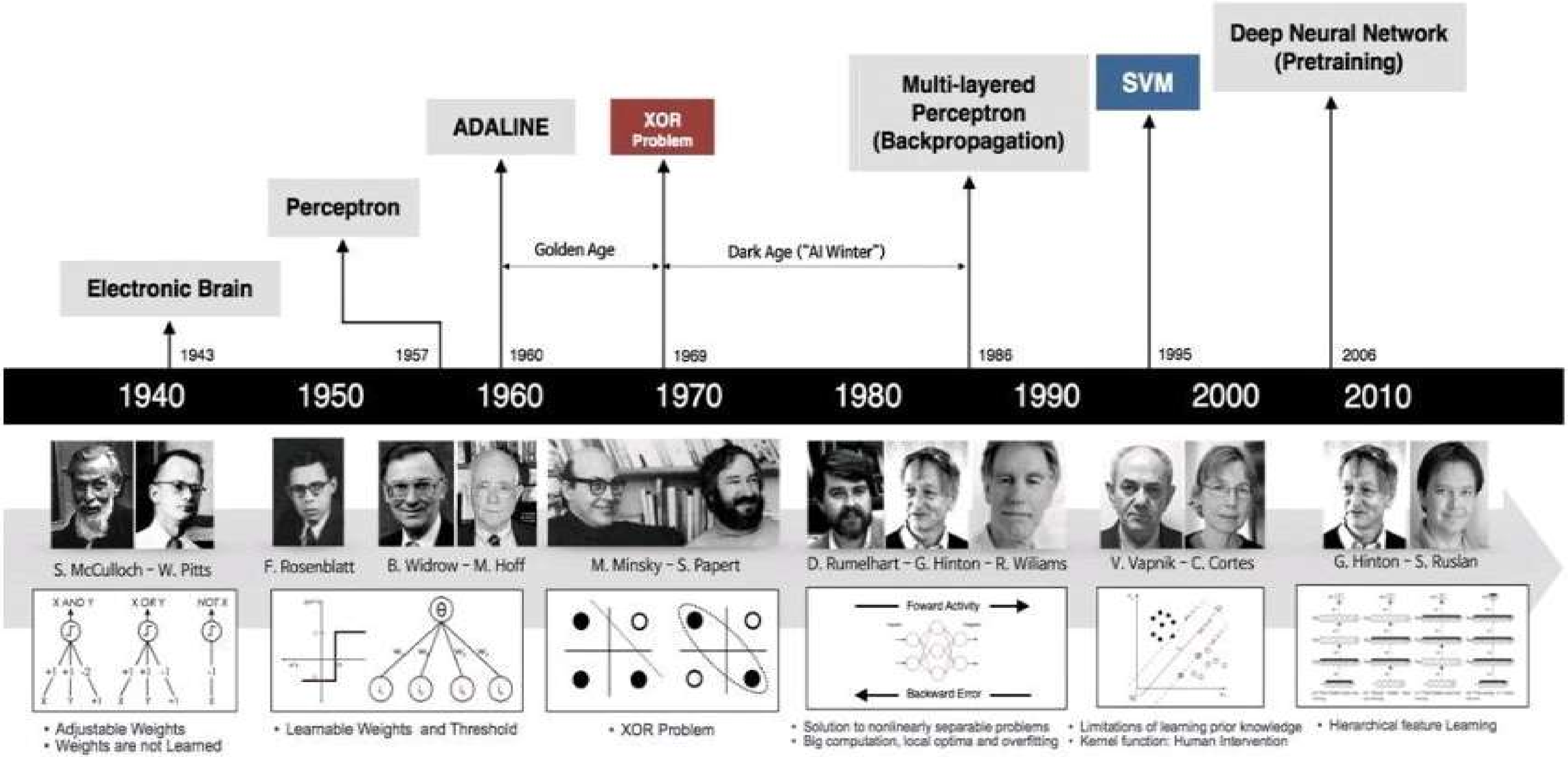
Artificial Neural Network (ANN) : Perceptron

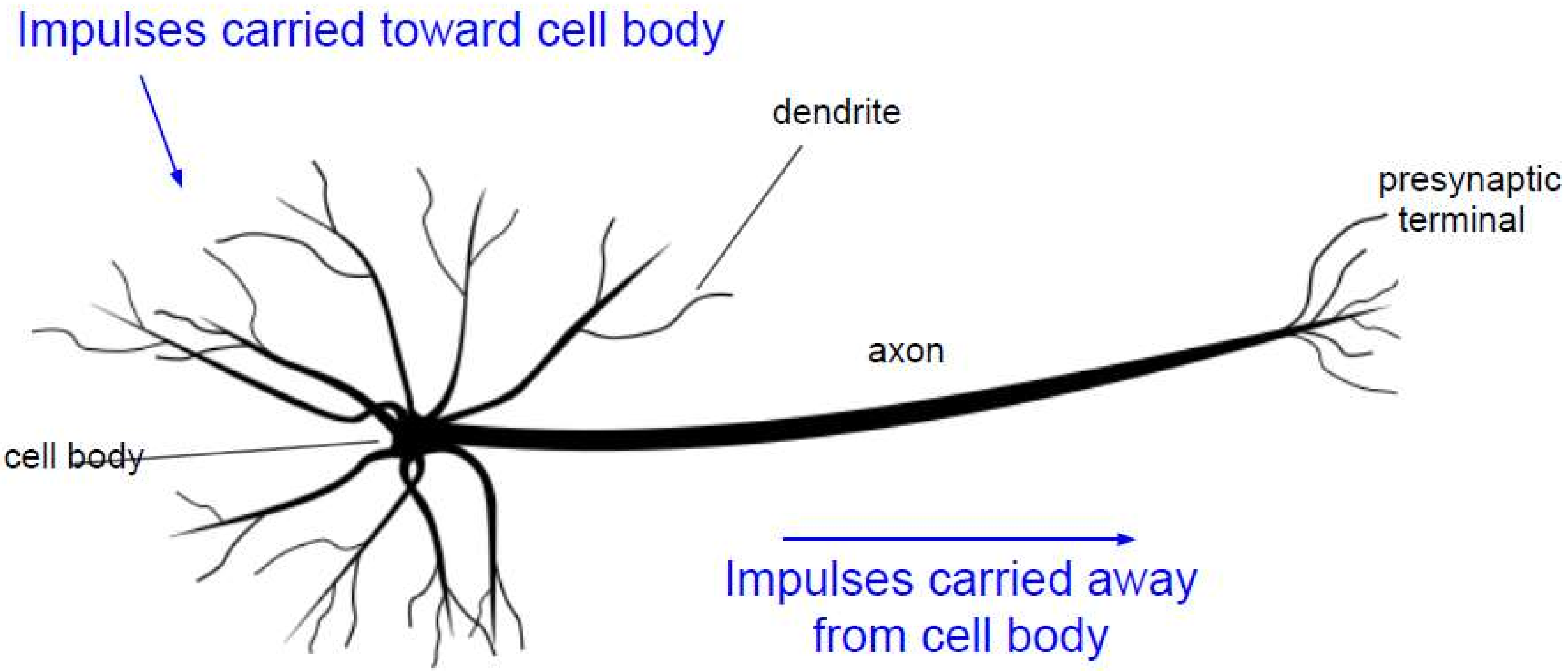
Sourav Karmakar

souravkarmakar29@gmail.com

A BRIEF HISTORY

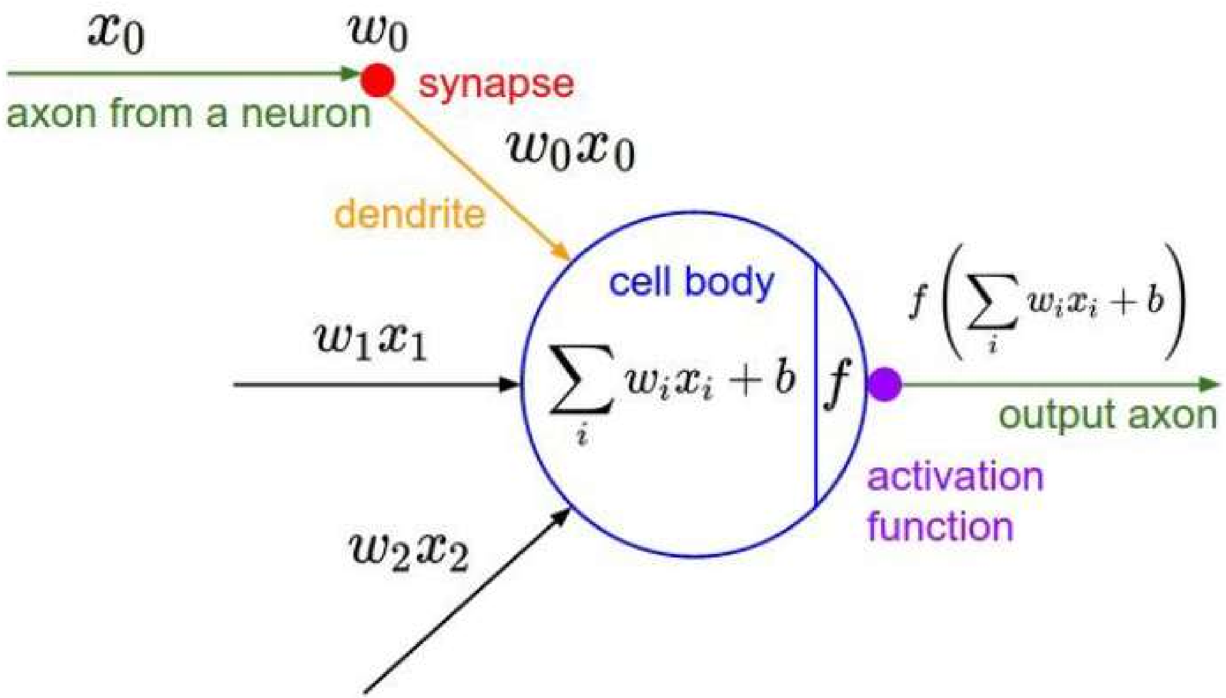


BIOLOGICAL NEURON



Biological Neuron

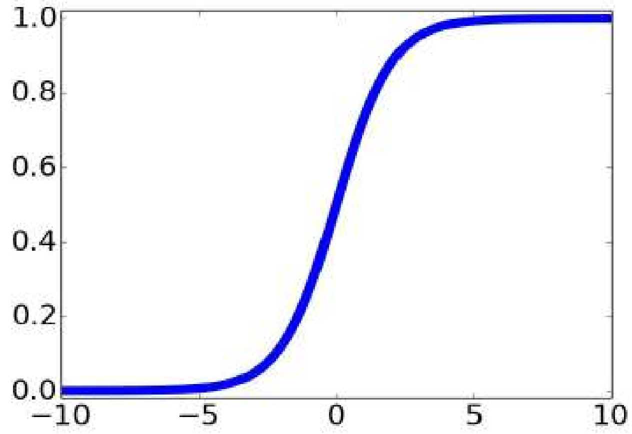
ARTIFICIAL NEURON



Artificial Neuron

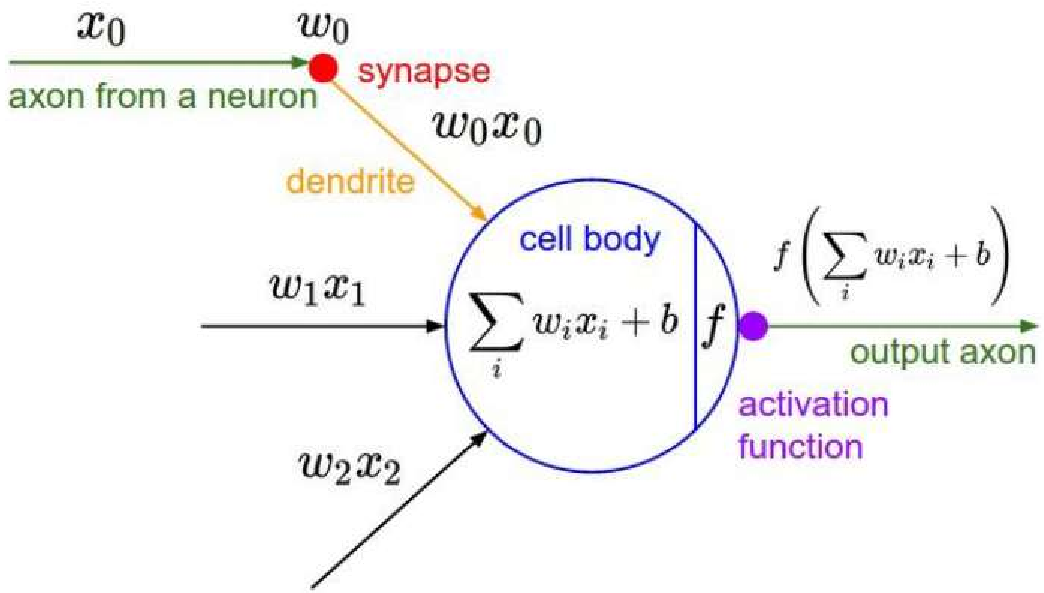
$f(.)$ is called the activation function

For example the activation function could be sigmoid activation function



- b is called the **bias**
- w_i are called the **synaptic weights** or simply **weights**
- This is a very crude approximation of biological neuron. Actual biological neuron are even more complex.

PERCEPTRON



- This model of Artificial Neuron is also called **McCulloch – Pitts Model.** [1943]

- **Rosenblatt** [1957] extended this idea and gave a learning rule to update the weight parameters from given dataset. This is called **Perceptron.**

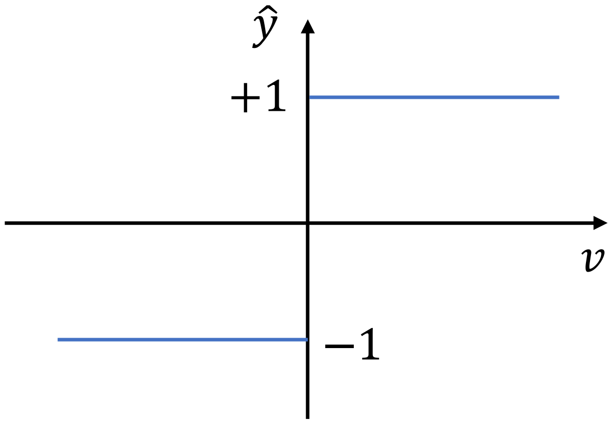
- **Activation Function (f):**

  In this case the activation function is signum function (sgn)

  Let, $v = \sum_i w_i x_i + b$, then output $\hat{y}$ is calculated as:

$$\hat{y} = \begin{cases} +1, & if \ \ v \geq 0 \\ -1, & if \ \ v < 0 \end{cases}$$

This is also called hard limiter



PERCEPTRON LEARNING RULE

- **Assumptions:**
  - Binary classification problem. The dataset is X. There are two classes denoted by C_1 and C_2 respectively.
  - Actual class label of datapoint $\vec{x}$ is denoted by y. It is denoted as following:

$$y = \begin{cases} +1, & if\ \vec{x} \in C_1 \\ -1, & if\ \vec{x} \in C_2 \end{cases}$$

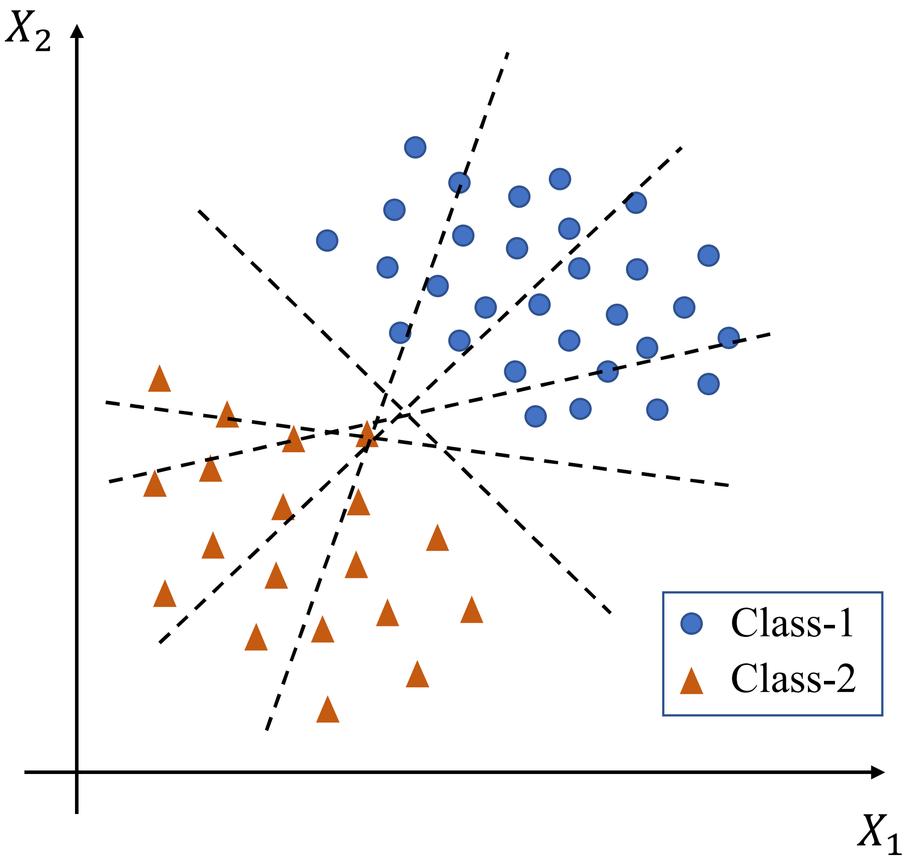
  - The weights are denoted by vector $\vec{w}$, and the bias by scaler b. If we consider another input (x_0) of value $+\,1$ then bias can be represented by weight w_0. In that case: $\sum_i w_i x_i + b = \vec{w}^T \vec{x}$

- **Algorithm:**
  - **Initialize:** Initialize the weight vector $\vec{w}(0)$ with some small random numbers. Iteration count $t \leftarrow 1$
  - ***Repeat***
  - ***for*** $\vec{x} \in X$, Compute $\hat{y} = \mathrm{sgn}(\vec{w}^T \vec{x})$, $\hat{y}$ is the computed output / label.
  - $\vec{w}(t+1) = \vec{w}(t) + \eta\ (y\ - \hat{y})\ \vec{x}$, y is the actual output / label. (from labelled training data)
  - ***Until*** $\left\| \vec{w}(t+1)\ -\ \vec{w}(t) \right\| < \epsilon$, ϵ is the user defined tolerance [check of convergence]
  - If not converged then $t\ \leftarrow t + 1$ and repeat the whole process.

$\boxed{\eta \text{ is called the learning rate.}}$

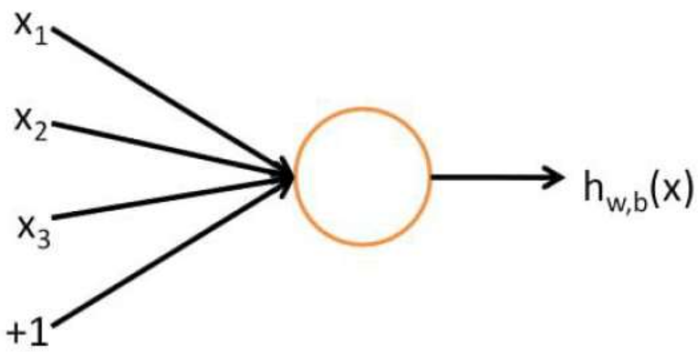
PERCEPTRON DECISION BOUNDARY

- Perceptron decision boundary is the hyperplane defined by $\vec{w}^T\vec{x} = 0 \; or \; \sum_i w_i x_i + b = 0$



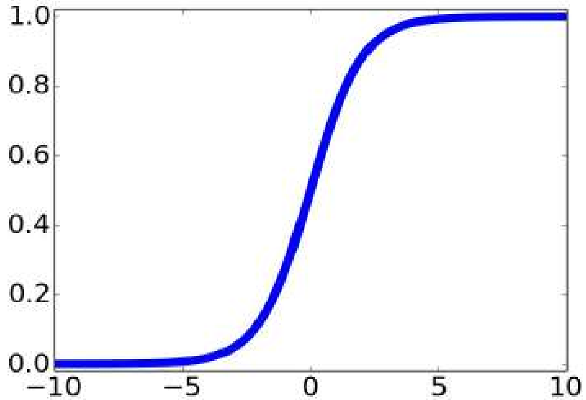
- Consider a binary classification problem. Dataset is shown in the figure beside.
- Random initialization of weight vector at the starting of the learning algorithm randomly fits a hyperplane (straight line in 2D) as shown beside.
- As we keep updating weights using the learning rule, the decision boundary keeps on changing.
- Till all the training datapoints fall correctly in the either side of the decision boundary.
- Thus perceptron acts as a linear classifier and can classify linearly separable data with high accuracy.

PERCEPTRON WITH SIGMOID ACTIVATION



This "neuron" is a computational unit that takes as input x_1, x_2, x_3 (and a $+1$ intercept term), and outputs $h_{W,b}(x) = f(\sum_{i=1}^{3} W_i x_i + b) = f(W^T x + b)$, where $f : \Re \mapsto \Re$ is called the activation function. Here we will choose $f(\cdot)$ to be the sigmoid function:

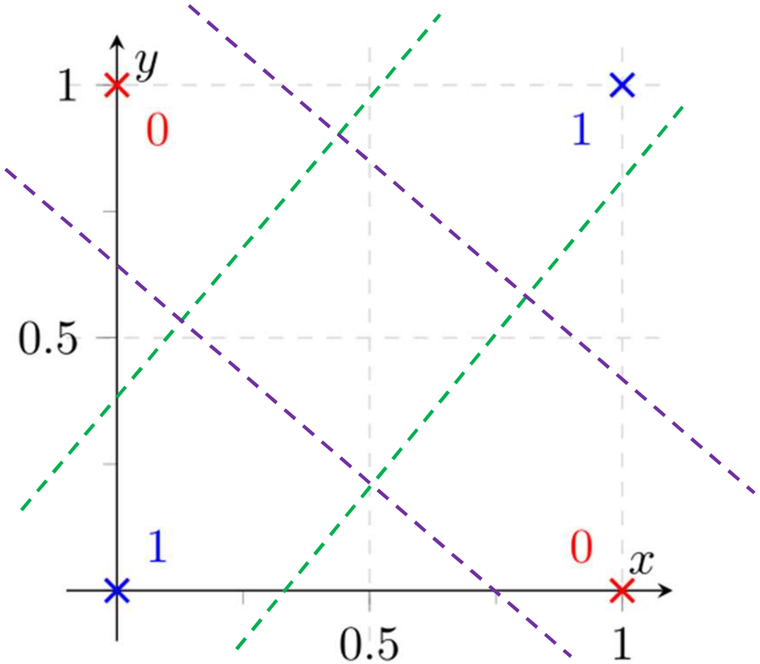
$$f(z) = \frac{1}{1 + e^{-z}}$$



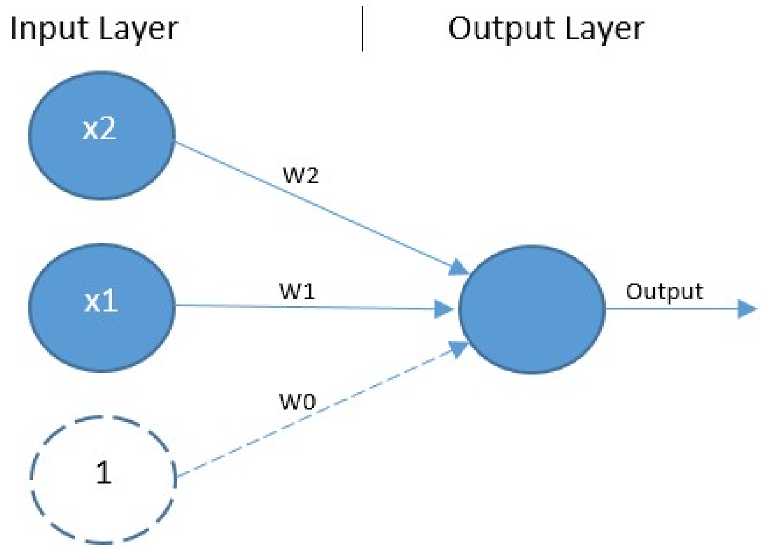
Thus, our single neuron corresponds exactly to the input-output mapping defined by logistic regression.

LIMITATIONS OF PERCEPTRON

XOR Problem:



Input 1	Input 2	Output
0	0	0
0	1	1
1	1	0
1	0	1



- Though perceptron works well for linearly separable dataset.
- It failed to solve XOR problem and other non-linearly separable datasets. Because there is no straight line that can form the decision boundary between the two classes.

Thank You