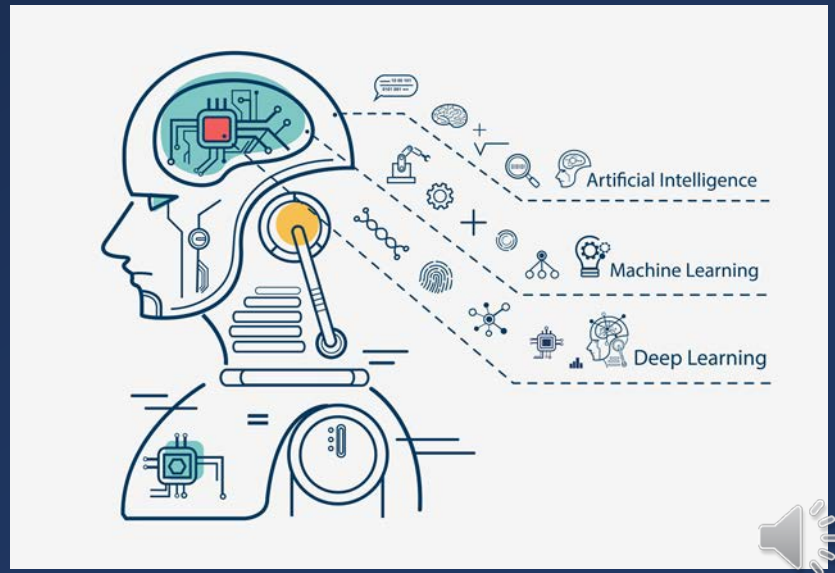
Big Data (MHI222956/MHI225101)

8.1 SVM & Neural Network

Support Vector Machine (SVM)



SVM (Support Vector Machine)

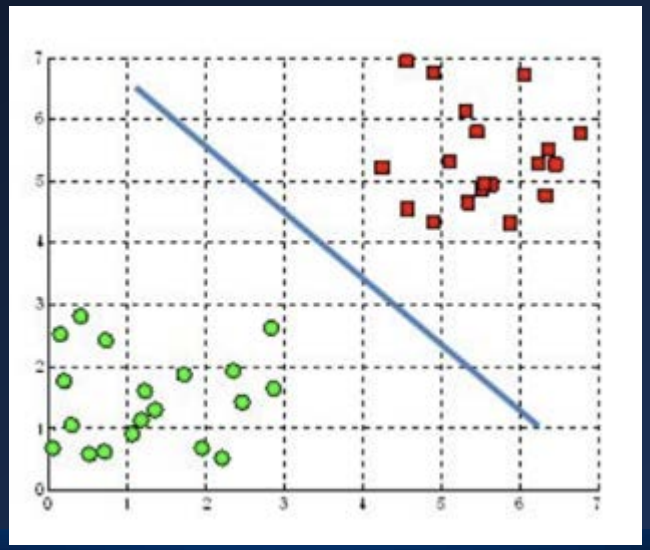
- A supervised machine learning algorithm which can be used for classification or regression challenges.

Plot each sample as a point in n-dimensional space

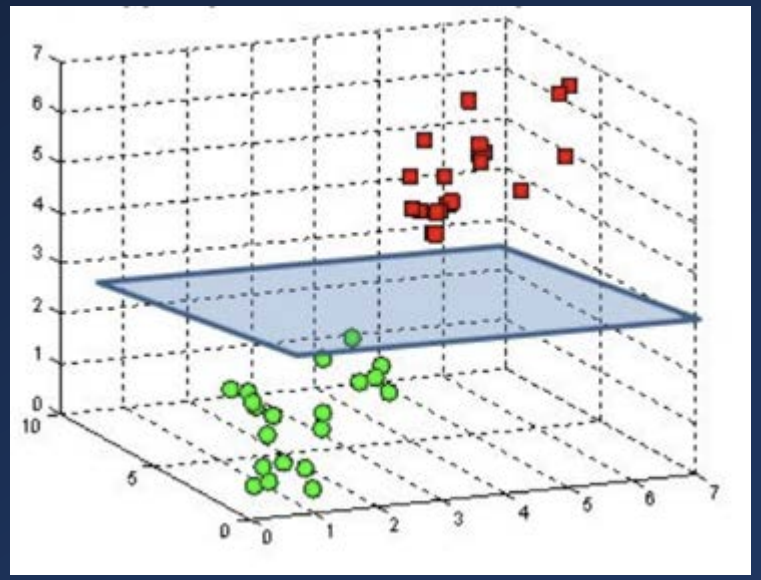
Find the hyperplane that different classes very well

A hyperplane is a generalization of a plane.

- In one dimension, an hyperplane is called a point
- In two dimensions, it is a line



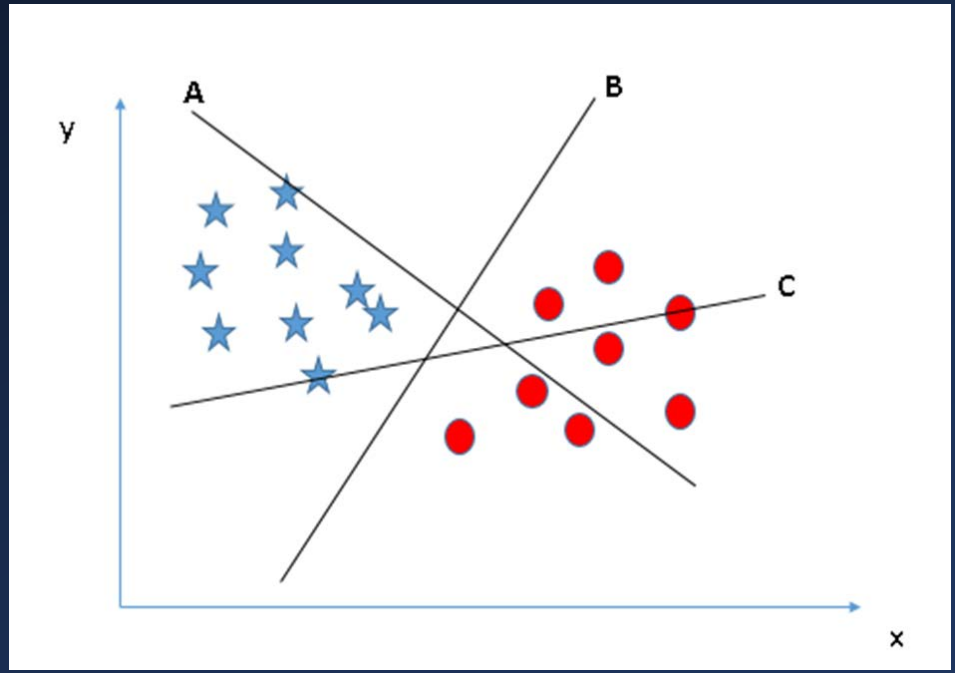
- In three dimensions, it is a plane



- In more dimensions you can call it a hyperplane

Start from 2D…

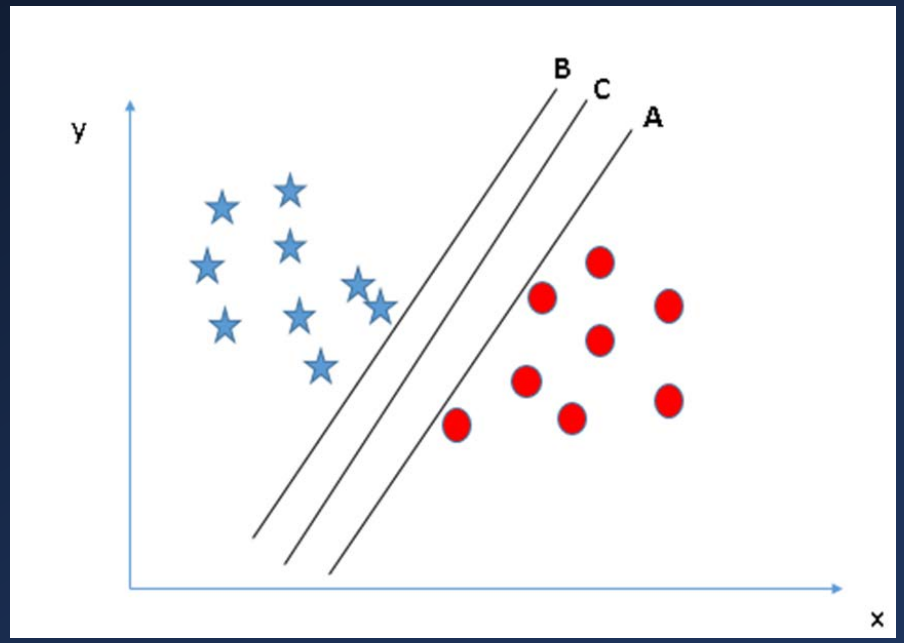
Which one is the right hyperplane to classify star and circle?



Answer: [B]

The second scenario

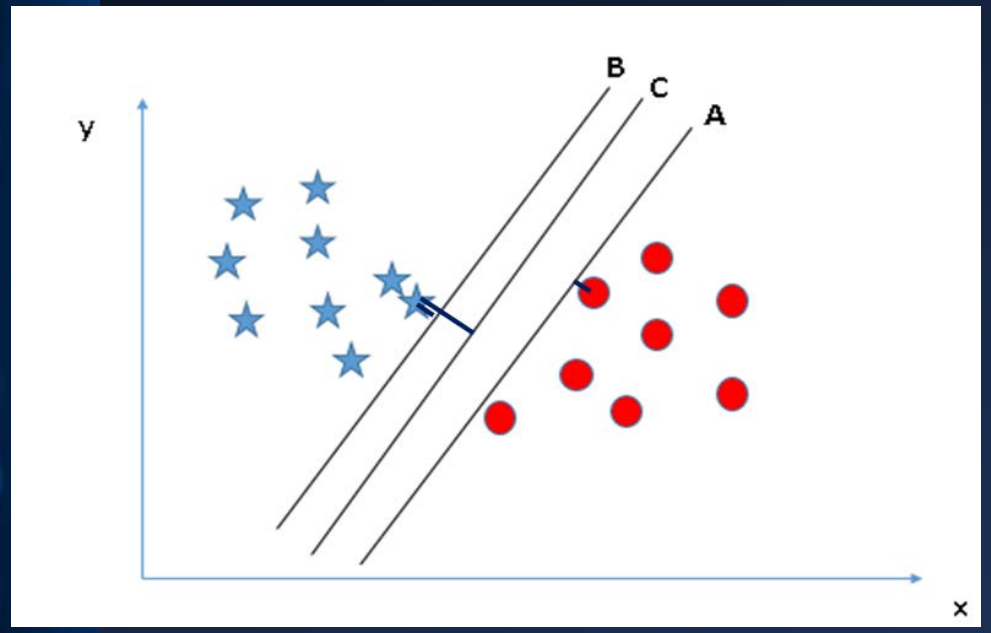
Then how about in the following scenario?



Valid vs. Optimal

Answer: [C]

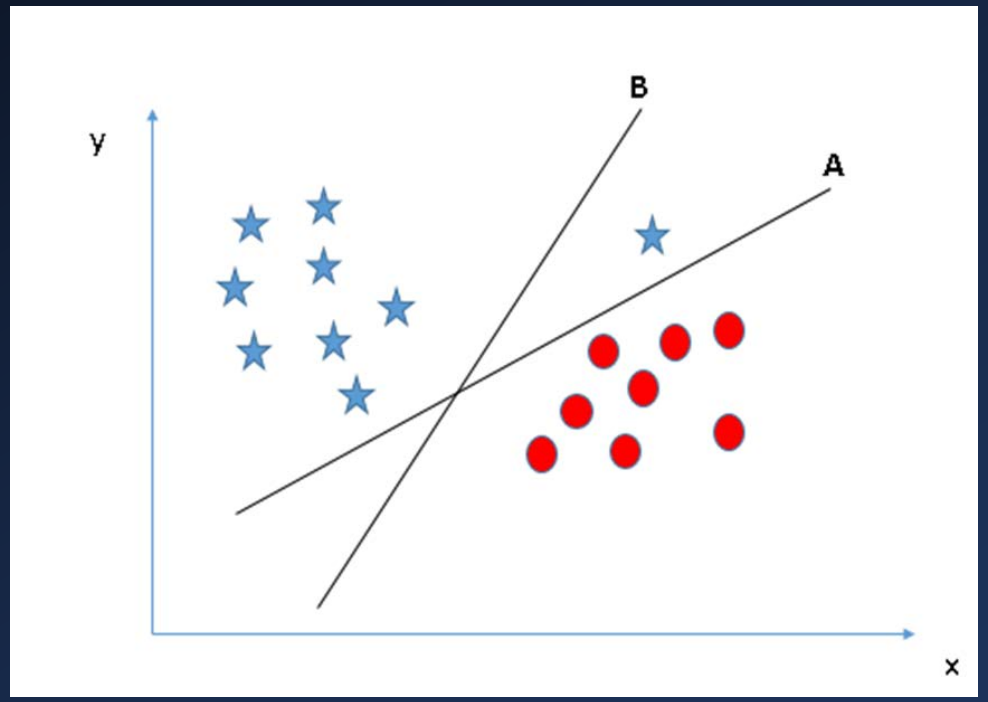
Margin



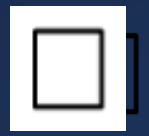
- Margin is the double of the distance between nearest data point (either class) and hyper plane

- The goal of a Support Vector Machine is to find the optimal separating hyperplane which **maximizes the margin** of the training data.

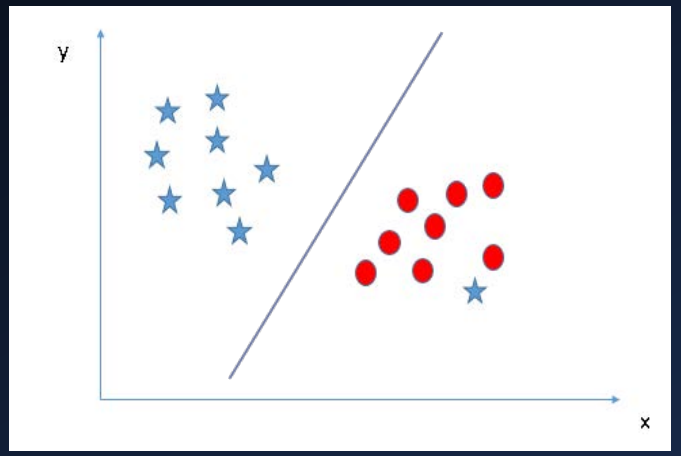
The third scenario



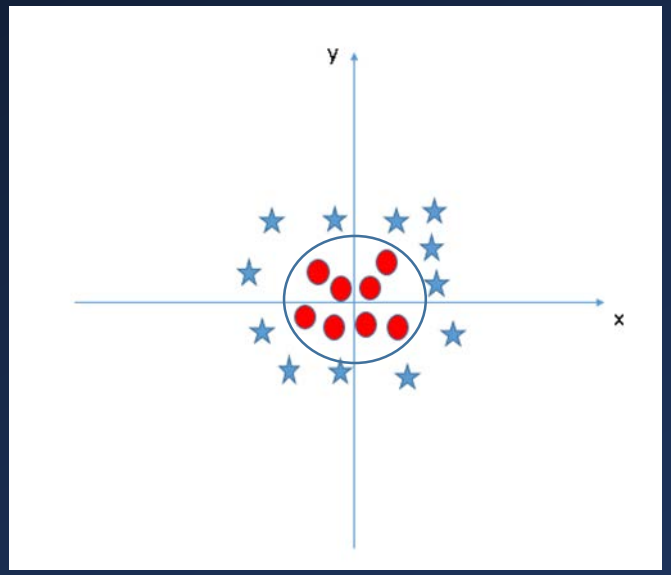
Margin vs. Accuracy

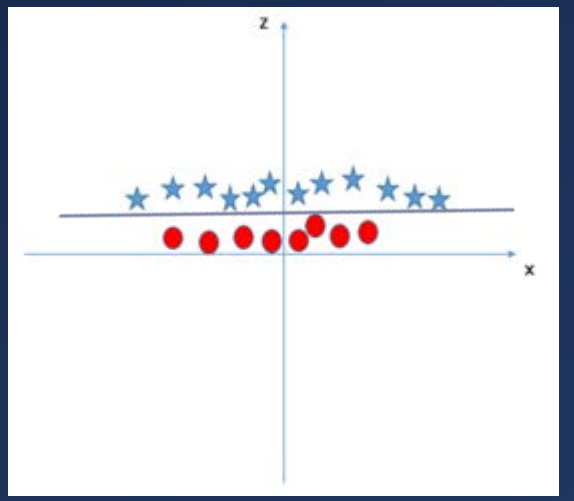


How about these scenarios?



Robust to outlier





Kernel for non-liner separation

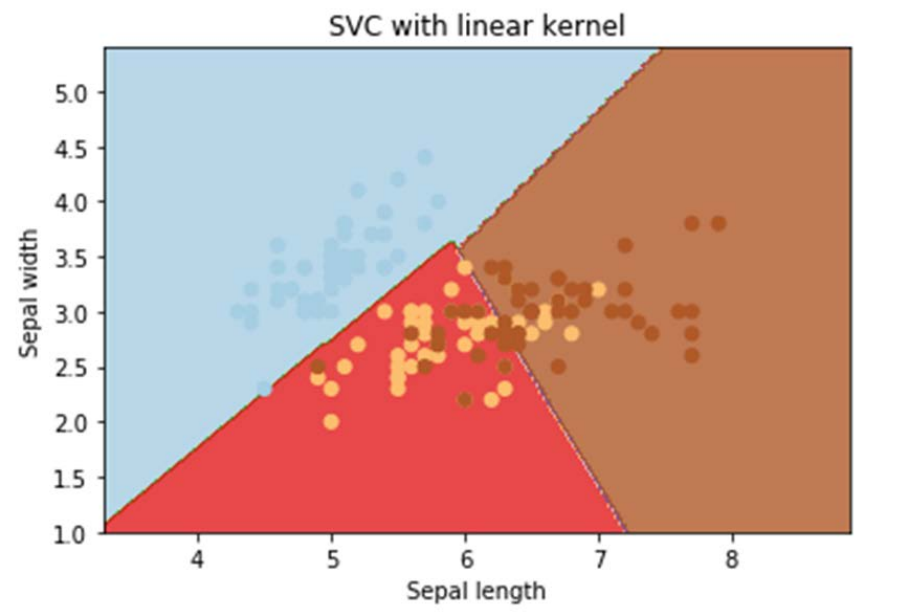
A SVM example

- Create a SVM model for the IRIS classification
  - Using two features only (sepal length and sepal width) to predict their class
  - In a SVM model, there are various hype-parameter available with kernel like, “linear”, “rbf”, ”poly”, “sigmoid”, etc.
  - Default kernel is “rbf”
  - “rbf”, “poly”, and “sigmoid” are useful for non-linear hyper-plane

A SVM example

- Using a linear SVM kernel

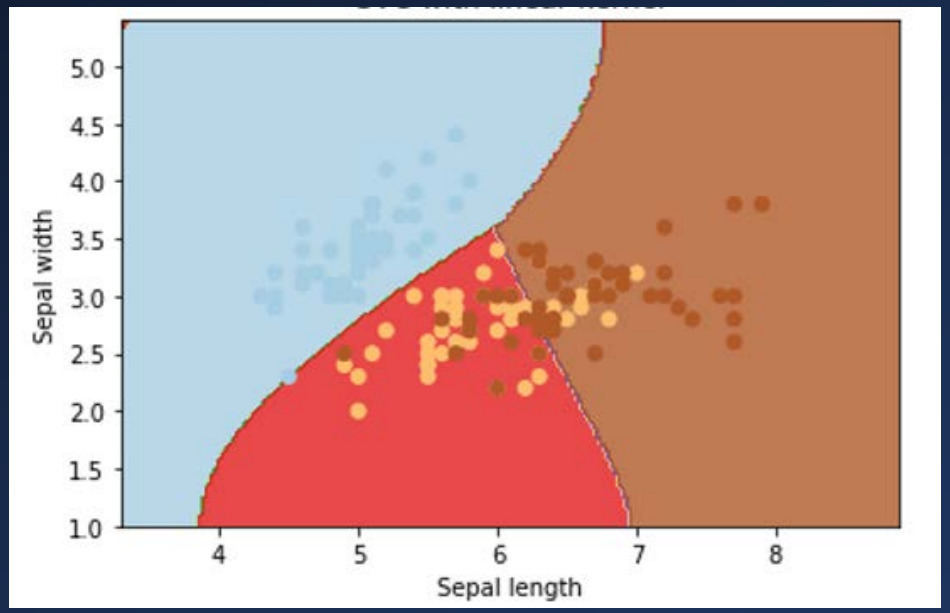
```
svc = svm.SVC(kernel='linear', C=1).fit(X, y)
```



A SVM example

- Using a rbf SVM kernel

```
svc = svm.SVC(kernel='rbf', C=1).fit(X, y)
```

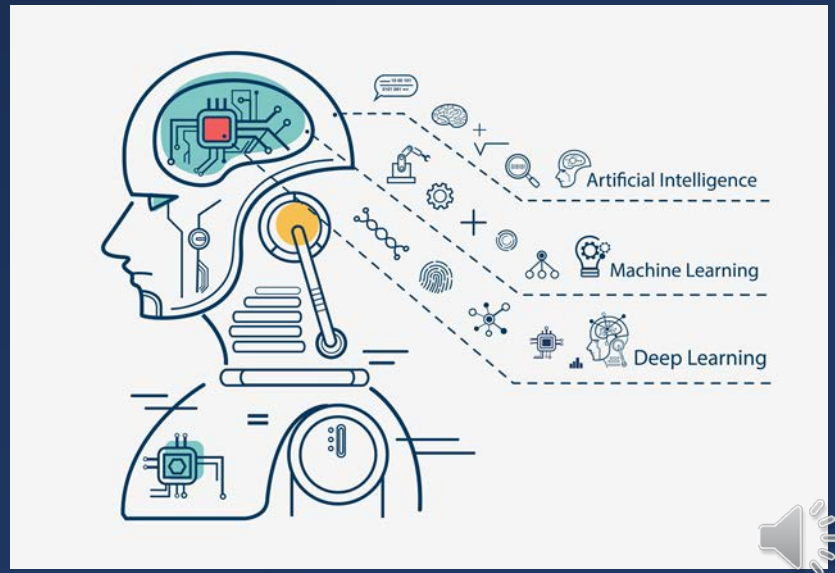


Pros and Cons

- It works really well with clear margin of separation
- It is effective in high dimensional spaces.
- It is effective in cases where number of dimensions is greater than the number of samples.
- It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

- It doesn’t perform well, when we have large data set because the required training time is higher
- It also doesn’t perform very well, when the data set has more noise i.e. target classes are overlapping
- SVM doesn’t directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

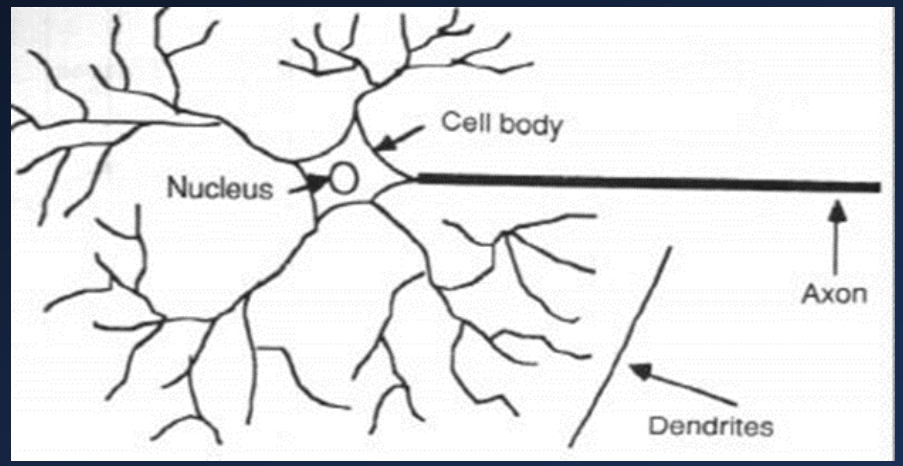
Neural Network (NN)

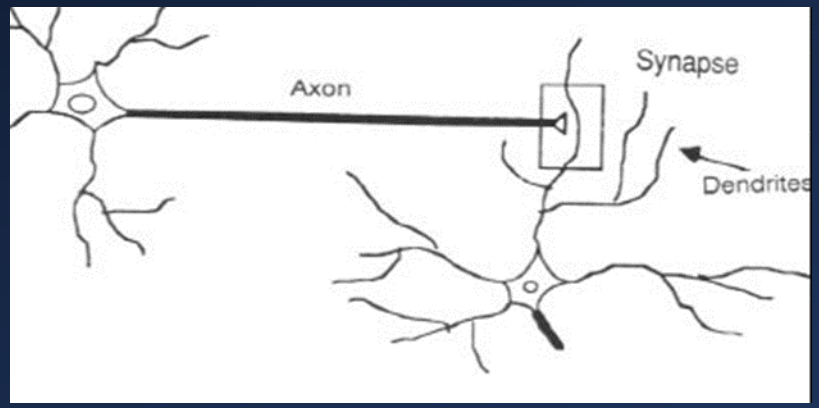


Human neurones

- Effortless for human to recognize those digitals
- Write a computer program to recognize those digits???
- Neural Networks – learn from training examples

- Neuron collects signal through dendrites
- Neuron sends out electrical activity through Axon

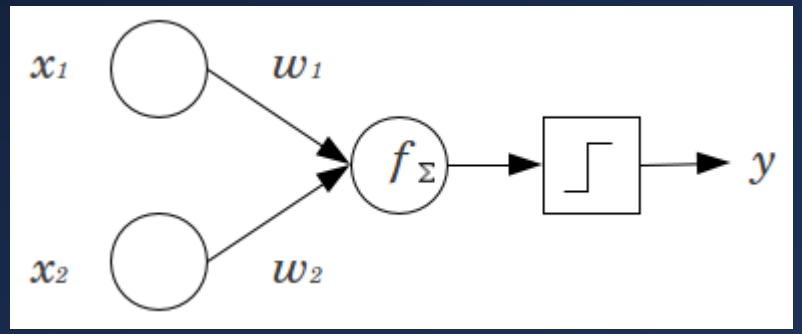




- Synapse convert the activity from axon into electrical effects that inhibit or excite activity in the connected neurons

Perceptrons

- Single-layer NN
- Takes several binary inputs
- Produces a single binary output



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

- Perceptron's bias

$$b \equiv -\text{threshold}$$

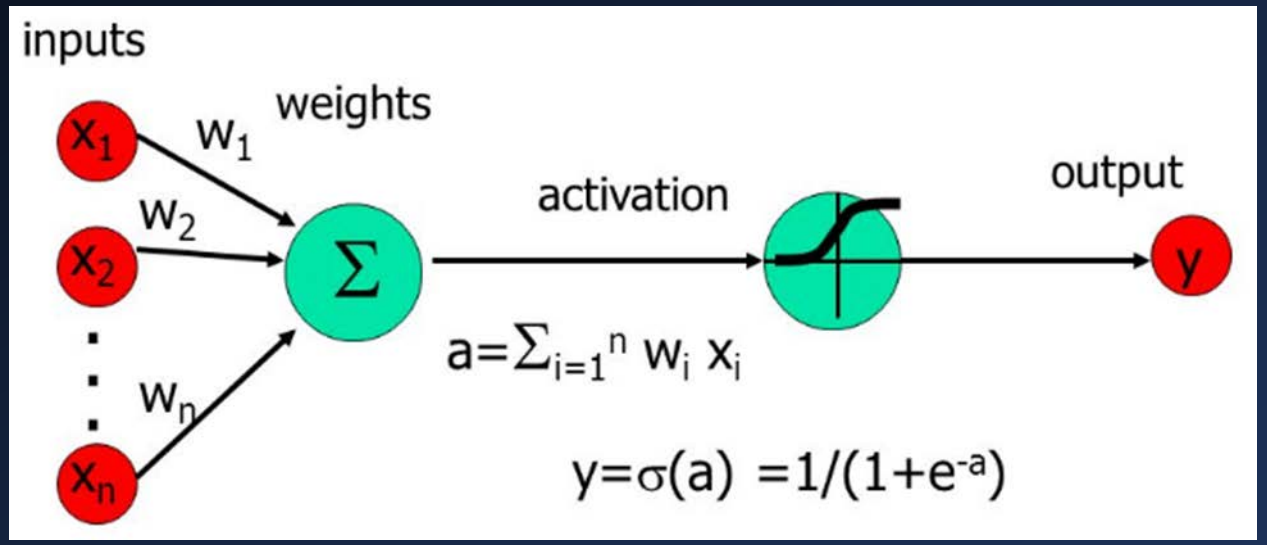
- A perceptron can weigh up different kinds of evidence in order to make decisions

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

- The bias is *a measure of how easy it is to get the perceptron to fire* (output =1)

- A small change in the weights or bias of any single perceptron in a network can sometimes cause the output of that perceptron to completely flip, say from 0 to 1.

Sigmoid neurons



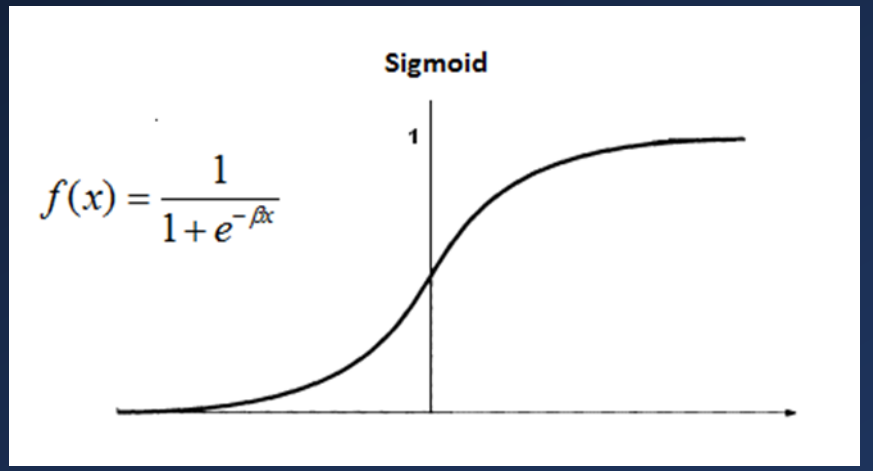
- Input can be any value between 0 and 1
- Has weights and overall bias
- Output is: $\sigma(w \cdot x + b)$

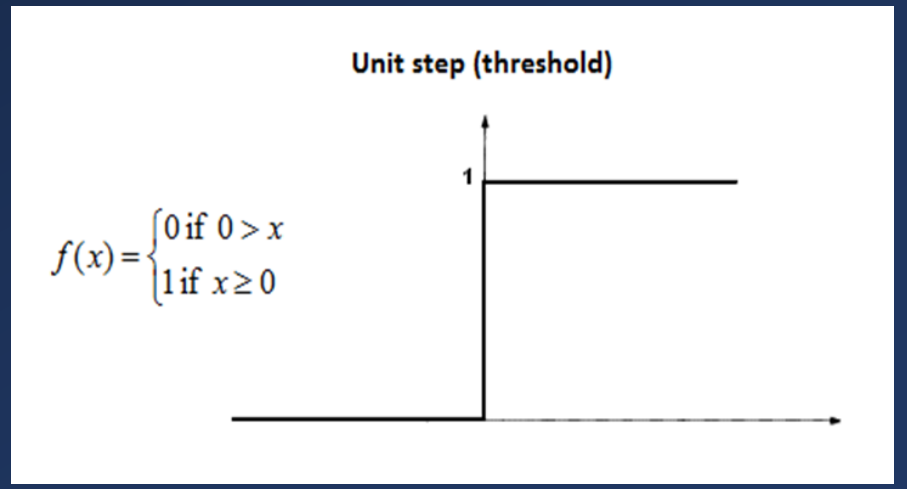
- σ is called the sigmoid function:

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}.$$

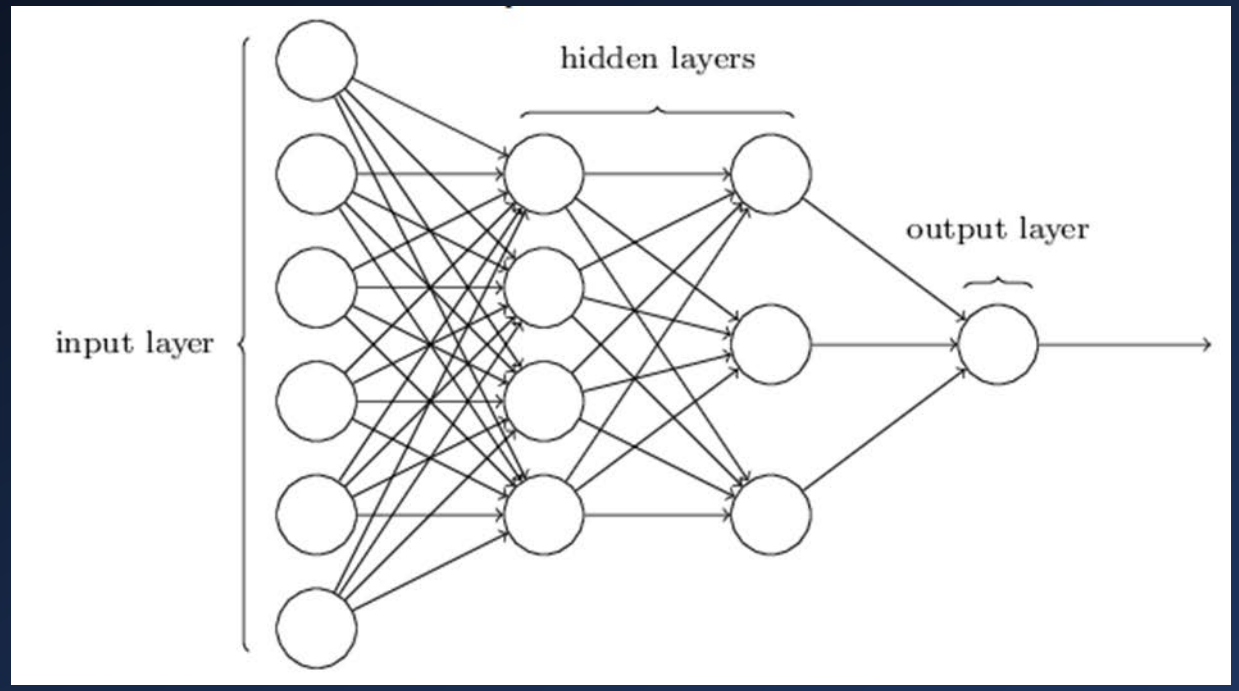
- So, the output of a sigmoid neuron is:

$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}$$

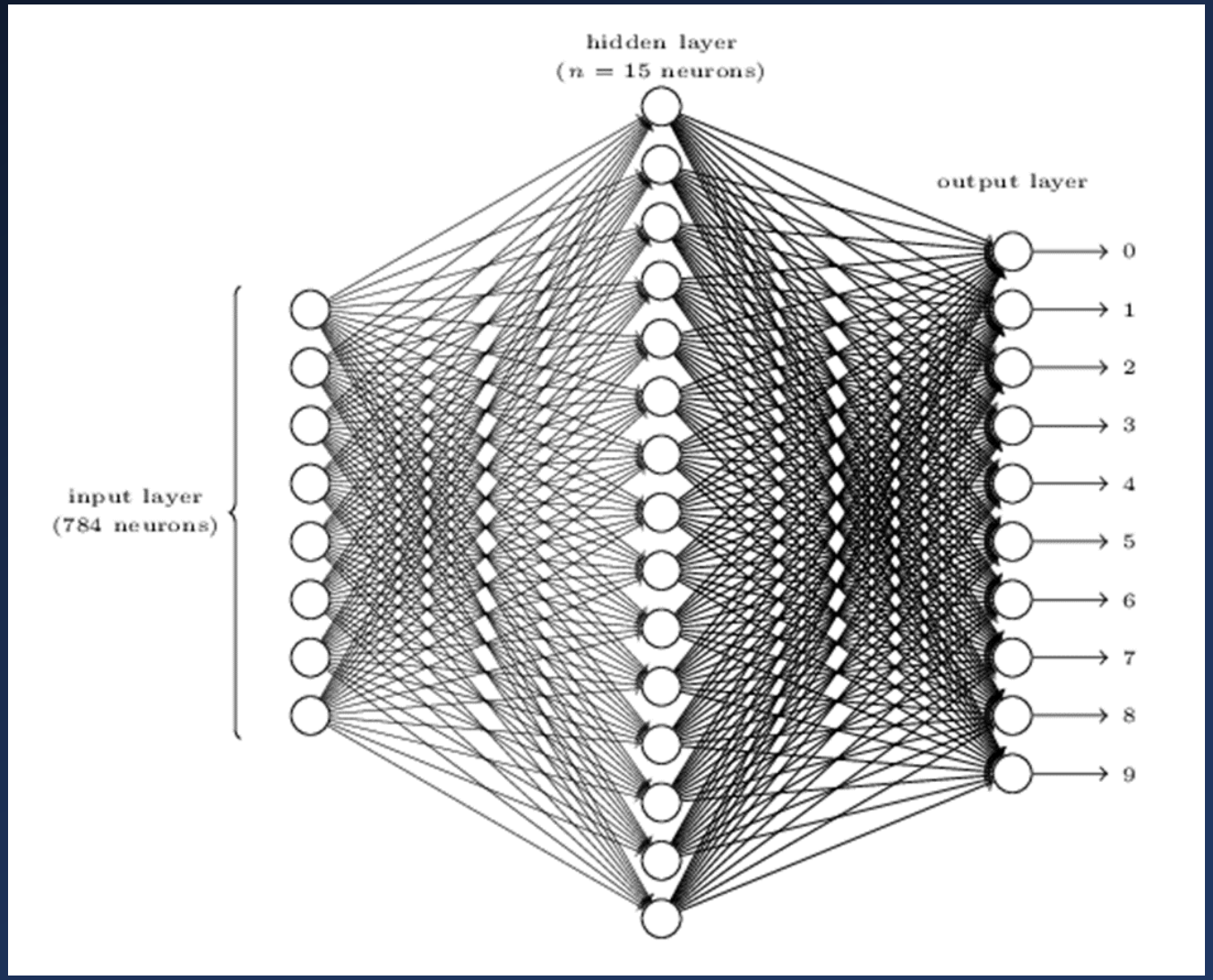




The architecture of Neural Networks



- The design of the input and output layers of a neural network is often straightforward.
- However, there can be quite an art to the design of the hidden layers.


hidden layer
(n = 15 neurons)
output layer
input layer
(784 neurons)
0
1
2
3
4
5
6
7
8
9

Feedforward NN

- There are no loops in the network: information is always fed forward, never fed back.

Feedback NN

- Also called *Recurrent neural network*
- Feedback loops are possible
- Less influential

Learning with gradient descent

- **Quadratic Cost function** (MSE - Mean Squared Error)

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

Where *w* denotes the collection of all weights in the network, *b* all the biases, *n* is the total number of training inputs, *a* is the vector of outputs from the network when *x* is input, and the sum is over all training inputs, *x*.

- The aim of training algorithm is to minimize the cost *C(w,b)* as a function of the weights and biases.
- Use a smooth cost function like the quadratic cost it turns out to be easy to figure out how to make small changes in the weights and biases so as to get an improvement in the cost.

- To use gradient descent to find the weights w_k and biases b_l which minimize the Quadratic cost, we have:

$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l}$$

- Then, the question is:

How to compute the gradient of the cost function?

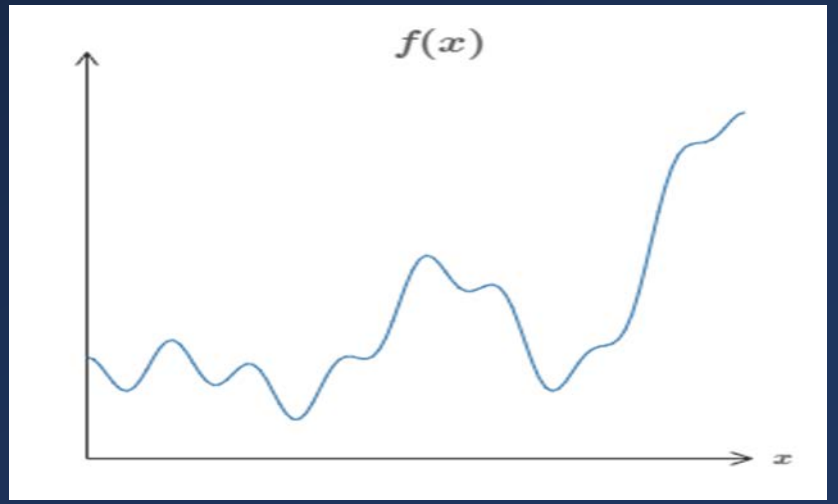
Backpropagation

- The workhorse of learning in neural networks
- Gives us detailed insights into how changing the weights and biases changes the overall behaviour of the network.

The Universal Approximation Theorem

- The standard multilayer feed-forward network with a single hidden layer, which contains finite number of hidden neurons, is a universal approximator among continuous functions on compact subsets of R_n, under mild assumptions on the activation function.

- A neural network can compute any function!



Pros and Cons

- Can be trained directly on data with hundreds or thousands input variables
- Once trained, predictions are fast
- Great for complex/abstract problems like image recognition
- Can significantly out-perform other models when the conditions are right (lots of high quality labelled data)

- Black box that not much can be gleaned from
- Training is computationally expensive
- Can suffer from “interference” in that new data cause it to forget some of what it learned on old data
- Often abused in cases where simpler solutions like linear regression would be best

Summary

- Support Vector Machine
- Neural Network