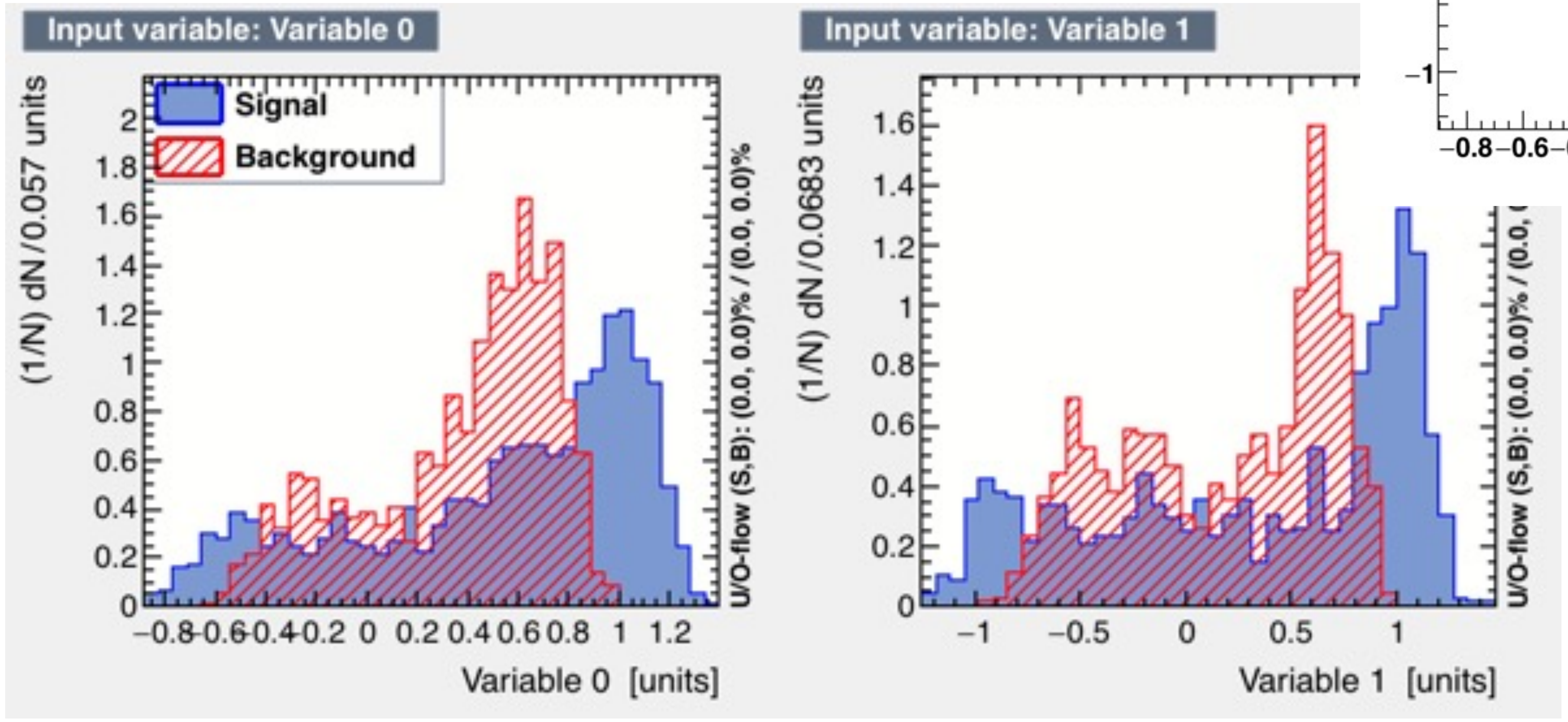


Data generated using Root/tutorials/tmva/createData.C

Why use MVA Analysis?

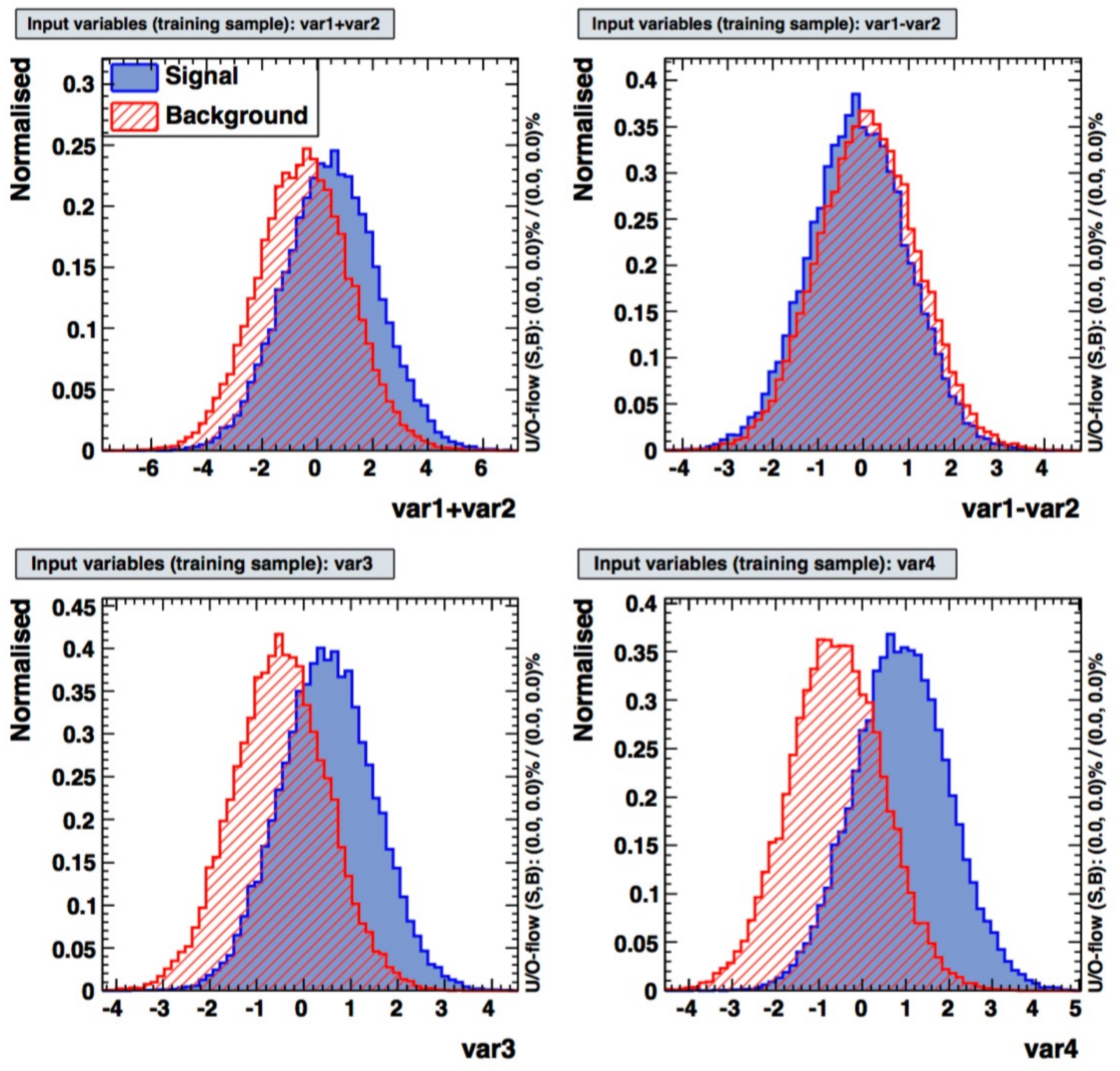
The actual boundary should be →



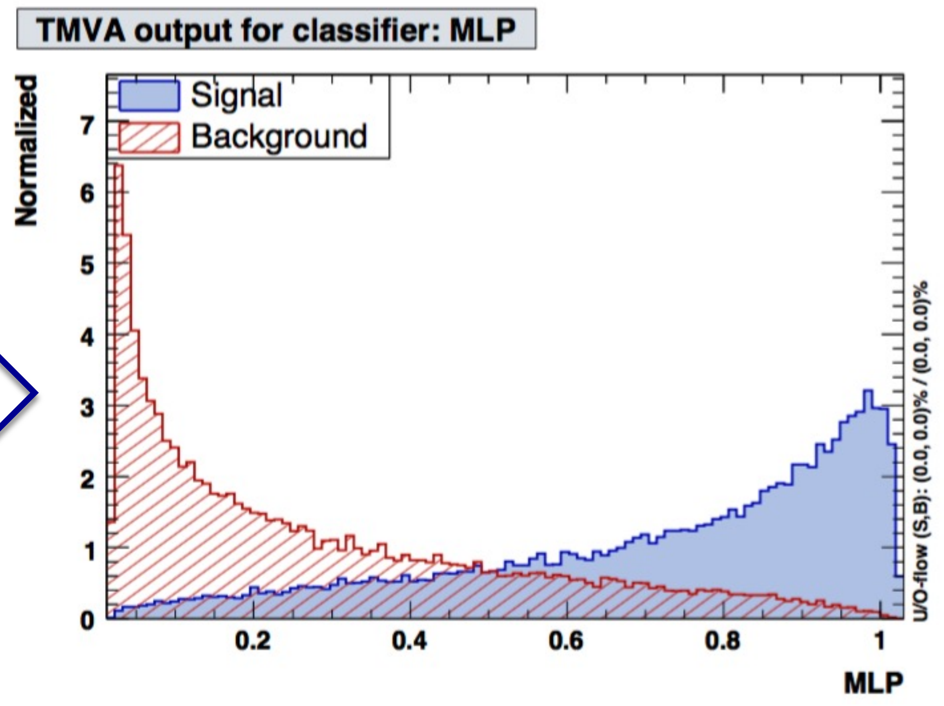


Why use MVA Analysis?

An Example



MVA

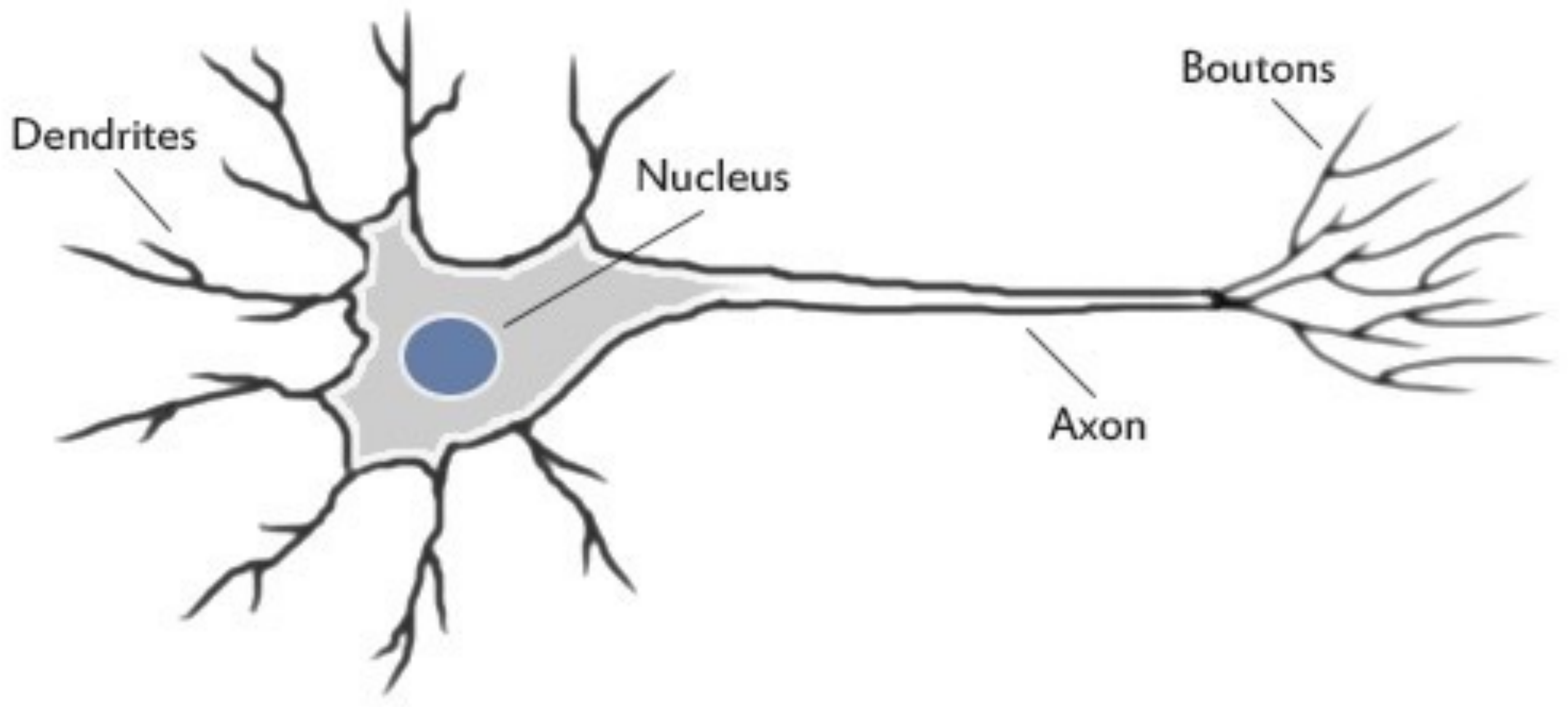


Combines many variables to a single variable, on which a cut can be applied depending on the required signal efficiency and purity

(figures from TMVA userguide)

Neural Network

Neural connections in human brain

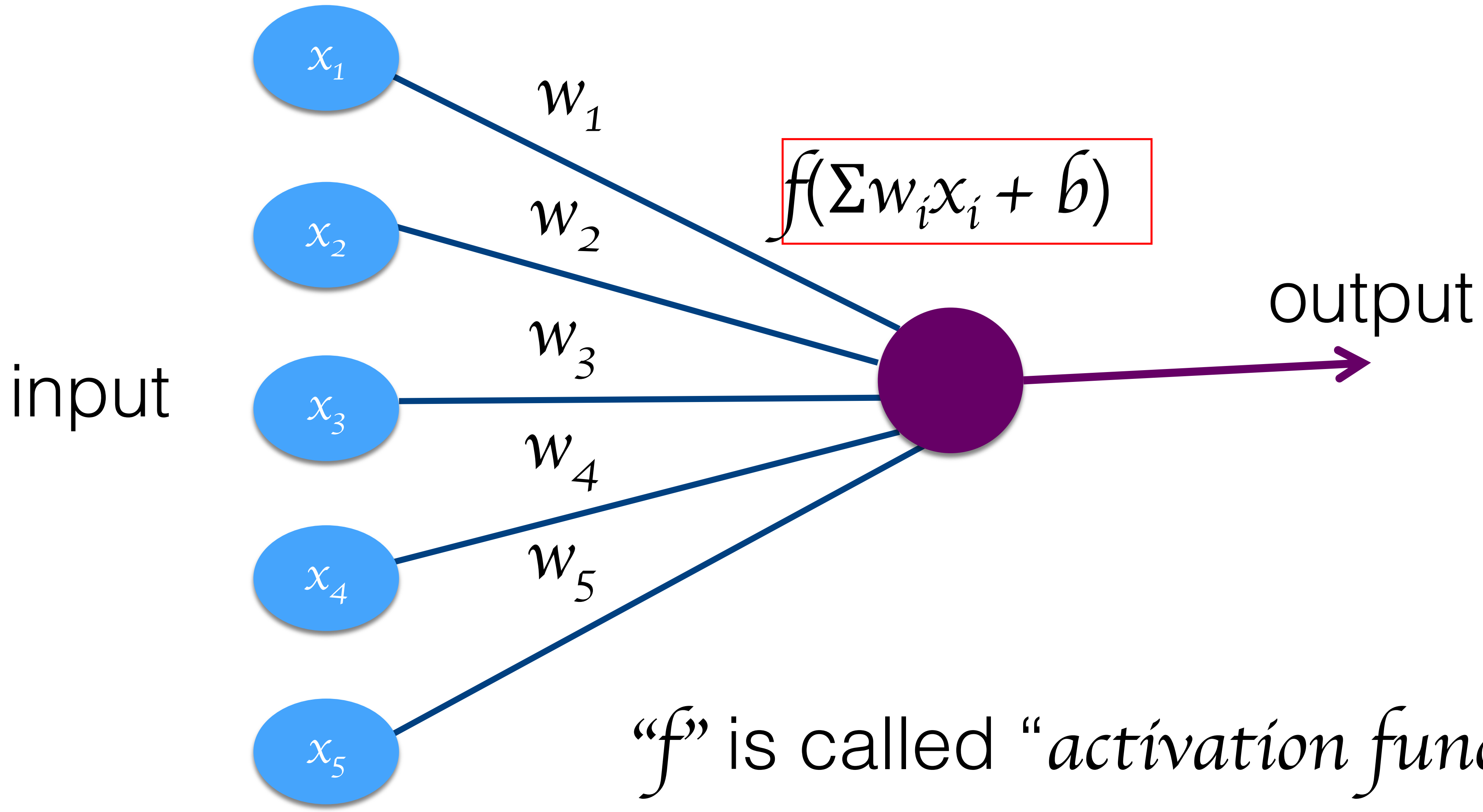


- Collects inputs from other neurons using dendrites
- Sums all the inputs, and fires, if the value is greater than a threshold
- The fired signal is then sent to other neurons through the Axon

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

Artificial Neuron

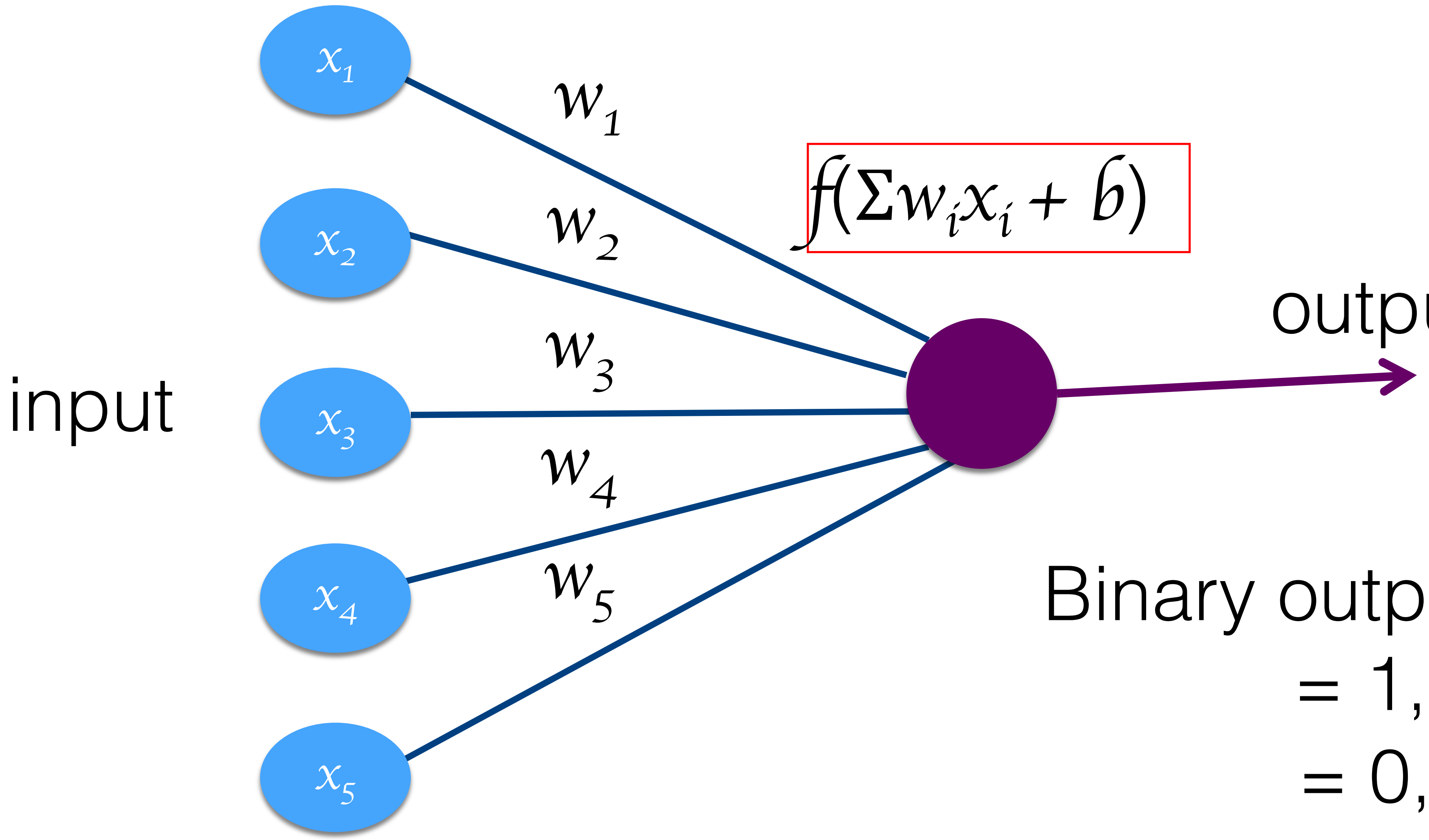
Model of an artificial neuron

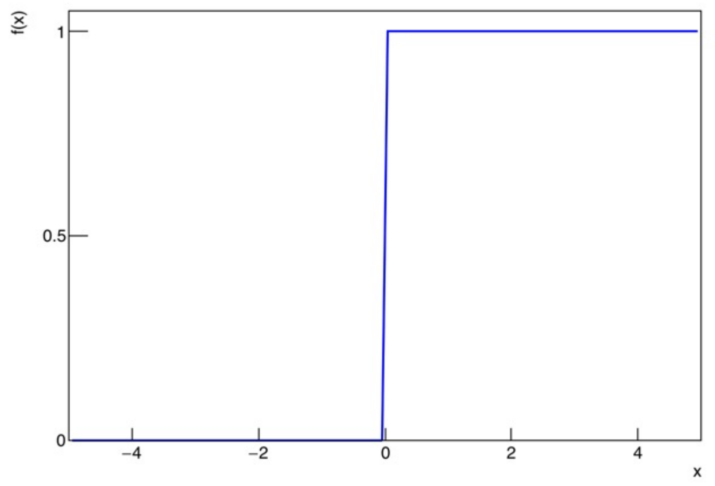


"f" is called "*activation function*"

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

Perceptron



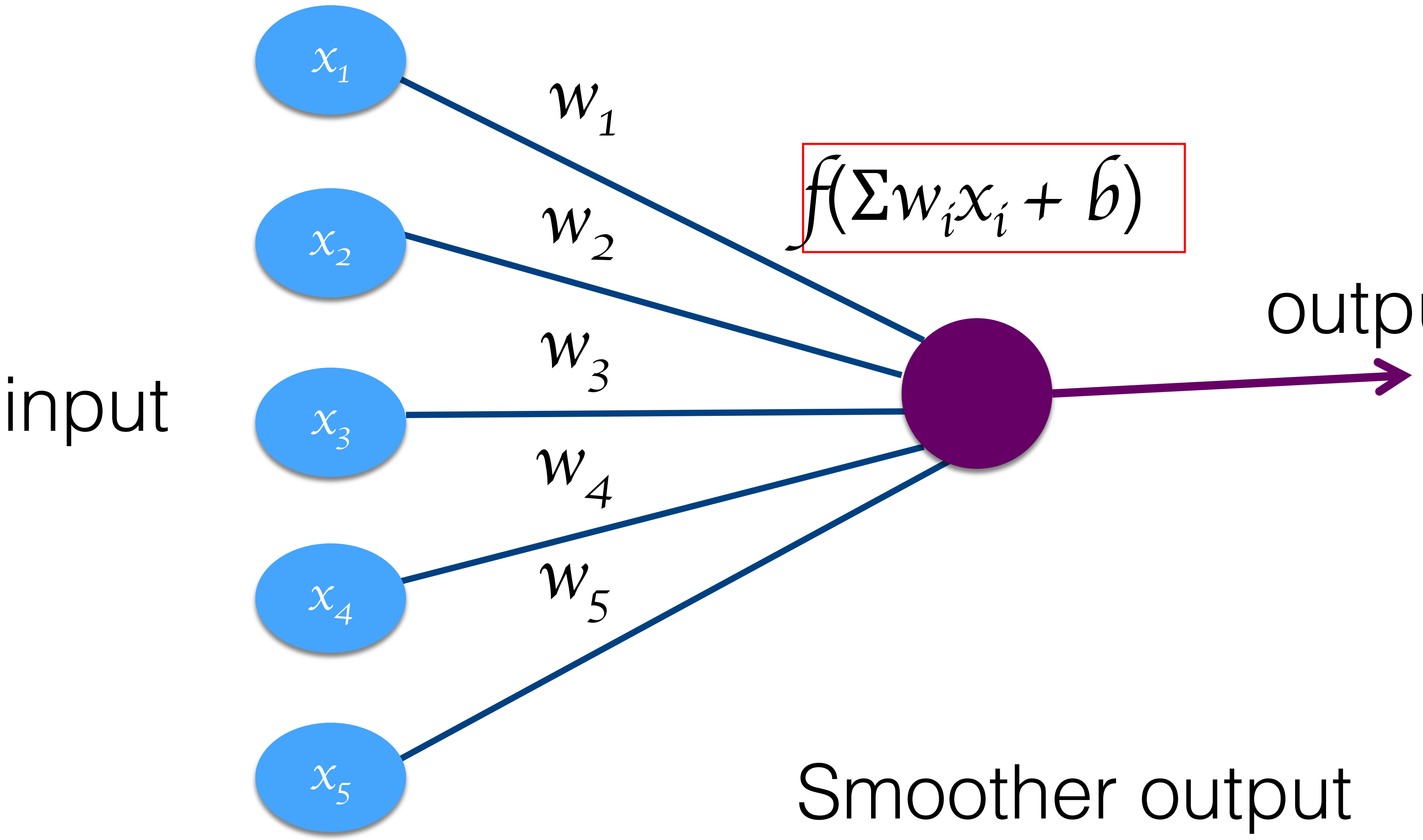


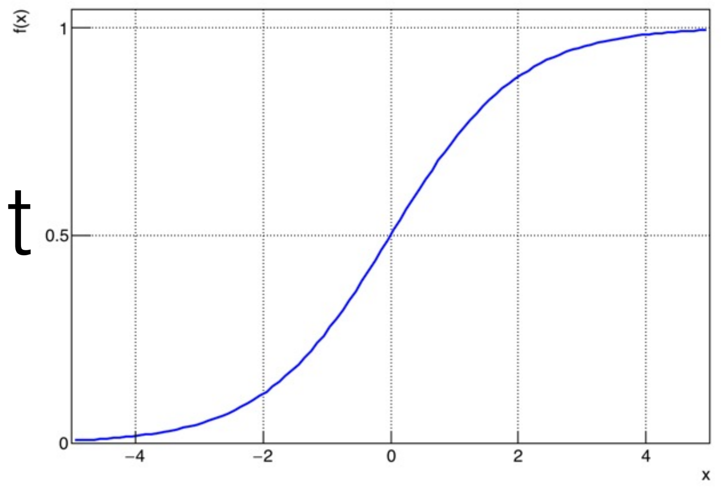
Binary output:

$$= 1, \text{ if } \Sigma w_i x_i + b > 0$$
$$= 0, \text{ if } \Sigma w_i x_i + b < 0$$

Sigmoid Neuron

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





Smoother output

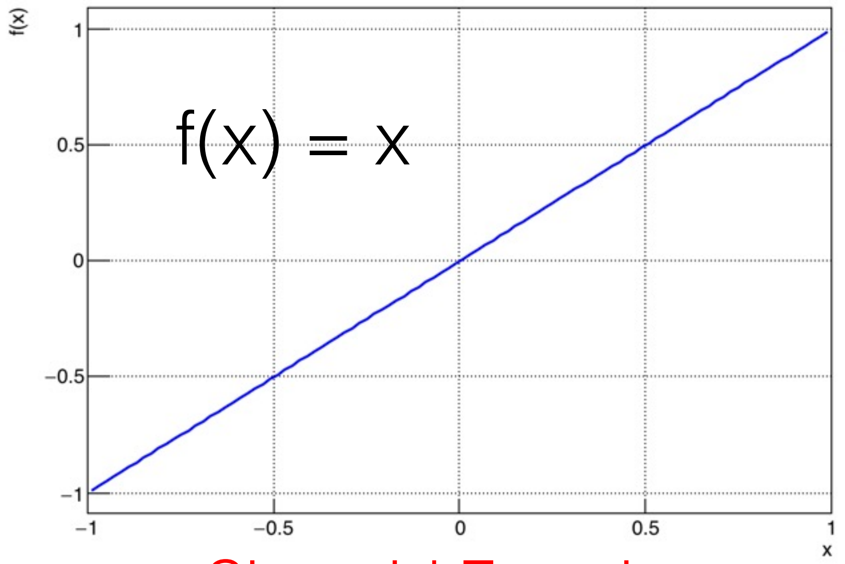
Small change in weight => small corresponding change in output

This property makes the learning possible

Activation Functions

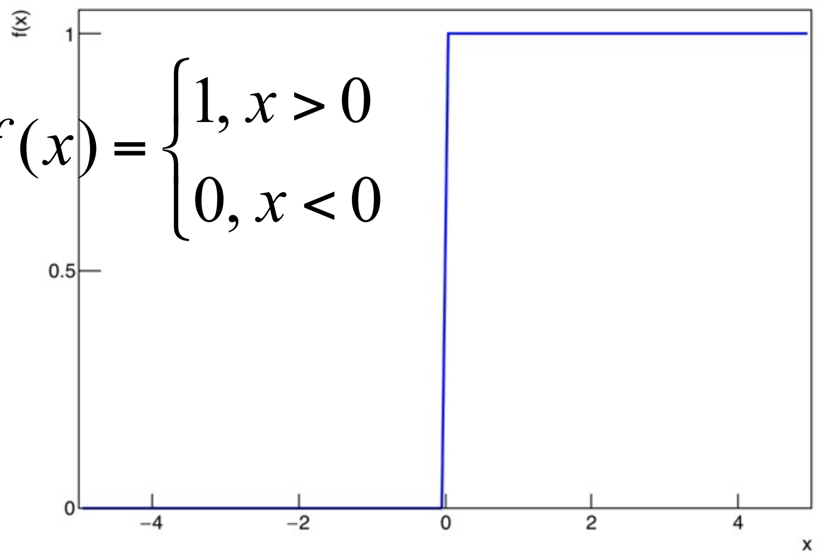
It determines at what threshold the neuron will fire
OR the frequency at which a neuron fires

Linear Function



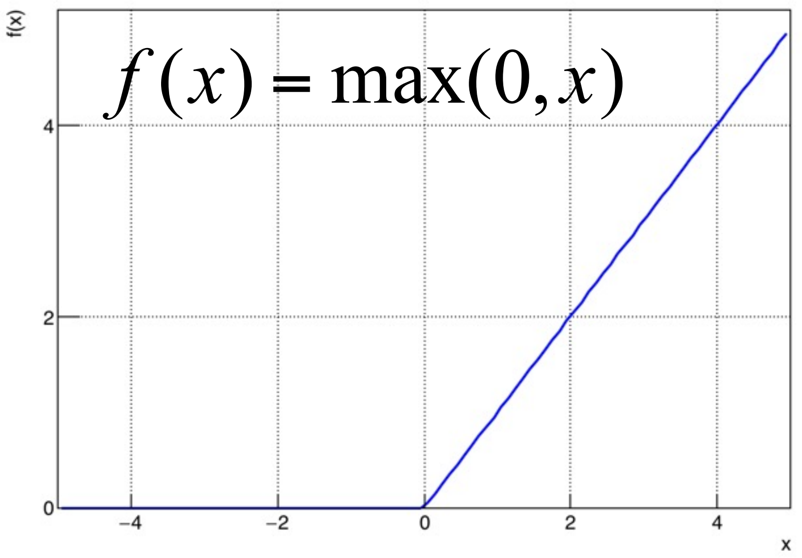
f(x) = x

Step Function



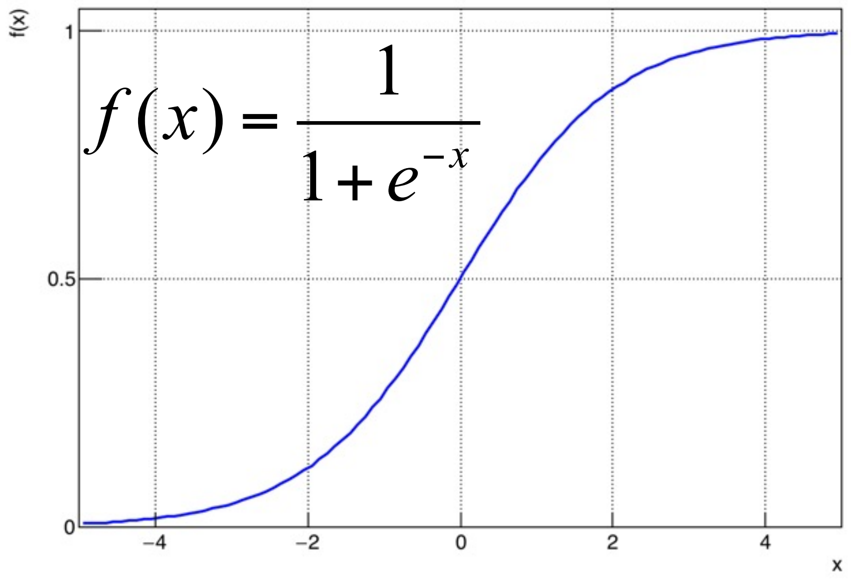
$$f(x) = \begin{cases} 1, x > 0 \\ 0, x < 0 \end{cases}$$

Rectified Linear Units (ReLU)



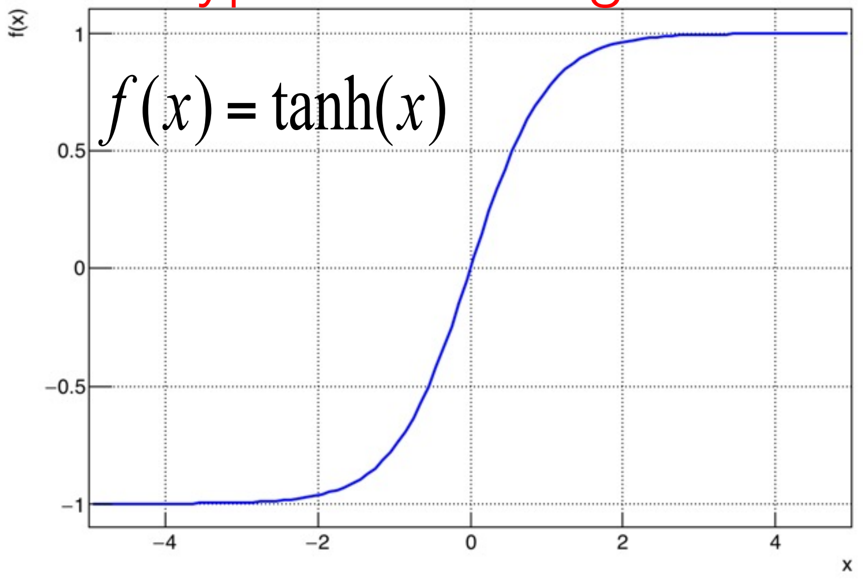
$$f(x) = \max(0, x)$$

Sigmoid Function



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

Hyperbolic Tangent



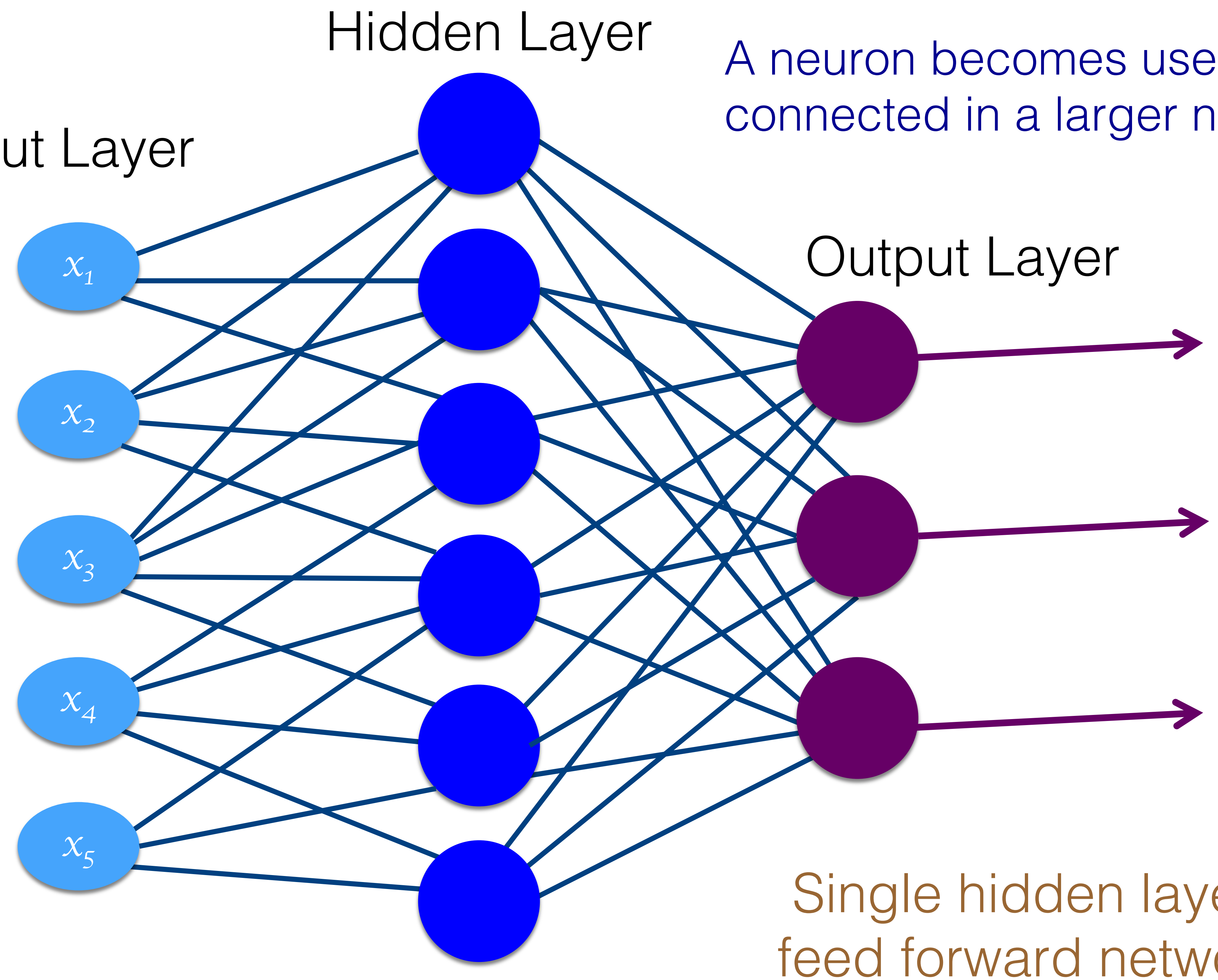
$$f(x) = \tanh(x)$$

Softmax

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

(used in output layer of a multiclassification network)

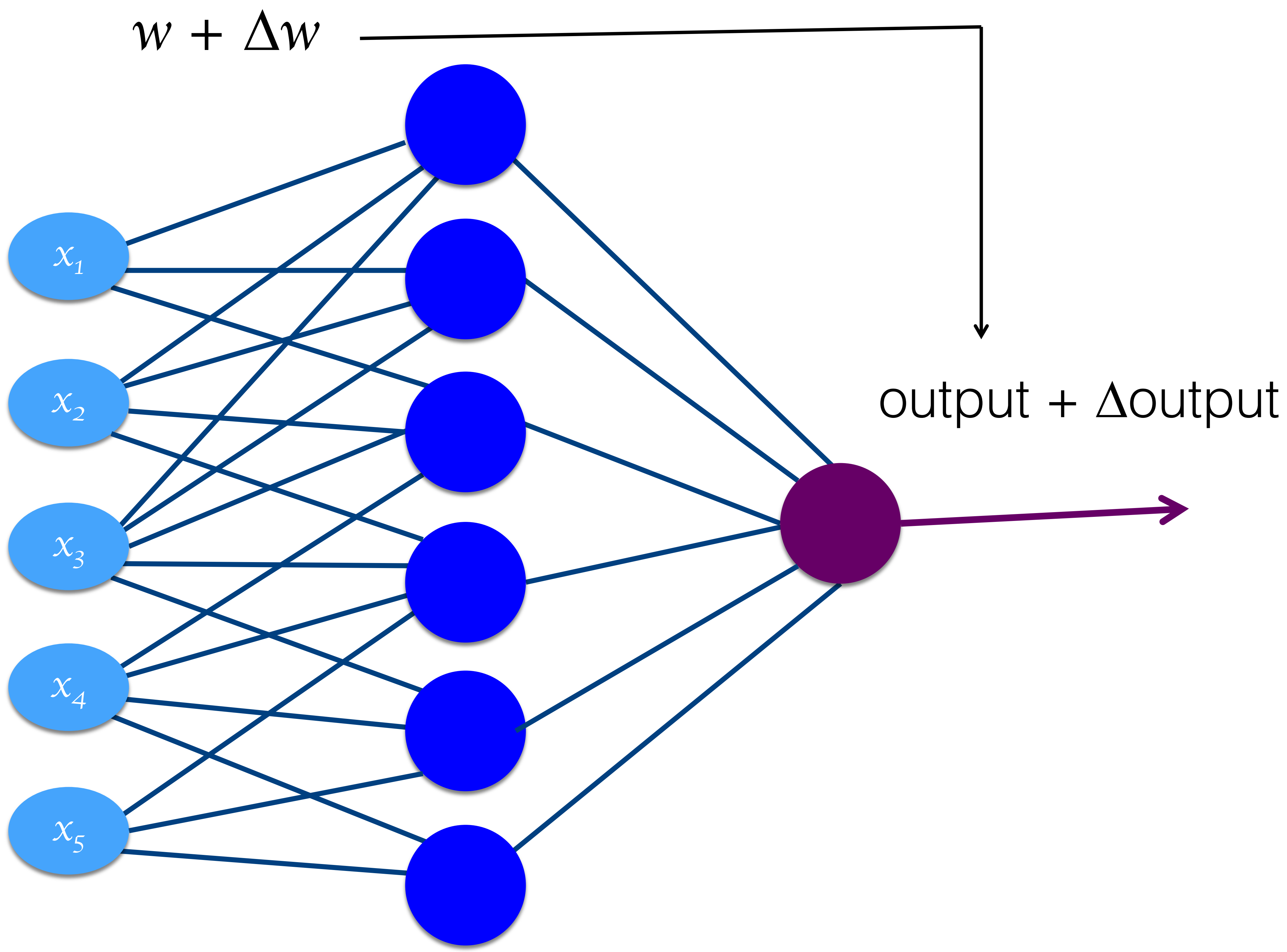
Artificial Neural Network



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

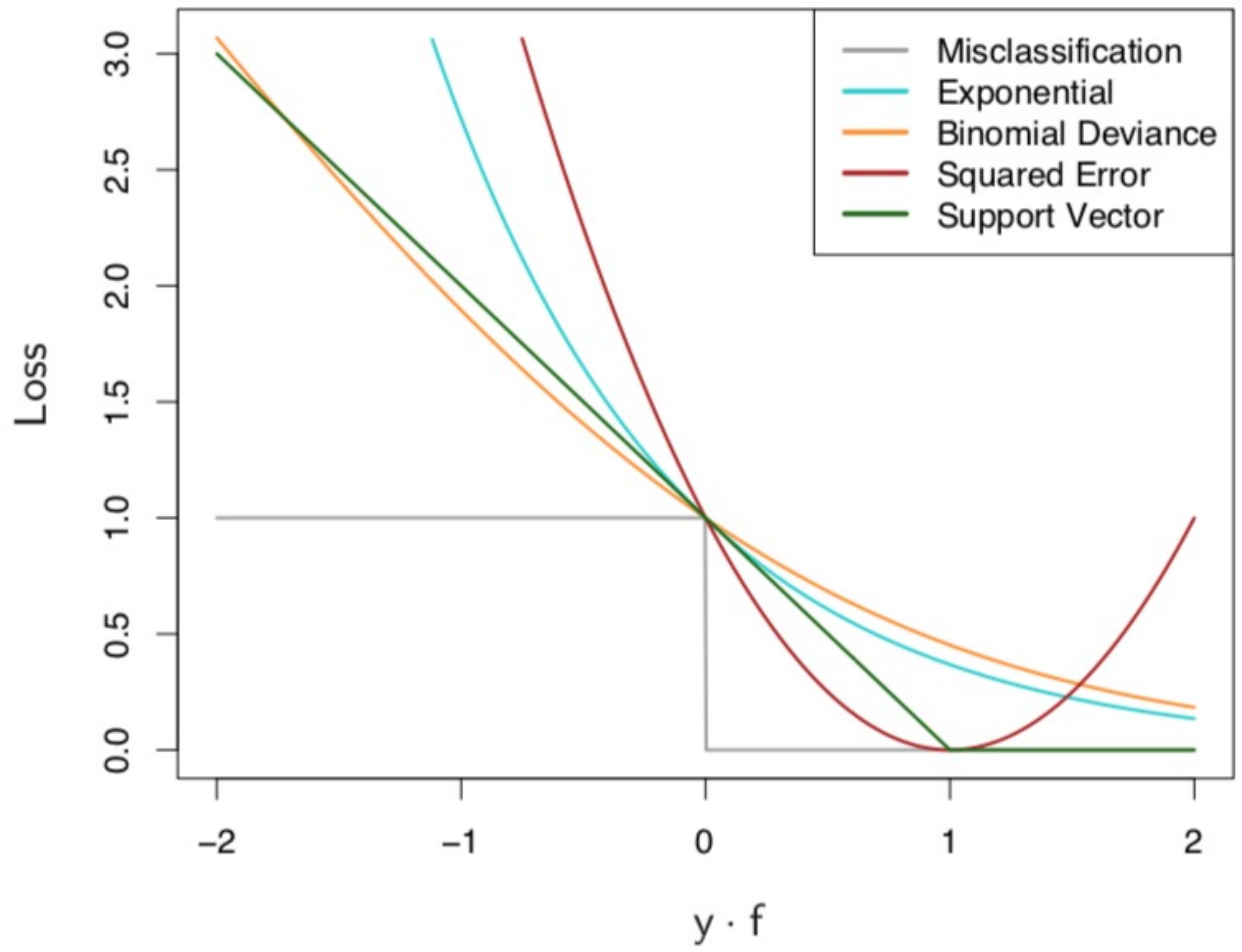
1. Mean Squared Error: $$L(w,b) = \frac{1}{2}\sum_{k=1}^{K}\sum_{i=1}^{N}\left(y_{ik} - \hat{y}_{ik}\right)^2$$

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

Loss Functions



T. Hastie et al.

FIGURE 10.4. *Loss functions for two-class classification. The response is* $y = \pm 1$*; the prediction is* f*, with class prediction* $\mathrm{sign}(f)$*. The losses are misclassification:* $I(\mathrm{sign}(f) \neq y)$*; exponential:* $\exp(-yf)$*; binomial deviance:* $\log(1 + \exp(-2yf))$*; squared error:* $(y - f)^2$*; and support vector:* $(1 - yf)_+$ *(see Section 12.3). Each function has been scaled so that it passes through the point* $(0, 1)$.

Backpropagation

Propagating in the backward direction to update the weights

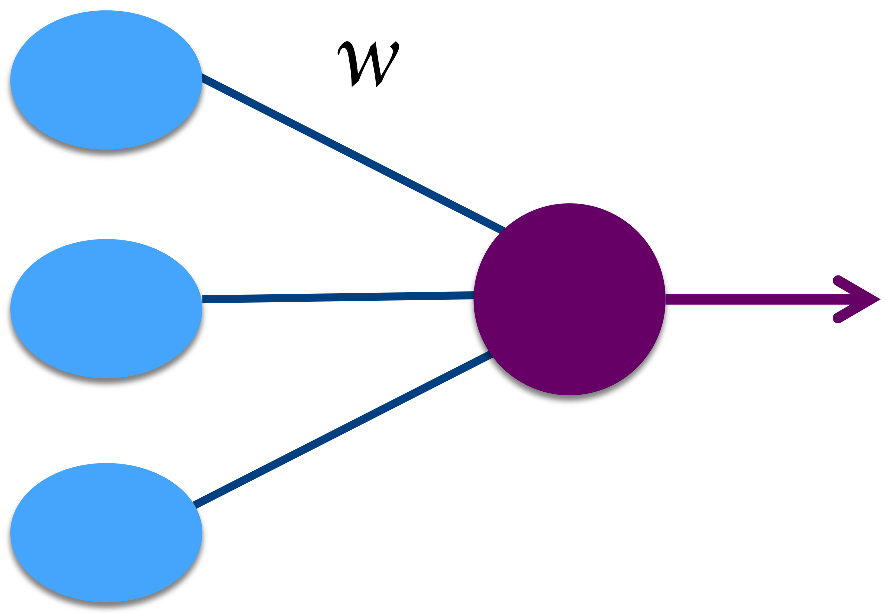
Steps:

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. ΔL is –ve)

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

Gradient descent

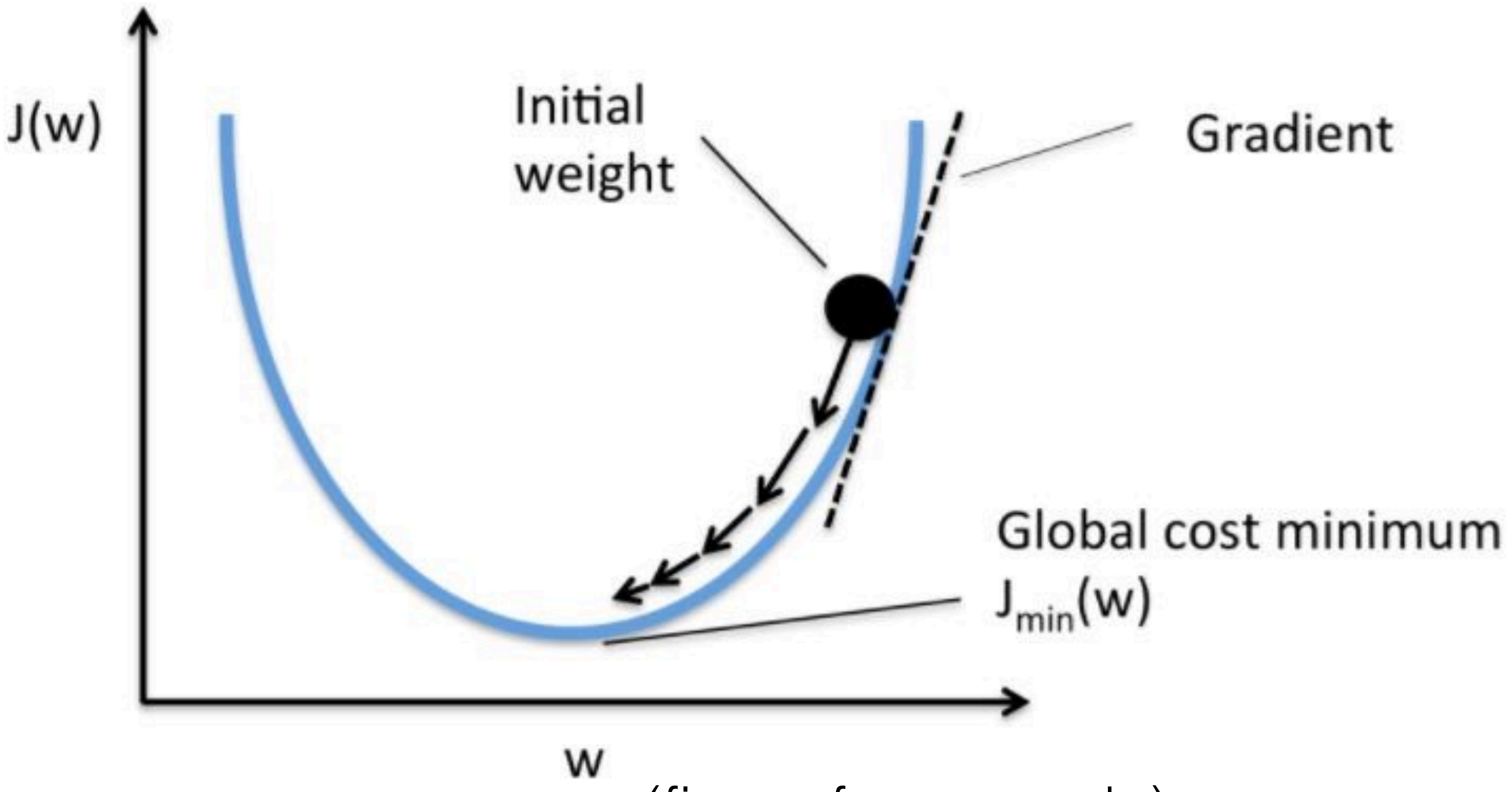
If we choose $\Delta w = -\eta \nabla L, \Rightarrow \Delta L \approx -\eta \,||\, \nabla L \,||^2 \Rightarrow \Delta L < 0$

Gradient descent

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

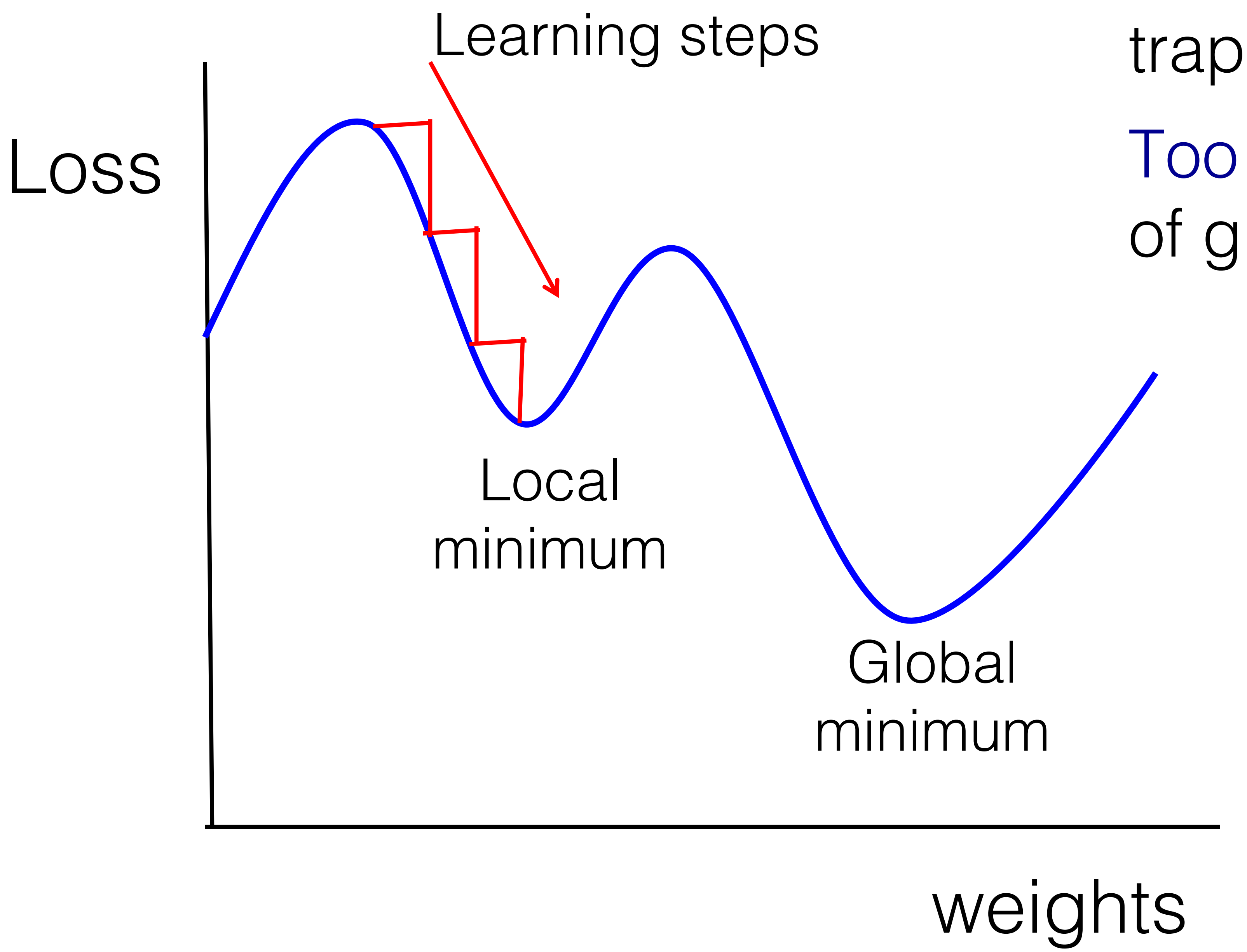
$$w_j(t+1) = w_j(t) - \eta \frac{\partial L}{\partial w_j}(t)$$

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

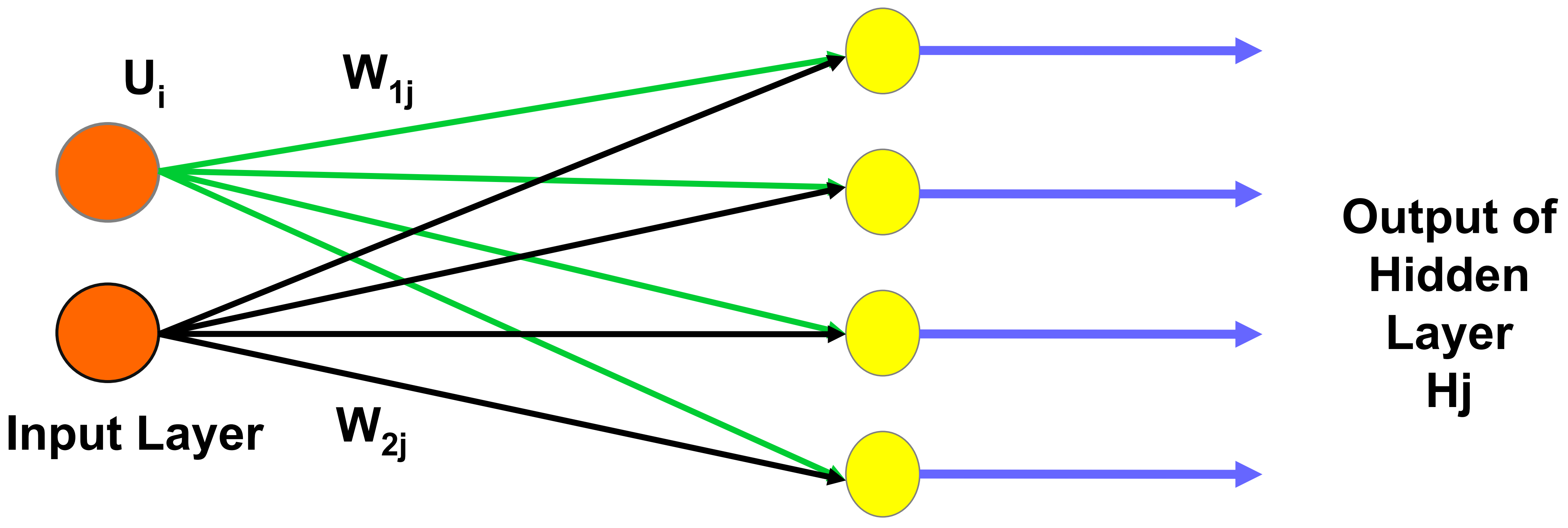
For the weights of the hidden layer node:

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

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

Artificial Neural Network (ANN)

Picture credit: Shamik Ghosh



[5]
[6]

Input vector U (2x1)

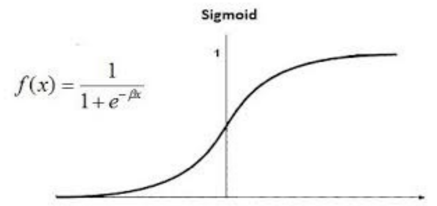
[3 7 1 3]
[-2 8 5 9]

Weight Matrix W_{ij} (2x4)

$\xrightarrow{W_{ij}^T \text{ x } U}$

[3]
[83]
[35]
[69]

Multiplied output vector (4x1)



$\longrightarrow$

[0.95]
[1.]
[1.]
[1.]

Output vector H (4x1)

Starting values of weights

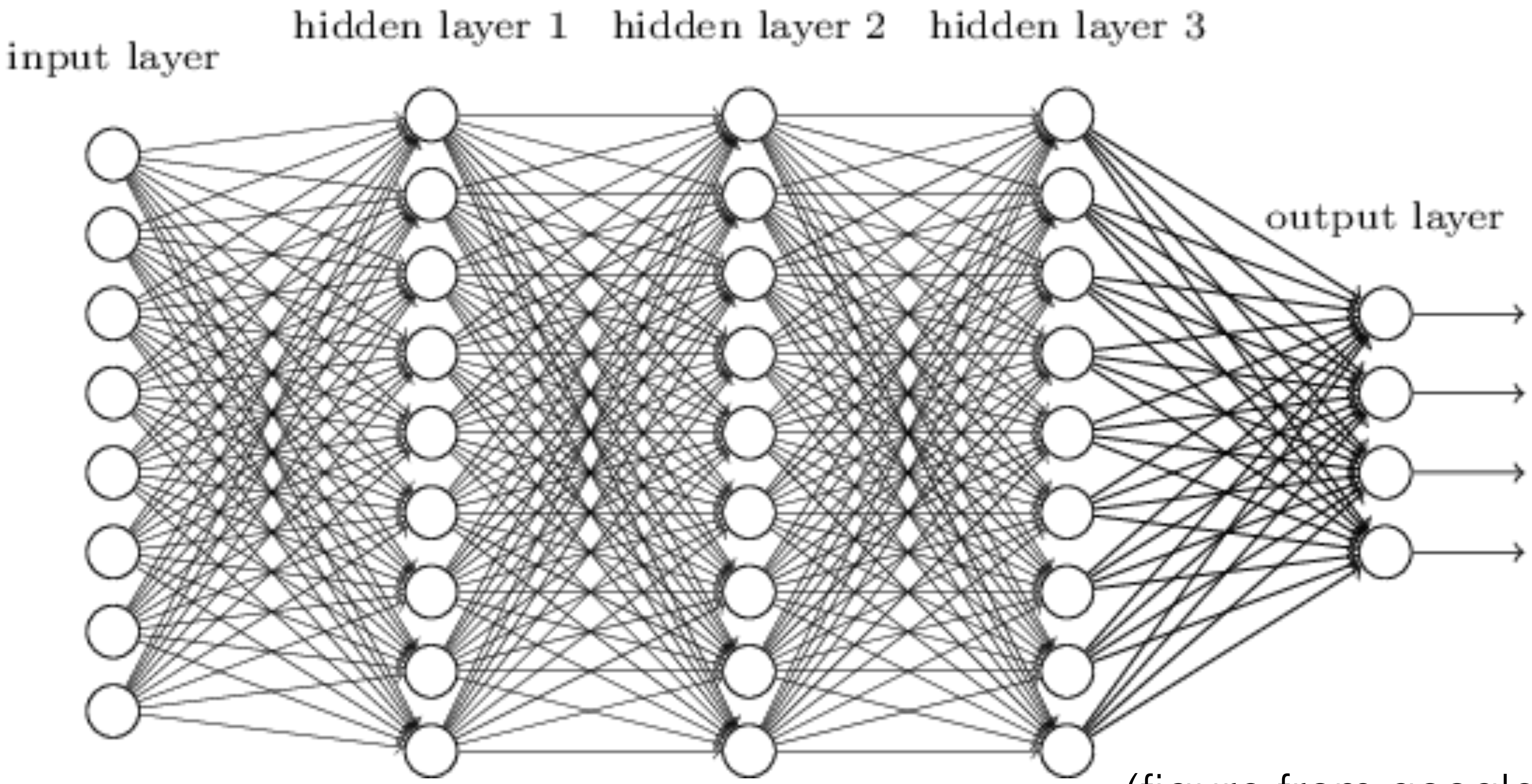
- Usually starting values for weights are chosen to be random values near zero.
- If the weights are near zero, then the operative part of the sigmoid is roughly linear => the neural network collapses into an approximately linear model
- Hence the model starts out nearly linear, and becomes nonlinear as the weights increase.
- Use of exact zero weights leads to zero derivatives and perfect symmetry, and the algorithm never moves.
- Starting instead with large weights often leads to poor solutions.

Stochastic gradient descent

- The updates on weights that we discussed are kind of a batch learning, i.e. parameter updates are averaged over all of the training cases
  - In this case the graph of loss vs epoch is quite smooth, and the loss keeps decreasing with epochs
  - But it can be very slow if the dataset is very large
- Deep learning models use large amount of data (more data => better model)
- In this case one can use stochastic gradient descent, i.e. update weights after passing each event
  - Loss may fluctuate over the training examples (not necessarily always decrease), but decreases in the long run
  - Converges faster for large dataset
- Combine batch processing with stochastic gradient descent, called "mini-batch gradient descent"
  - Use a batch of a fixed number of training examples which is less than the actual dataset

Deep Neural Network

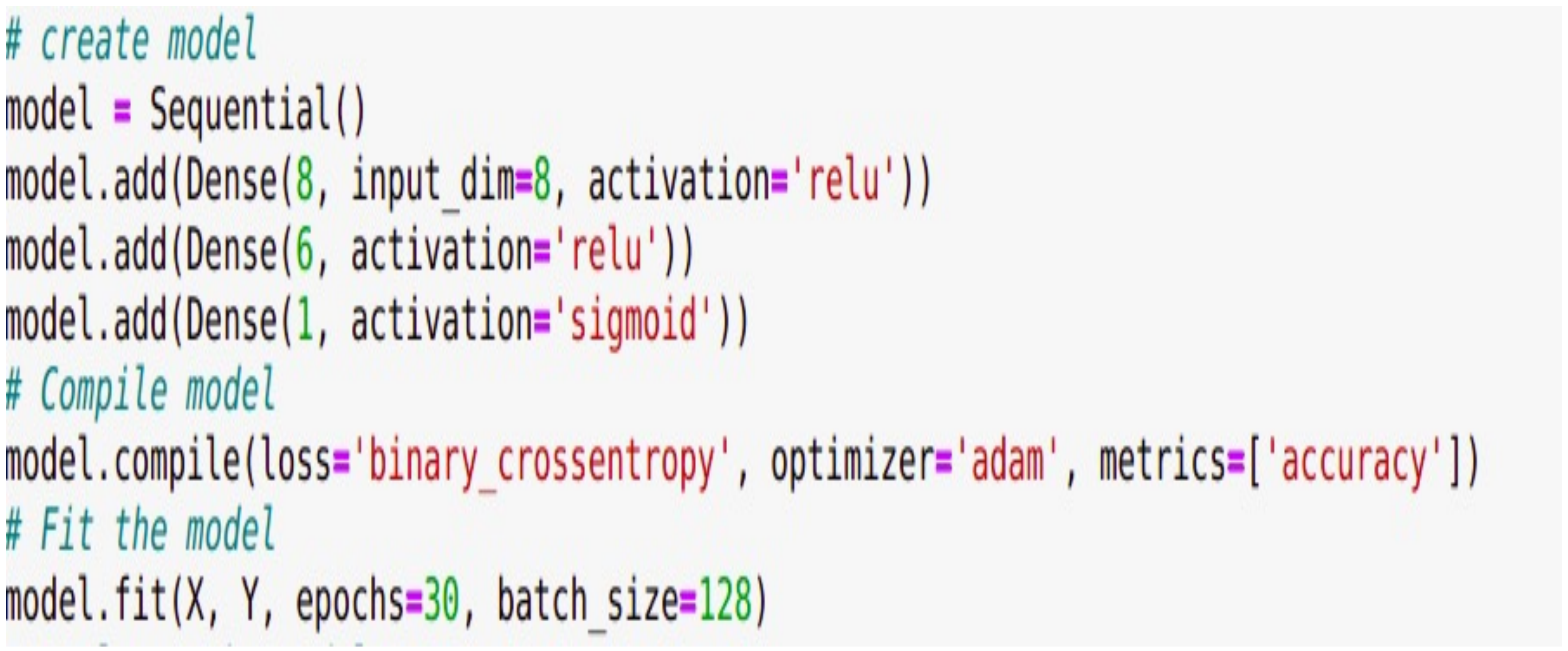
- Networks with many-layer structure - two or more hidden layers
- Deep learning techniques are based on stochastic gradient descent and backpropagation, but also introduce new ideas
- Deep nets have ability to build up a complex hierarchy of concepts



(figure from google)

Practical information

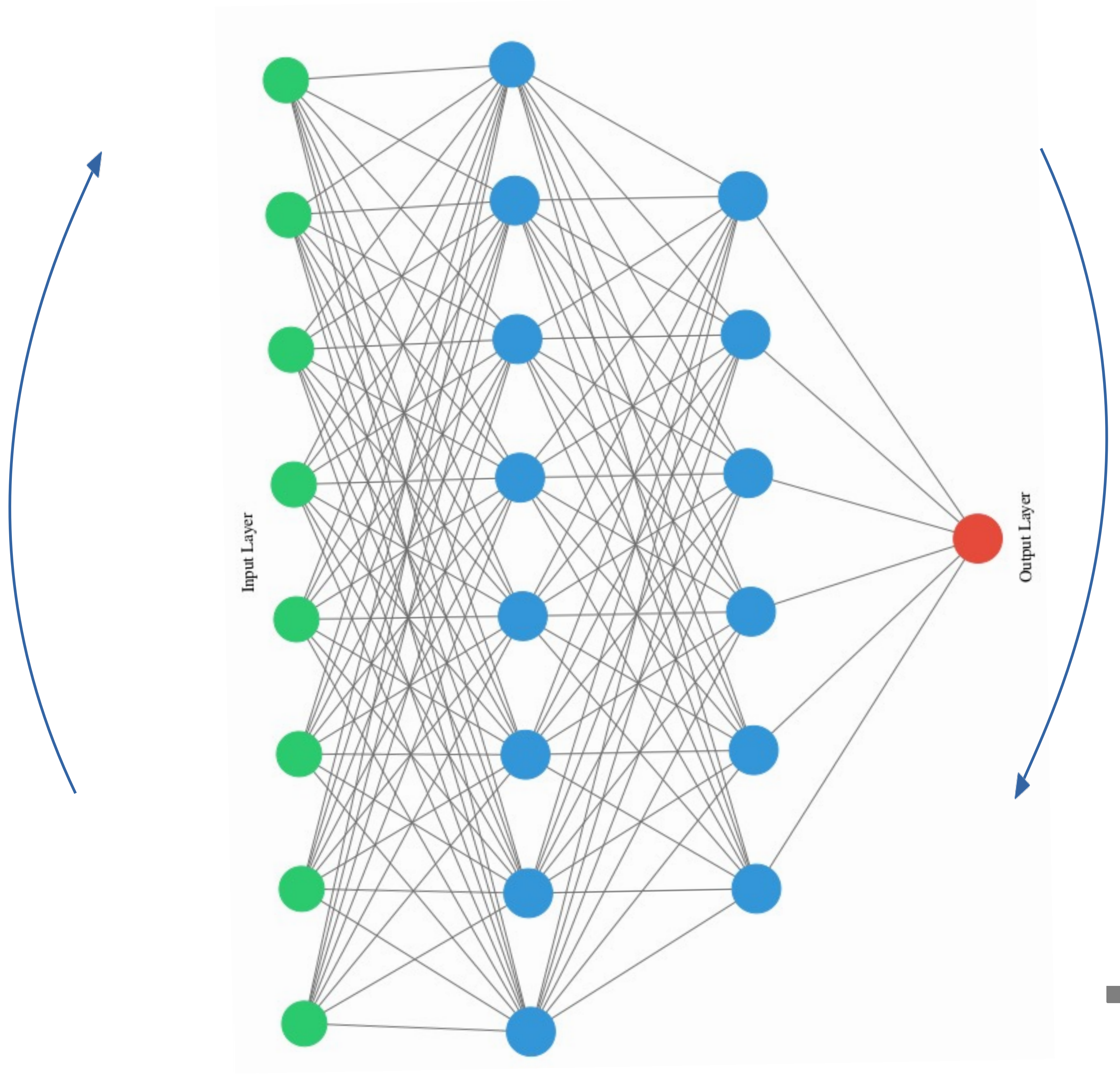
- To construct a neural network model you need to specify the following:
- The number of hidden layers and neurons in each layer.
- Activation function(eg Relu or tanh),Cost function(here cross entropy)
- Batch size,learning rate(here 'adam' optimises the learning rate for Gradient decent)
- Number of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



```
# 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)
```

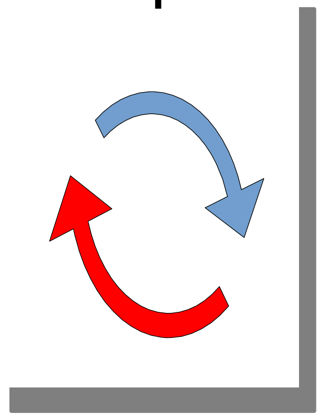
Lets see what this model looks like(next slide)

How this model looks like

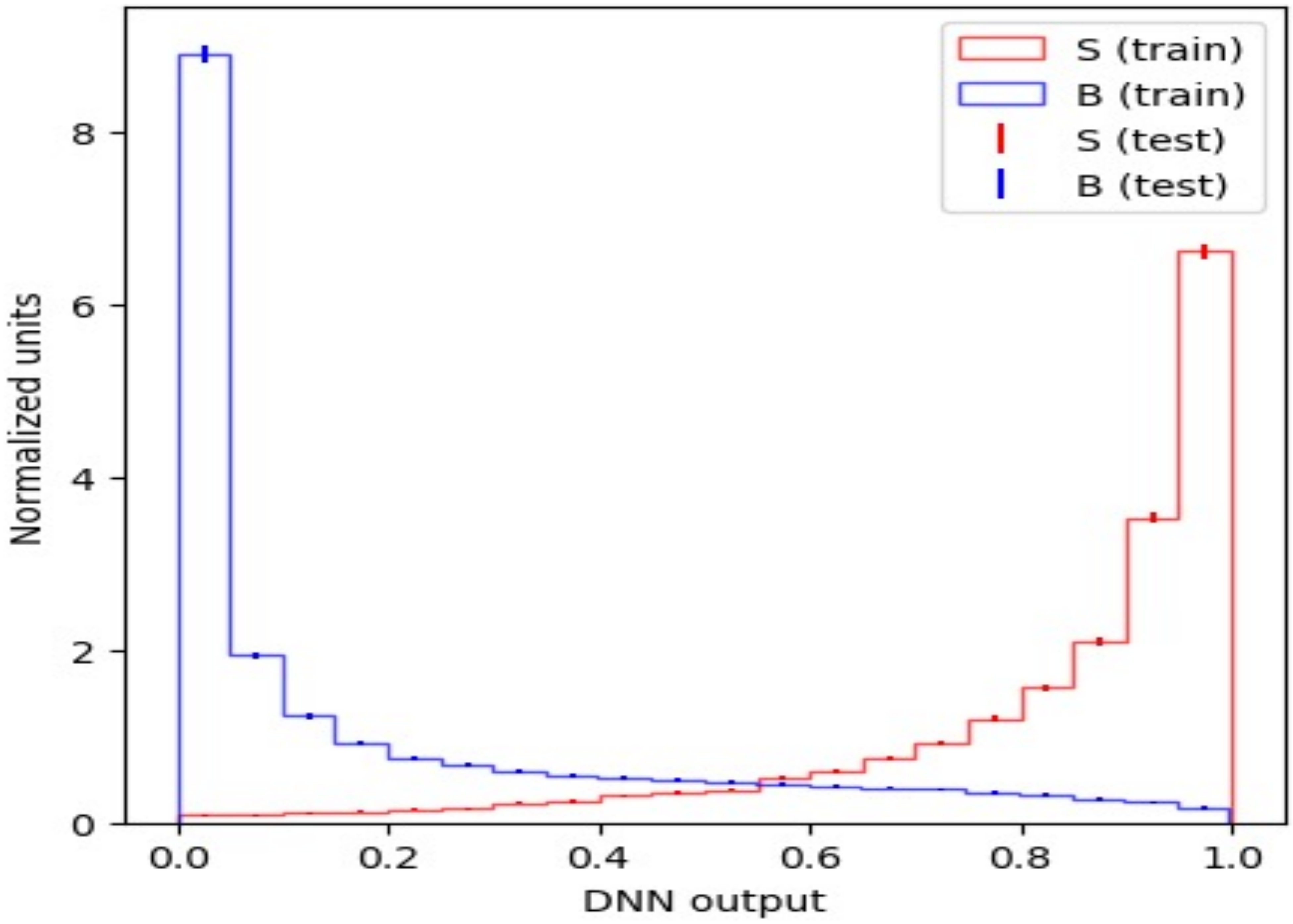


Calculate the output and error

Repeat for N=30 epoch cycles

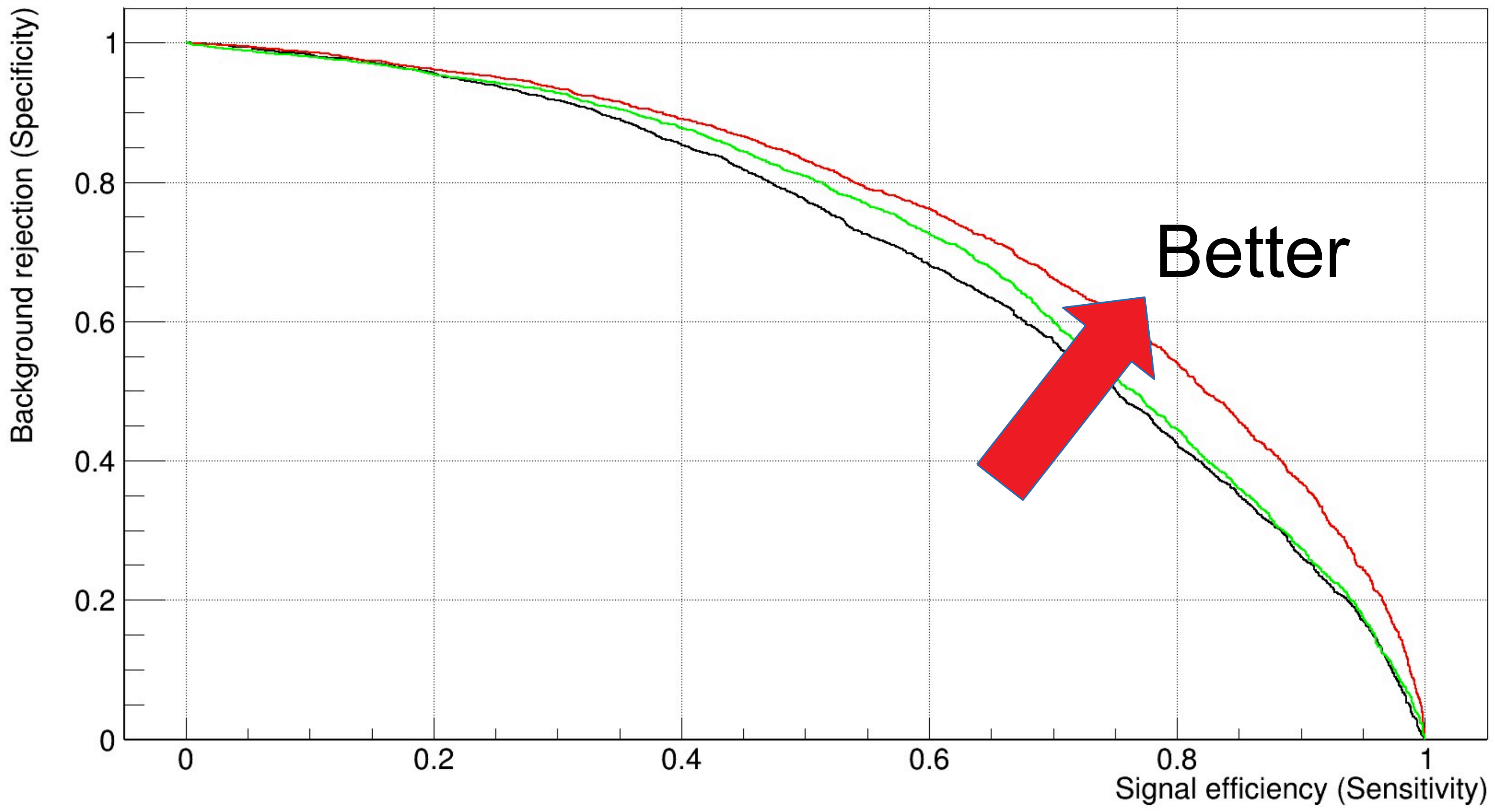


Classifier Output



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)

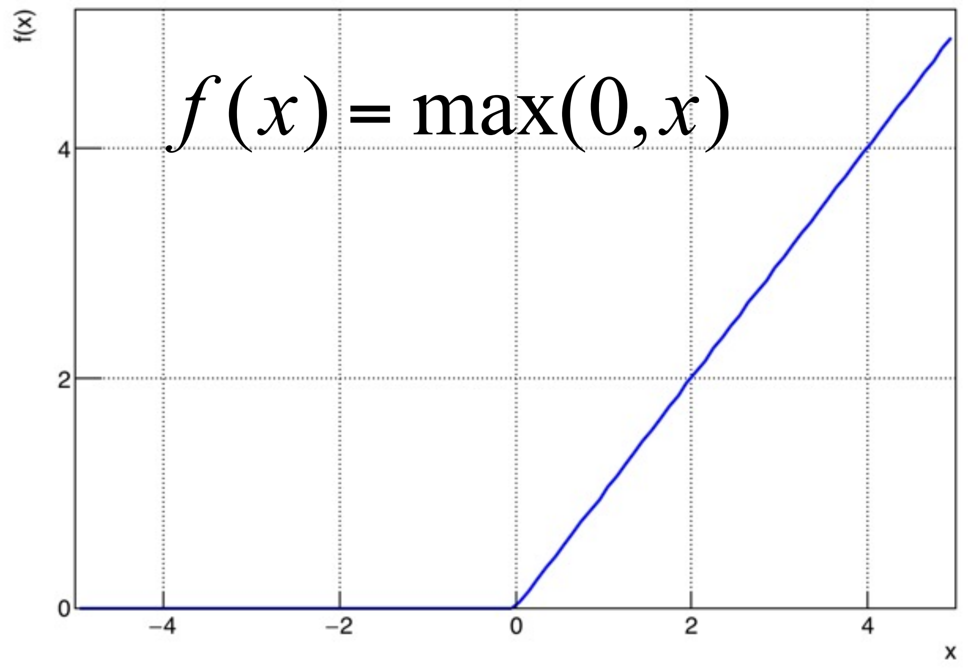


Choice of Activation function

- Usually nonlinear activation functions are preferred as they allow the nodes to learn more complex structures in the data.
  - Widely used ones: sigmoid and tanh
- A general problem with both these functions is that they saturate.
  - i.e. only sensitive to changes around their midpoint of their input
- Saturation happens regardless of whether the summed activation from the node provided as input contains useful information or not.
  - Challenging for the learning algorithm to continue to adapt the weights to improve the performance of the model.
- Layers deep in large networks using these nonlinear activation functions fail to receive useful gradient information.
  - the gradient diminishes dramatically as it is propagated backward through the network.
  - The error may be so small by the time it reaches layers close to the input of the model that it may have very little effect.
- This is called the vanishing gradient problem ➔ prevents deep networks from learning effectively.

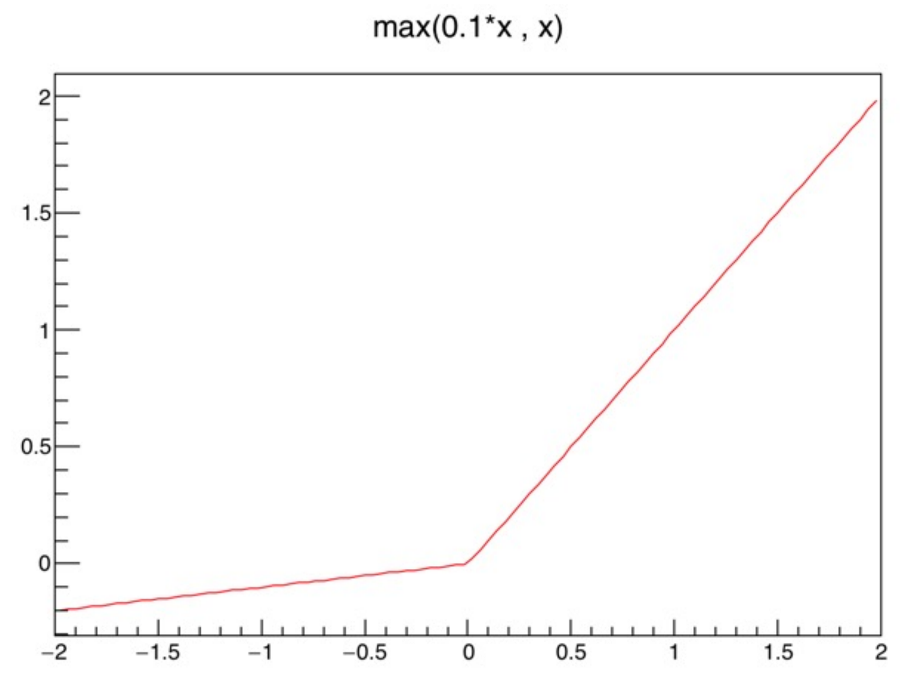
Choice of Activation function

- The solution to these issues are the use of "rectified linear activation unit" or "ReLU"
- It looks and acts like a linear function, but is a nonlinear function allowing complex relationships in the data to be learned
  - A neural network is easier to optimize when its behavior is linear or close to linear.
- This is the default activation function in modern deep learning networks



Choice of Activation function

- Some popular extensions to the ReLU relax the non-linear output of the function to allow small negative values.
- For the ReLU activation function, the gradient is 0 for all the values of inputs that are less than zero, which would deactivate the neurons in that region and may cause dying ReLU problem.
- The Leaky ReLU modifies the function to allow small negative values when the input is less than zero.
- It is an improved version of the ReLU activation function



Validation of NN performance

Usually, the performance of the NN model is accessed using an independent data set, called “test dataset”, which is not used in training the model

Typically, the whole dataset is divided randomly to a “training dataset” and “test dataset” before starting the training.

Training dataset: The sample of data used to fit the model.

Test dataset: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

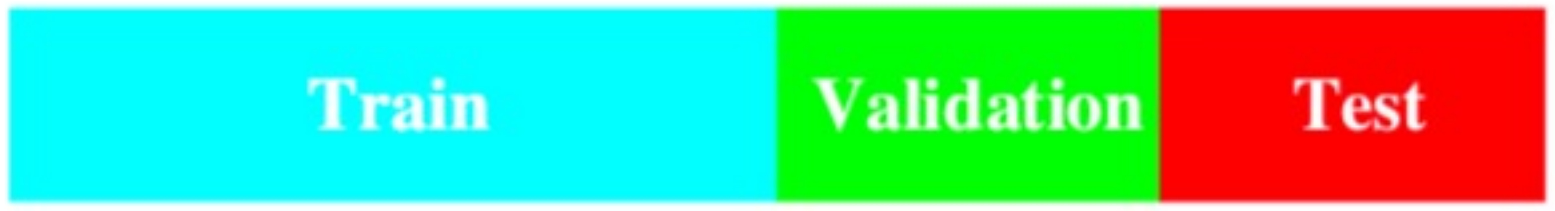
In addition, training dataset can be split further to get a “validation dataset”, that can be used to get an unbiased evaluation of the model while tuning model hyperparameters, e.g. choosing number of hidden units

Validation dataset: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.

> The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.

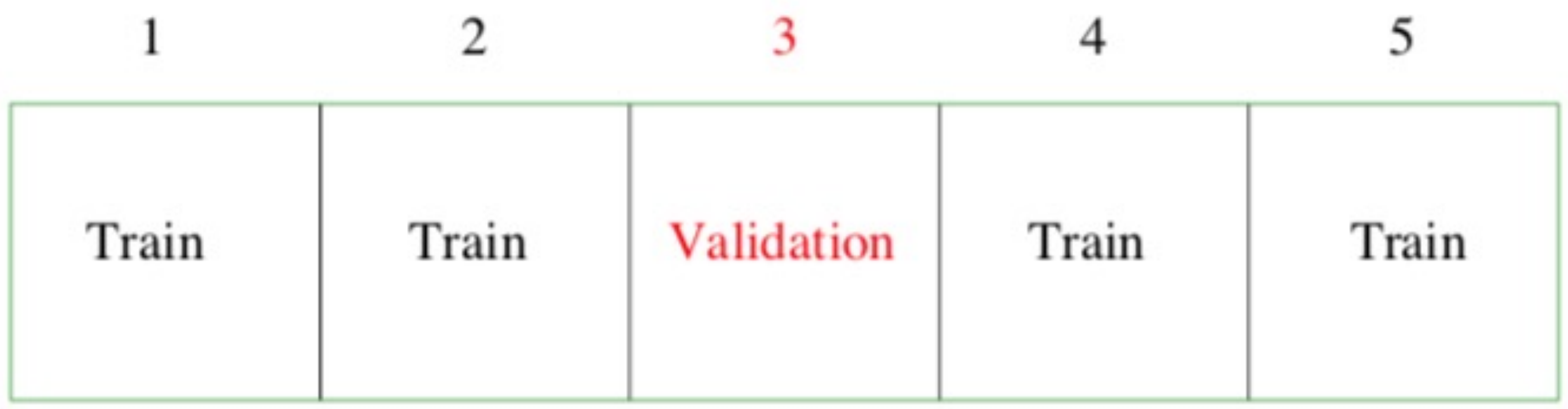
Validation of NN performance

A typical split might be 50% for training, and 25% each for validation and testing:



Cross-Validation

- The simplest and most widely used method for estimating prediction error
- If dataset size is not large enough, it is difficult to set aside a validation set to assess the performance of prediction model
- K-fold cross- validation uses part of the available data to fit the model, and a different part to test it.
- e.g., for K = 5

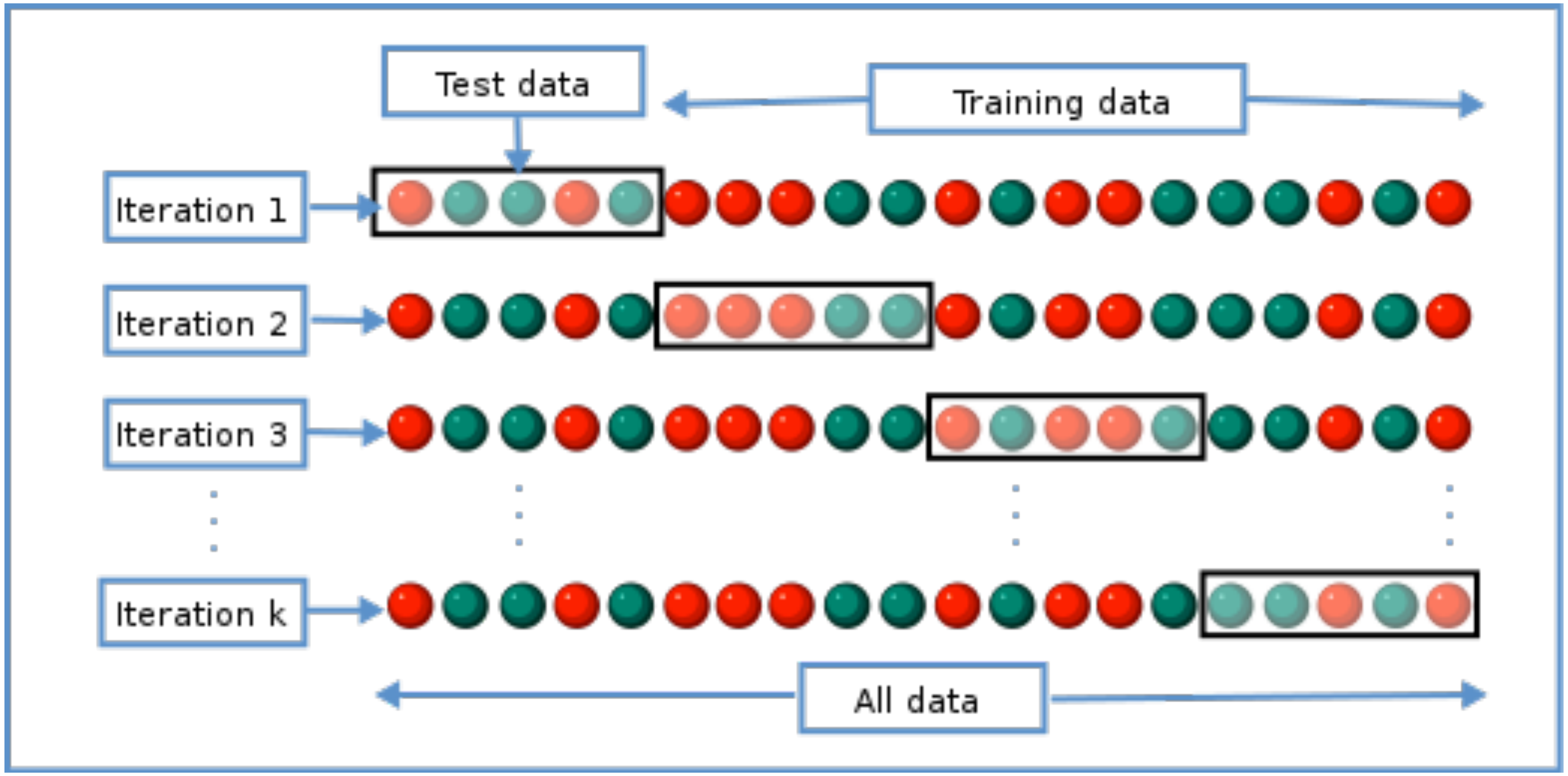


Repeat this for k = 1,2,...,K and combine the K estimates of prediction error.

$$\mathrm{CV}(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(x_i)).$$

Typical choices of K are 5 or 10

K-Fold Cross Validation

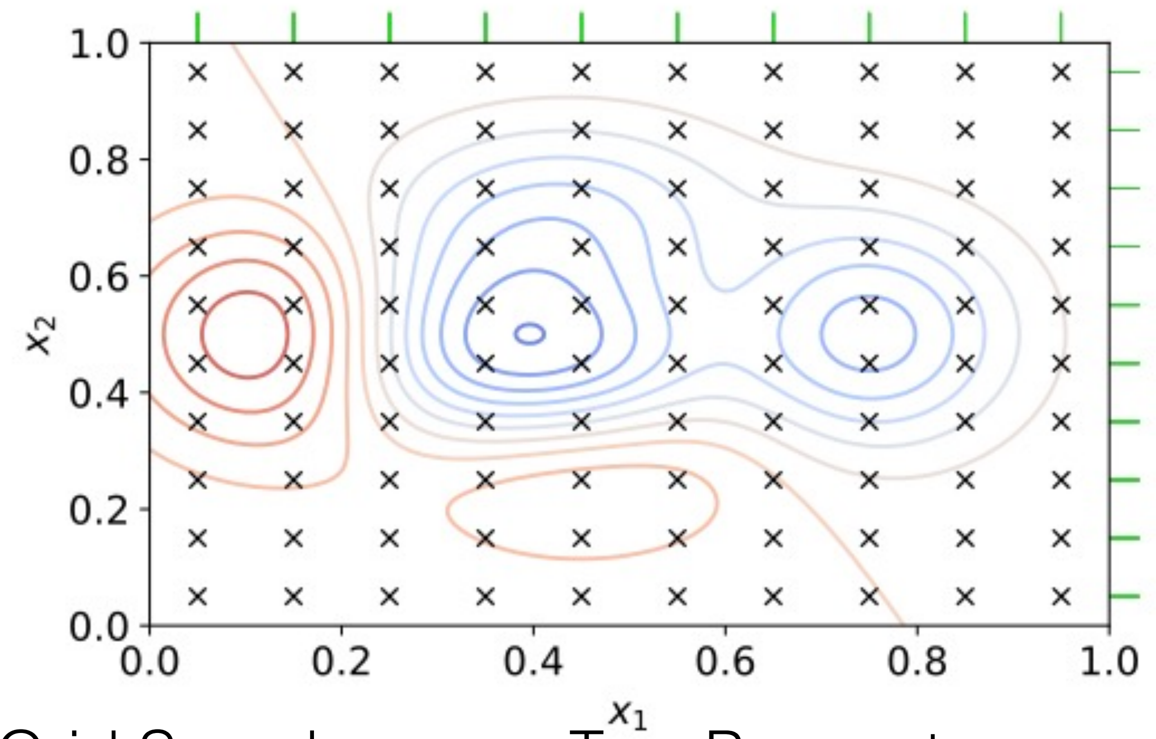


Ref: https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/

Hyperparameter Tuning

- Various techniques to perform hyperparameter tuning

  e.g. Manual Search, Random Search, Grid Search, Bayesian Optimizations, etc
- e.g., one can use GridSearchCV in scikit-learn, that performs grid-search with k-fold cross validation



Grid Search across Two Parameters

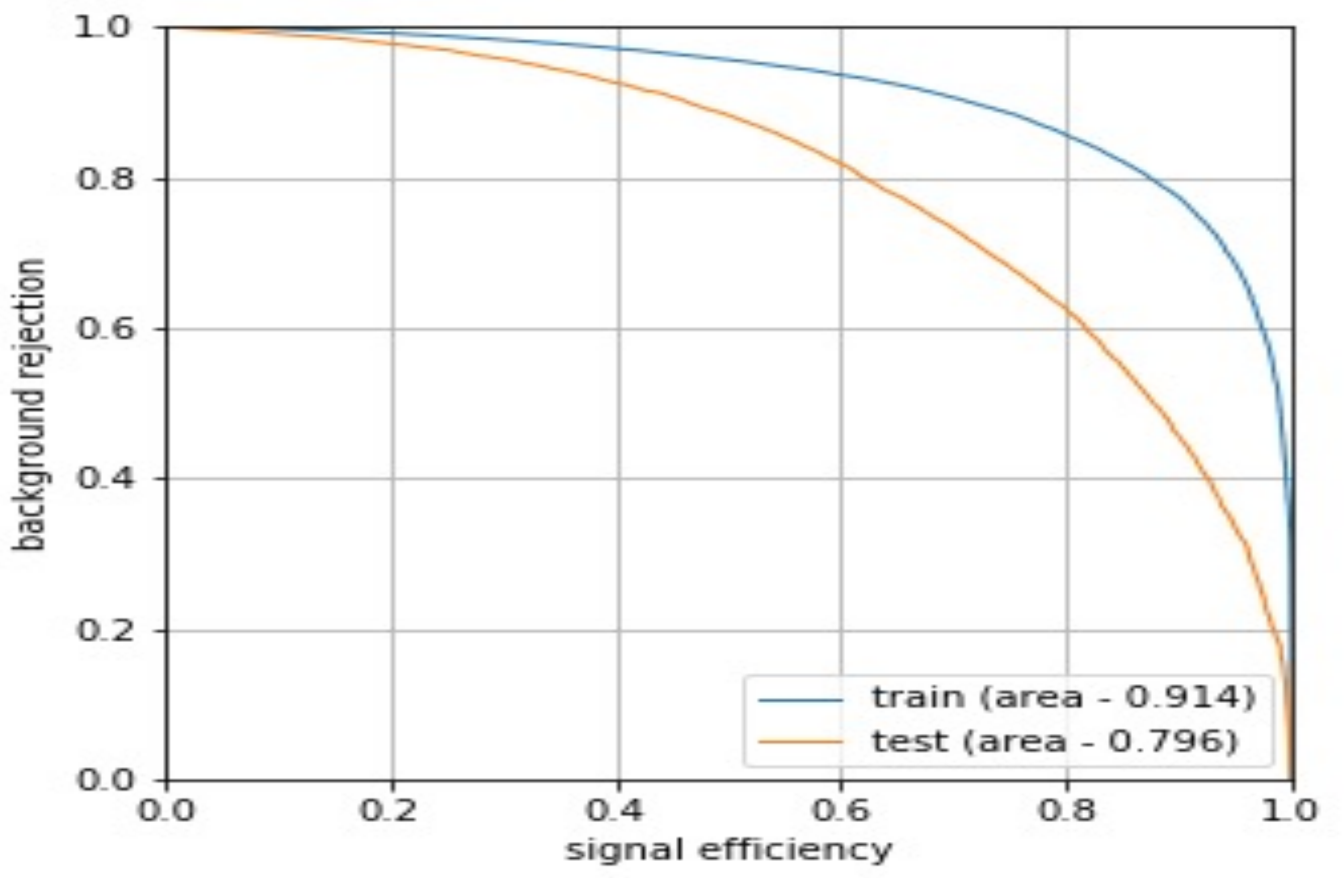
Ref: https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/)

Overtraining

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

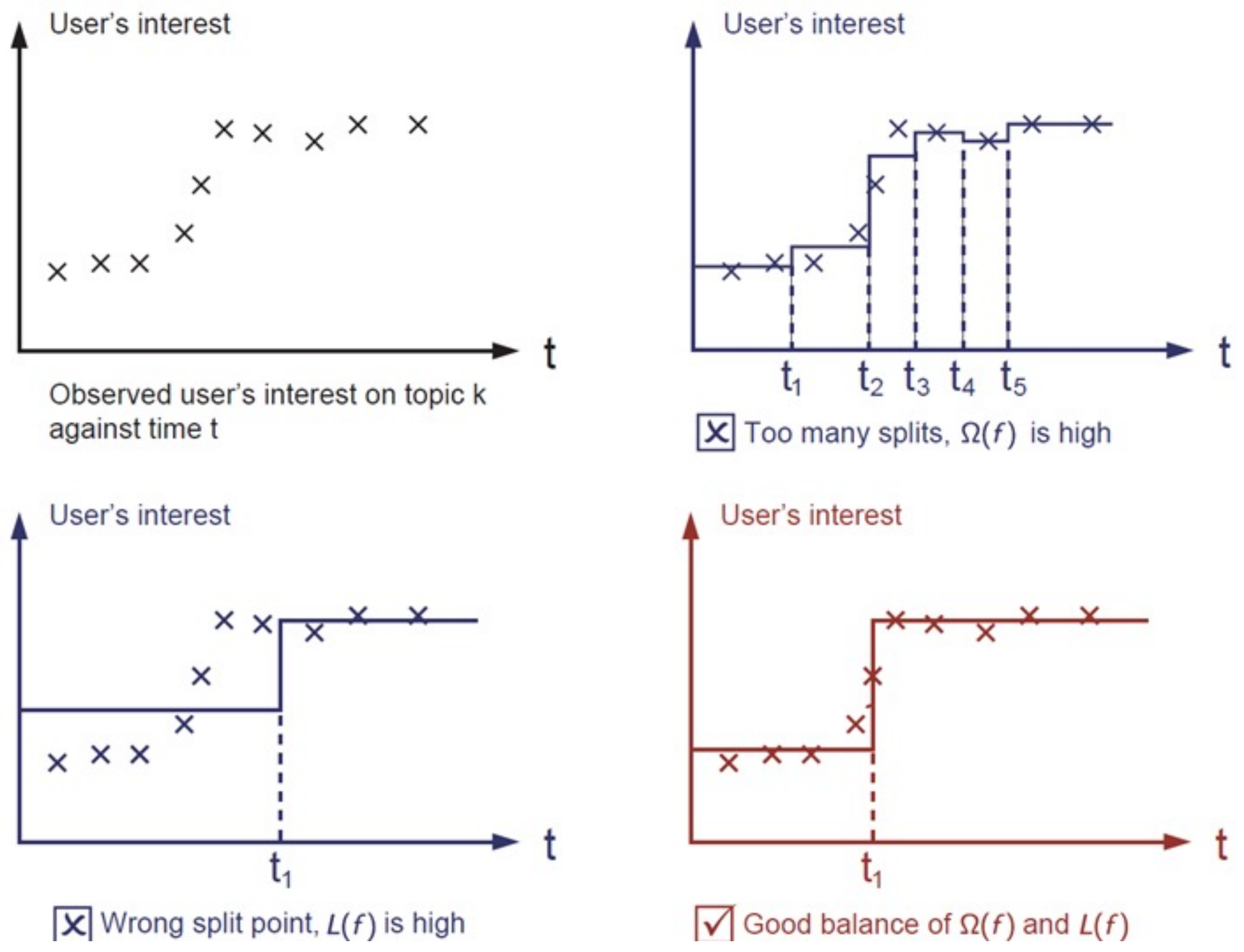
- Leads to poor performance in other samples
- Mostly due to small training sample size, or data that is too homogenous
- Over sensitive to some features of the training data

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



Regularization

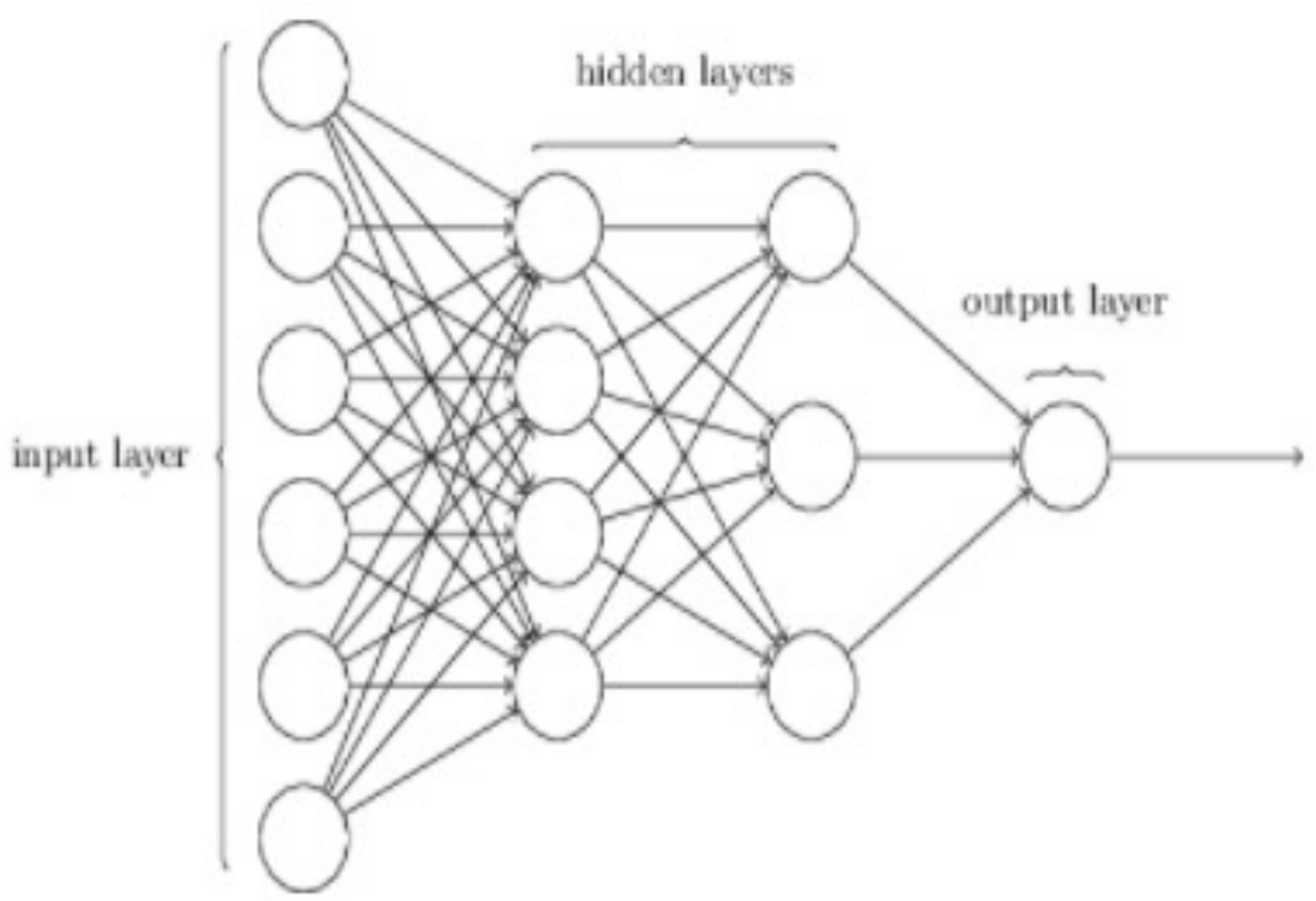
- Fitting the training data too well can lead to overfitting and degrade future predictions
- The regularization controls the complexity of the model, which helps us to avoid overfitting

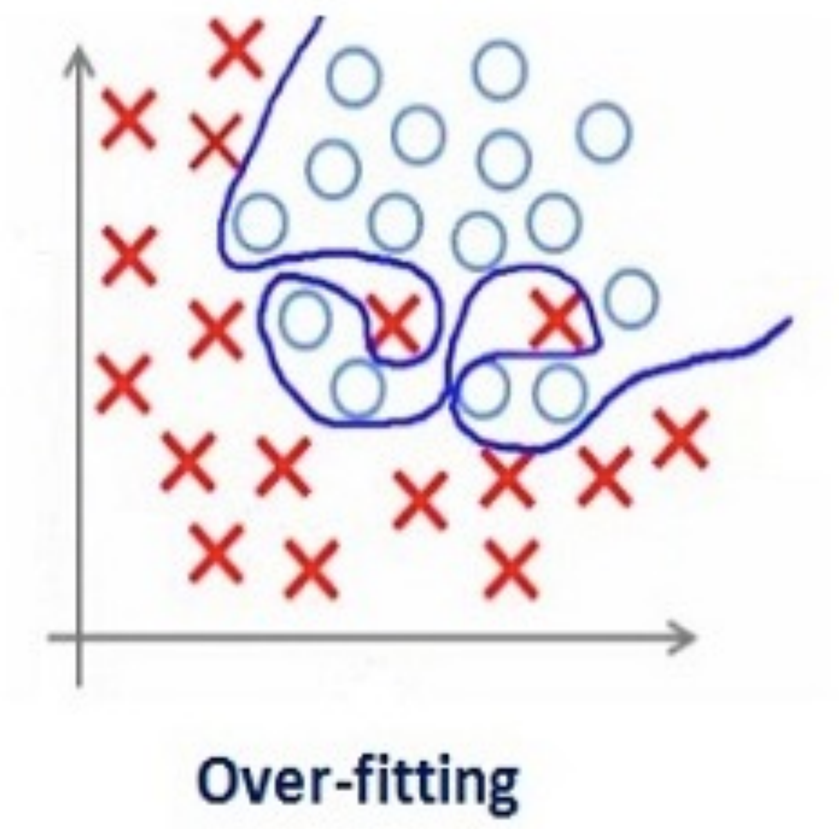

User's interest
t
Observed user's interest on topic k against time t
User's interest
t_1 t_2 t_3 t_4 t_5
t
☒ Too many splits, $\Omega(f)$ is high
User's interest
t_1
t
☒ Wrong split point, $L(f)$ is high
User's interest
t_1
t
☑ Good balance of $\Omega(f)$ and $L(f)$

(ref: XGBoost userguide)

Regularization

Increase in complexity of the model can lead to overfitting



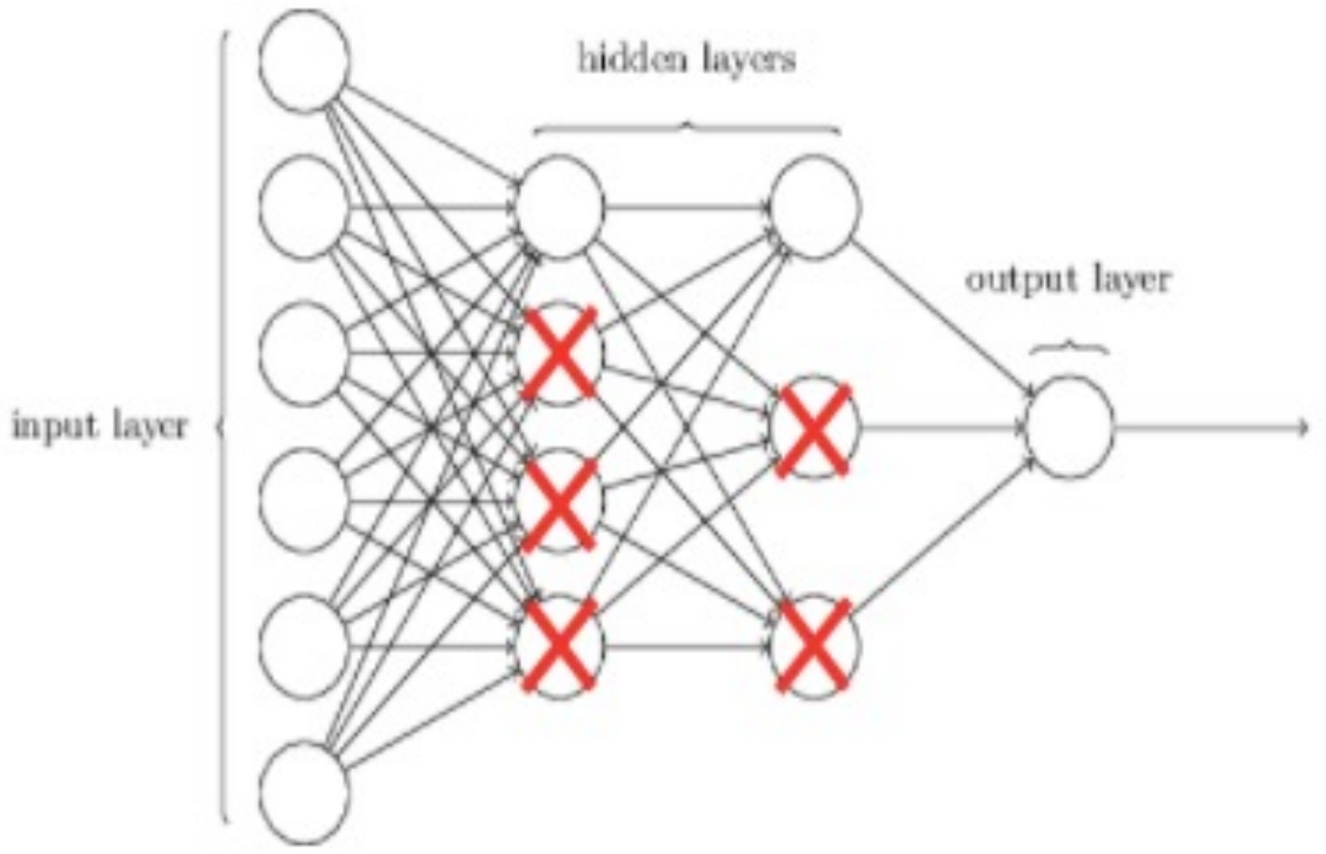


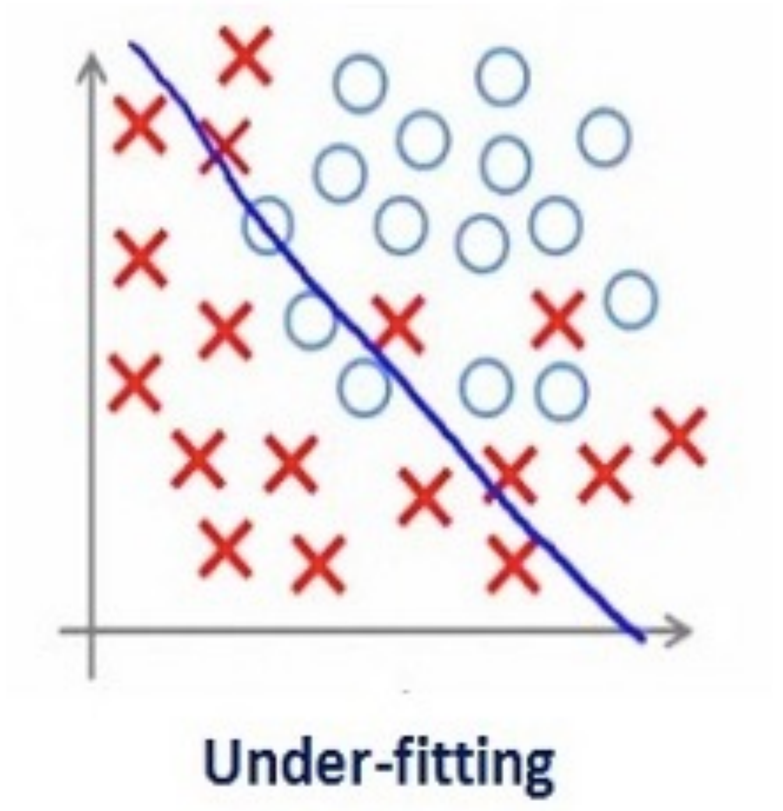
Ref:
https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

Regularization

Regularization penalizes the coefficients (weight matrices of the nodes)

Assume regularization coefficient is so high that some of the weight matrices are nearly equal to zero ➔ much simpler network



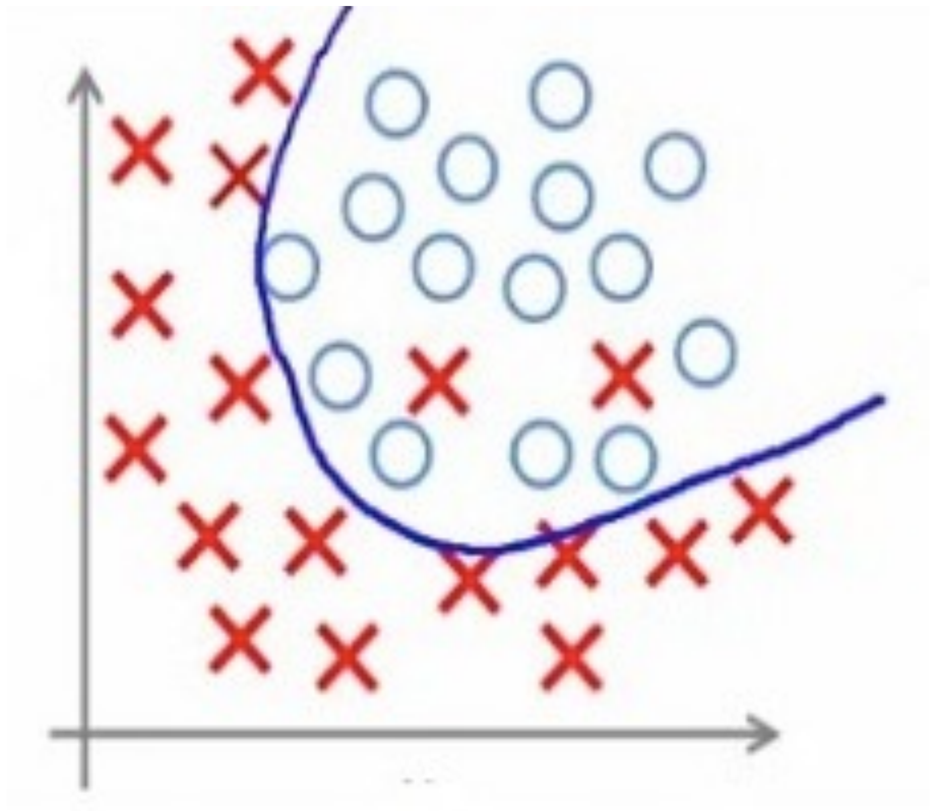


Ref: https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

Regularization

Regularization penalizes the coefficients (weight matrices of the nodes)

Need to optimize the value of regularization coefficient in order to obtain a well-fitted model, e.g. like this



Appropriate-fitting

Ref: https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

Regularization Techniques

Different regularization techniques:

1. L1 & L2 Regularization
2. Dropout
3. Early stopping
4. Data Augmentation

L1 & L2 Regularization

- Add regularization term to the cost function
  - Reduces complexity of the network by penalizing weights

$$obj = L(w) + \Omega(w) \longleftarrow \boxed{\text{Regularization term}}$$

- L1 Regularization:

$$obj = L(w) + \lambda \sum |w|$$

Penalizes weights to remain small. Can make some of the weights to be zeros

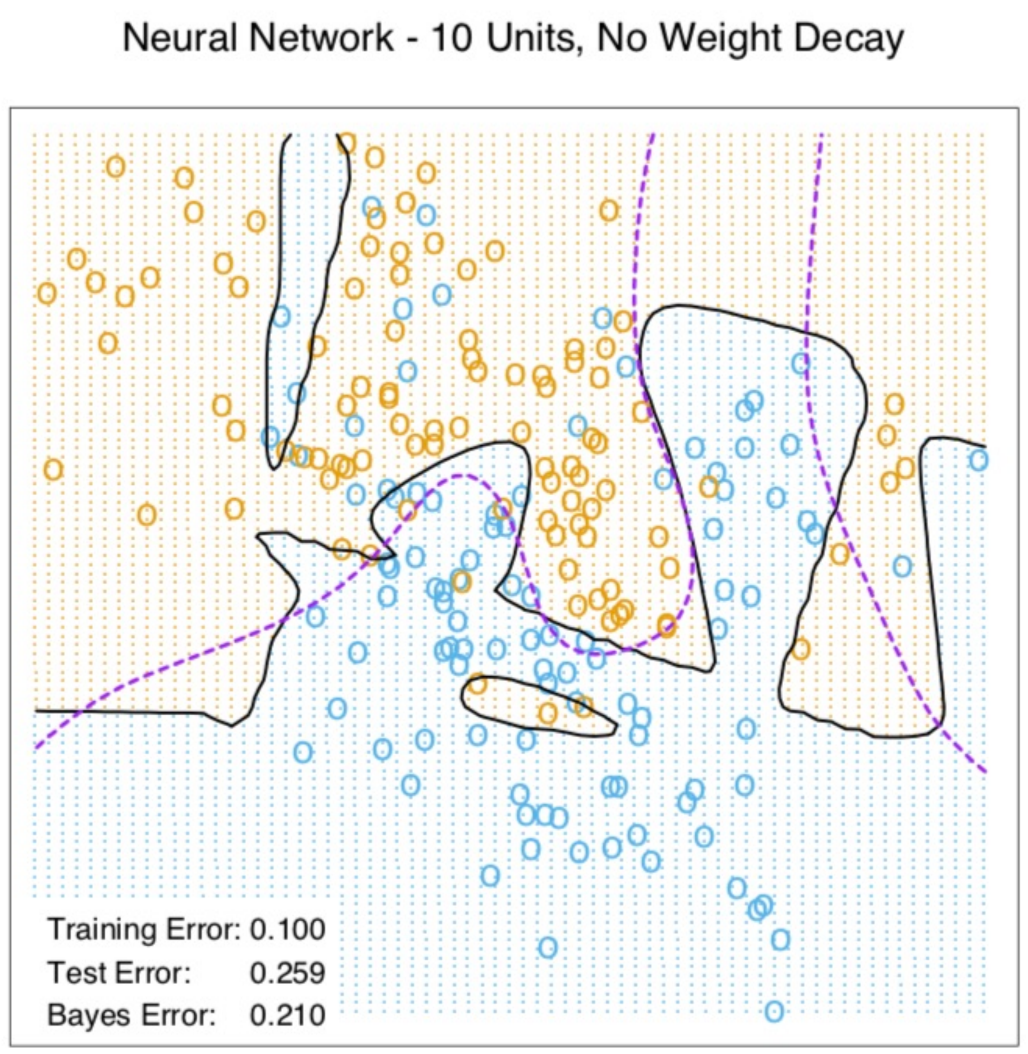
- L2 Regularization:

$$obj = L(w) + \lambda \sum |w|^2$$

Makes weights close to zero, but not zero.
aka "weight decay"

λ is a hyperparameter, large λ => small w

Weight decay



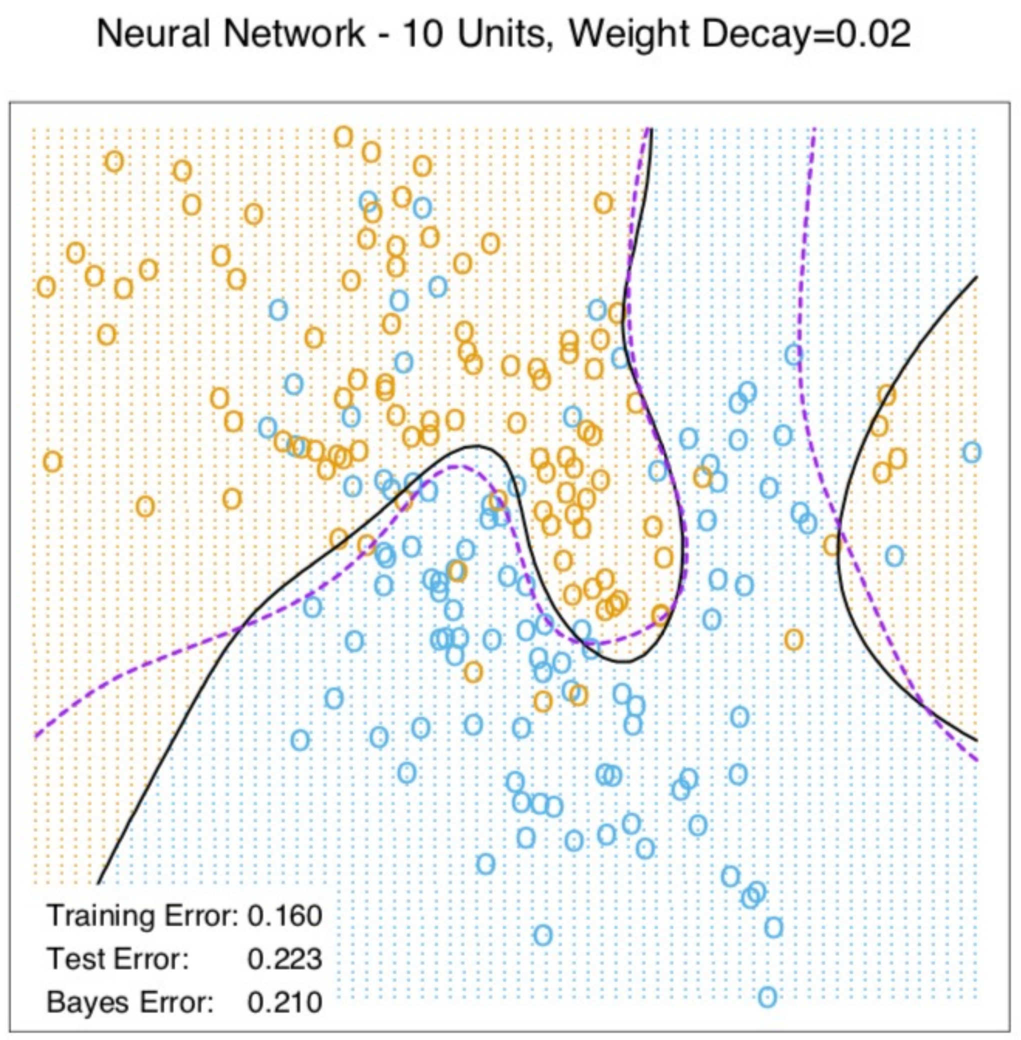


FIGURE 11.4. *A neural network on the mixture example of Chapter 2. The upper panel uses no weight decay, and overfits the training data. The lower panel uses weight decay, and achieves close to the Bayes error rate (broken purple boundary). Both use the softmax activation function and cross-entropy error.*

Ref: T. Hastie et al.

Weight decay

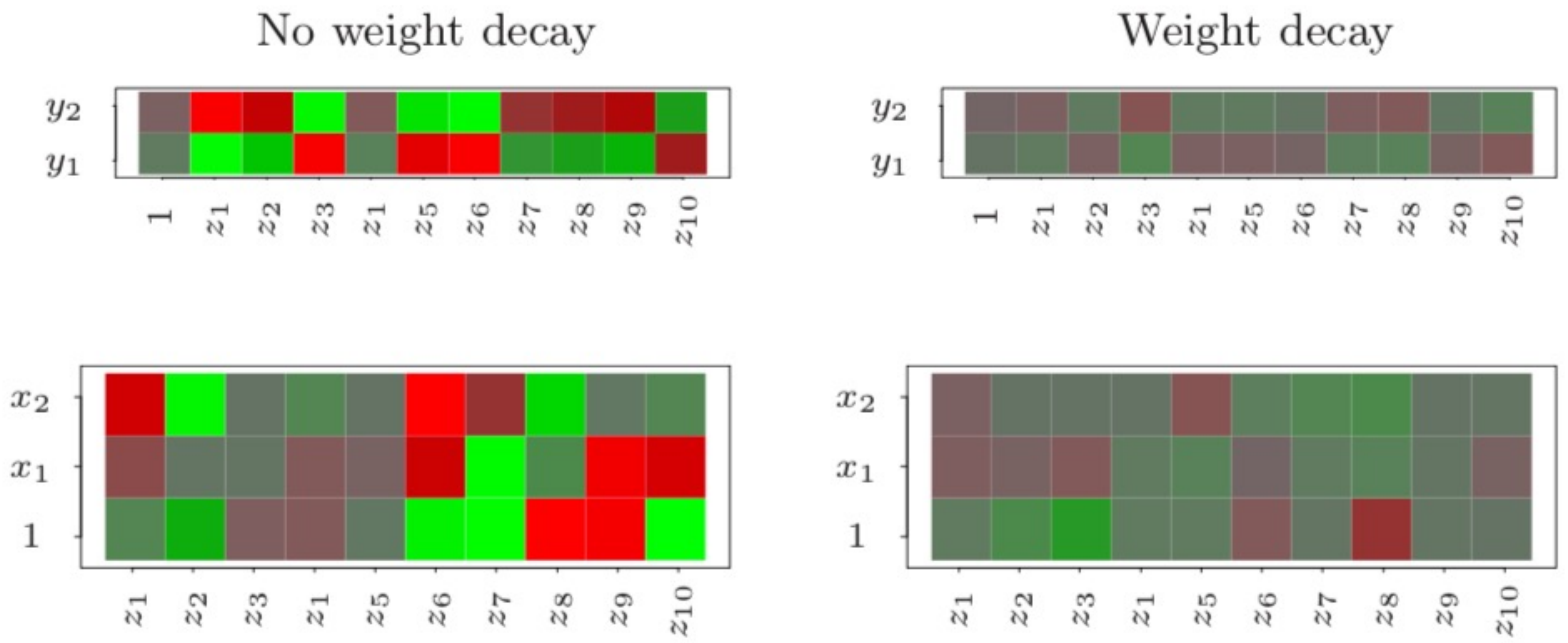


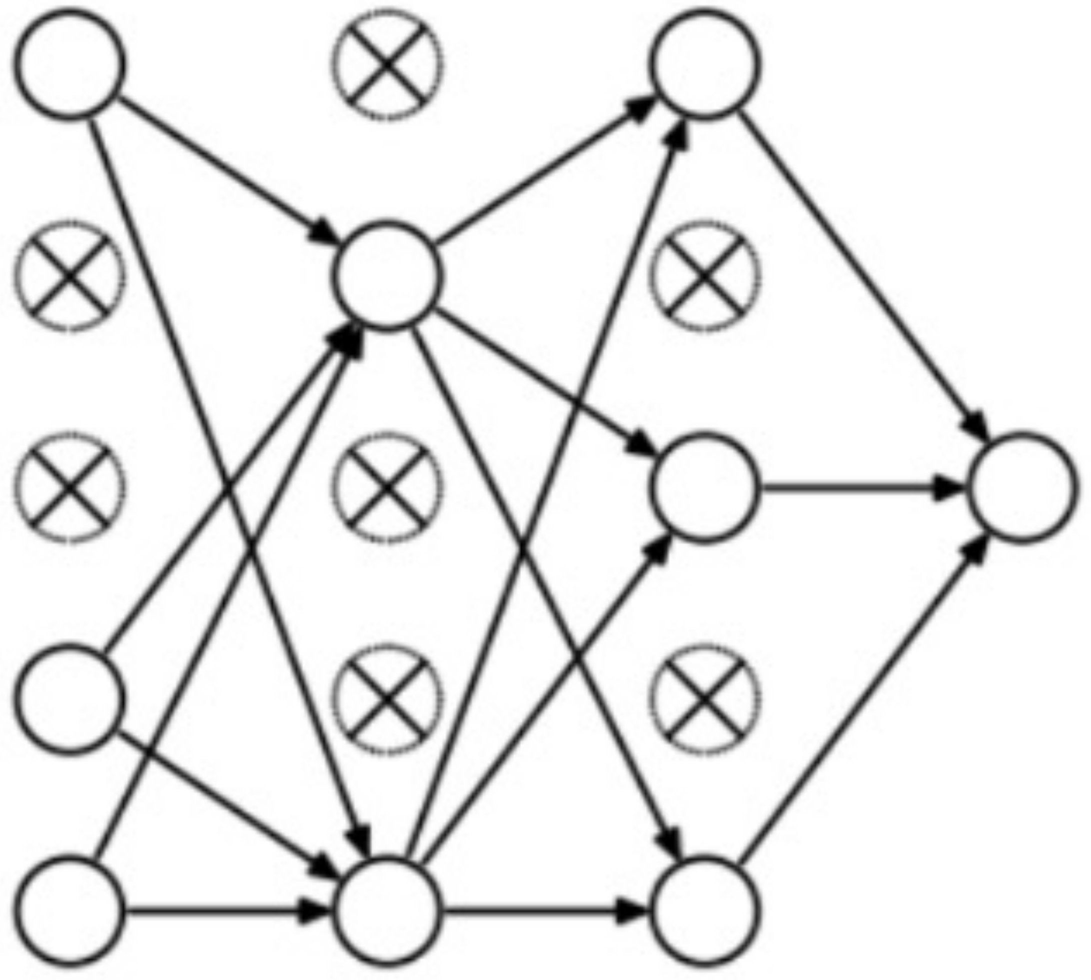
FIGURE 11.5. *Heat maps of the estimated weights from the training of neural networks from Figure 11.4. The display ranges from bright green (negative) to bright red (positive).*

Ref: T. Hastie et al.

Dropout

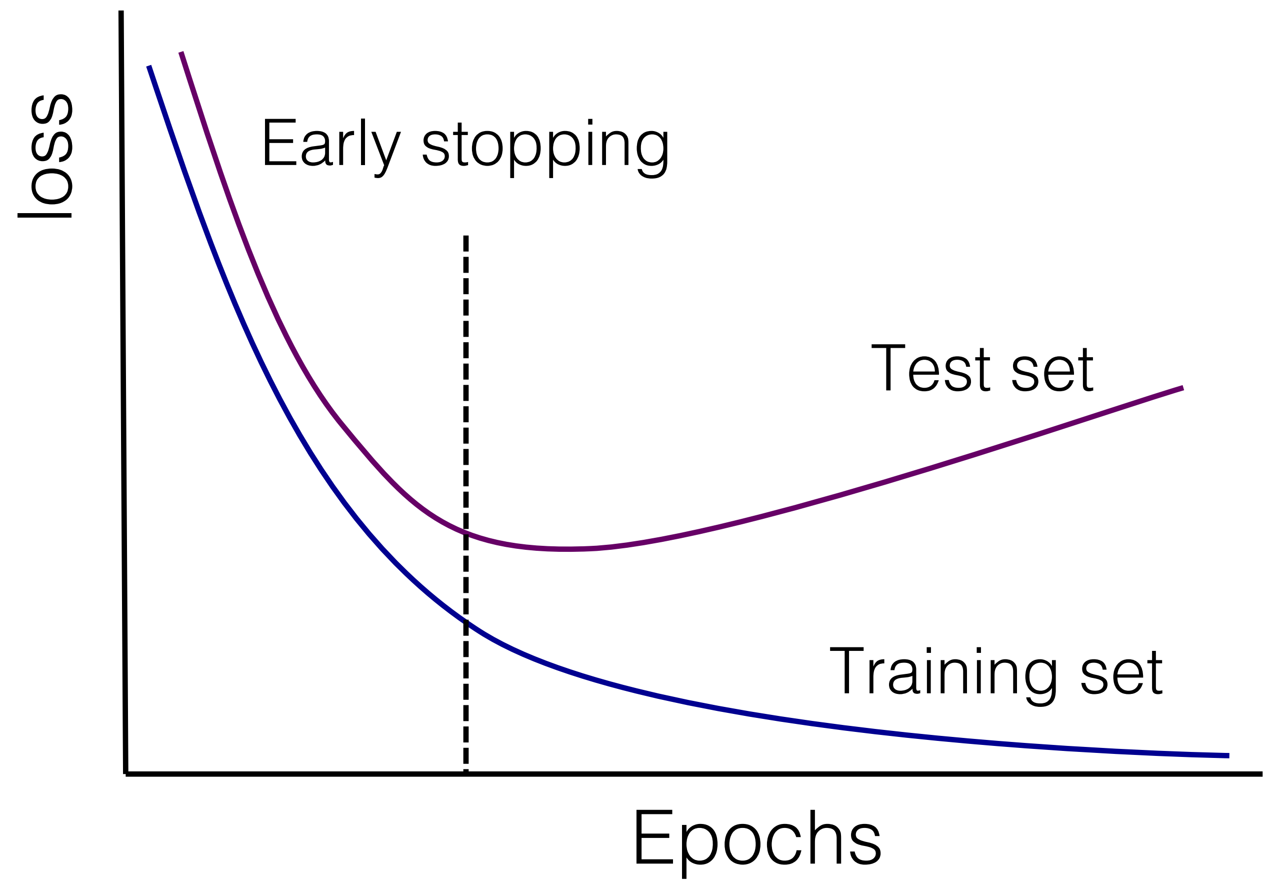
- You can randomly drop a predefined set of neurons during each epoch.
- So, each iteration has a different set of nodes
- Can be thought of as an ensemble technique in machine learning

Ref: https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/



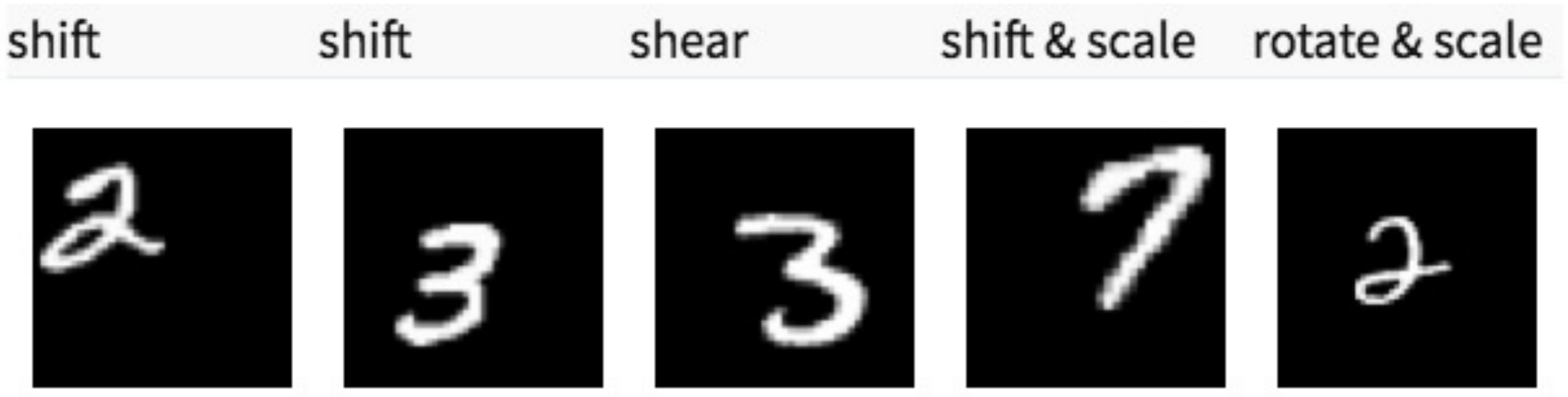
Early Stopping

- In most neural network packages like Keras or DNN in TMVA, the training set can divided into a training and validation set. Monitor the valiation error for each epoc and stop when the valiation error is no more improving (or starts to increase).



Augmenting training sample

- The simplest way to reduce overtraining is by increasing the training statistics
  - Not always possible, labeled data are too costly
- Training sample can be augmented by generating new events from the existing training sample (e.g. by process of flipping, rotating, scaling, shifting etc.. of the images)



Ref:
https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

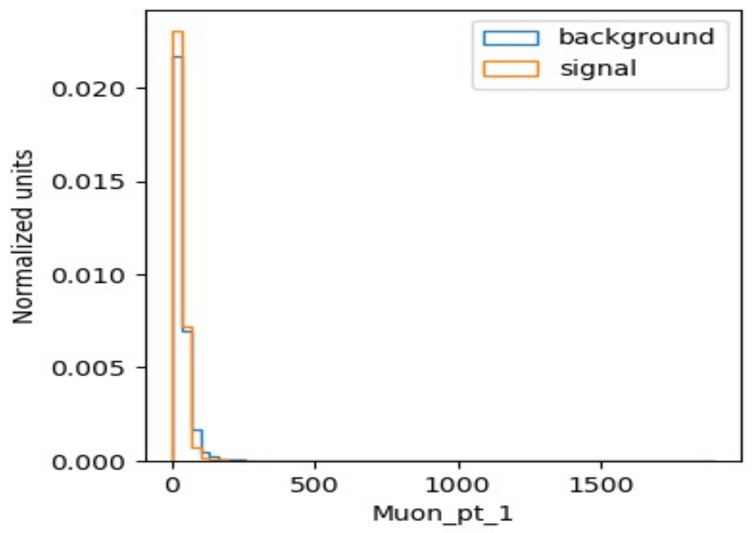
Scaling of the Inputs

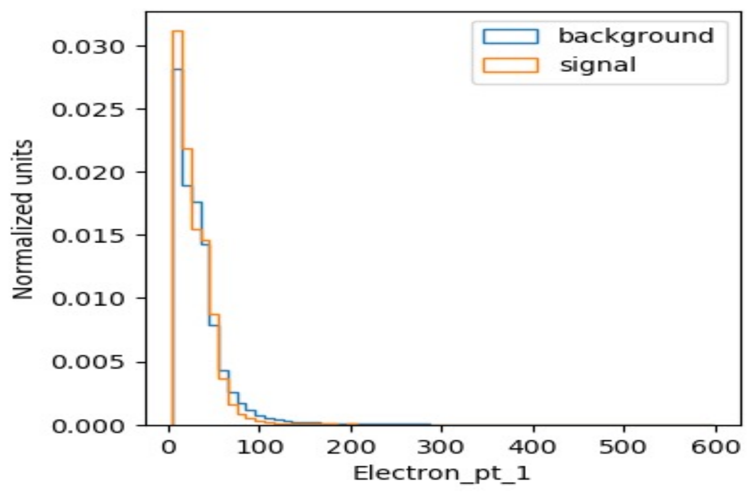
- The scaling of the inputs determines the effective scaling of the weights in the bottom layer
  - Can have a large effect on the quality of the final solution
- Usually, it is best to standardize all inputs to have mean zero and standard deviation one
  - Ensures all inputs are treated equally in the regularization process,
  - Allows one to choose a meaningful range for the random starting weights.

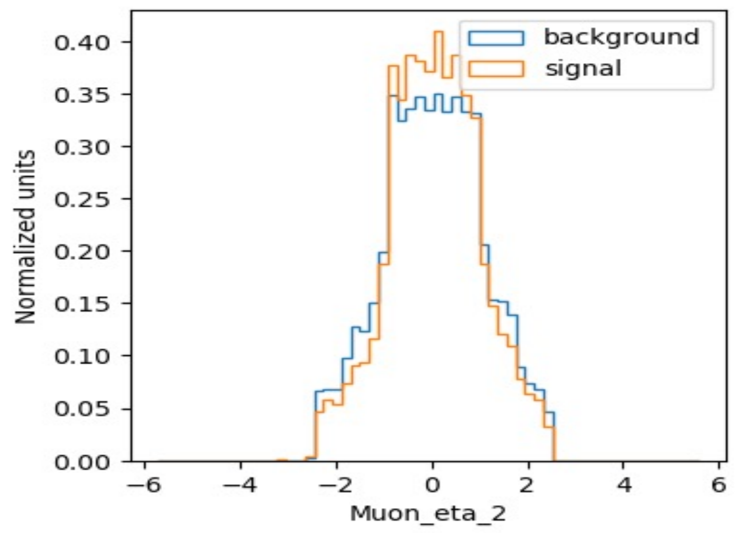
Example from Physics

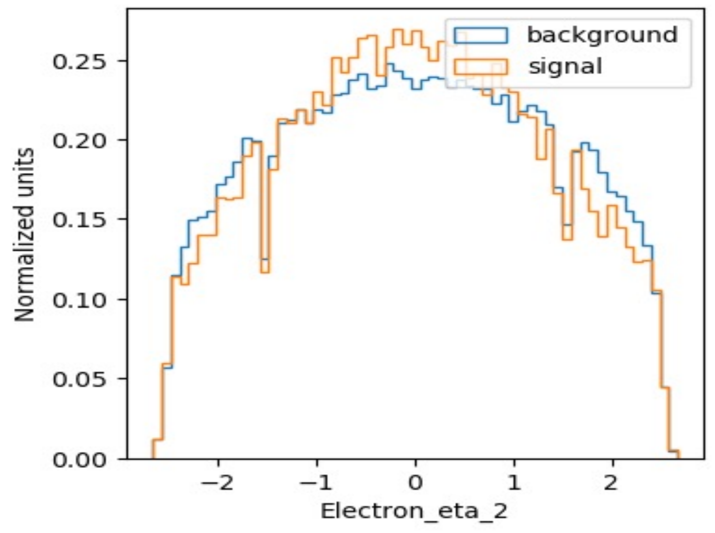
Separate $H \rightarrow ZZ^* \rightarrow 2e2\mu$ (signal*) from $ZZ \rightarrow 2e2\mu$ (background*)

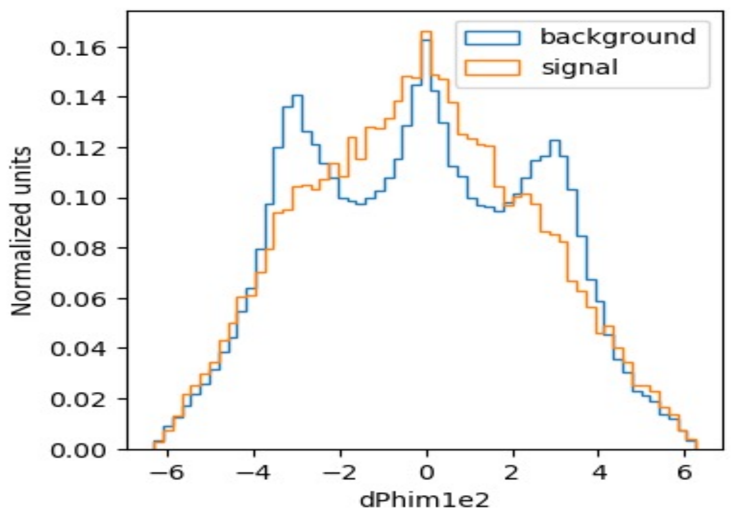
Input Variables used : p_T of individual leptons, Delta Phi combinations of each lepton pair, eta of each lepton(total 14 variables)

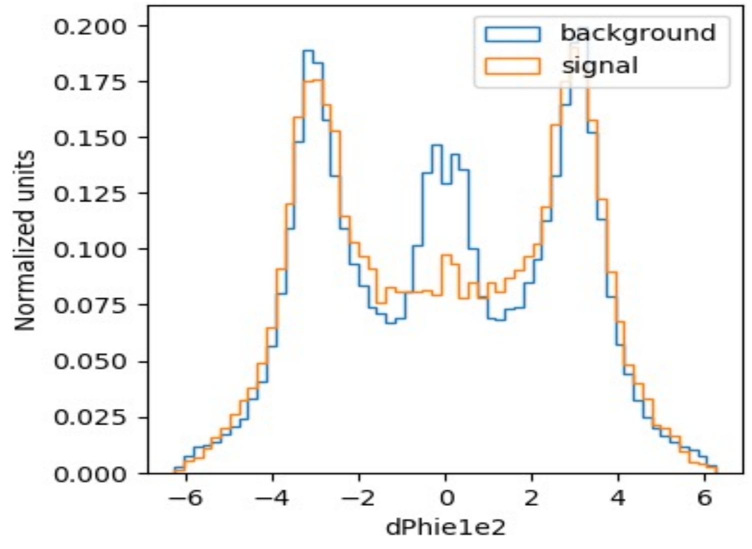








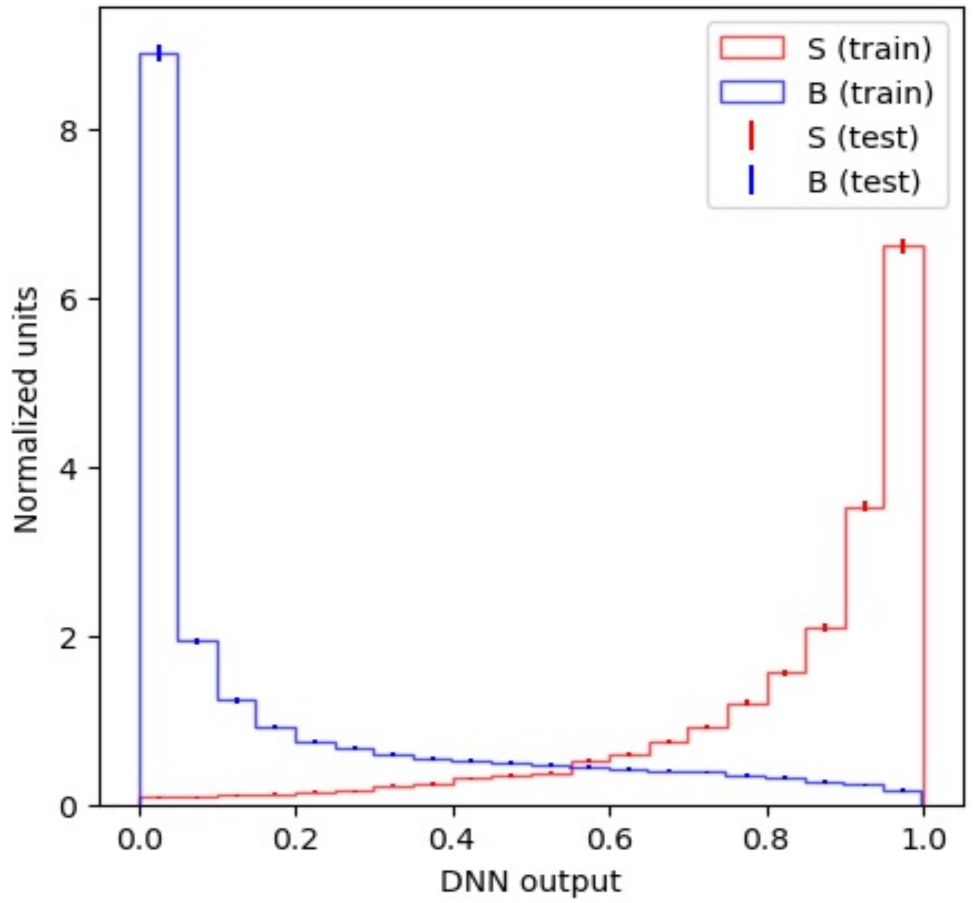


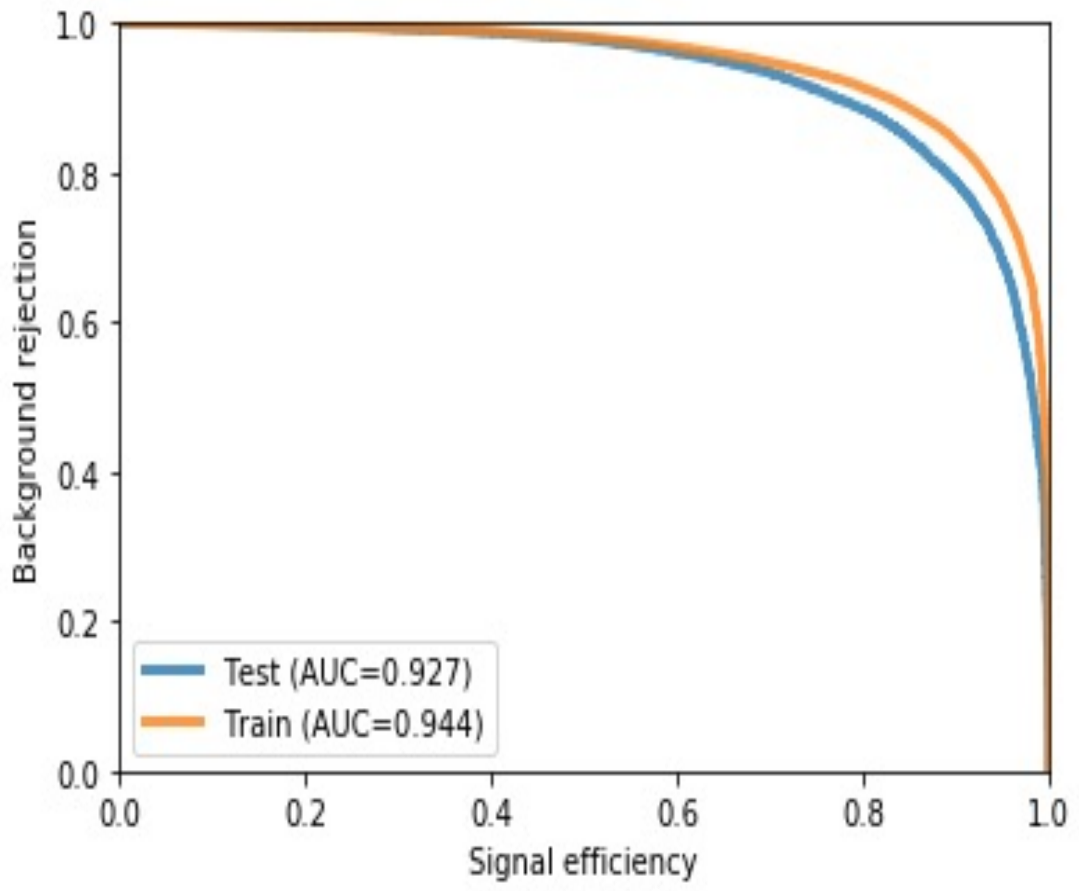


* Both signal and background mc samples taken from cms open data

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

References

1. The elements of statistical learning, Hastie, Tibshirani, Friedman
2. Neural Networks, Freeman & Skapura
3. http://neuralnetworksanddeeplearning.com/
4. https://machine-learning-for-physicists.org/
5. TMVA userguide

Thanks