Introduction To Clustering

Clustering

Considered to be the most important technique of unsupervised learning.

A cluster is a collection of data objects which are

- similar to one another within the same group (class or category)
- different from the objects in the other clusters.

Clustering is

- an unsupervised learning technique
- predefined classes and prior information which defines how the data should be labelled into separate classes

Uses

- to discover hidden patterns of interest or structure in data
- sometimes as a pre-processing step in other algorithms

Why Cluster

Clustering allows us to find hidden relationship between the data points in the dataset.

Examples:

- In marketing, customers are segmented according to similarities to carry out targeted marketing.
- Given a collection of text, we need to organize them, according to the content similarities to create a topic hierarchy
- Detecting distinct kinds of pattern in image data (Image processing). It’s effective in biology research for identifying the underlying patterns.

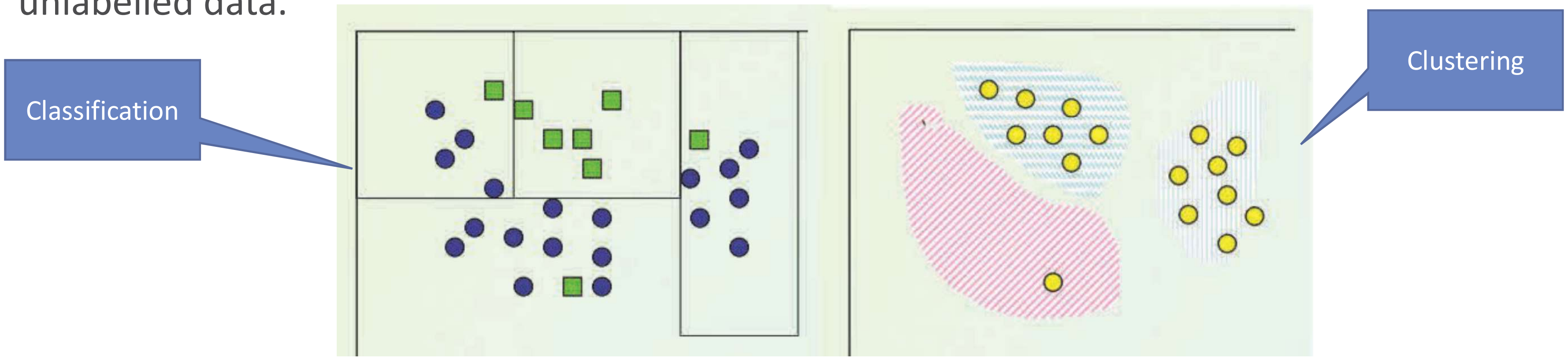
Classification vs Clustering

Classification

- In Supervised learning a model learns a method for predicting the instance class from a pre-labelled (classified) instances.

Clustering

- In unsupervised learning a model tries to find **"natural"** grouping of instances for a given unlabelled data.



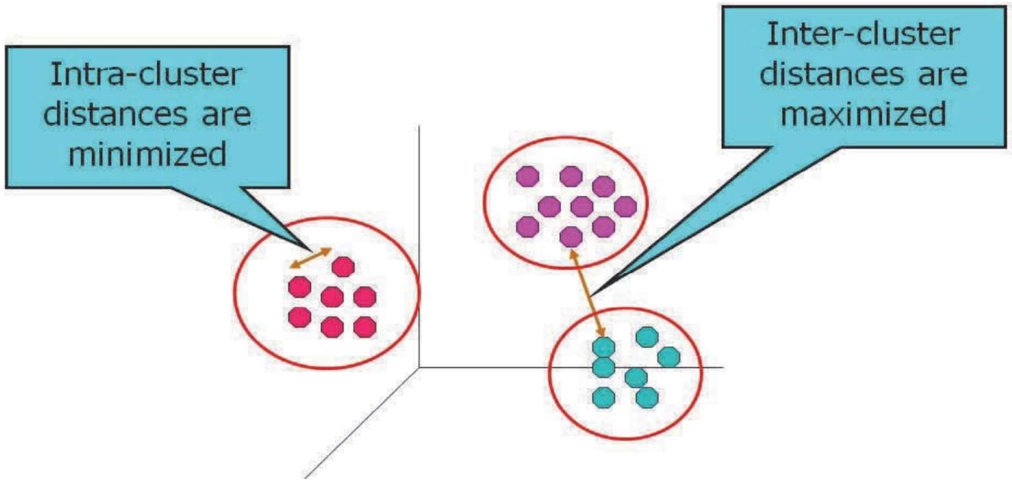
How to define Clustering Algorithms

Clusters are created by

- reducing the distance between objects in the same cluster – **intra-cluster** minimization
- increasing the distance between objects in other clusters – **inter-cluster** maximization

Intra-cluster minimization
The closer the objects in a cluster, the more likely they belong to the same cluster.

Inter-cluster Maximization
This makes the separation between two clusters. The main goal is to maximize the distance between 2 clusters.

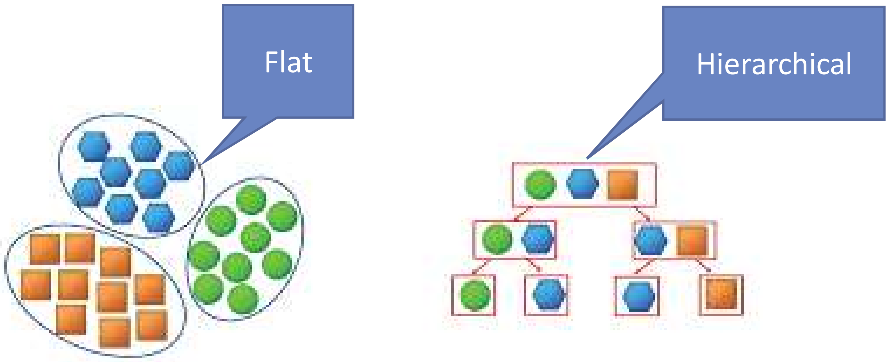


Types of Algorithm

Many types of algorithm, using different techniques

Flat or partitioning algorithm

Tries to divide the dataset of interest into predefined number of groups/ clusters.
All groups/ clusters are independent of each other.
e.g.: **K-means**



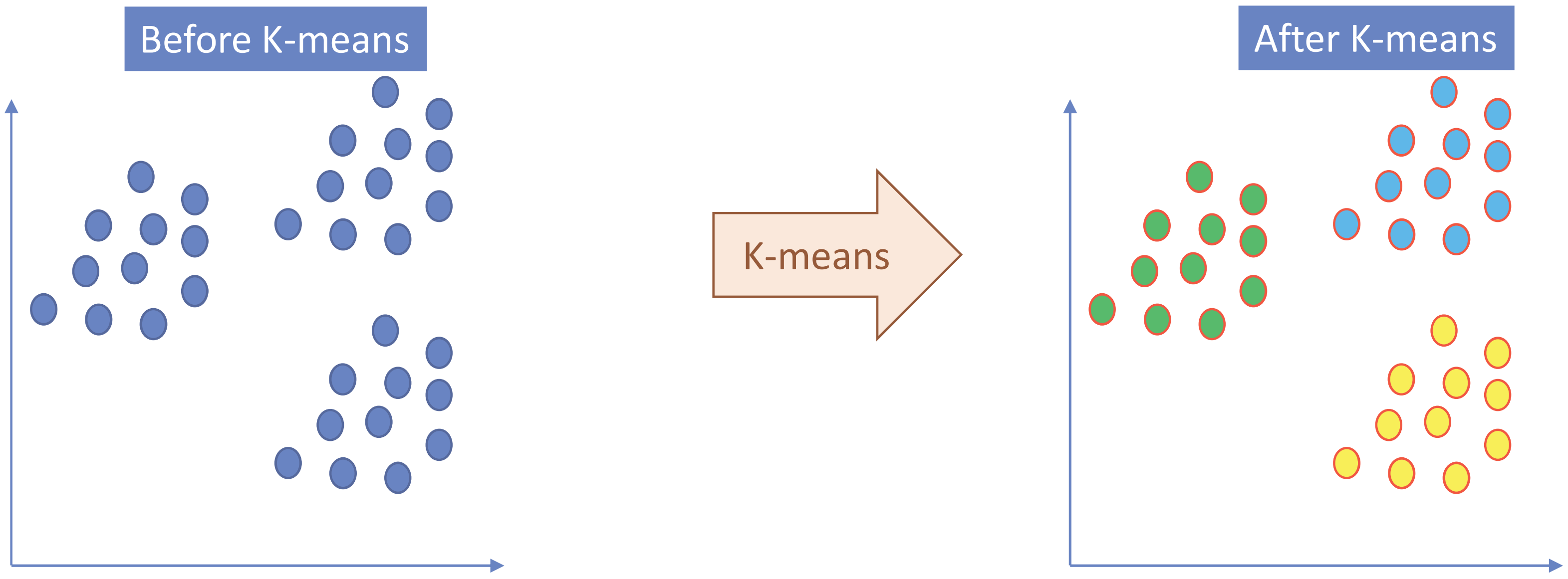
Hierarchical Clustering algorithm

Does not partition
Multiple steps which run from a single cluster containing all the data points to n clusters containing single data point.
This algorithm is further classified into **Divisive** and **Agglomerative** Methods.

K-means Clustering

K-means

Can work with multi-dimensional data.



K-means

Can work with multi-dimensional data.

K-Means algorithm

1 – select number (k) of clusters

2 – Select at random K points (centroids)

Not necessarily from the dataset

3 – Assign each datapoint to the closest centroid

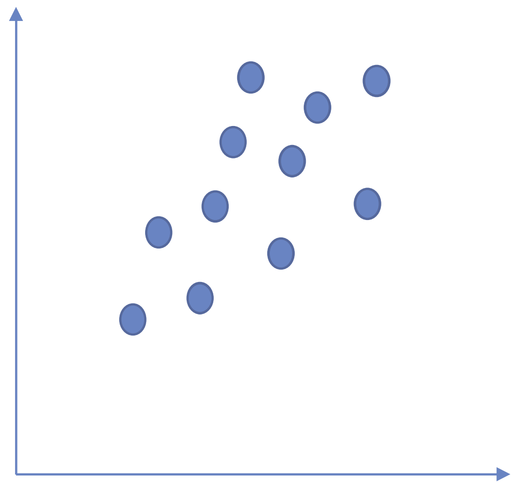
Form K-clusters

4 – Compute and place the new centroid of each cluster

5 – Reassign each datapoint to the new closest centroid

If any reassignment took place, go to step 4, otherwise the model is ready.

K-Means – Step 1

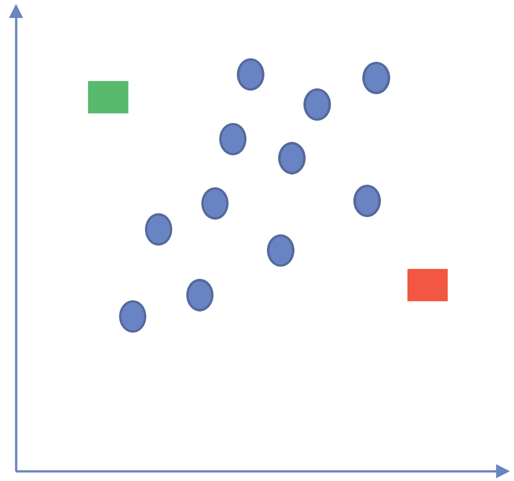


Step 1

Choose the number K of clusters

e.g K = 2

K-Means – Step 2

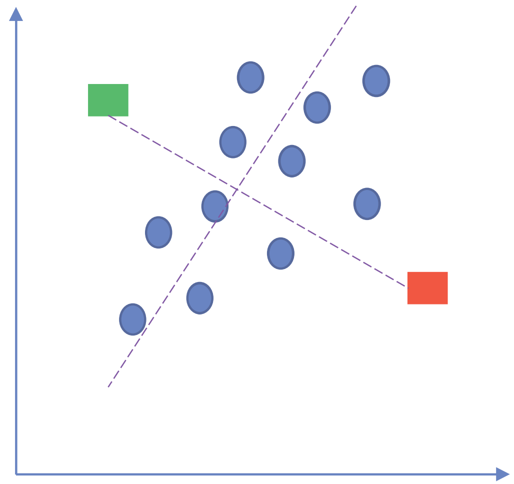


Step 2

Select at random K points (centroids)

Not necessarily from your dataset

K-Means – Step 3

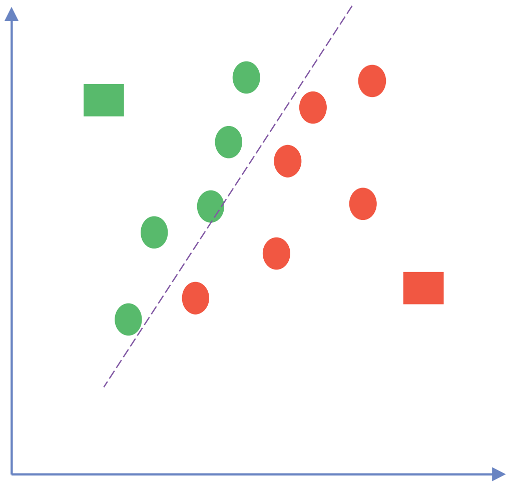


Step 3

Assign each datapoint to the closest centroid

Form K-clusters

K-Means – Step 3

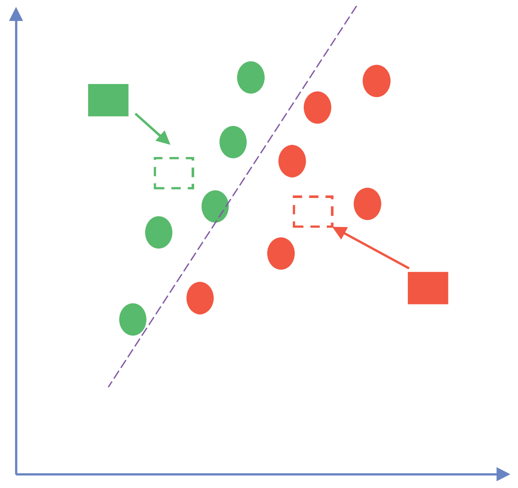


Step 3

Assign each datapoint to the closest centroid

Form K-clusters

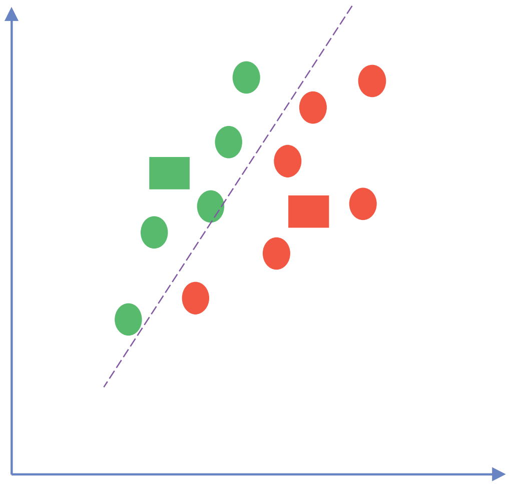
K-Means – Step 4



Step 4

Compute and place the new centroid of each cluster

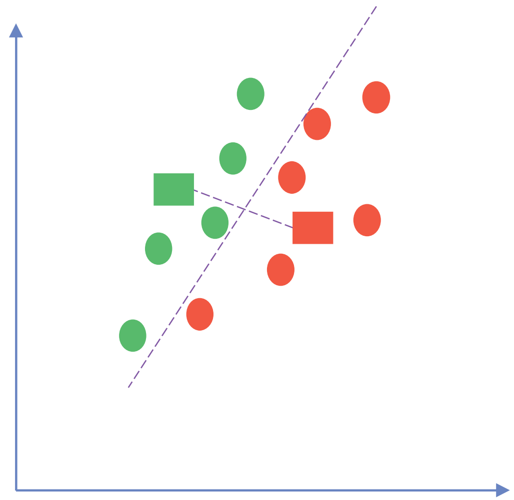
K-Means – Step 4



Step 4

Compute and place the new centroid of each cluster

K-Means – Step 5

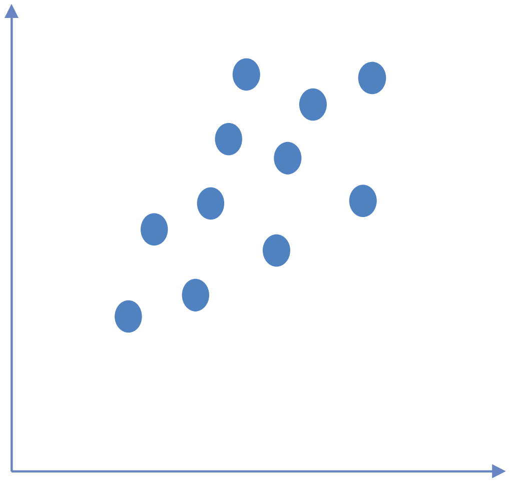


Step 5

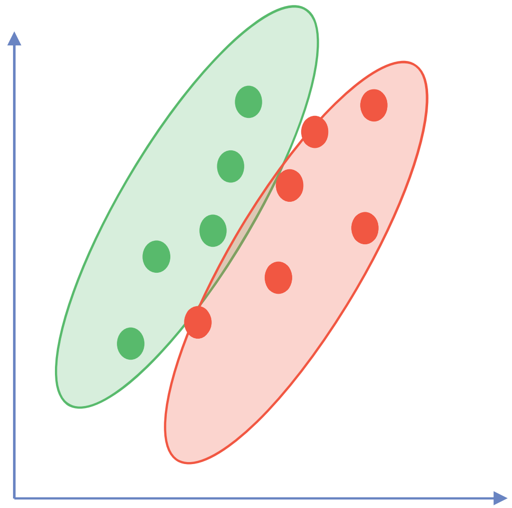
Reassign each datapoint to the new closest centroid

If any reassignment took place, go to step 4, otherwise finish

K-Means

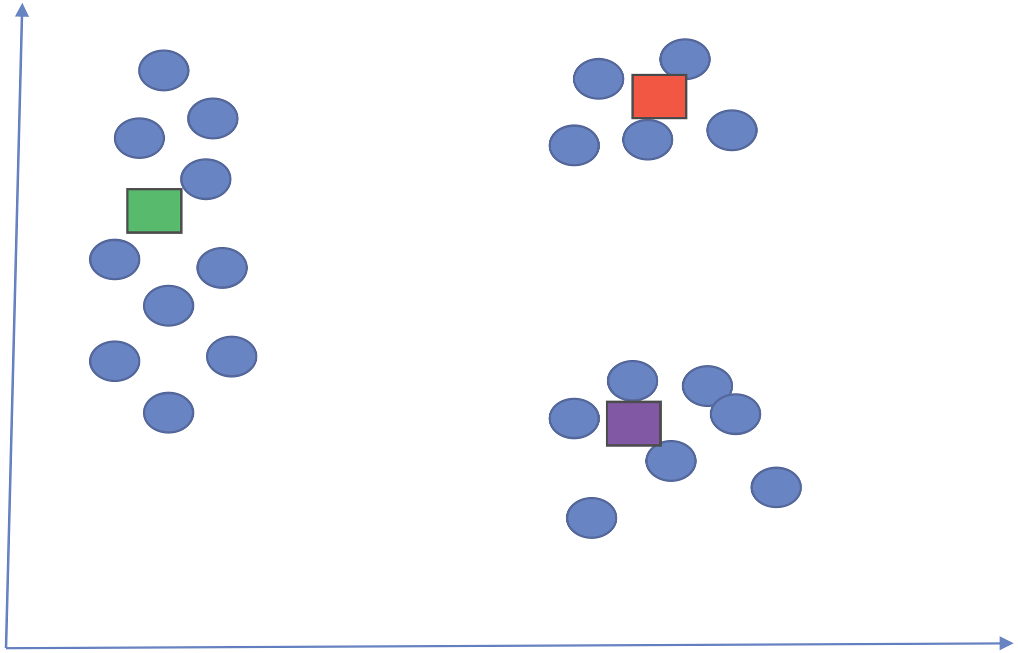


Before K-Means



After K-Means

Random Initialization Trap

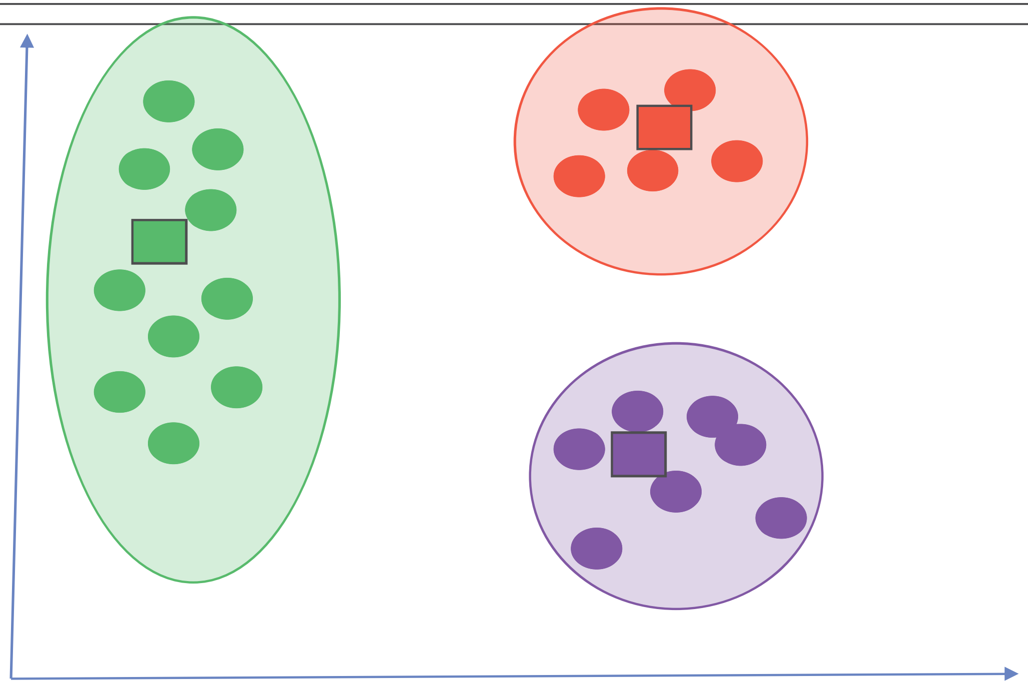


Scenario 1

You **manually** choose initial centroids

Based on your intuition

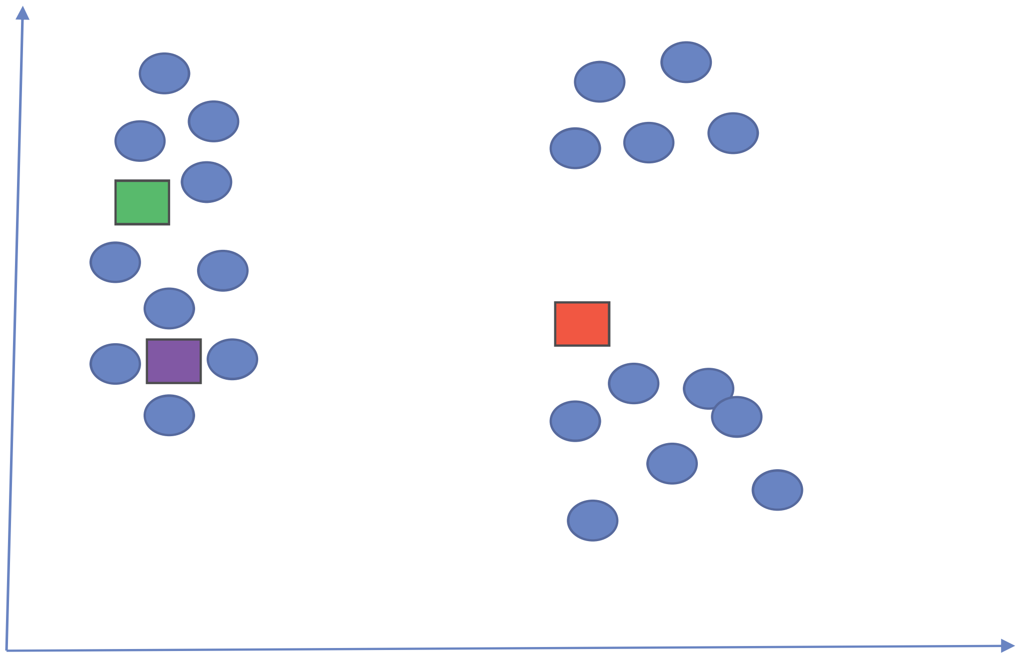
Random Initialization Trap



Scenario 1

K-means might cluster as follows

Random Initialization Trap

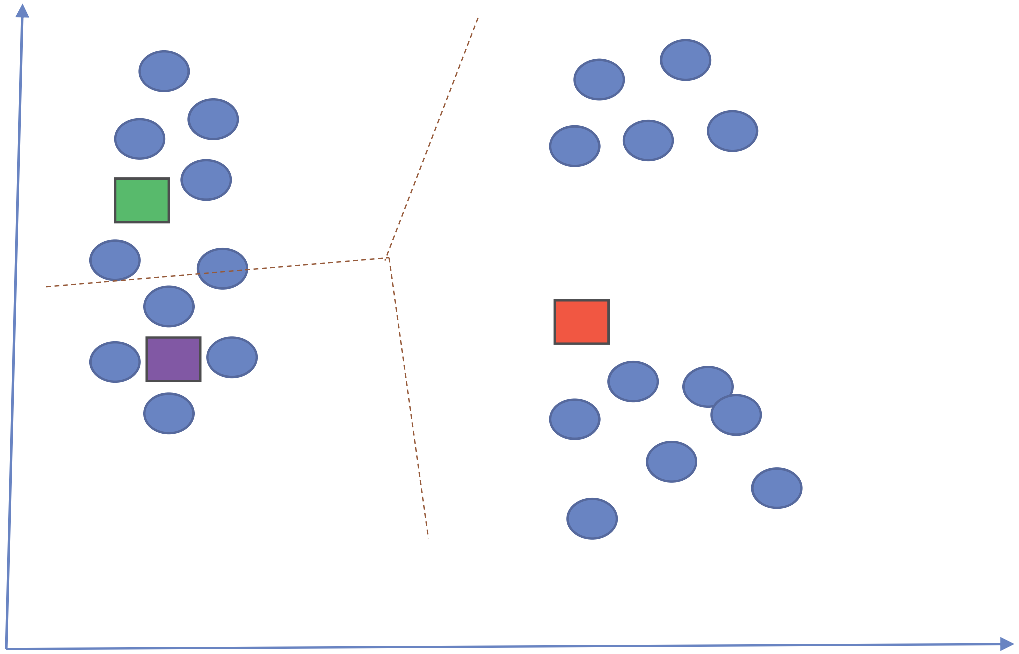


Scenario 2

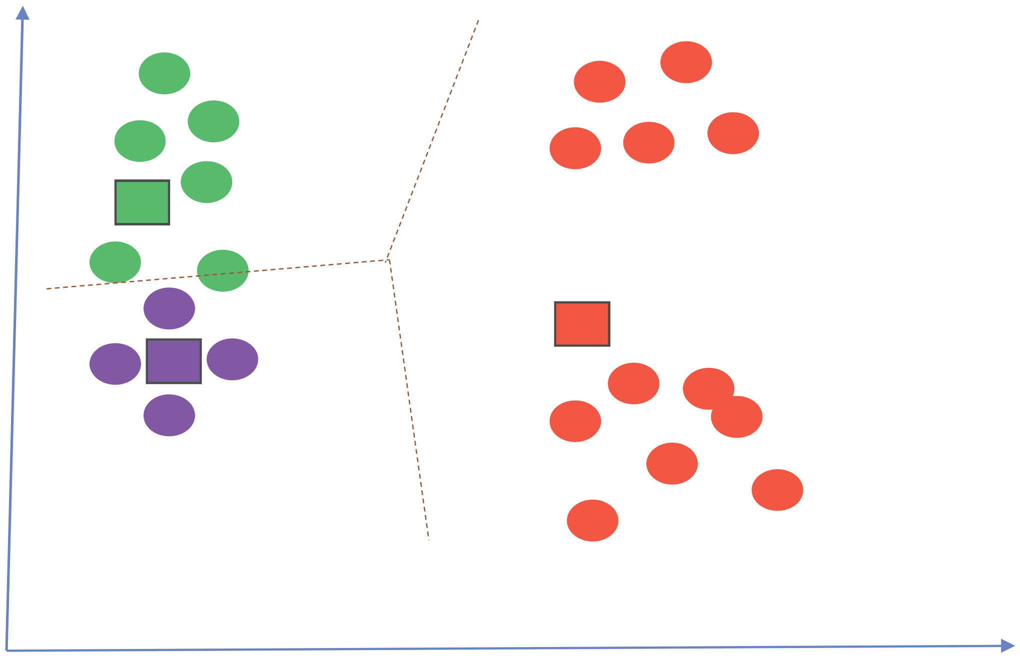
You **randomly** choose initial centroids

Based on your intuition

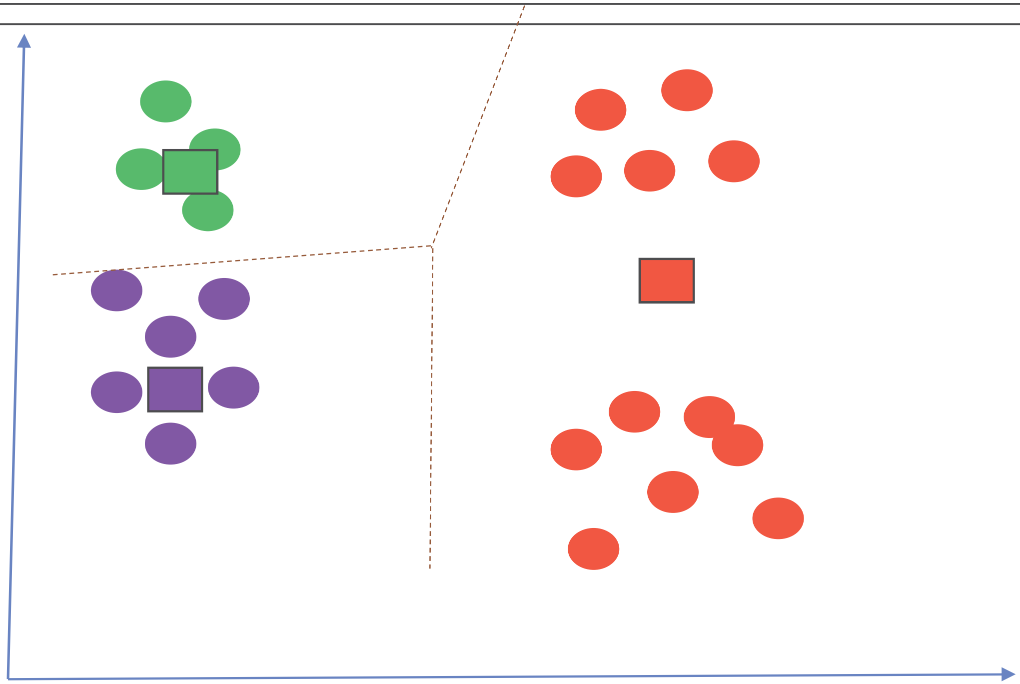
Random Initialization Trap



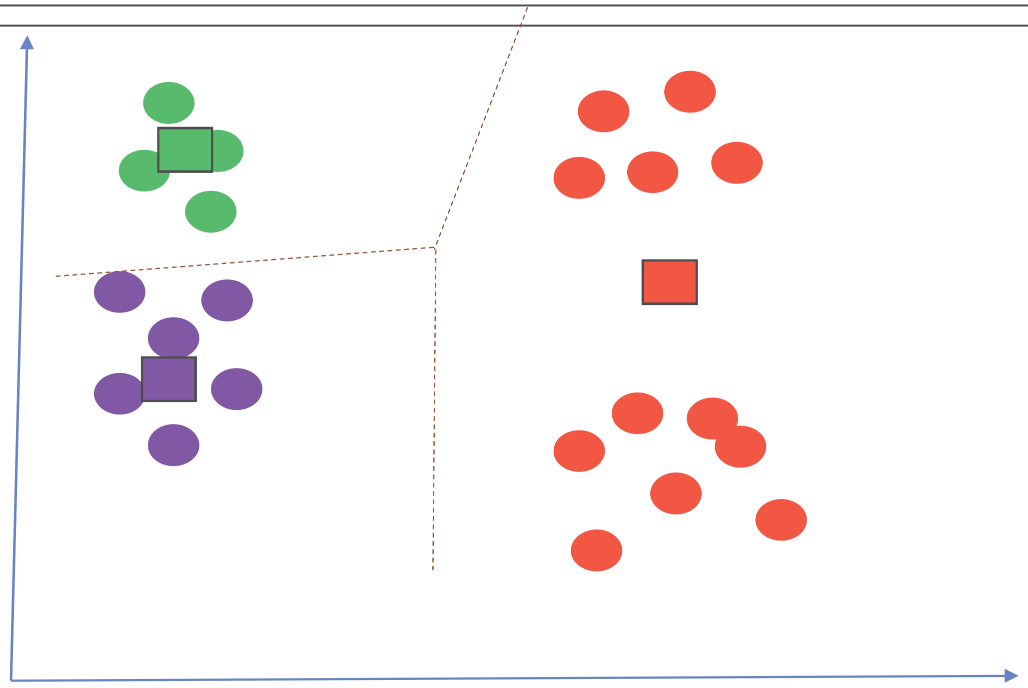
Random Initialization Trap



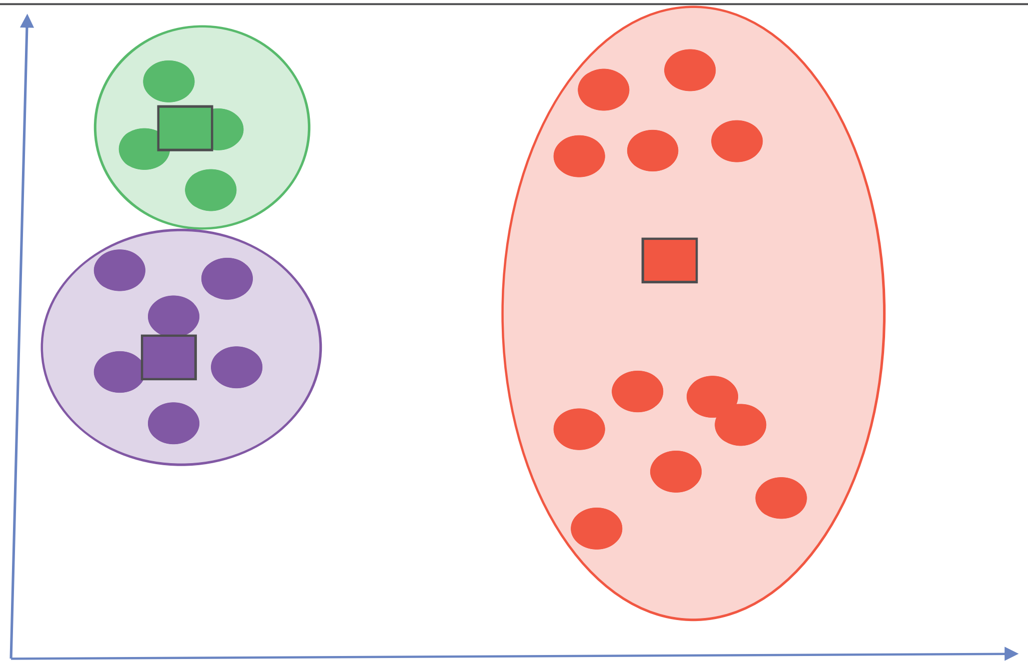
Random Initialization Trap



Random Initialization Trap



Random Initialization Trap

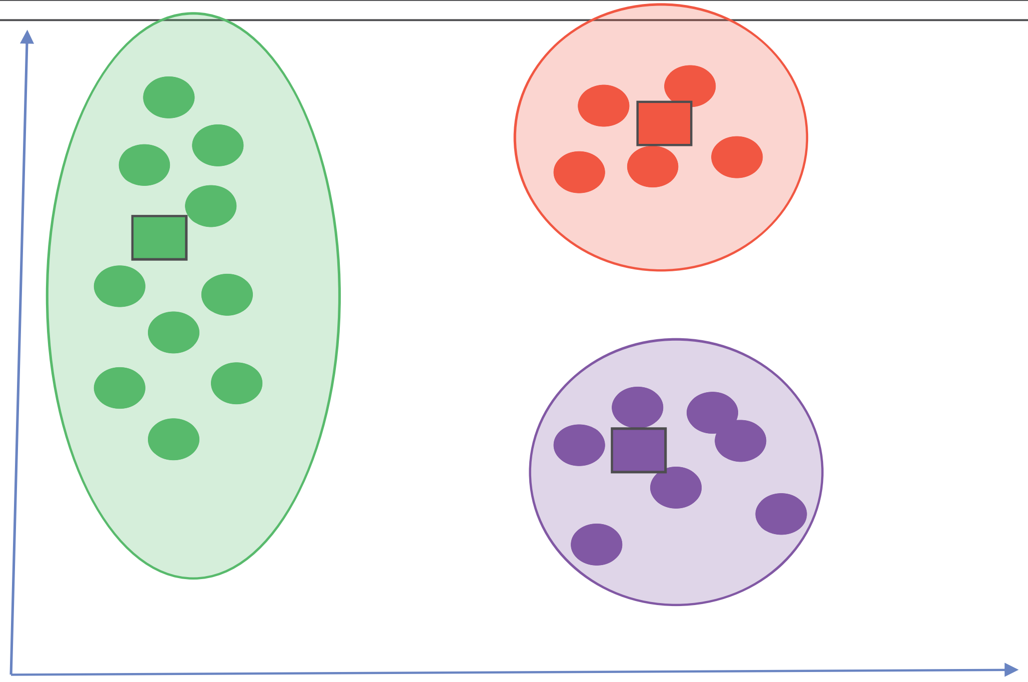


Scenario 2

Model converges with a False Result

Due to poor initial choice of centroids

Random Initialization Trap



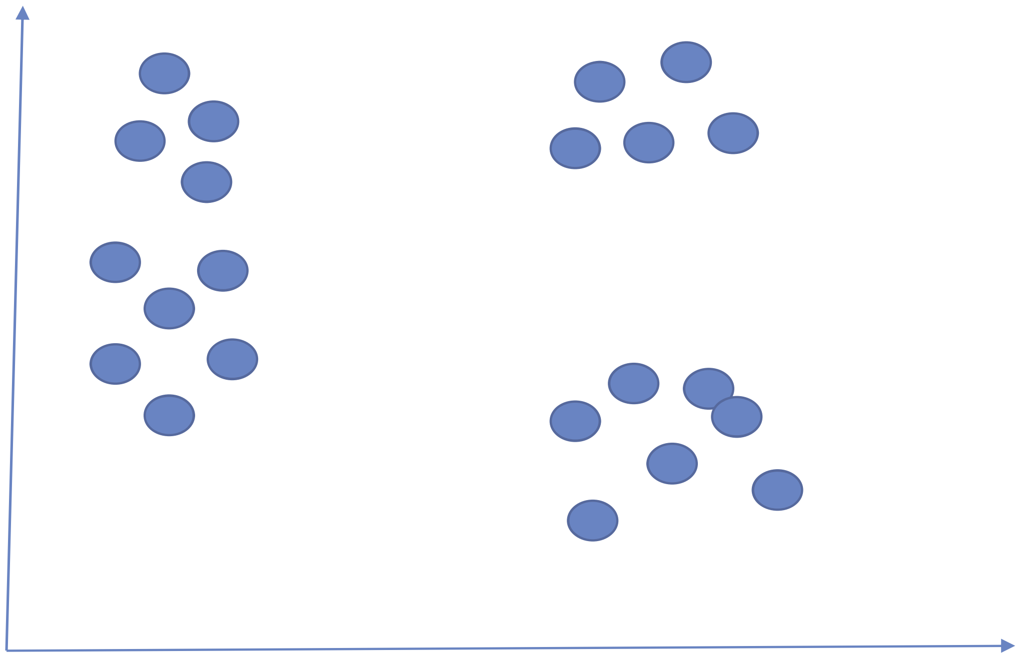
Scenario 1 - Original

Model converges with a True Result

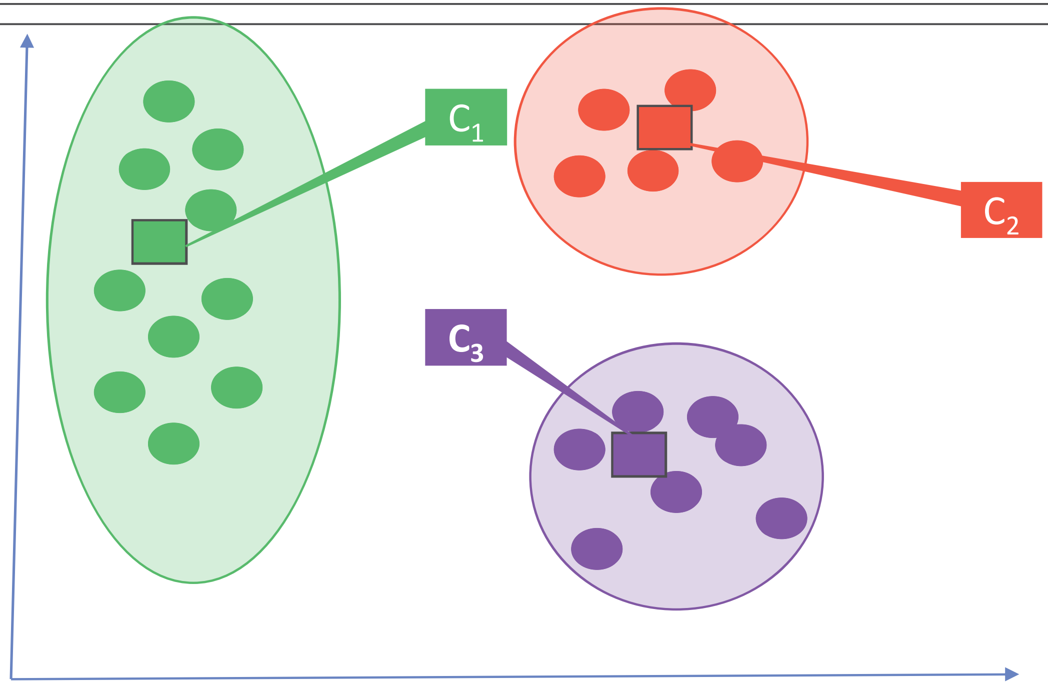
Due to good initial choice of centroids

K-means++ algorithm will fix this
Complicated & beyond this course
Most ML libraries have it as a *hyper parameter*
Happens in the background

Choosing the Right number of Clusters



Choosing the Right number of Clusters



Is 3 the correct number of clusters

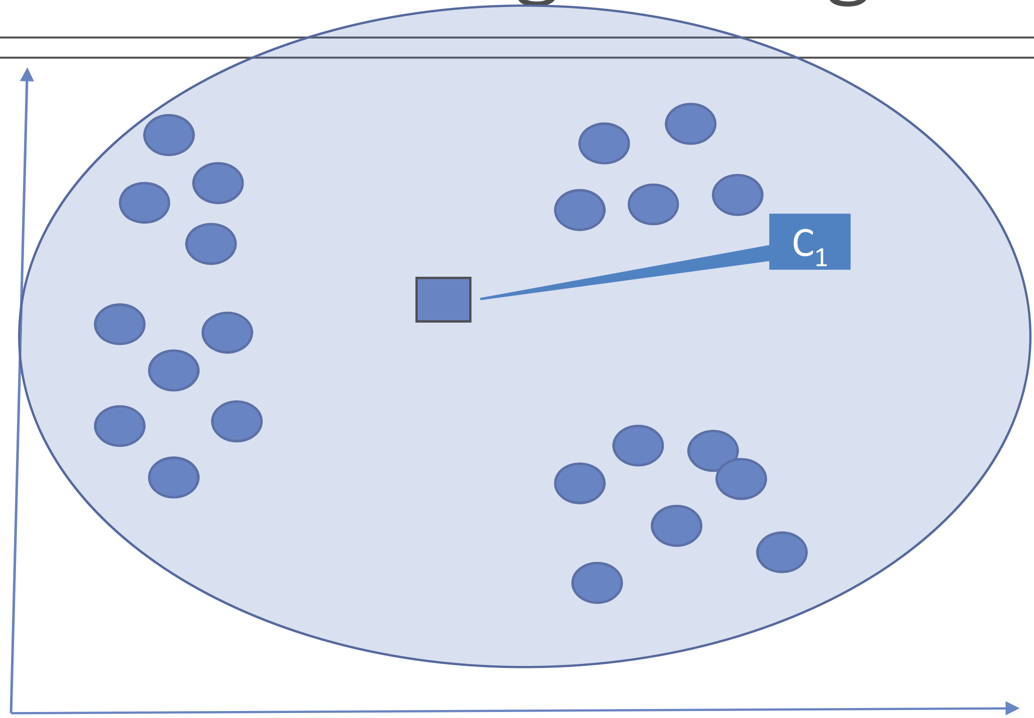
Why not 2 clusters ?

Why not 4 clusters ?

Calculate WCSS

$$WCSS = \sum_{P_i \text{ in } C1} distance(P_i, C_1)^2 + \sum_{P_i \text{ in } C2} distance(P_i, C_2)^2 + \sum_{P_i \text{ in } C2} distance(P_i, C_2)^2$$

Choosing the Right number of Clusters



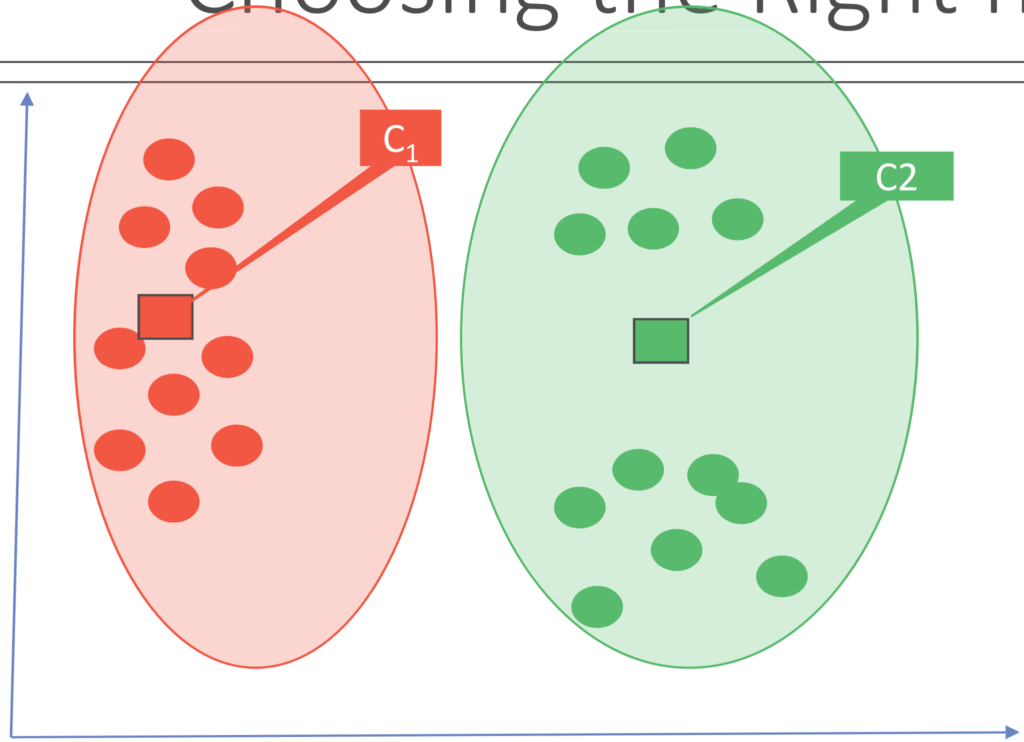
For 1 Cluster

Lots of points in the dataset

WCSS usually large

$$WCSS = \sum_{P_i \text{ in } C1} distance(P_i, C_1)^2$$

Choosing the Right number of Clusters



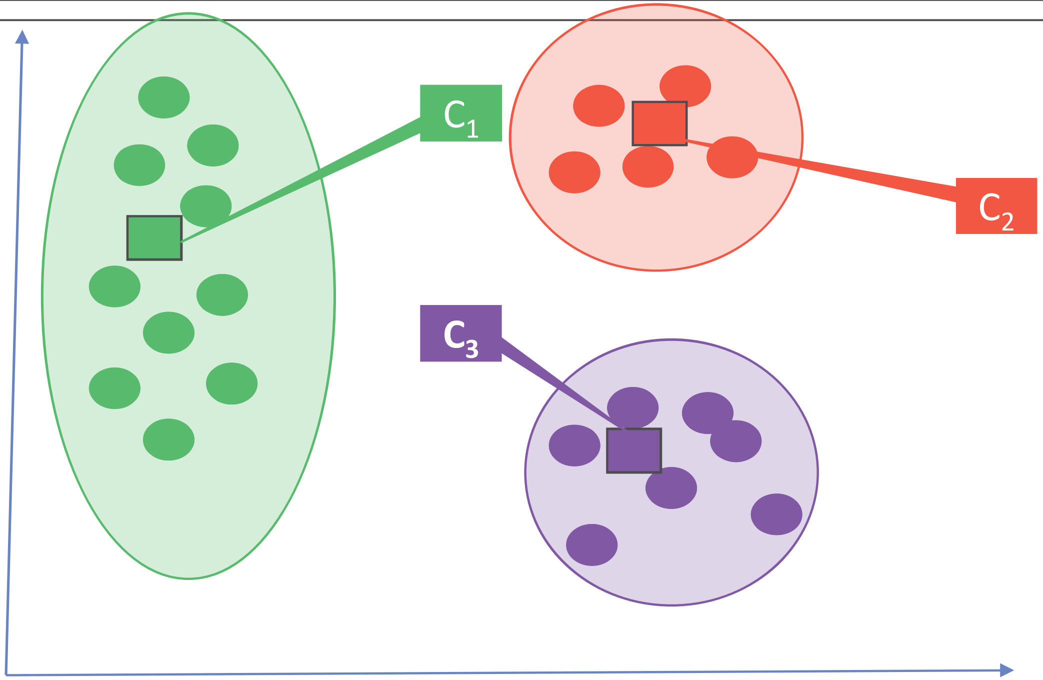
For 2 Clusters

Fewer points in each dataset

WCSS smaller than

$$WCSS = \sum_{Pi\ in\ C1} distance(P_i, C_1)^2 + \sum_{Pi\ in\ C2} distance(P_i, C_2)^2$$

Choosing the Right number of Clusters



For 3 Clusters

Smaller again

$$WCSS = \sum_{P_i \text{ in } C1} distance(P_i, C_1)^2 + \sum_{P_i \text{ in } C2} distance(P_i, C_2)^2 + \sum_{P_i \text{ in } C2} distance(P_i, C_2)^2$$

Choosing the Right number of Clusters

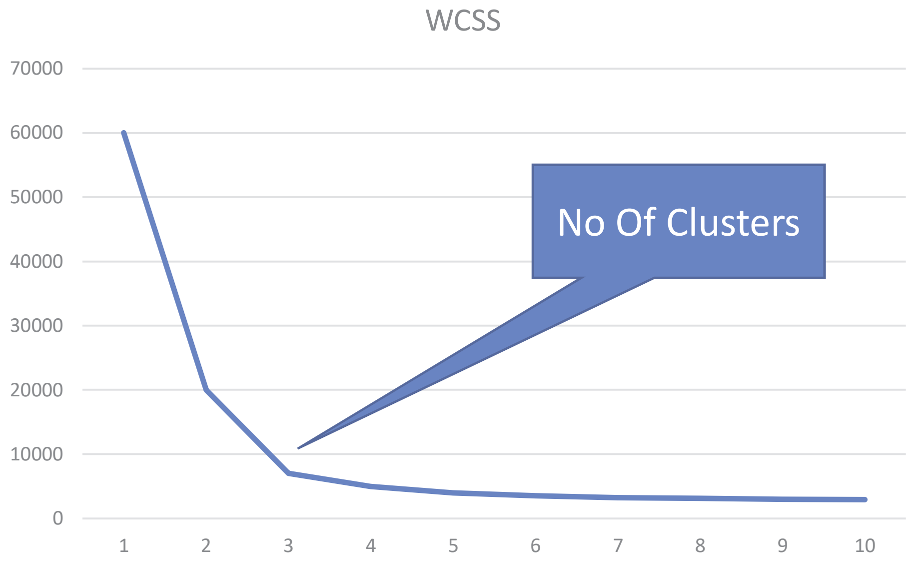
How many clusters are possible?

Ans – As many points / elements in the dataset
Each point is its own cluster

In the limit, WCSS equals ZERO

WCSS decreases as number of clusters increase

Use "**elbow**" method –
Trial and error
Scientists judgement



?

ARTIFICAL NERURAL NETWORKS

01 - Regression

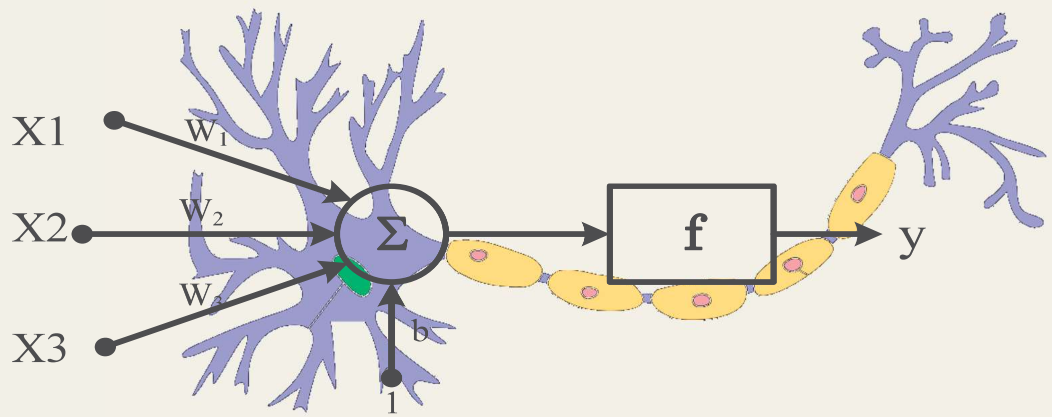
A.N.N. - Introduction

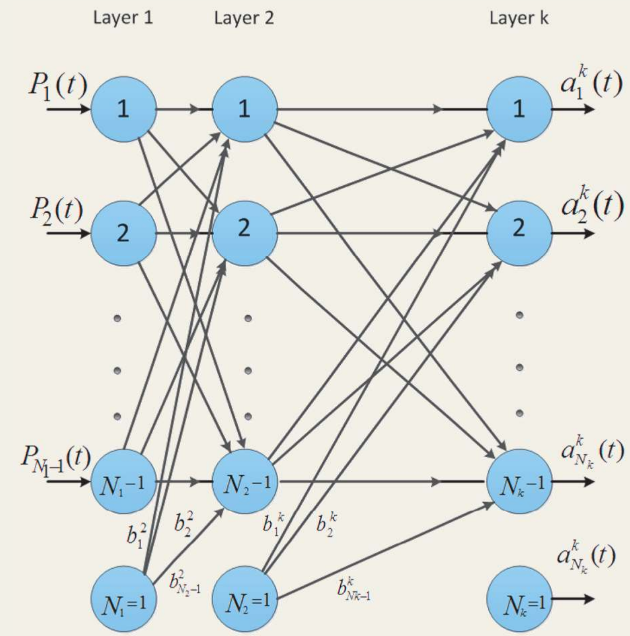
The brain has over 100 billion neurons communicating through electrical and chemical signals.

Neurons communicate with each other and help us see, think, and generate ideas.

Human brain learns by creating connections among these neurons.

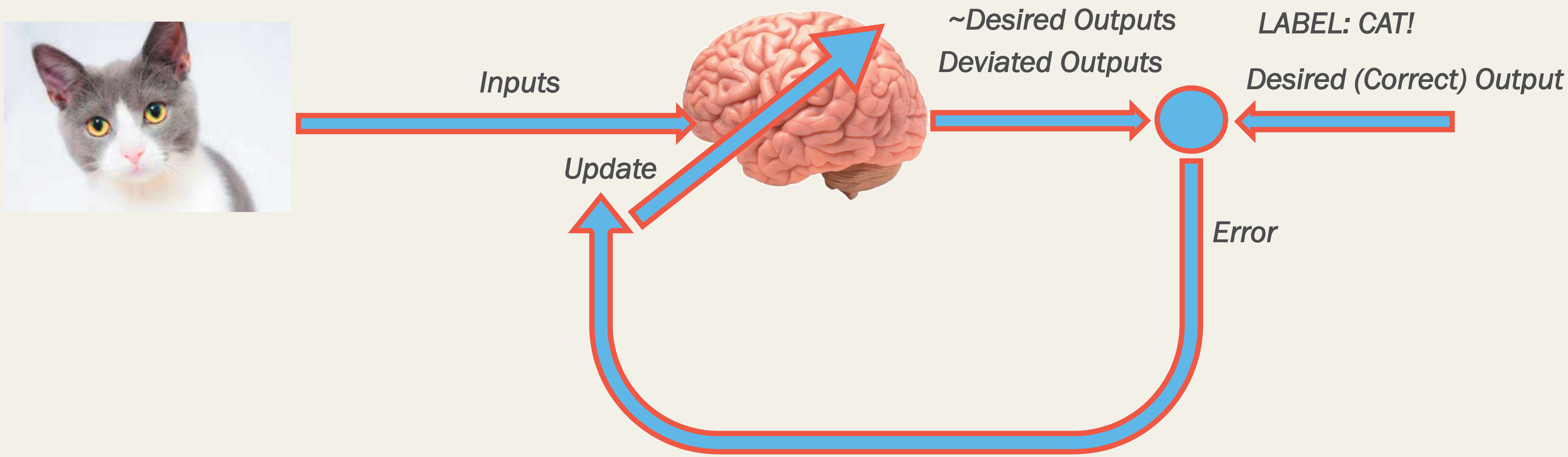
ANNs are information processing models inspired by the human brain.





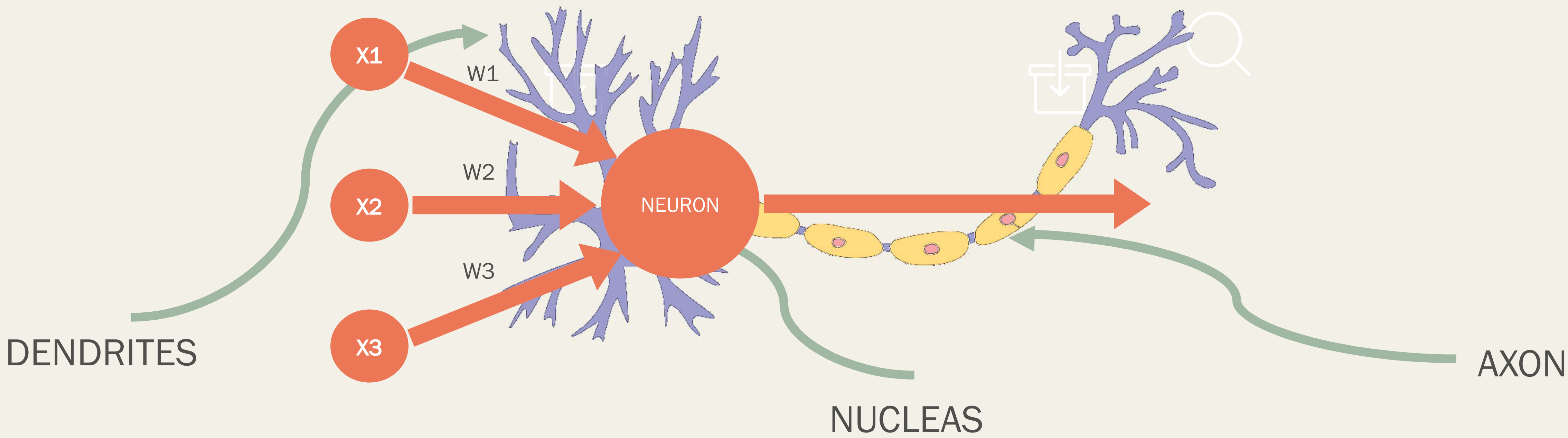
How to humans learn ?

Humans learn from experience (by example)



Neuron Mathematical Model

The neuron collects signals from input channels named dendrites, processes information in its nucleus, and then generates an output in a long thin branch called axon.

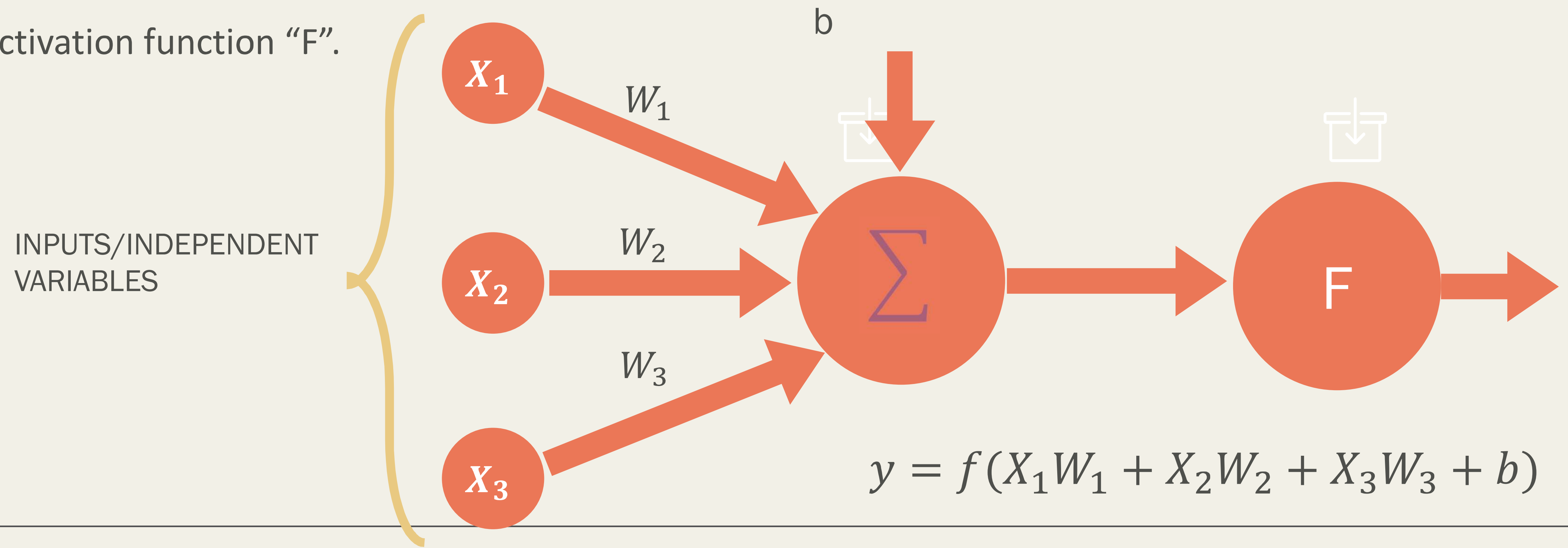


Neuron Mathematical Model

Bias allows to shift the activation function curve up or down.

Number of adjustable parameters = 4 (3 weights and 1 bias).

Activation function “F”.

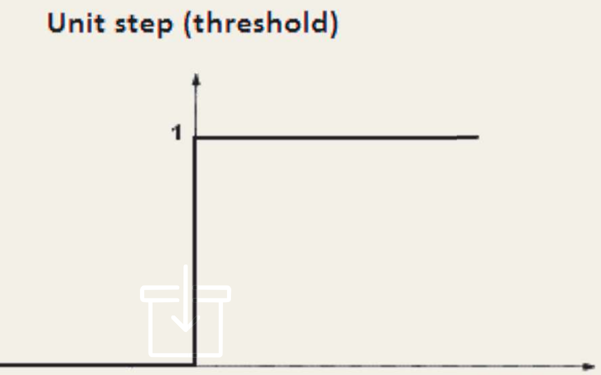


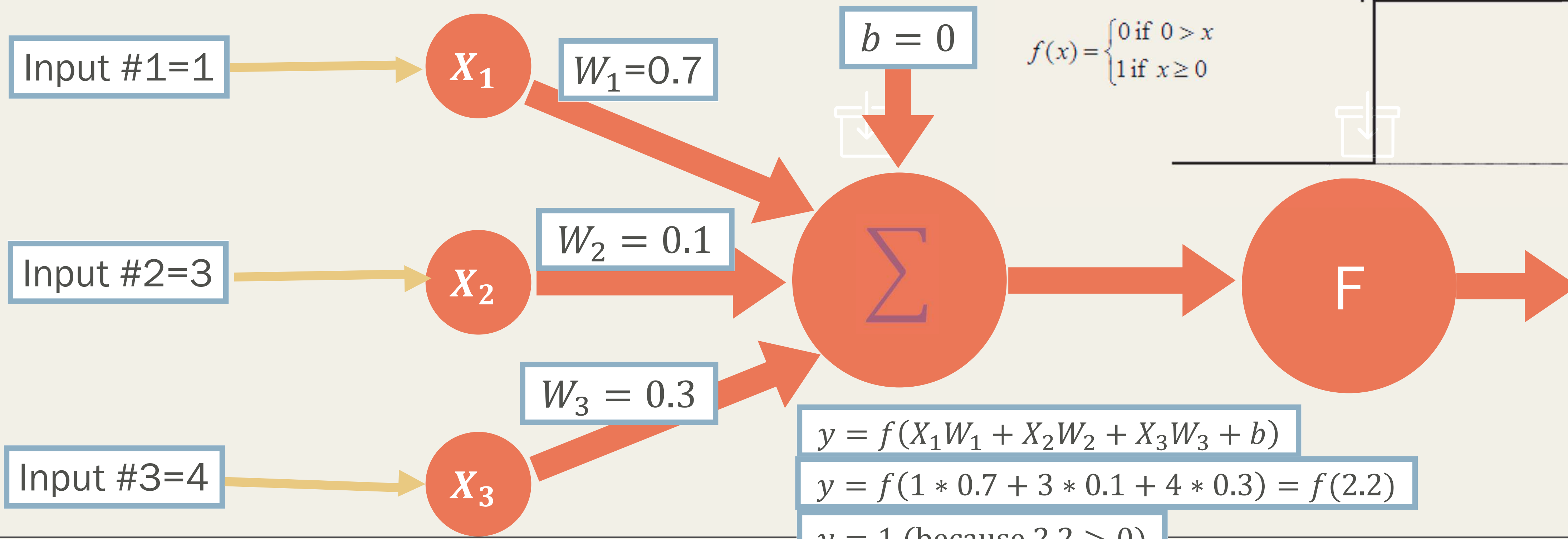
$$y = f(X_1W_1 + X_2W_2 + X_3W_3 + b)$$

Single Neuron Model

Assume that the activation function is a **Unit Step Activation Function**.

The activation functions is used to map the input between (0, 1).





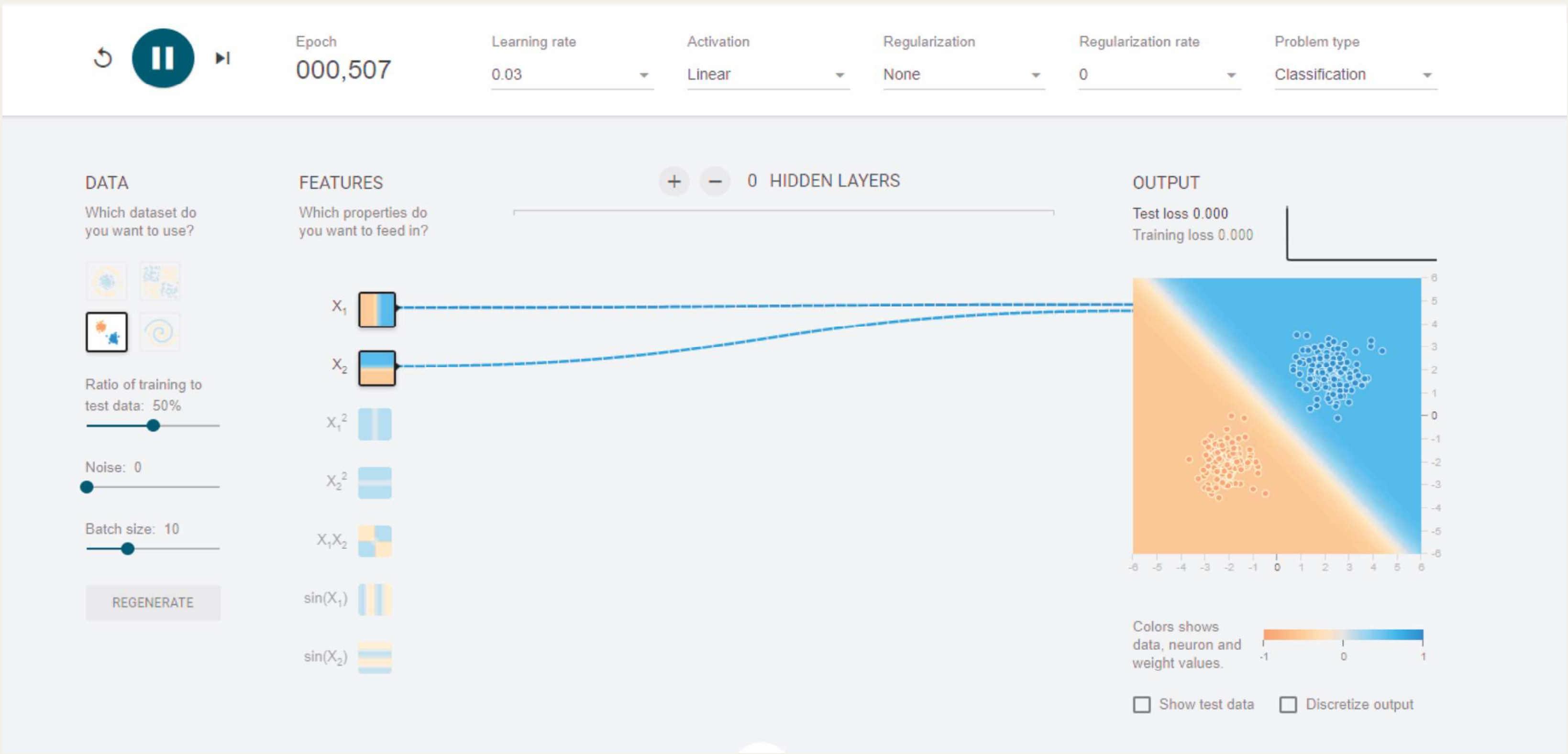
$y = f(X_1W_1 + X_2W_2 + X_3W_3 + b)$

$y = f(1 * 0.7 + 3 * 0.1 + 4 * 0.3) = f(2.2)$

$y = 1$ (because $2.2 > 0$)

Single Neuron Model

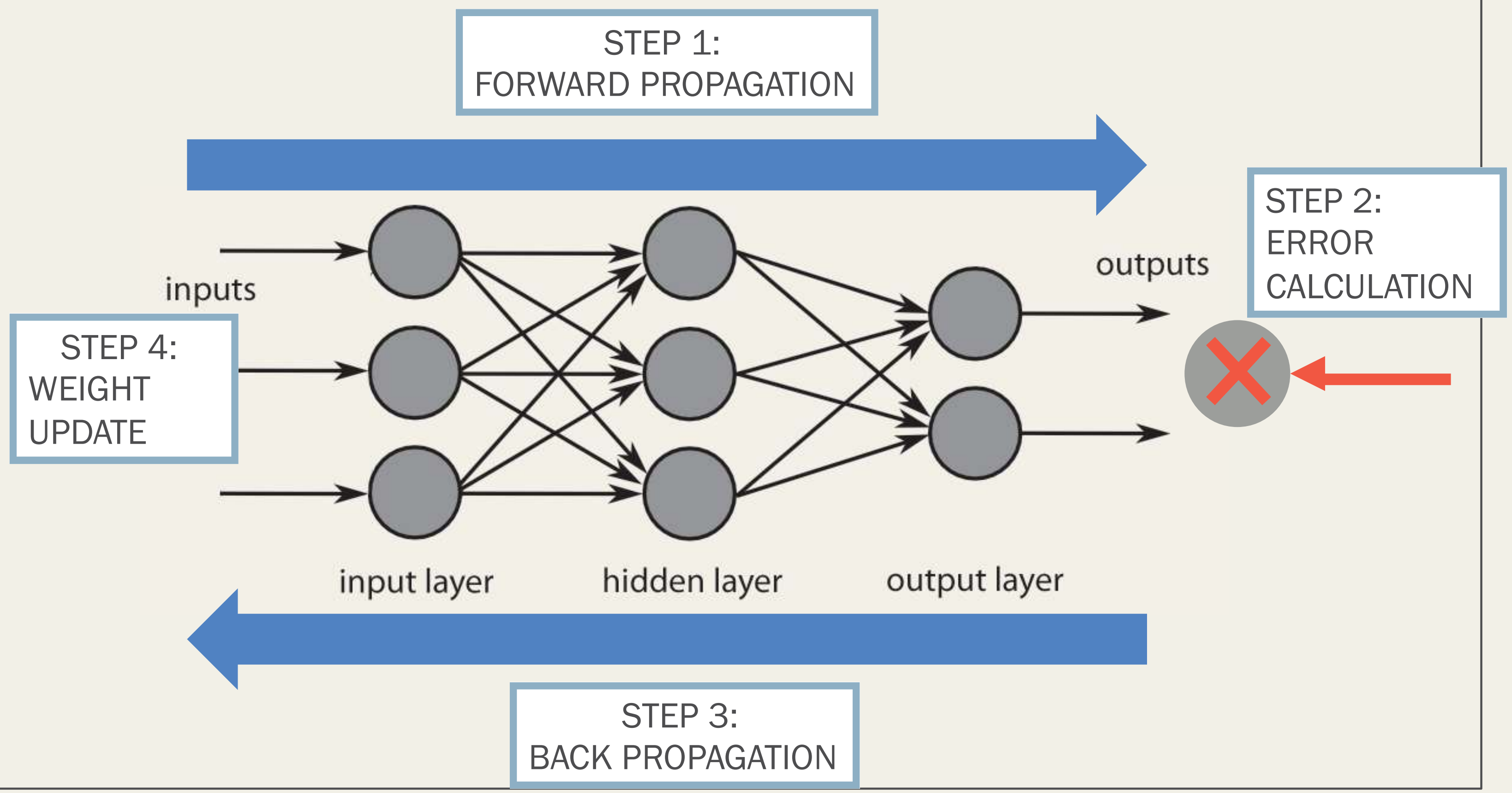
Try a neural network out : https://playground.tensorflow.org



TRAINING A NETWORK

Back Propagation

A method used to train ANNs by calculating gradient needed to update network weights.



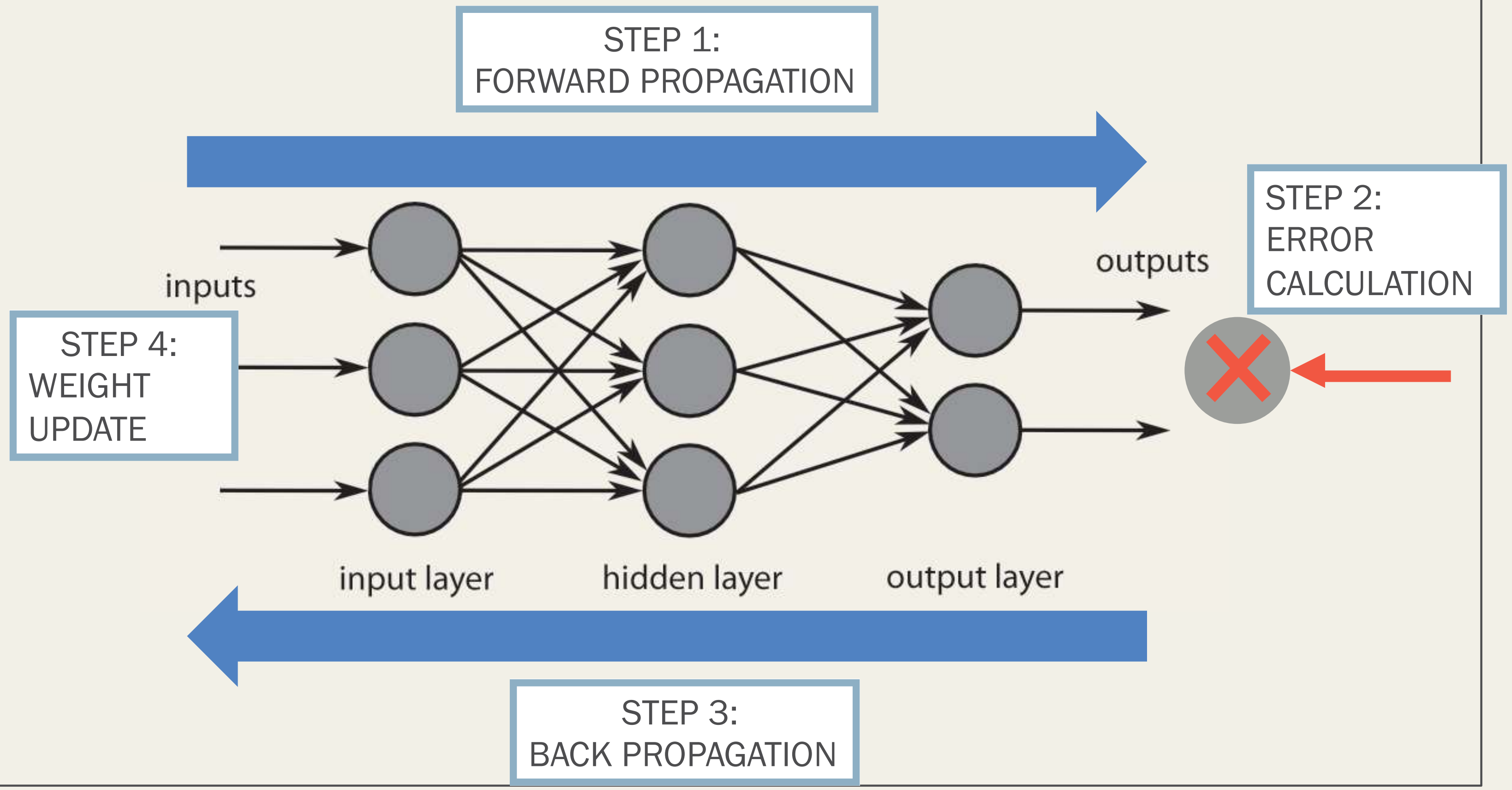
Often used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function.

Back Propagation – Phase 1 - Propagation

1. Propagation forward through the network to generate the output value(s)

2. Calculation of the cost (error term)

3. Propagation of output activations back through network using training pattern target in order to generate the deltas (difference between targeted and actual output values)

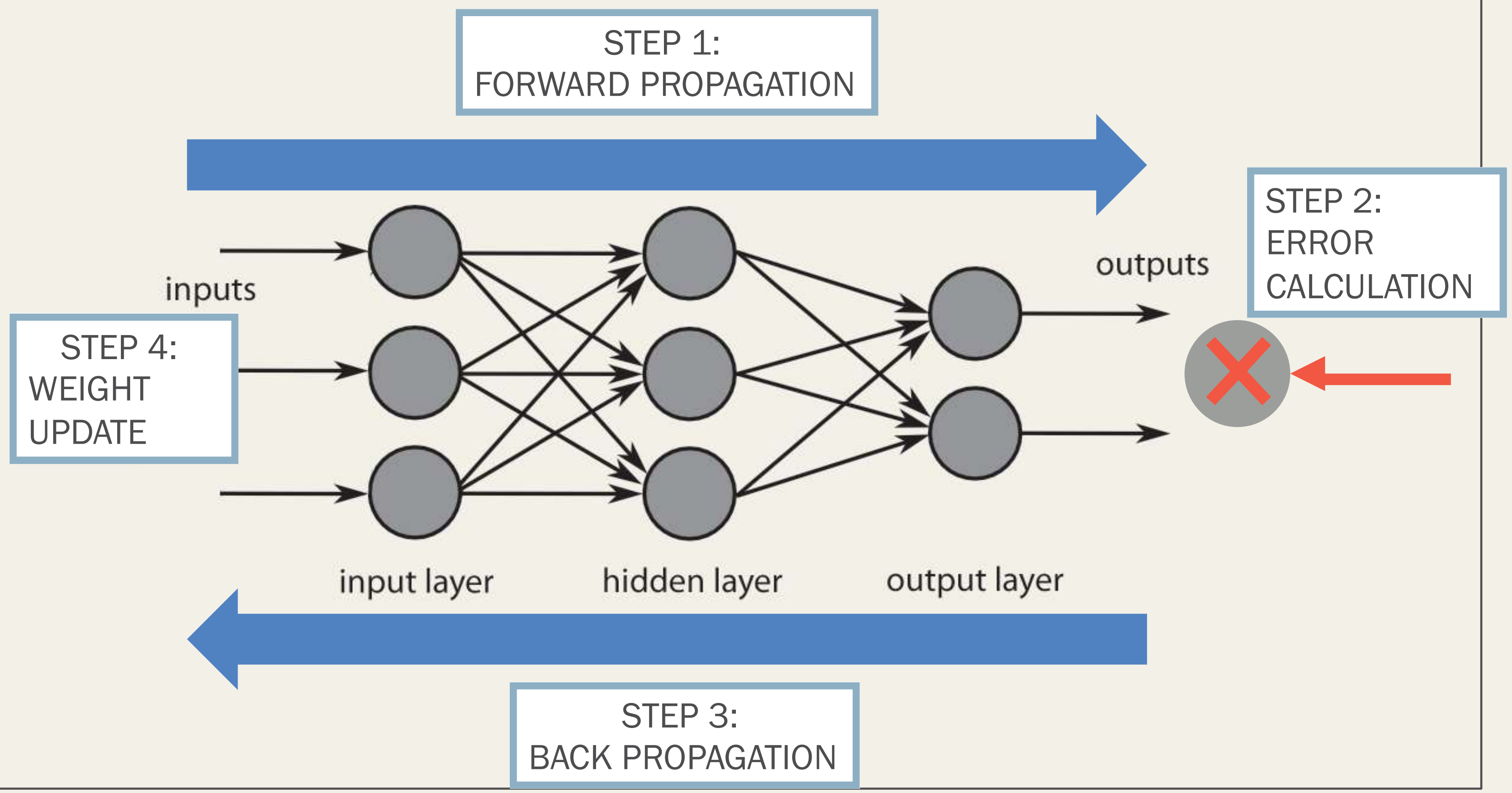


Back Propagation – Phase 2 – Weight Update

1. Calculate weight gradient

2. A ratio (percentage) of the weight's gradient is subtracted from the weight.

3. This ratio influences the speed and quality of learning and called learning rate. The greater the ratio, the faster neuron train, but lower ratio, more accurate the training is.



MULTI NEURON MODEL

2 Neurons

The network is represented by a matrix of weights, inputs and outputs.

Total Number of adjustable parameters = 8:

Weights = 6

Biases = 2

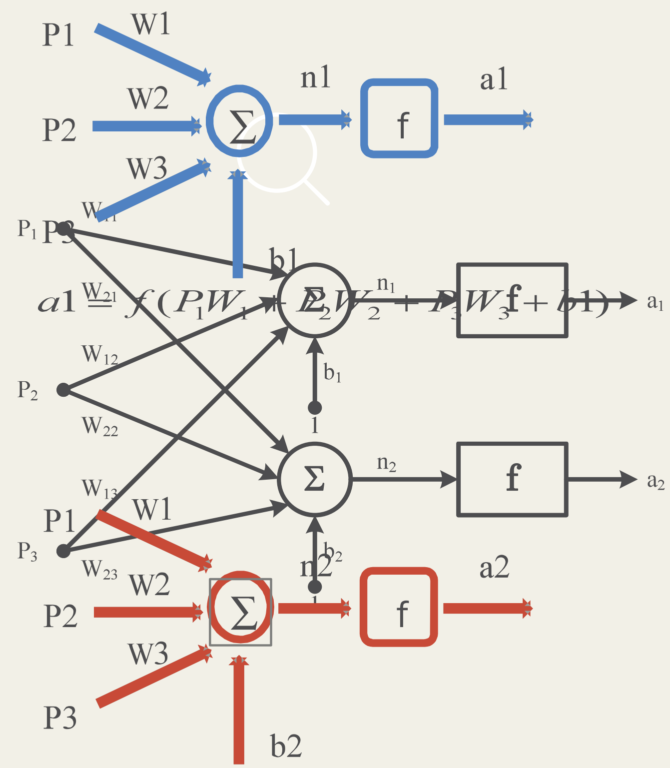
Matrix Representation

$$P = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

$$W = \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \end{bmatrix}$$

$$b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$a = f(W \times P + b)$$

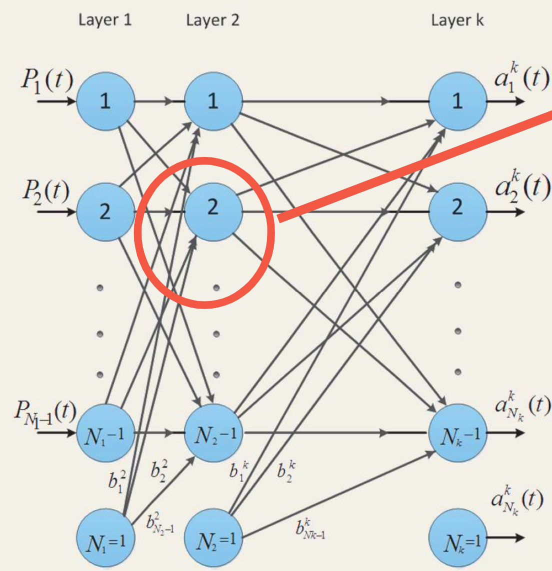


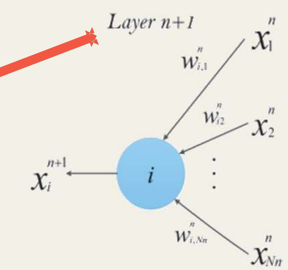
$$a2 = f(P_1W_1 + P_2W_2 + P_3W_3 + b2)$$

Multi Neuron Network - Matrices

$$P = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_{N_1} \end{bmatrix}$$

$$\begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1,N_1} \\ W_{21} & W_{22} & & W_{2,N_1} \\ \vdots & & \ddots & \vdots \\ W_{m-1,1} & W_{m-1,2} & \cdots & W_{m-1,N_1} \\ W_{m,1} & W_{m,2} & & W_{m,N_1} \end{bmatrix}$$





$$x_i^{n+1}(t) = \varphi\left(\sum_{j=1}^{N_n} w_{i,j}^n \, x_j^n(t)\right)$$

Node *(n+1, i)* representation

Non-Linear Sigmoid Activation function

$$\varphi(w) = \frac{1}{1+e^{-w}}$$

m: number of neurons in the hidden layer

N_1*: number of inputs*

Questions

?