Logical Rhythm - Class 3

August 27, 2018

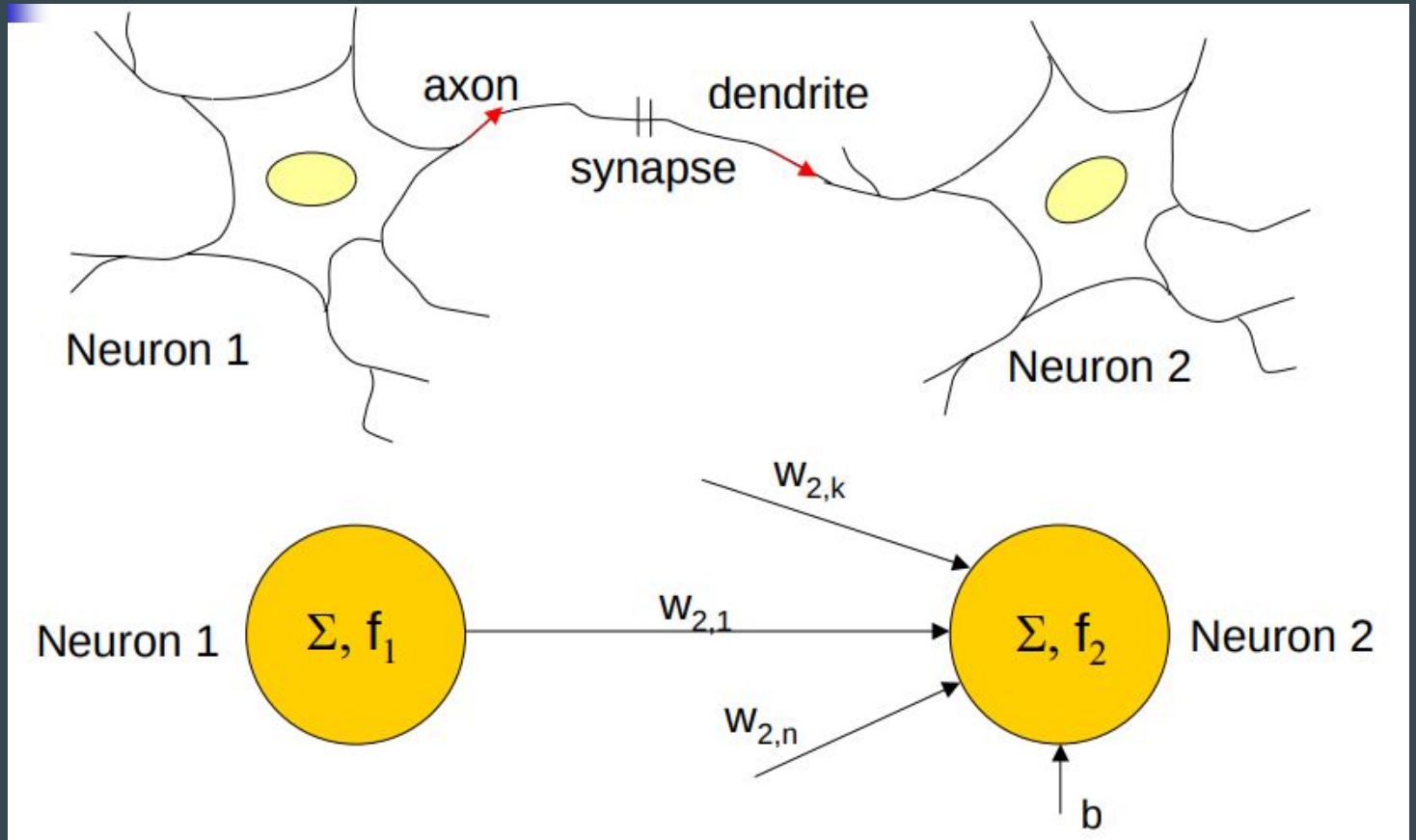
In this Class

- Neural Networks (Intro To Deep Learning)
- Decision Trees
- Ensemble Methods(Random Forest)
- Hyperparameter Optimisation and Bias Variance Tradeoff

Biological Inspiration for Neural Networks

- Human Brain: ≈ 10^{11} neurons (or nerve cells)
  - Dendrites: incoming extensions, carry the signals in
  - Axons: outgoing extensions, carry the signals out
  - Synapse: connection between 2 neurons
- Learning :
  - Forming new connections between the neurons
  - Modifying existing connections

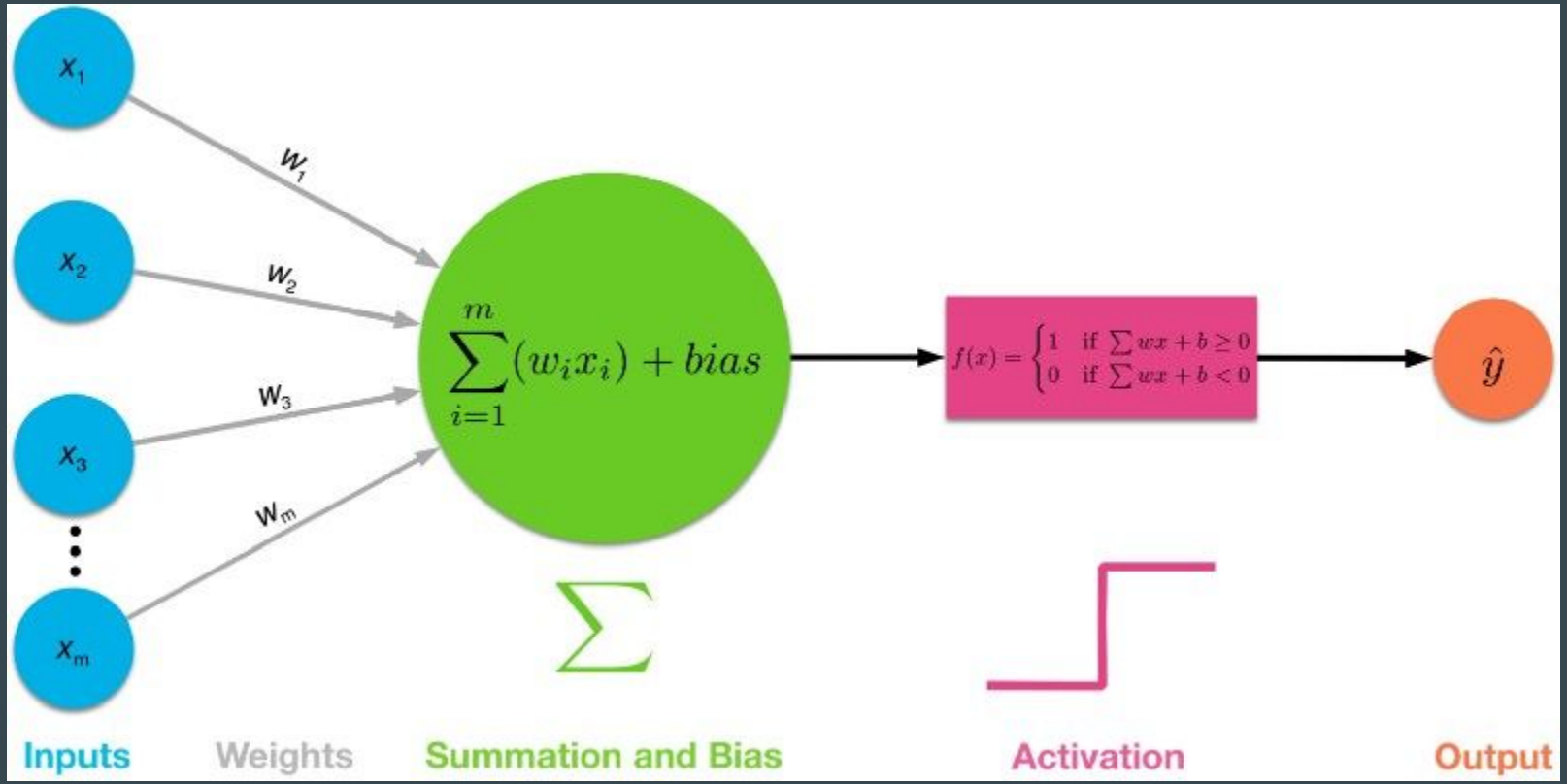
From Biology to the Artificial Neuron



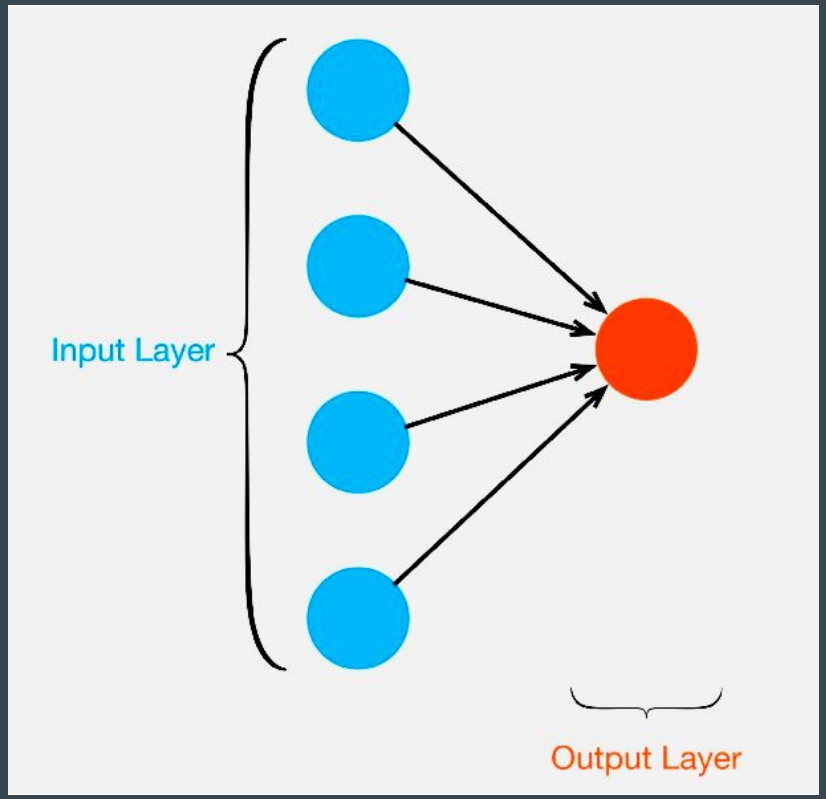
Relating Human Neuron with Artificial neuron

1. The **weight w** models the **synapse** between two biological neurons.
2. Each neuron has a **threshold** that must be met to activate the neuron, causing it to “fire.” The threshold is modeled with the activation/transfer function.

Single Perceptron == Single Neuron

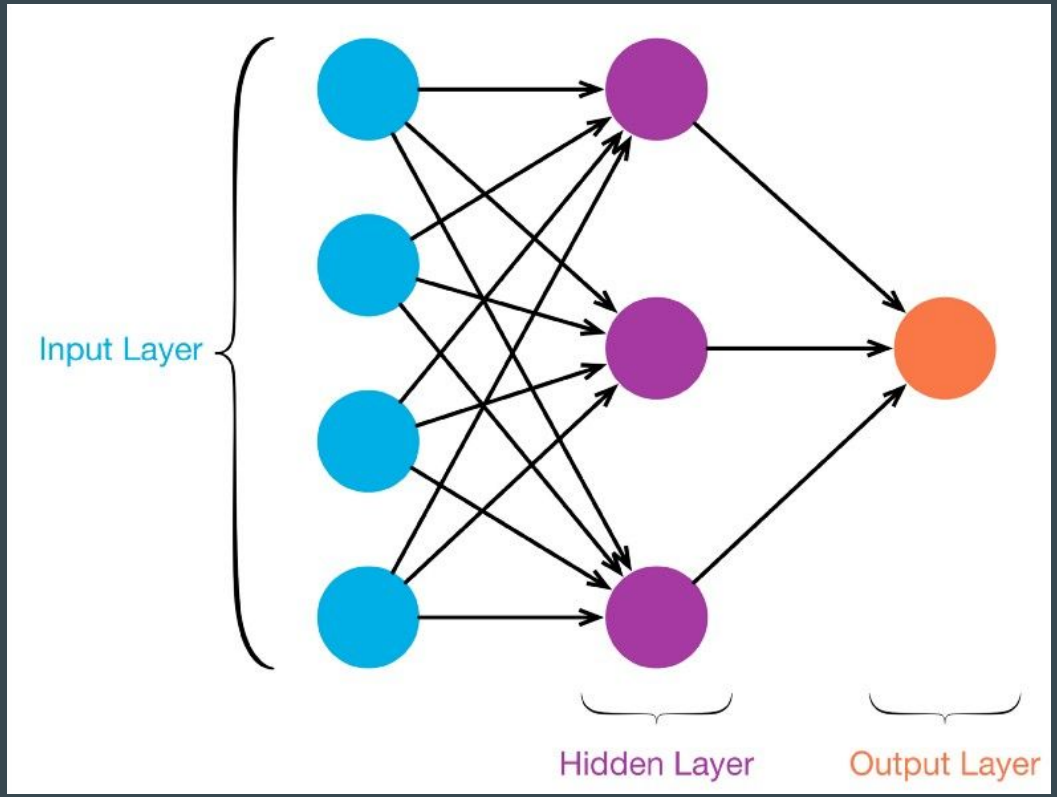


One Layer Of Perceptrons



- SLP has power equivalent to a linear model
- i.e SLP are only capable of learning linearly separable patterns

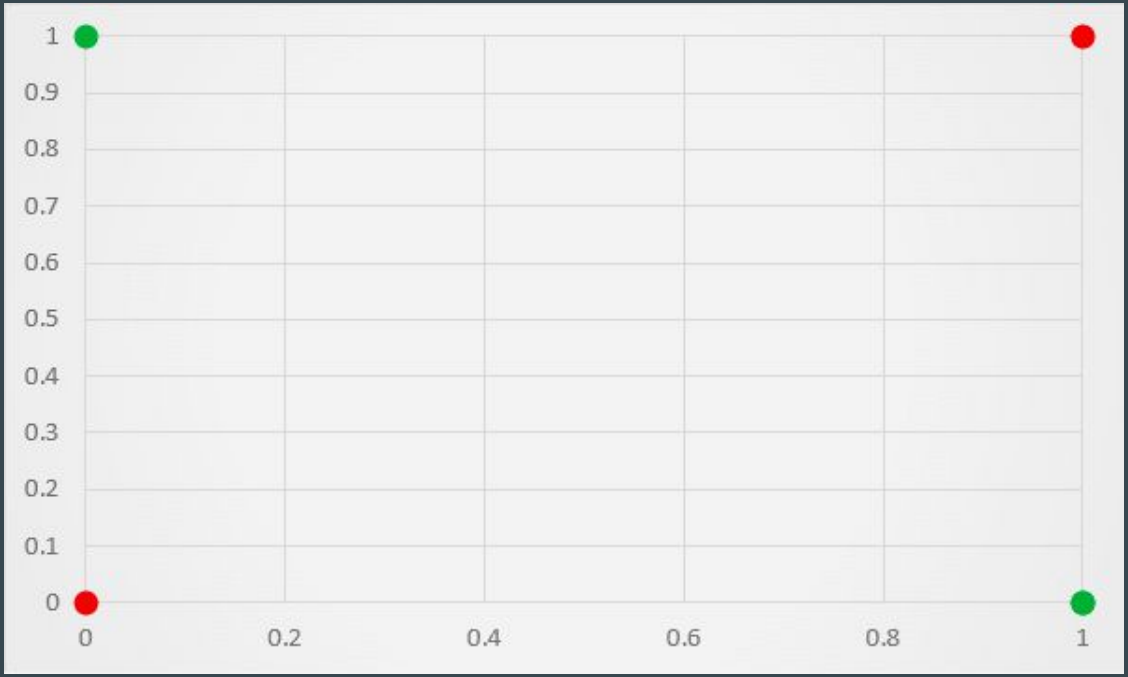
The BreakThrough - Multiple Layers of Perceptron Units



Wait a minute !.....Why multiple layers ?
Why is SLP not sufficient ?????

Welcome to The XOR Problem

Plot of values for XOR



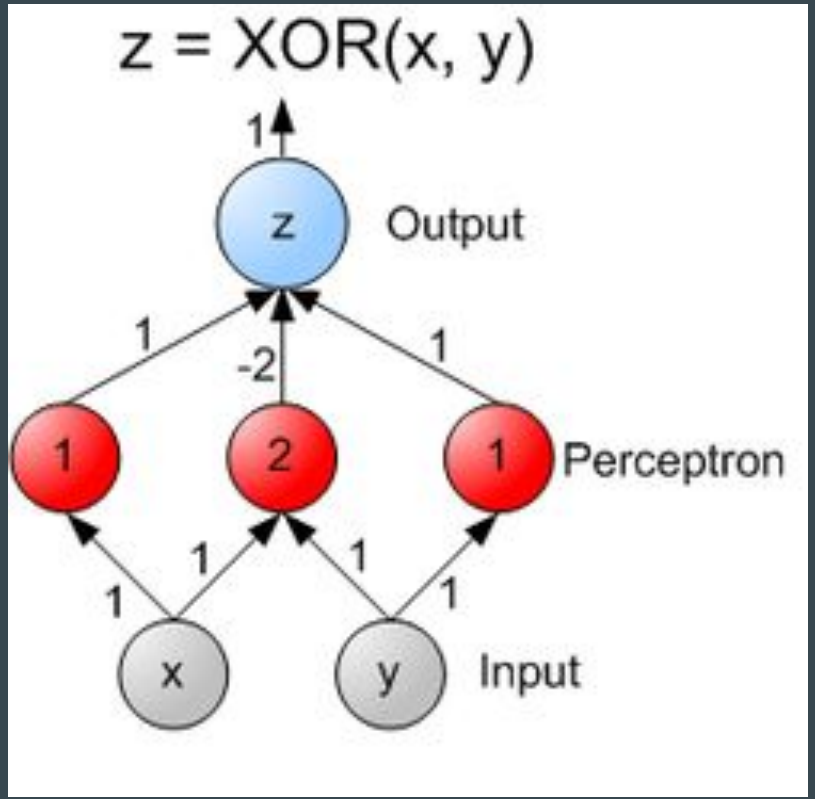
in1	in2	out
0	0	0
0	1	1
1	0	1
1	1	0

No SLP can represent XOR function

- Single-layer perceptrons are only capable of learning linearly separable patterns; in 1969 in a famous monograph entitled Perceptrons, Marvin Minsky and Seymour Papert showed that it was impossible for a **single-layer perceptron network** to learn an XOR function

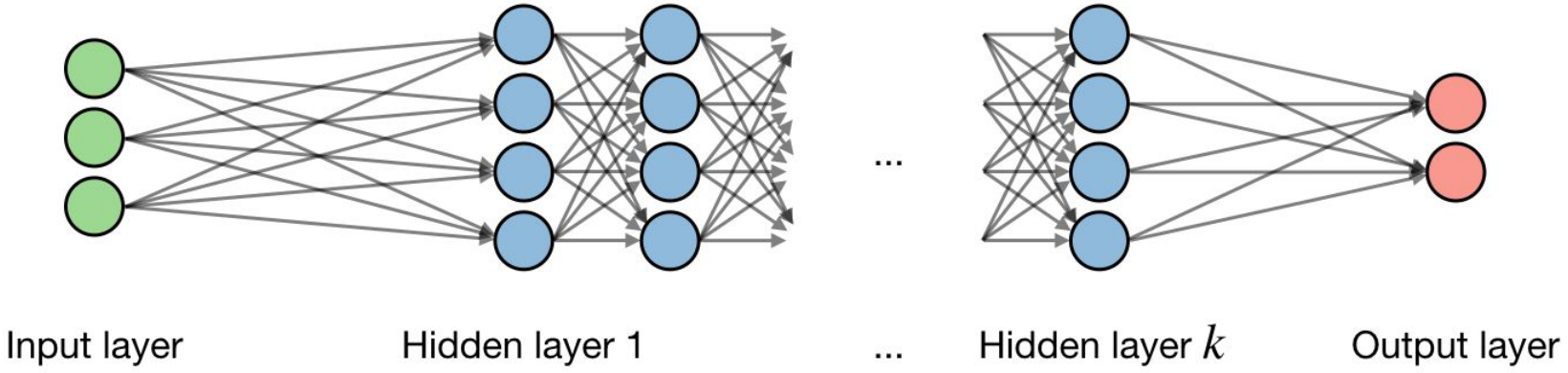
MLP can model XOR function

Eg.



Neural Networks

- Neural networks are a class of models that are build with layers. Commonly used types of neural networks include convolutional and recurrent neural networks.
- The terminology :



“My mom always said:
‘Life is like a box of neural nets. You never know what you gonna get.’”

Components of ANN

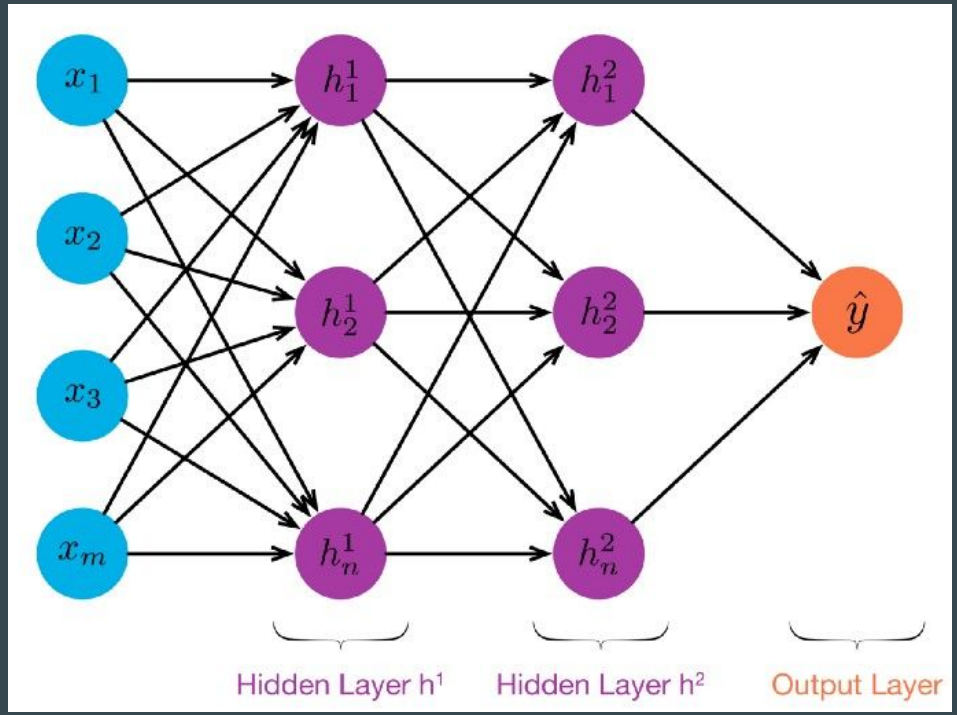
1. Input Layer (features)
2. Weight Matrix (W,b)
3. Hidden Layers
4. Output Layer

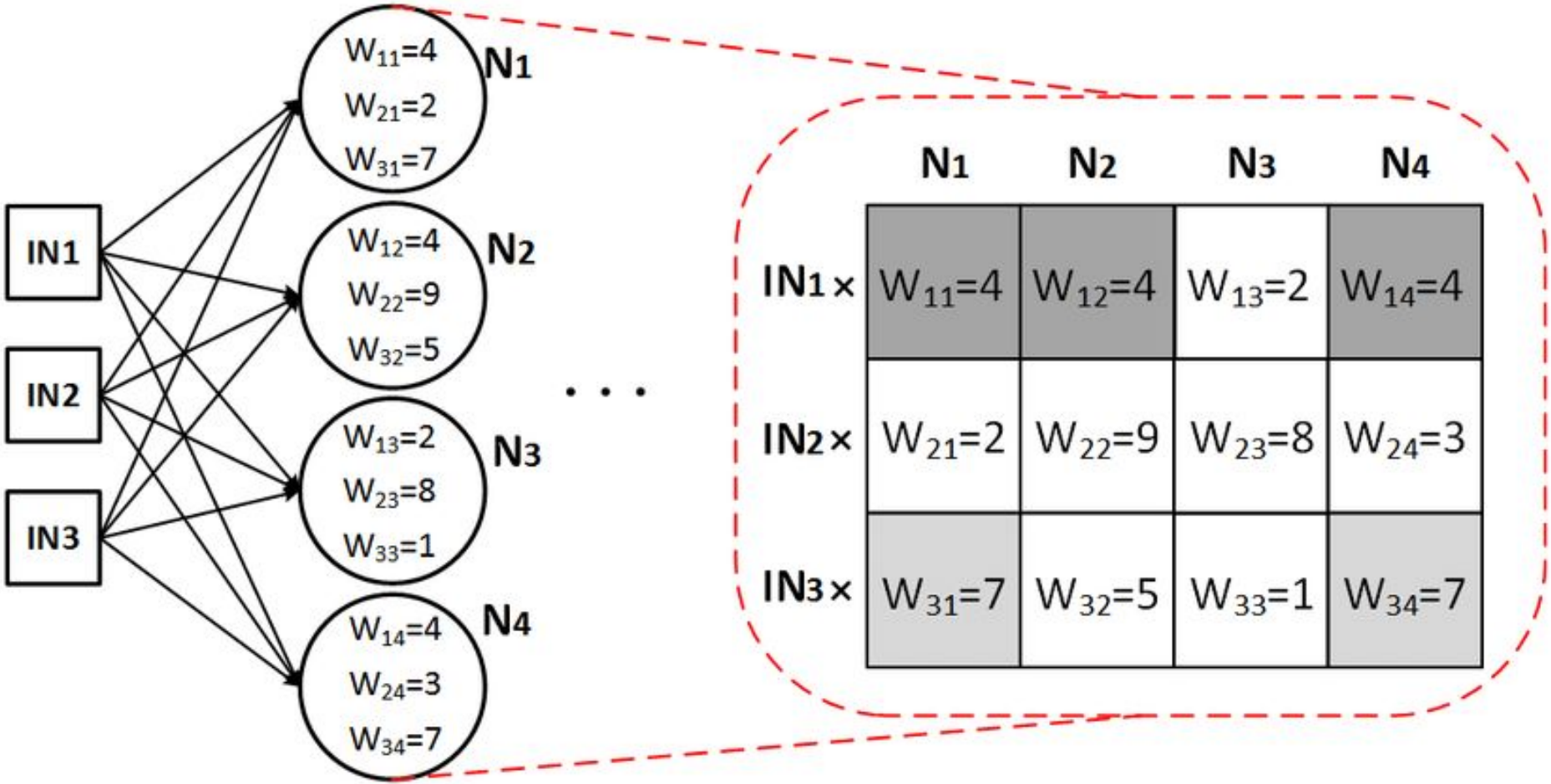
First, Some Notations

Weight Matrix

- First index (i) indicates the neuron # the input is entering (the “to” index)
- Second index (j) indicates the element # of input vector p that will be entering the neuron (the “from” index”)

$$w_{i,j} = w_{to,from}$$




IN1
IN2
IN3
W11=4
W21=2
W31=7
N1
W12=4
W22=9
W32=5
N2
W13=2
W23=8
W33=1
N3
W14=4
W24=3
W34=7
N4
N1 N2 N3 N4
IN1 × W11=4 W12=4 W13=2 W14=4
IN2 × W21=2 W22=9 W23=8 W24=3
IN3 × W31=7 W32=5 W33=1 W34=7

The Goal

In a fully connected ANN

To find Weights(W) and bias units (b) such that error at the output layer is minimum.

Getting (Dendritic) Input to a neuron

Simple Dot Product

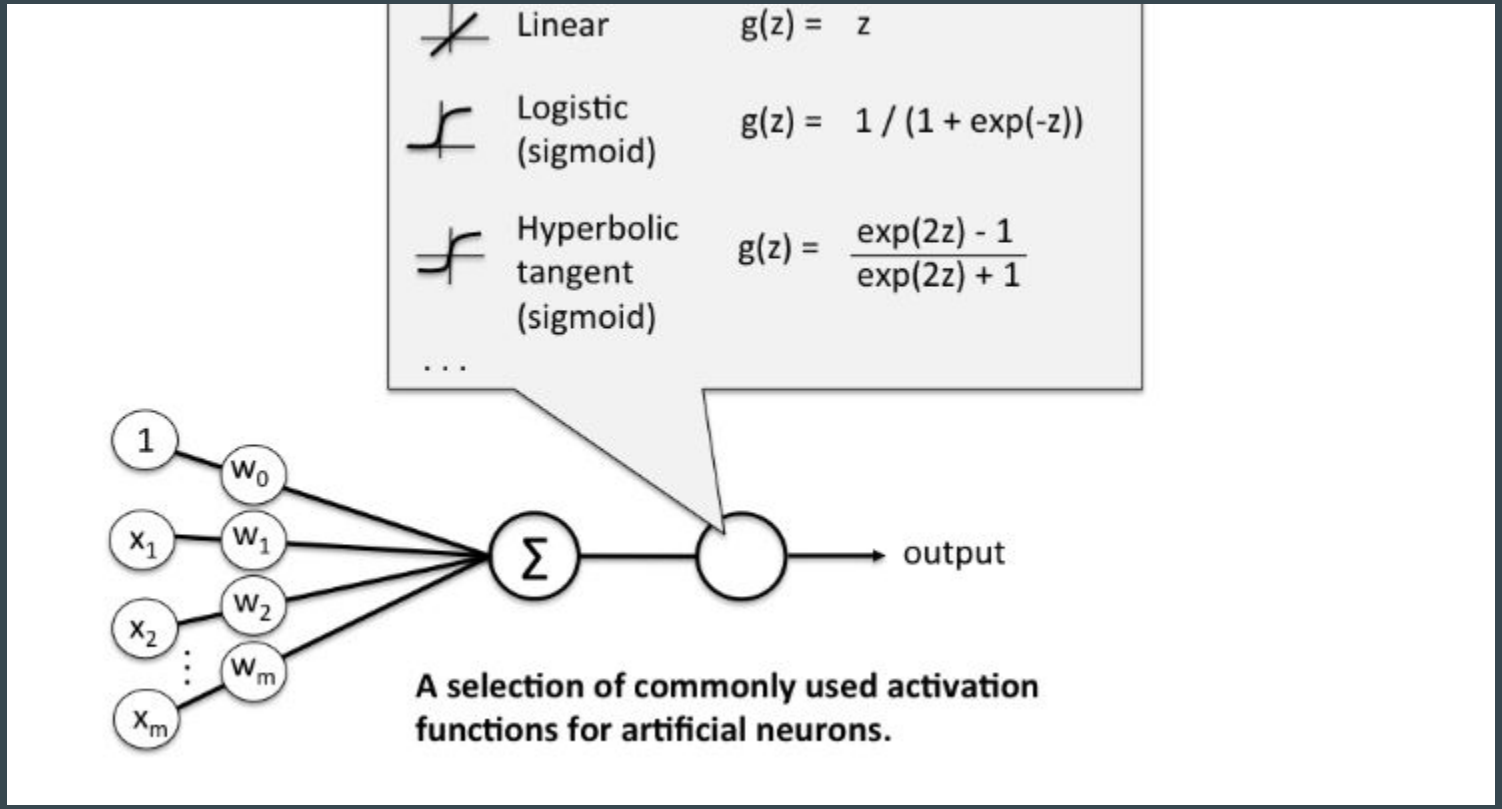
$$z^{(i)} = w^T x^{(i)} + b$$

Single layer == linear model :

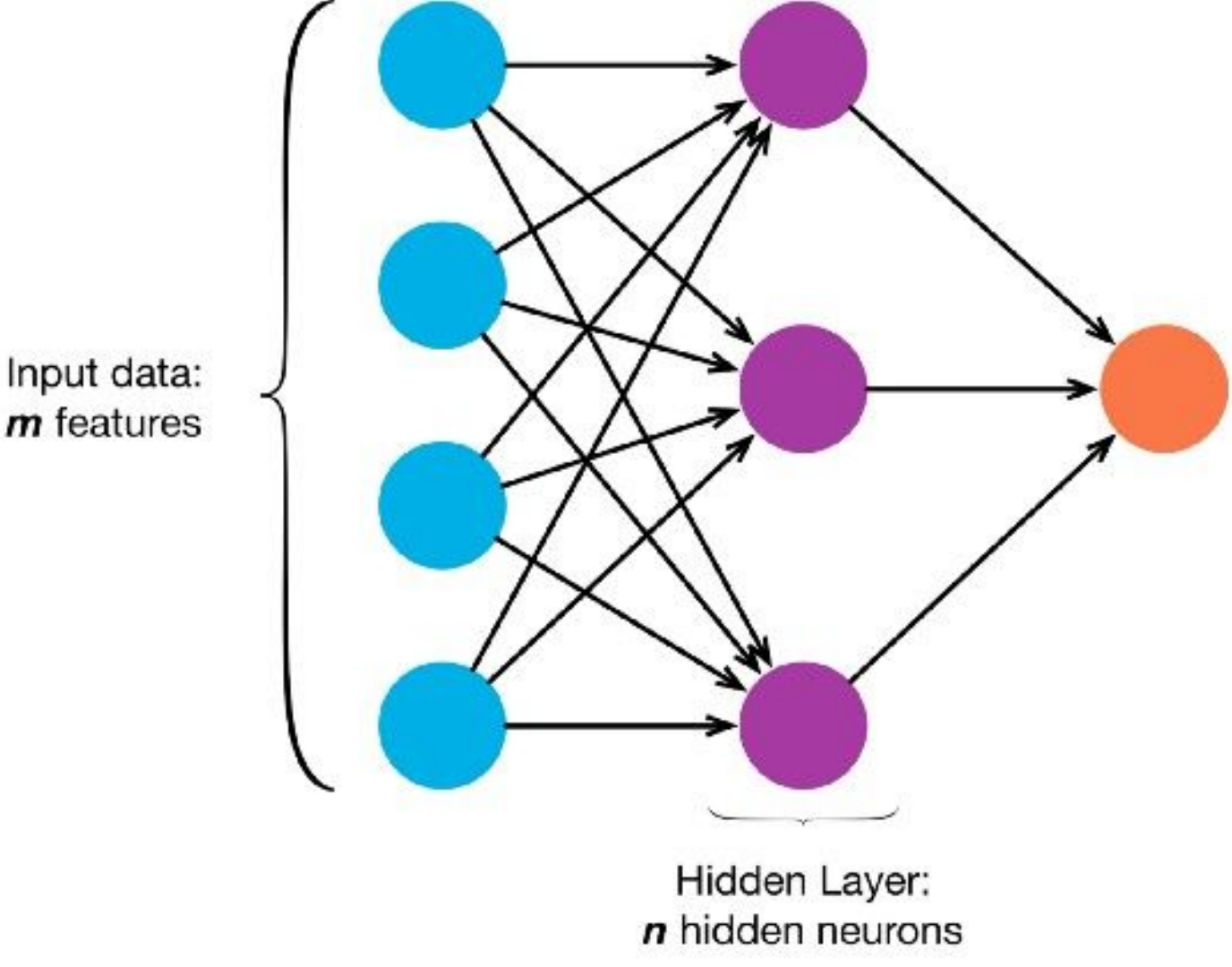
$$W \cdot X = w_1 x_1 + w_2 x_2 + \ldots + w_m x_m = \sum_{i=1}^{m} w_i x_i$$

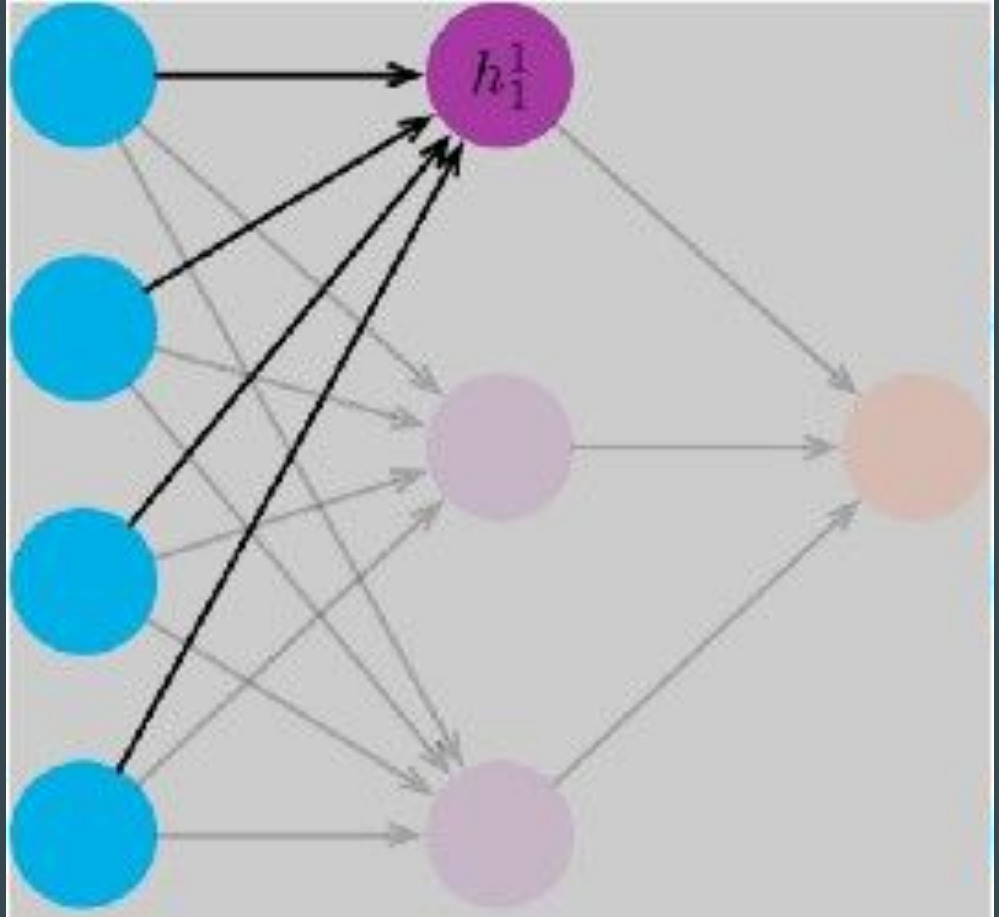
Getting (Axonal) Output from a neuron

Simply apply an activation function to Z(sum of products)

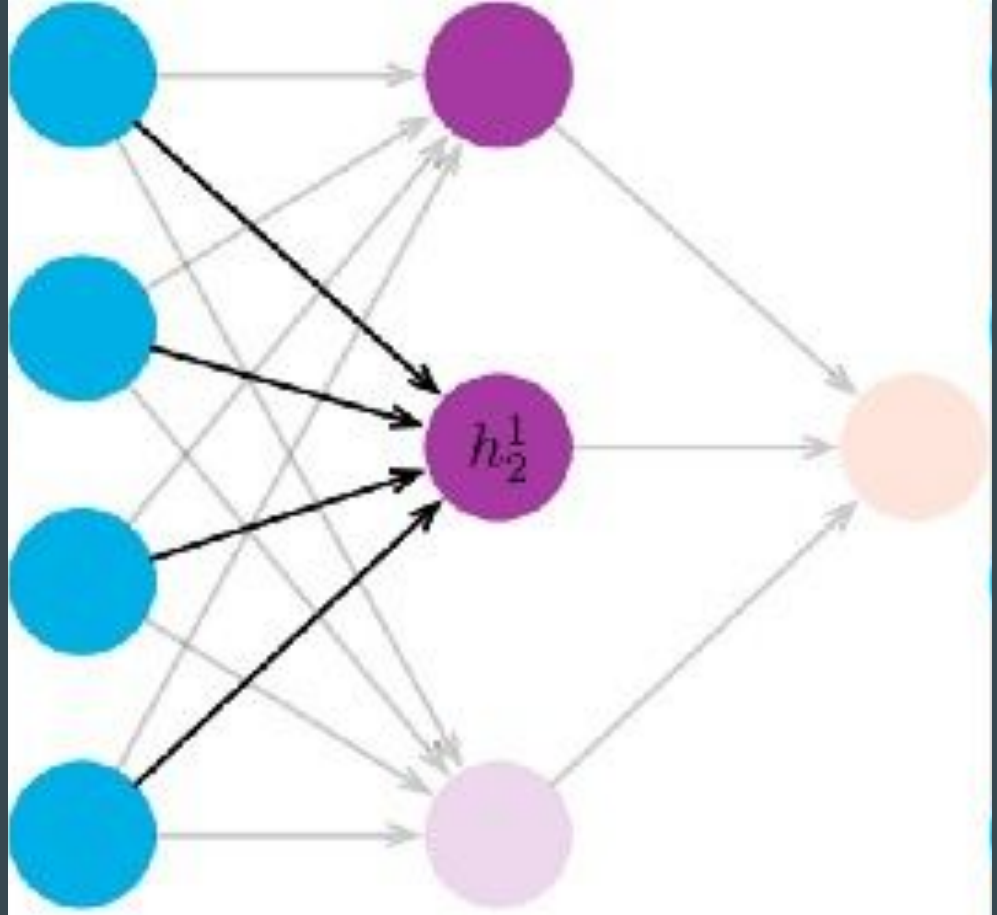


Doing it for every single neuron

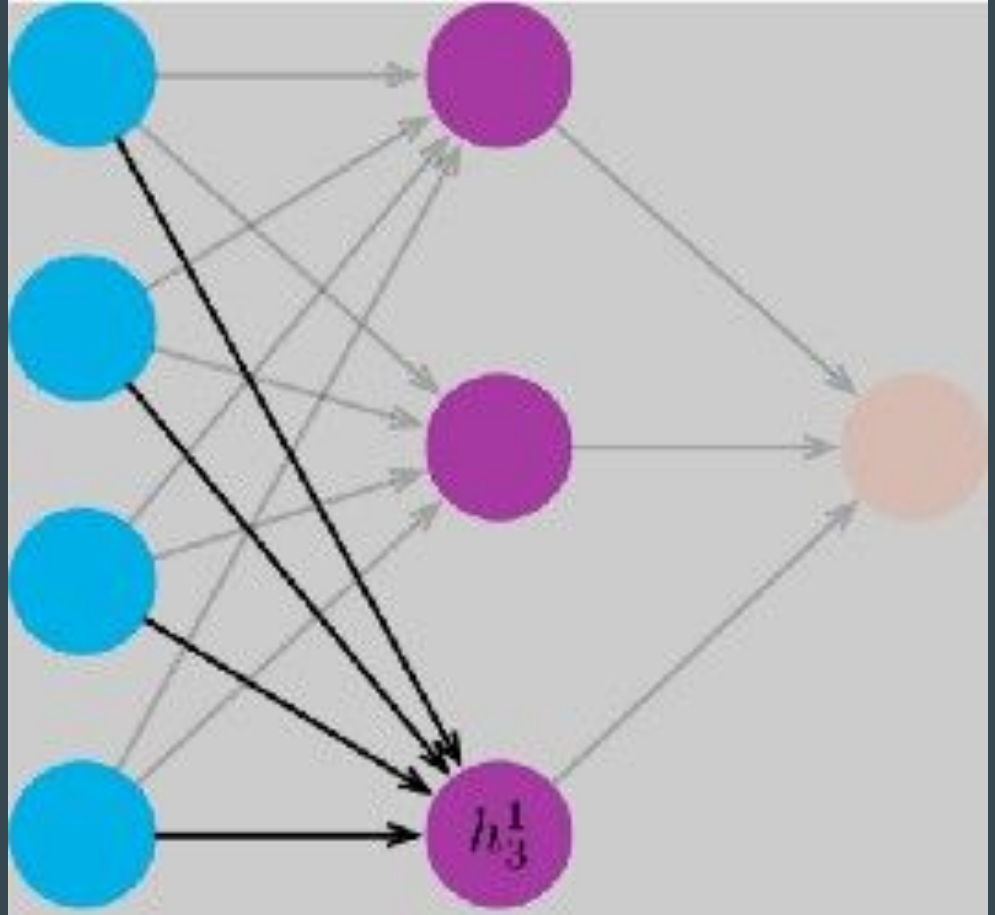

Input data:
m features
Hidden Layer:
n hidden neurons



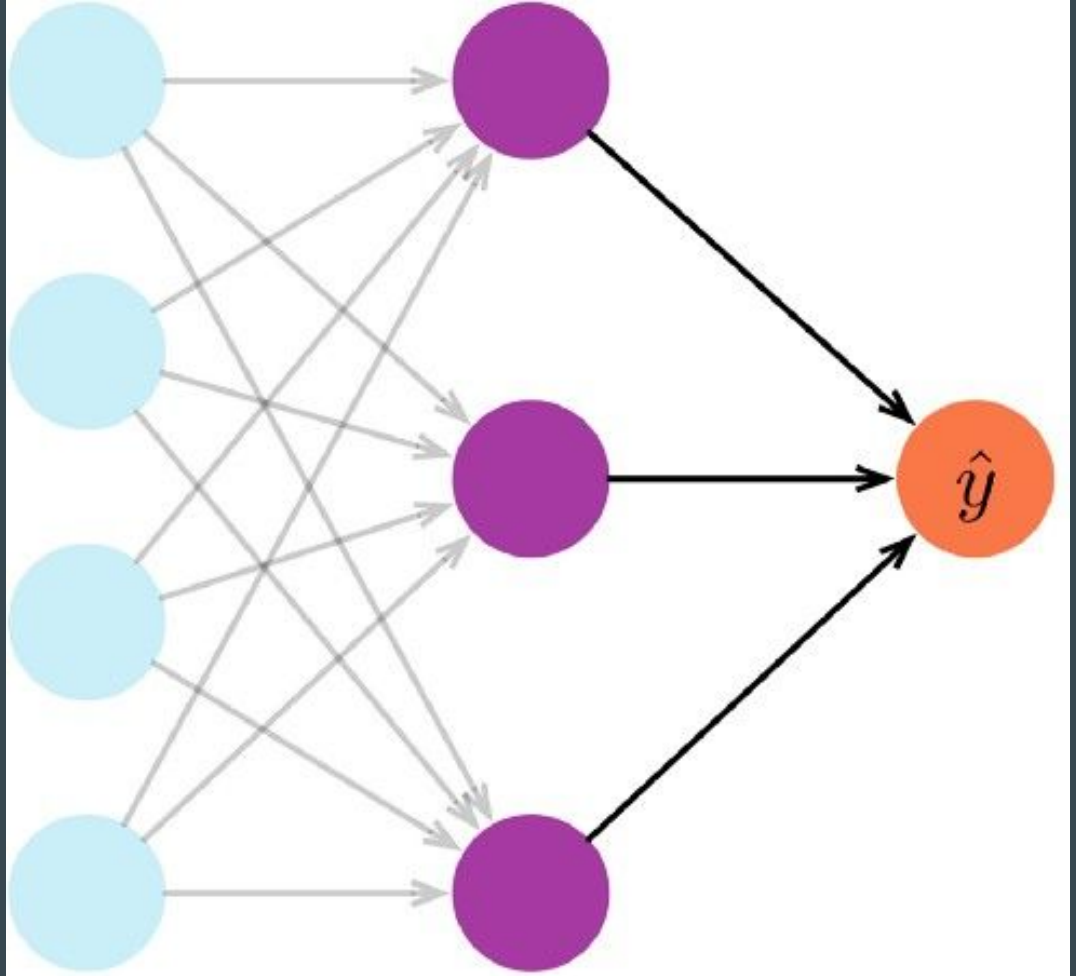
(1) W^1: $w_1^1, w_2^1, ..., w_m^1$



(2) W^2: $w_1^2, w_2^2, ..., w_m^2$



(n) W^n: $w_1^n, w_2^n, \ldots, w_m^n$



$$W_1^h : (w_1^{h_1}, w_2^{h_1}, ..., w_n^{h_1})$$

You See it?

Parallelisation : Why we need GPUs ?

Forward Propagation

Given one training example (x, y):

Forward propagation:

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

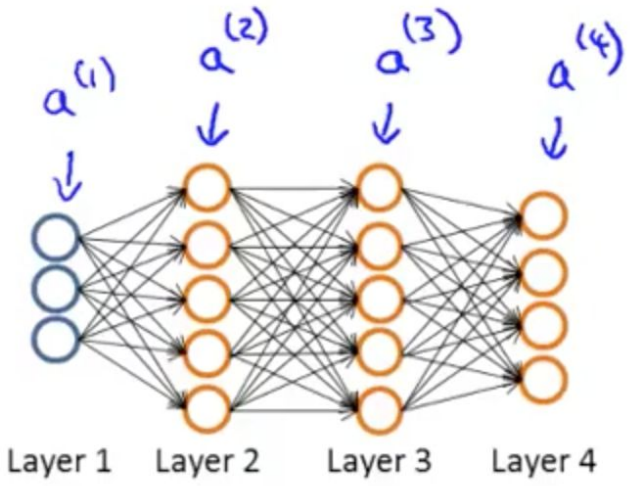
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$

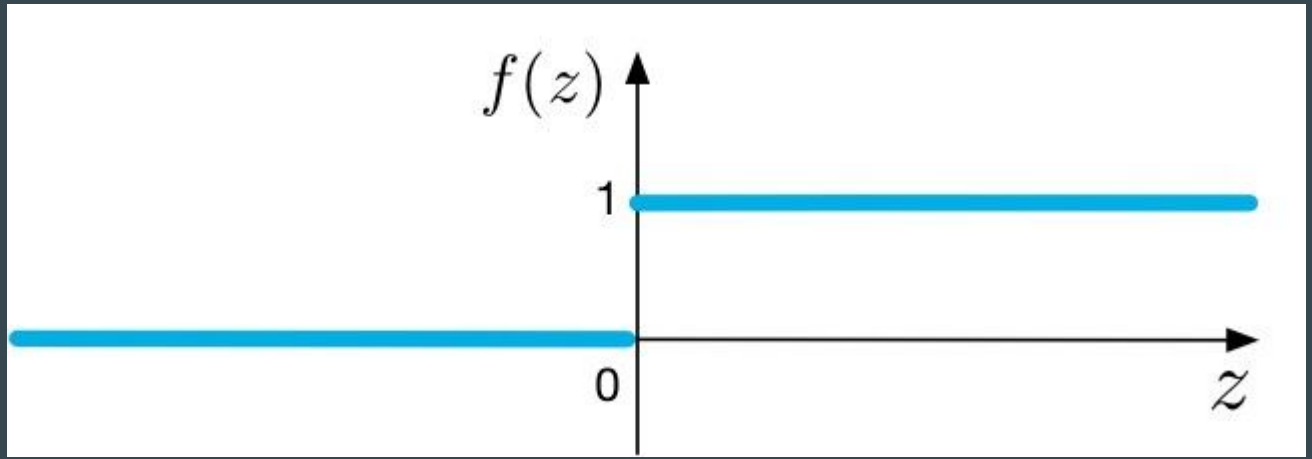
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$

$$a^{(4)} = h_\Theta(x) = g(z^{(4)})$$



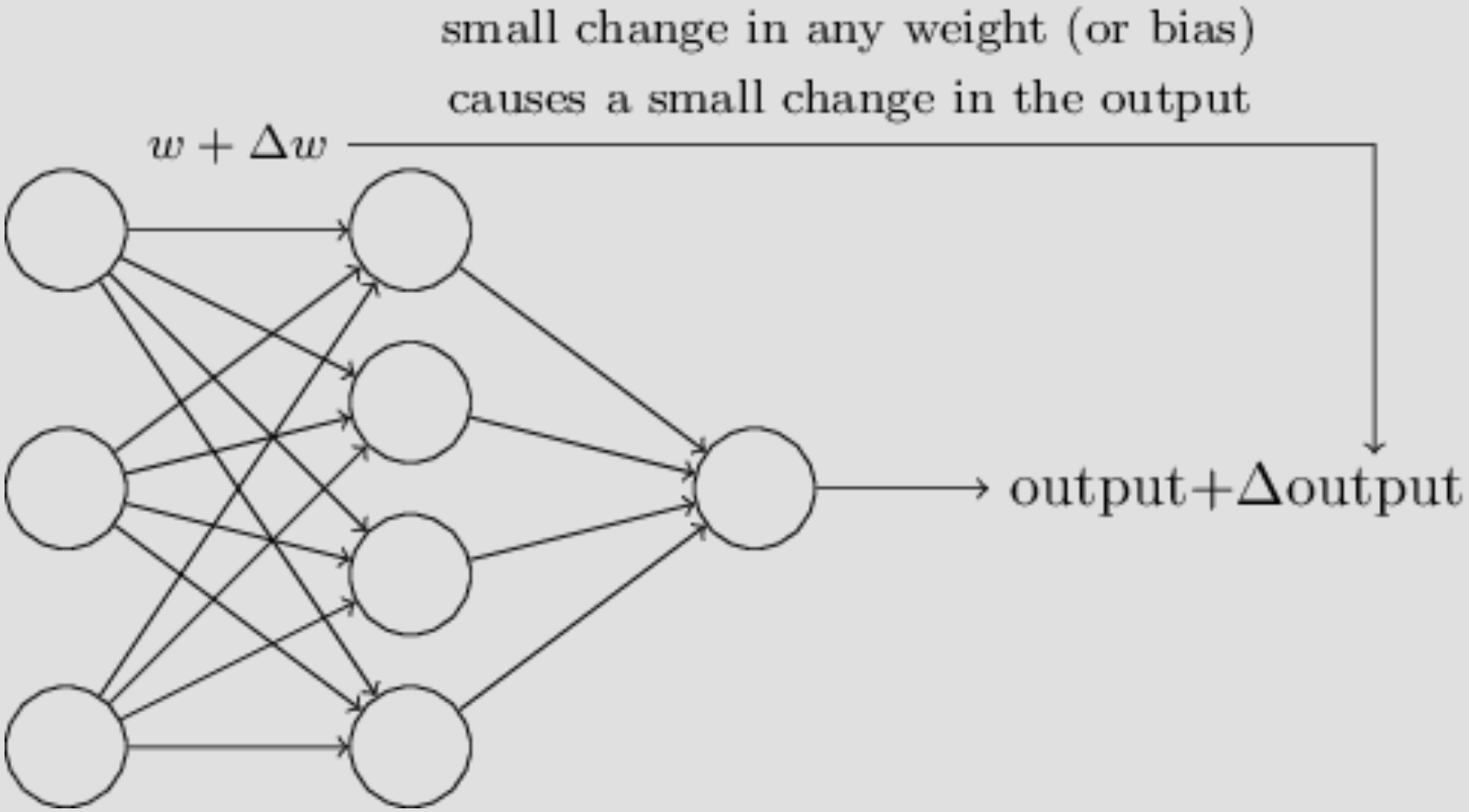
Activation Function

- If the activation function is linear, then you can stack as many hidden layers in the neural network as you wish, and the final output is still a linear combination of the original input data.
- Perceptron's default activation function is Heaviside Step Function:



Activation Function Properties

- So basically, a small change in any weight in the input layer of our perceptron network could possibly lead to one neuron to suddenly flip from 0 to 1.
- Which could again affect the hidden layer's behavior, and then affect the final outcome.
- We want a learning algorithm that could improve our neural network by **gradually changing the weights**, not by flat-no-response or sudden jumps.
- If we can't use an activation function to gradually change the weights, then it shouldn't be the choice.

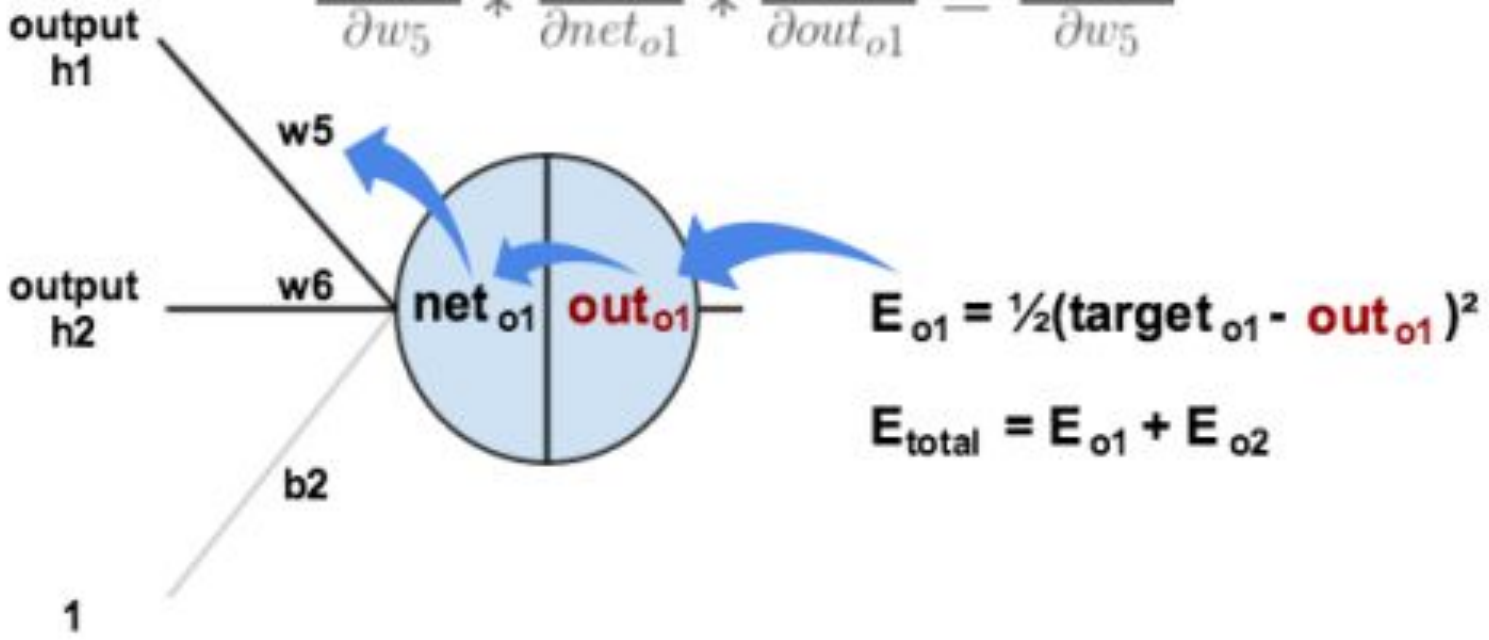

small change in any weight (or bias)
causes a small change in the output
$w + \Delta w$
output+Δoutput

NP-complete problem

trying to find an acceptable set of weights for an MLP network manually would be an incredibly laborious task, and is an NP-complete problem (Blum and Rivest, 1992)

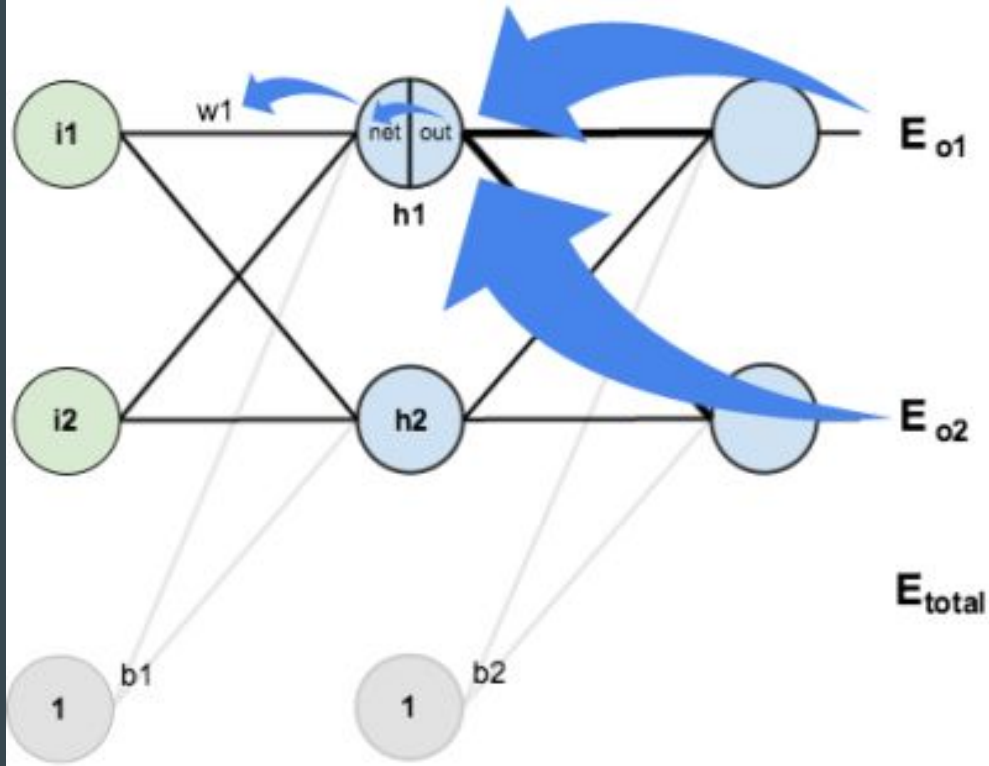
Backpropagation (The Genie)

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



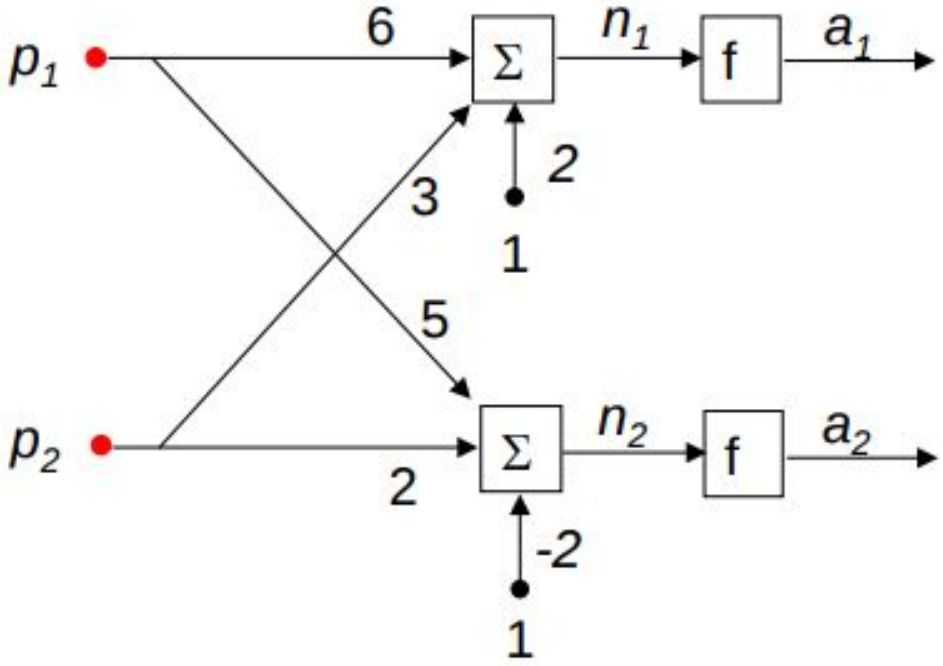
$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



$E_{total} = E_{o1} + E_{o2}$


Google



a = compet(Wp + b)

where compet(*n*) = 1, neuron w/max n

0, else

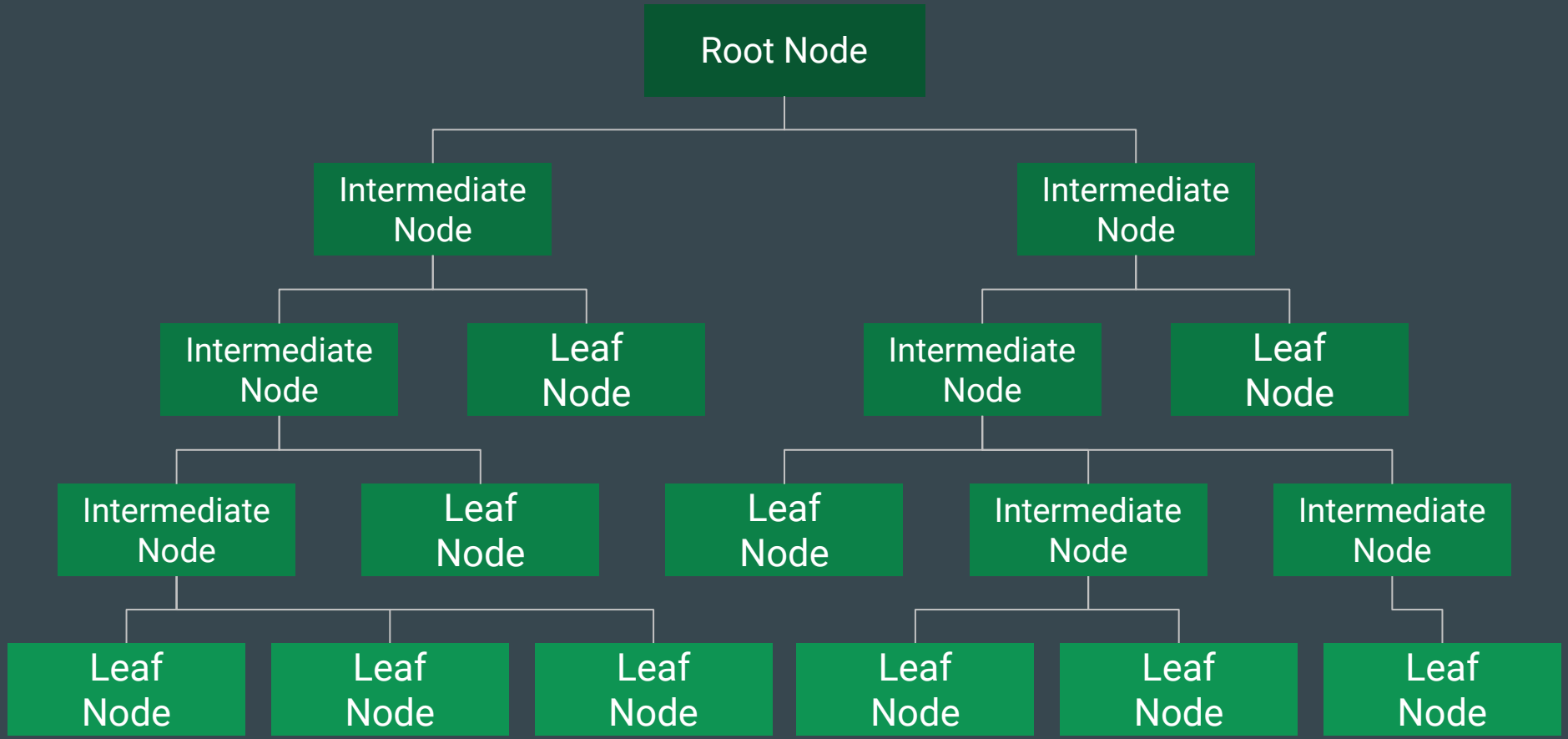
1) For the neural network shown, find the weight matrix **W** and the bias vector **b**.

2) Find the output if f = "compet" and the input vector is $\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

Decision Tree

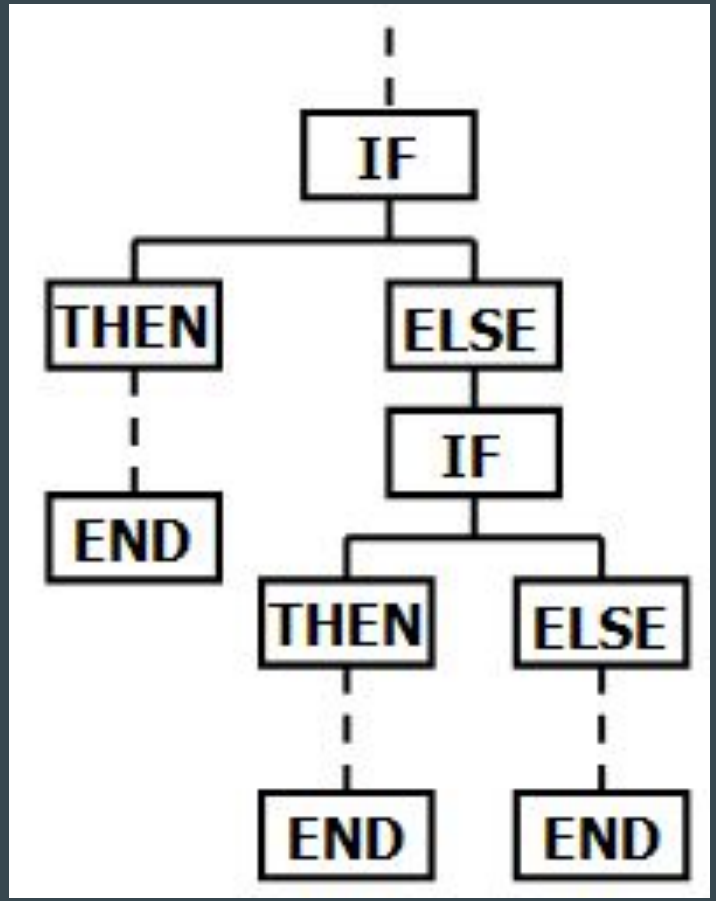
- Introduction
- Intuition
- Building Trees
  - Splitting Criterion
  - Multi-Way branching
- Problems with D-Trees

A brief overview of TREE Data Structure



What are Decision Trees?

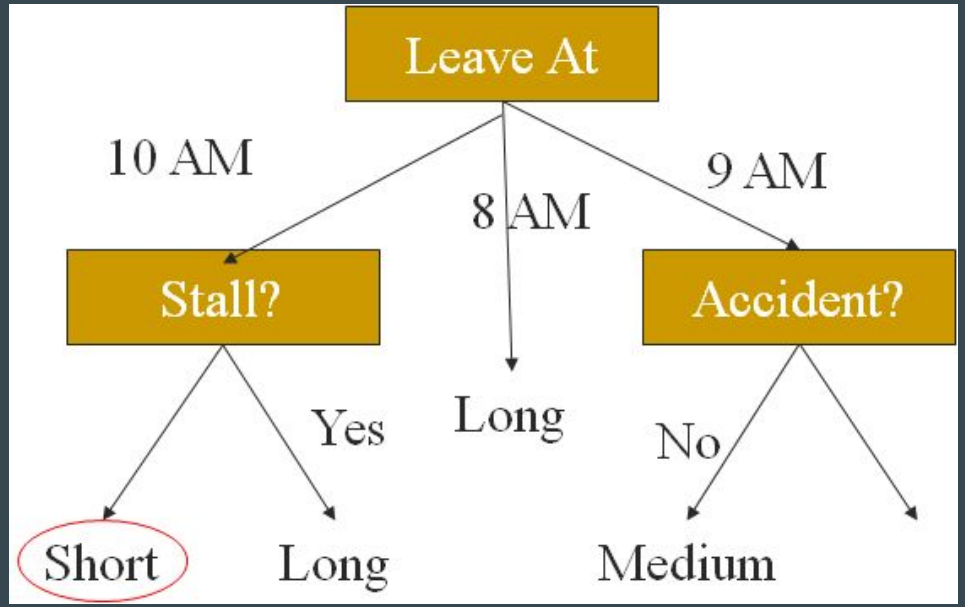
- A predictive model based on a branching series of Boolean tests
- These smaller Boolean tests are less complex than a one-stage classifier
- Powerful Algorithms.



Random Forests, a Decision Tree based algorithm was used for body part recognition in Microsoft Kinect.

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/BodyPartRecognition.pdf

An Example - Predicting commute time



If we leave at 10 AM and there are no cars stalled on the road, what will our commute time be?

SHORT

Inductive Learning

- In this decision tree, we made a series of Boolean decisions and followed the corresponding branch
  - Did we leave at 10 AM?
  - Did a car stall on the road?
  - Is there an accident on the road?

By answering each of these yes/no questions, we then came to a conclusion on how long our commute might take.

The system tries to induce a general rule from a set of observed instances => **Inductive Learning**

Why not make an if-else ladder?

```
if hour == 8am
        commute time = long
else if hour == 9am
        if accident == yes
                commute time = long
        else
                commute time = medium
else if hour == 10am
        if stall == yes
                commute time = long
        else
                commute time = short
```

- Cumbersome programming
- Decision Tree can be built recursively
- All attributes do not appear in each decision path.
    - Using an if-else ladder for a dataset with N features, we will have N! Conditions to check
    - Decision Trees reduce this number greatly
- Also, all attributes may not even appear in the tree.

Learning Decision Trees

- Split the records based on an attribute test that optimizes certain criterion.
- Determine how to split the records
  - How to specify the attribute test condition?
  - How to determine the best split?
- Determine when to stop splitting

An Algorithm

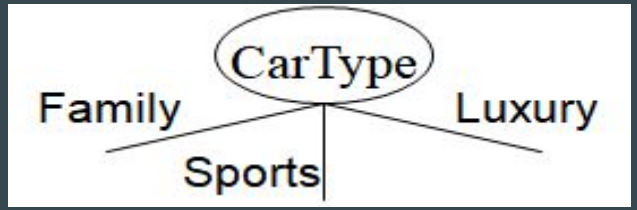
BuildTree(DataSet,Output)

- If all output values are the same in DataSet, return a leaf node that says “predict this unique output”
- If all input values are the same, return a leaf node that says “predict the majority output”
- Else find attribute X with highest Info Gain
- Suppose X has n_X distinct values (i.e. X has arity n_X).
  - Create and return a non-leaf node with n_X children.
  - The i’th child should be built by calling

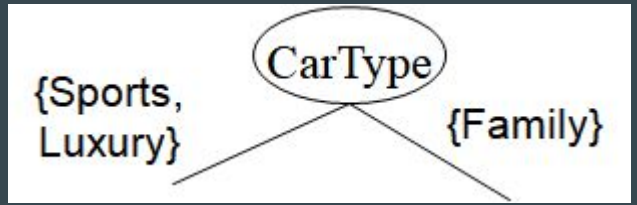
    BuildTree(DS_i,Output)
  - Where DS_i built consists of all those records in DataSet for which X = ith distinct value of X.

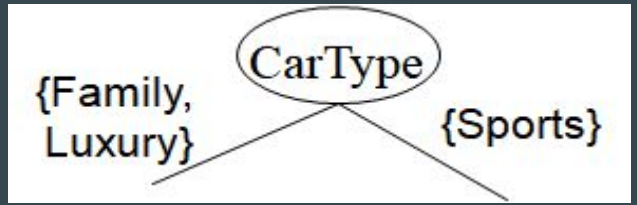
How to split records?

Multi-way split: Use as many partitions as distinct values.



Binary split: Divides values into two subsets. Need to find optimal partitioning.





NB: Splitting done along a single feature

How to split records?

Node Impurity Index: Records are split such that successive nodes are more and more homogeneous in class distribution.

Measurement of Node Impurity:

- Gini Index
- Entropy
- Misclassification error

- Gini Index for a given node t :

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

(NOTE: $p(j \mid t)$ is the relative frequency of class j at node t).

- Maximum (1 - $1/n_c$) when records are equally distributed among all classes, implying least interesting information
- Minimum (0.0) when all records belong to one class, implying most interesting information

- Entropy at a given node t:

$$Entropy(t) = -\sum_j p(j \mid t) \log p(j \mid t)$$

(NOTE: $p(j \mid t)$ is the relative frequency of class j at node t).

- Measures homogeneity of a node.
  - Maximum (log n_c) when records are equally distributed among all classes implying least information
  - Minimum (0.0) when all records belong to one class, implying most information

How to split records?

- Information Gain:

$$GAIN_{split} = Entropy(p) - \left(\sum_{i=1}^{k} \frac{n_i}{n} Entropy(i) \right)$$

Parent Node, p is split into k partitions;

n_i is number of records in partition i

  - Measures Reduction in Entropy achieved because of the split. Choose the split that achieves most reduction (maximizes GAIN)
  - Used in ID3 and C4.5

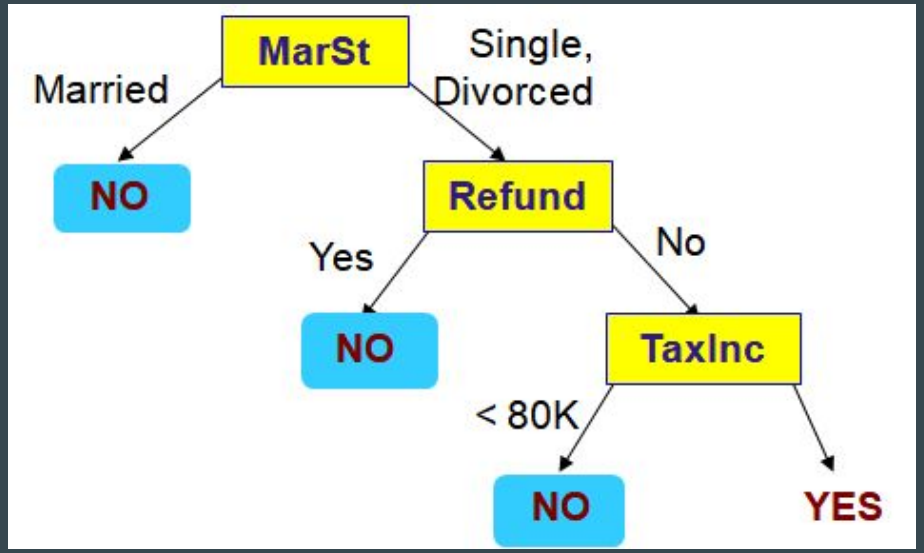
- Used in the BuildTree(DataSet,Output) function.

When to Stop?

Base Case One: If all records in current data subset have the same output then don’t recurse => Stop expanding a node when all the records belong to the same class

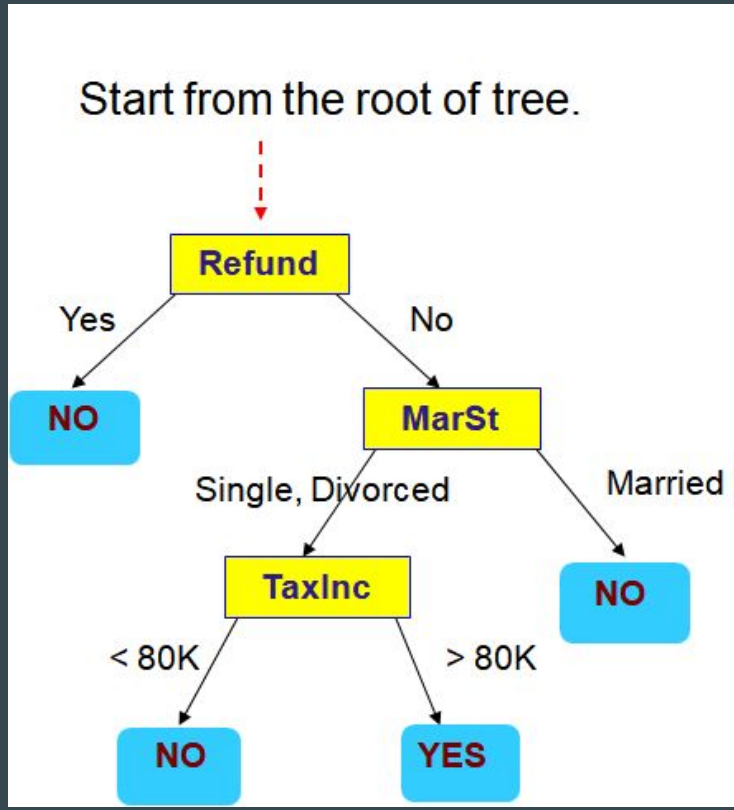
Base Case Two: If all records have exactly the same set of input attributes then don’t recurse => Stop expanding a node when all the records have similar attribute values

Decision Tree in Action



		categorical	categorical	continuous	class
Tid	Refund	Marital Status	Taxable Income	Cheat	
1	Yes	Single	125K	No	
2	No	Married	100K	No	
3	No	Single	70K	No	
4	Yes	Married	120K	No	
5	No	Divorced	95K	Yes	
6	No	Married	60K	No	
7	Yes	Divorced	220K	No	
8	No	Single	85K	Yes	
9	No	Married	75K	No	
10	No	Single	90K	Yes	

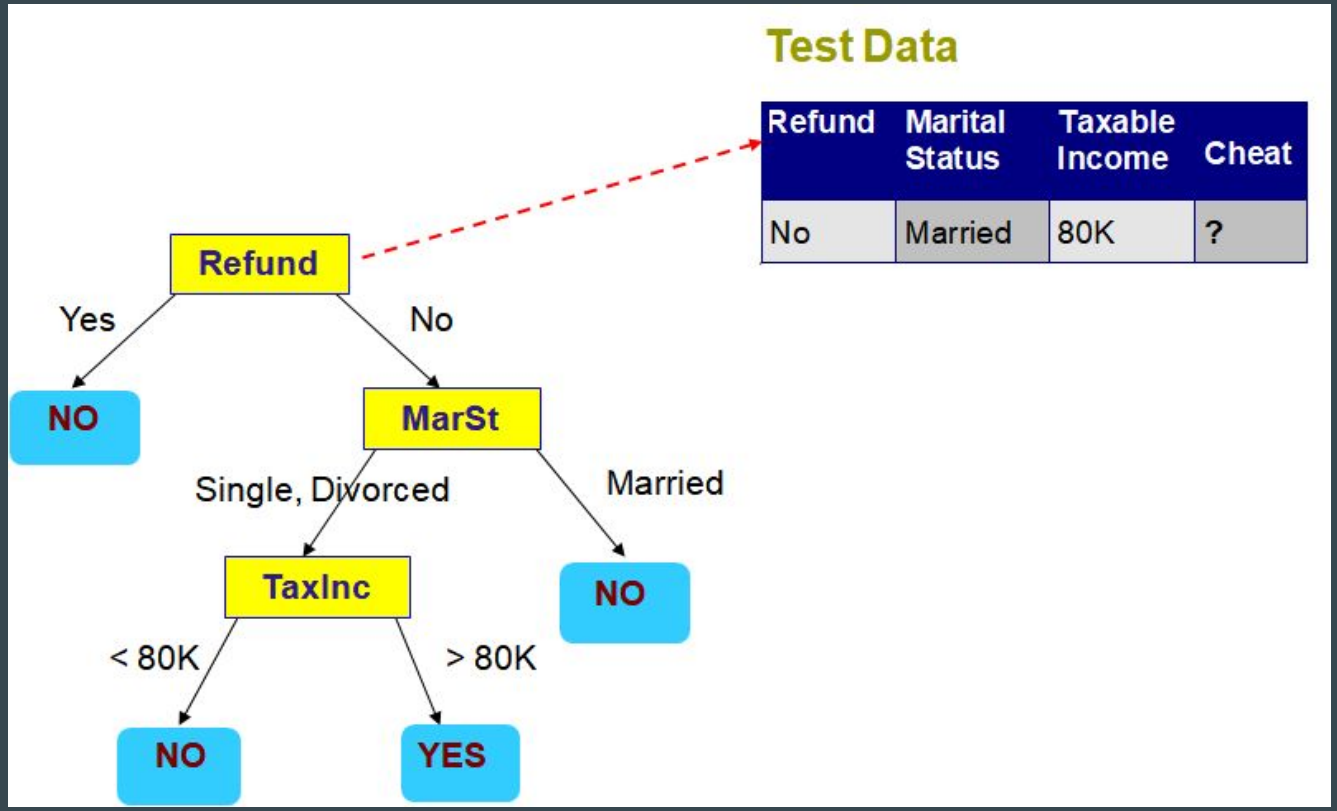
Decision Tree in Action



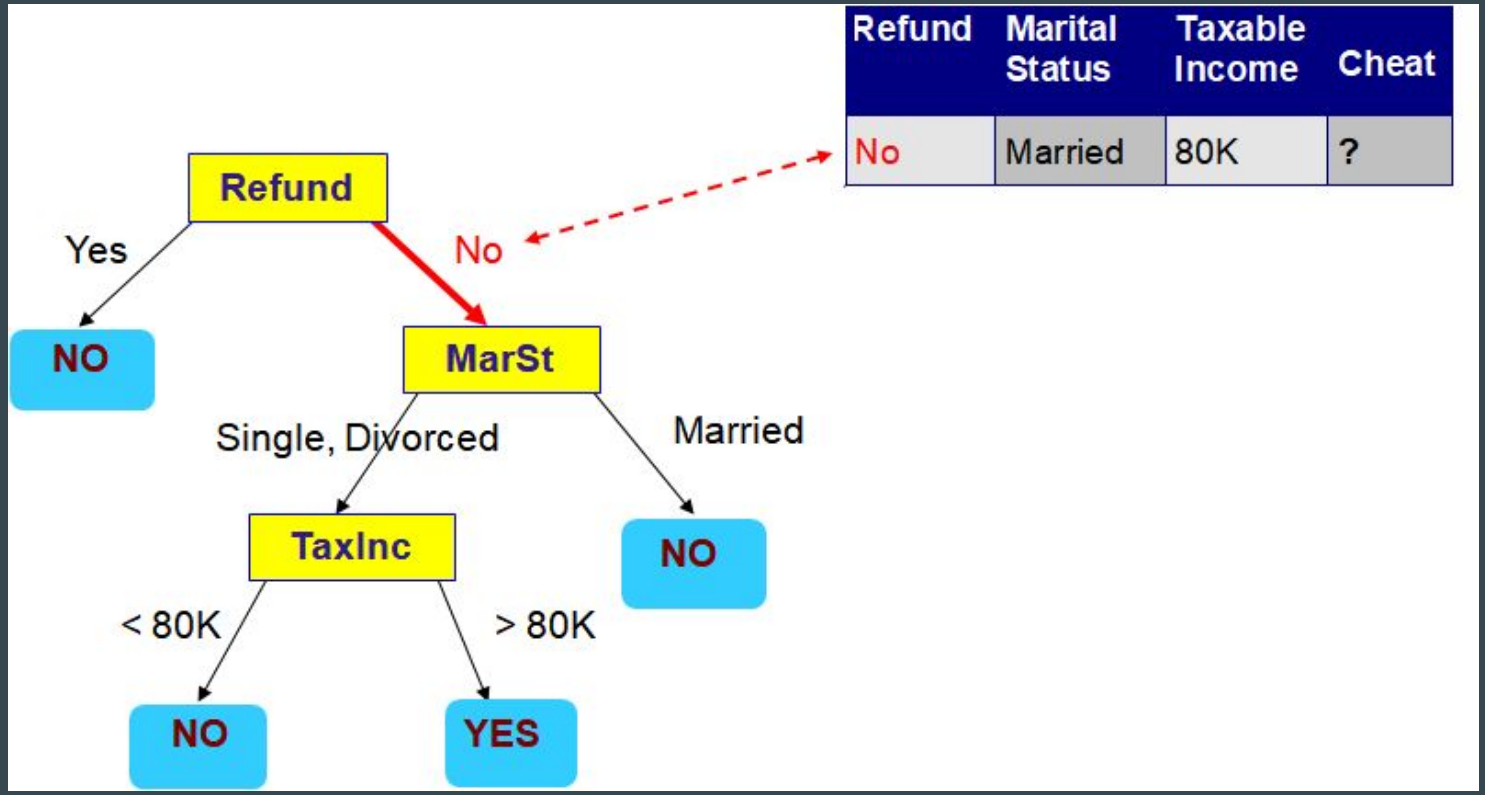
Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

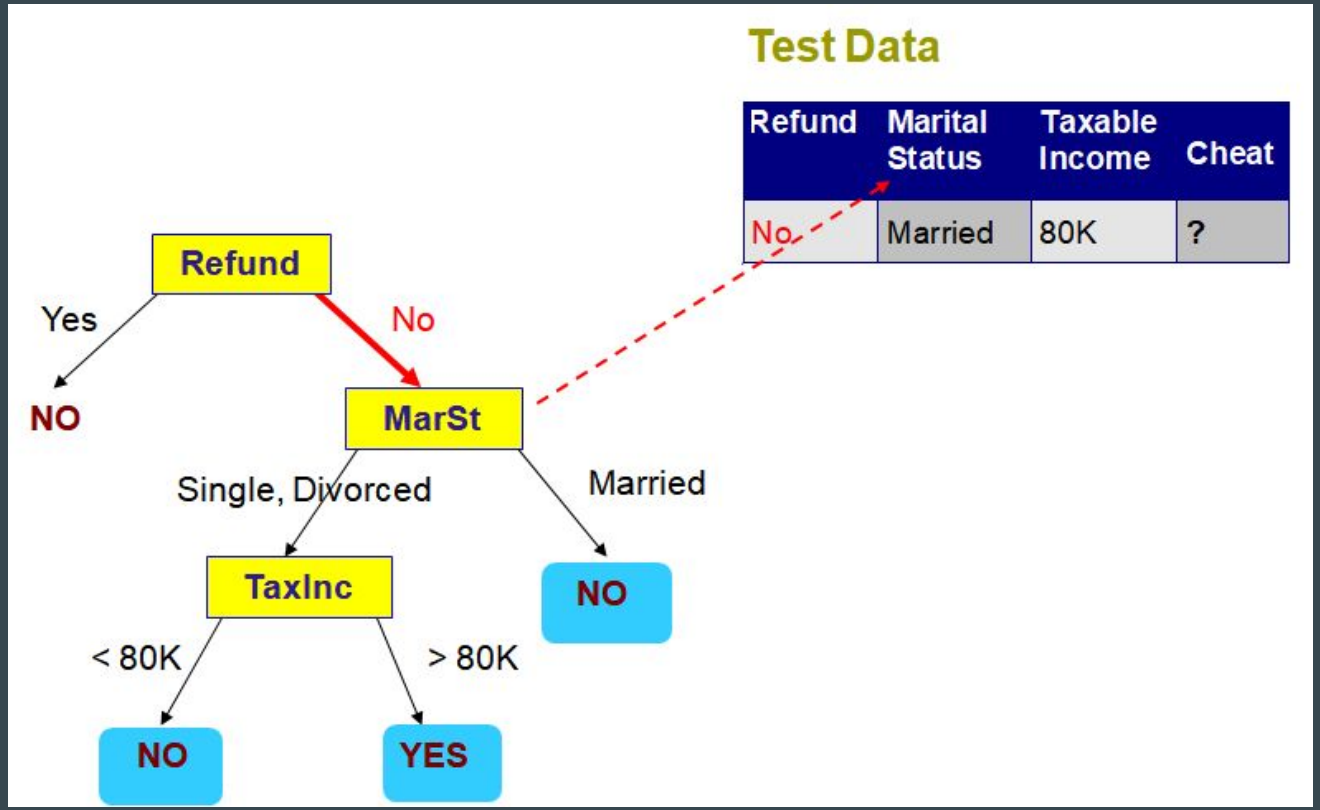
Decision Tree in Action



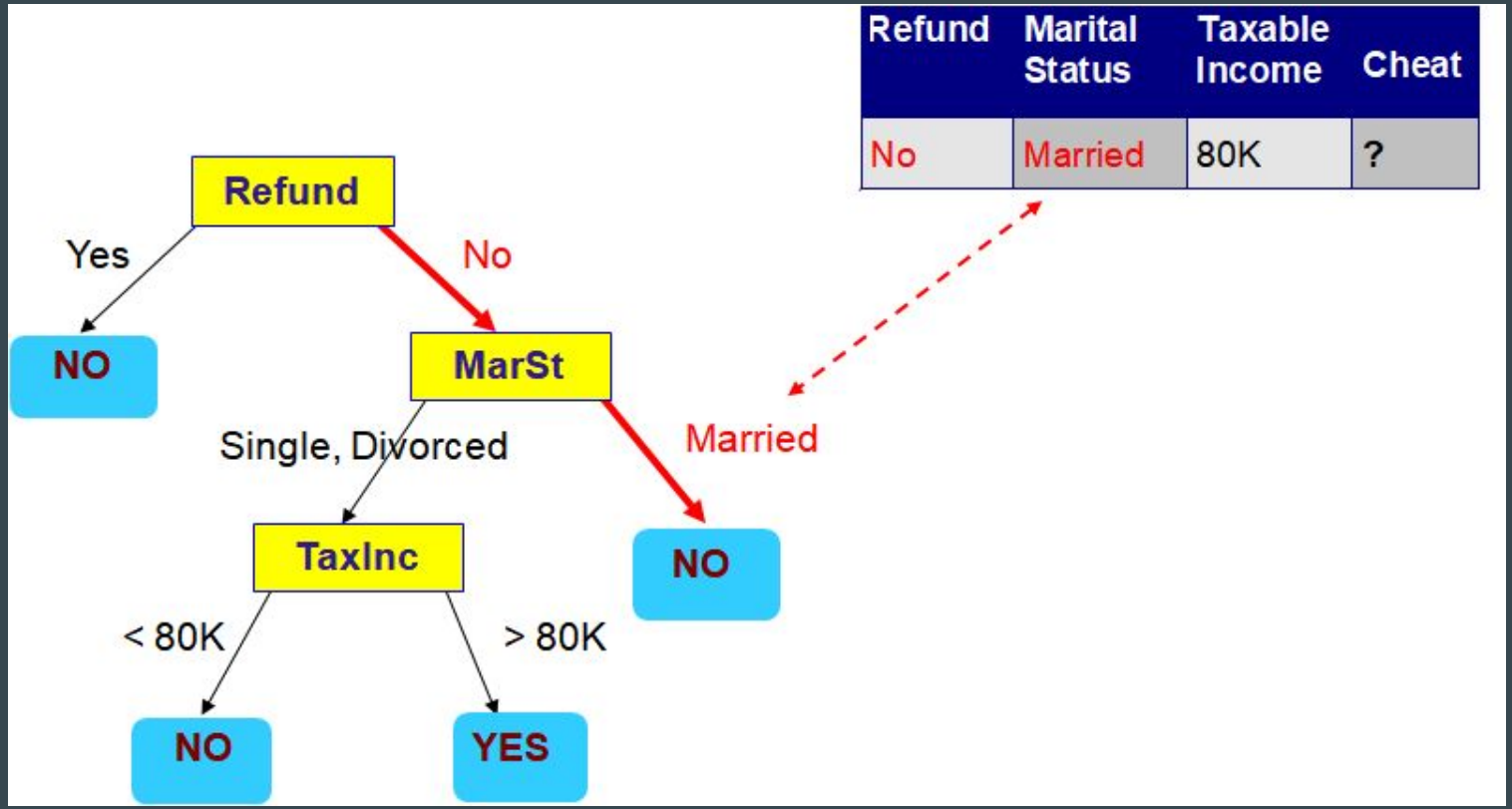
Decision Tree in Action



Decision Tree in Action

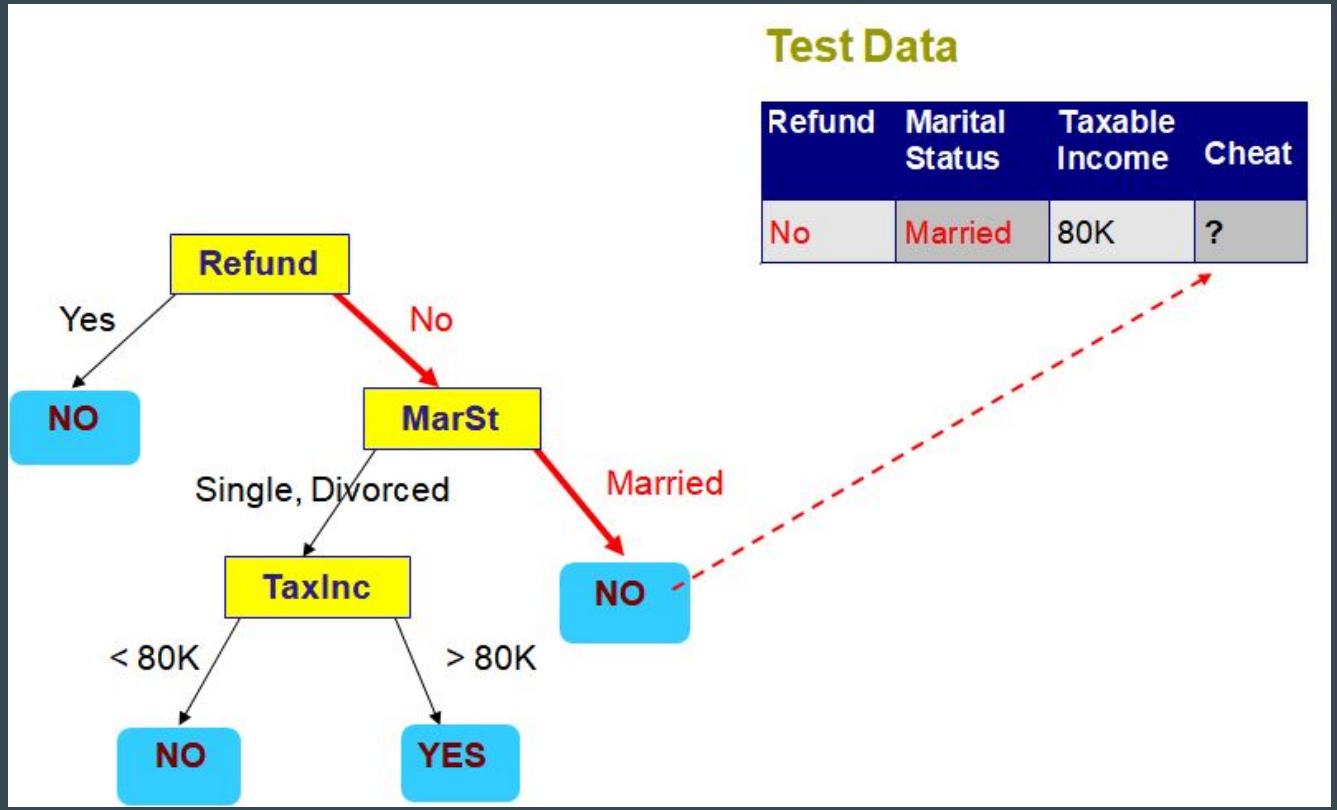


Decision Tree in Action



Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Decision Tree in Action



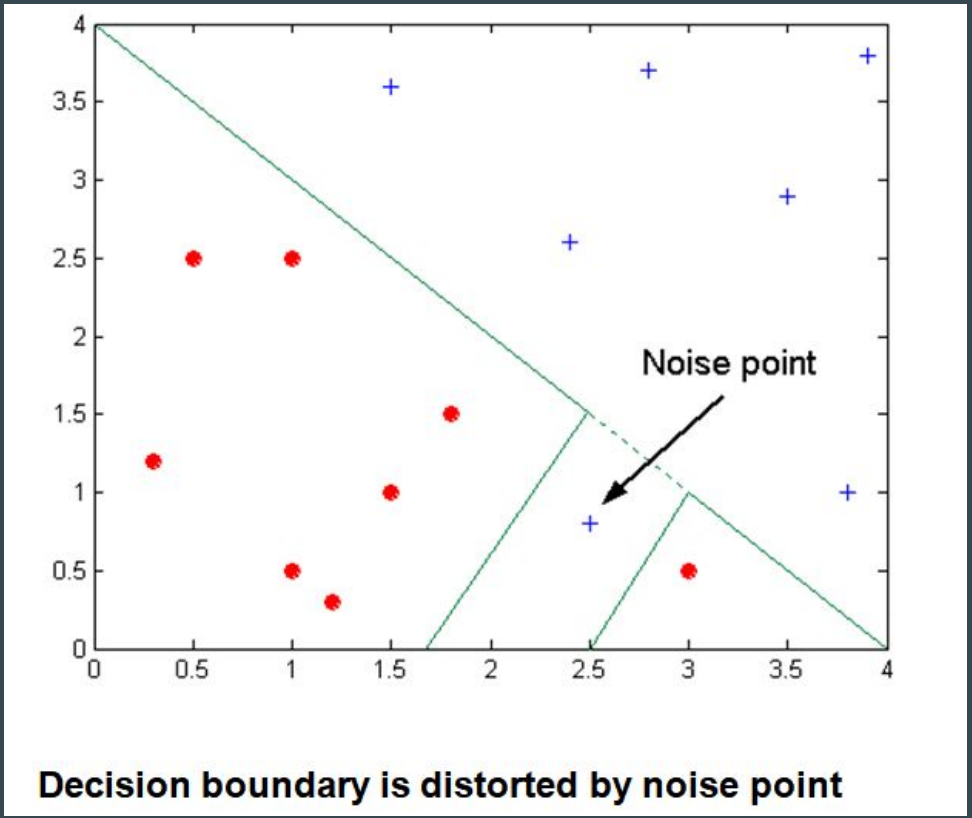
Overfitting

A major Problem in Decision Trees

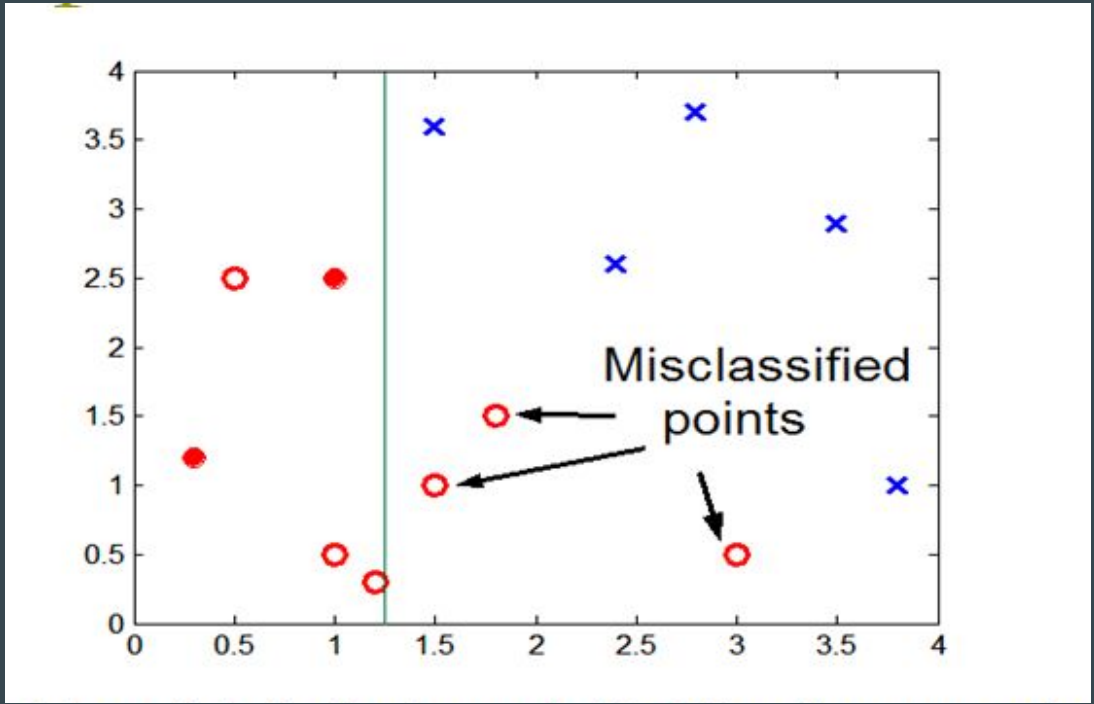
If your machine learning algorithm fits noise (i.e. pays attention to parts of the data that are irrelevant) it is overfitting

If your machine learning algorithm is overfitting then it may perform less well on test set data.

Overfitting due to Noise

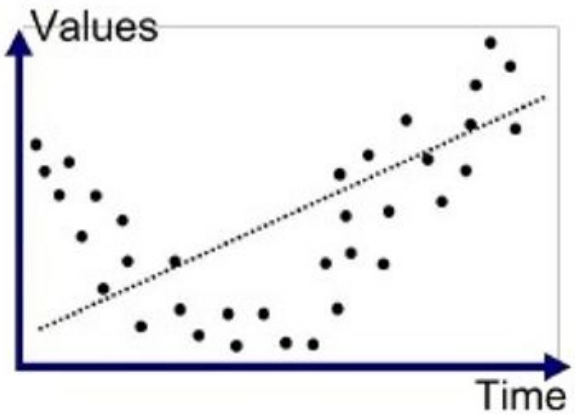


Overfitting due to Insufficient Data

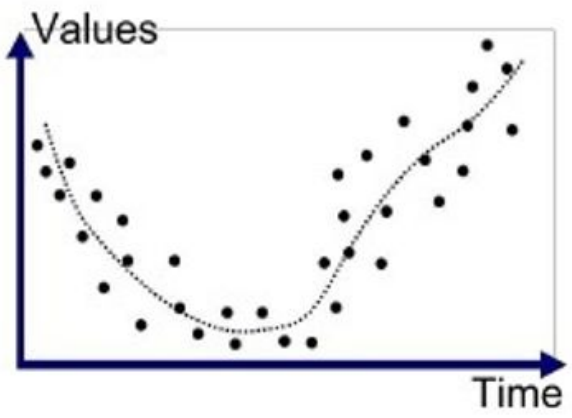


Lack of data points in the lower half of the diagram makes it difficult to predict correctly the class labels of that region

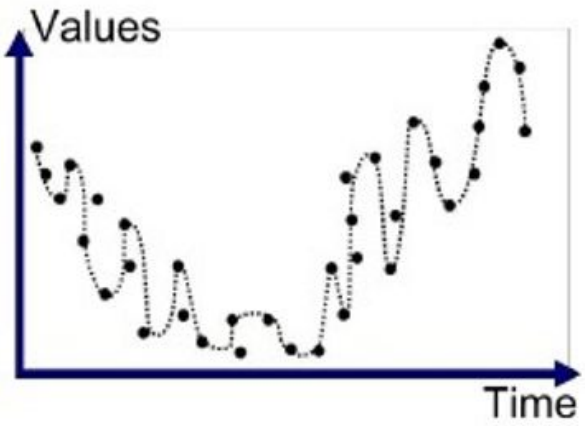
- Insufficient number of training records in the region causes the decision tree to predict the test examples using other training records that are irrelevant to the classification task



Underfitted



Good Fit/Robust



Overfitted

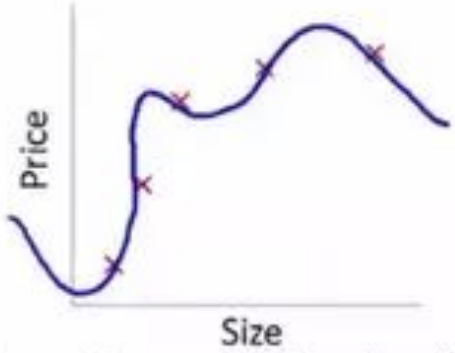
Bias and Variance in Model - Reason for non-robust fit

$\theta_0 + \theta_1 x$

High bias
(underfit)

$\theta_0 + \theta_1 x + \theta_2 x^2$

"Just right"



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

High variance
(overfit)

Ensemble Methods

Overcoming Problems with D-Trees

- Bagging
  - Random Forests
- Boosting
  - Gradient Boosting

Bagging

- Trains multiple estimators on the dataset
- Prediction is a calculated statistical measure on predictions of all estimators
- Example- Random Forest
  - constructs a multitude of randomised decision trees at training time and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees

Boosting

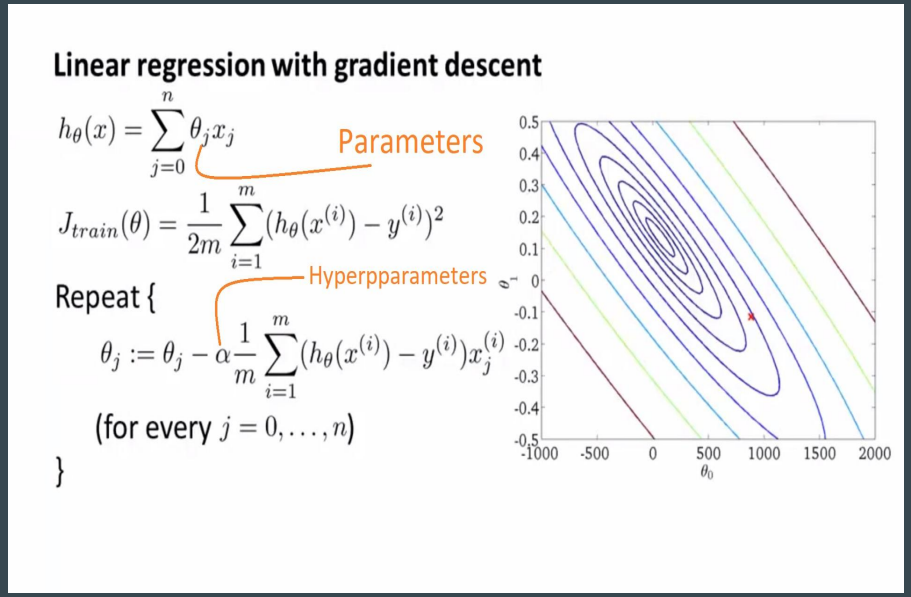
- Makes a weak learner learn iteratively on the samples it performs worse
- Keeps spawning new estimators on a modified data set of large number of samples on which the estimator had performed badly in the previous iteration
- The weak learner must be able to perform better than random on the data, else the algorithm fails.
- Example - Gradient Boosting
    - boosting can be interpreted as an optimization algorithm on a suitable cost function

Hyperparameter Optimization

- Difference between model parameters and hyperparameters
- General Conventions of choosing hyperparameters
- Examples

Difference between model parameters and hyperparameters

- Parameters of a model can be learnt using goodness of fit on data
- Hyperparameters of model can not be learnt using goodness of fit on data, rather they are figured out using the performance metrics, often, through manual tweaking.



Convention for choosing Hyper Parameters

- **Cross Validation Set** : A portion of training set, kept aside for cross-validating the learning task of the model
- The cross-validation set is used, in iterations, with different configurations of hyperparameters to see which set performs better
- Generally, learning rates and regularization rates are increased or decreased by a factor of 3 in successive epochs.
  $\alpha=3\alpha$ or $\alpha=\alpha/3$
- Same heuristic could be applied for the depth of tree in decision trees and dropout rate in neural networks.

Vectorised Hypothesis

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$. $(x_0^{(i)} = 1)$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n \quad (x_0 = 1)$$

$$= \theta^T x$$

θ^T: $(n+1) \times 1$ matrix

$\theta^T x$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$