INTRODUCTION TO NEURAL NETWORKS

Lecture 4. Perceptron

Dr. Jingxin Liu

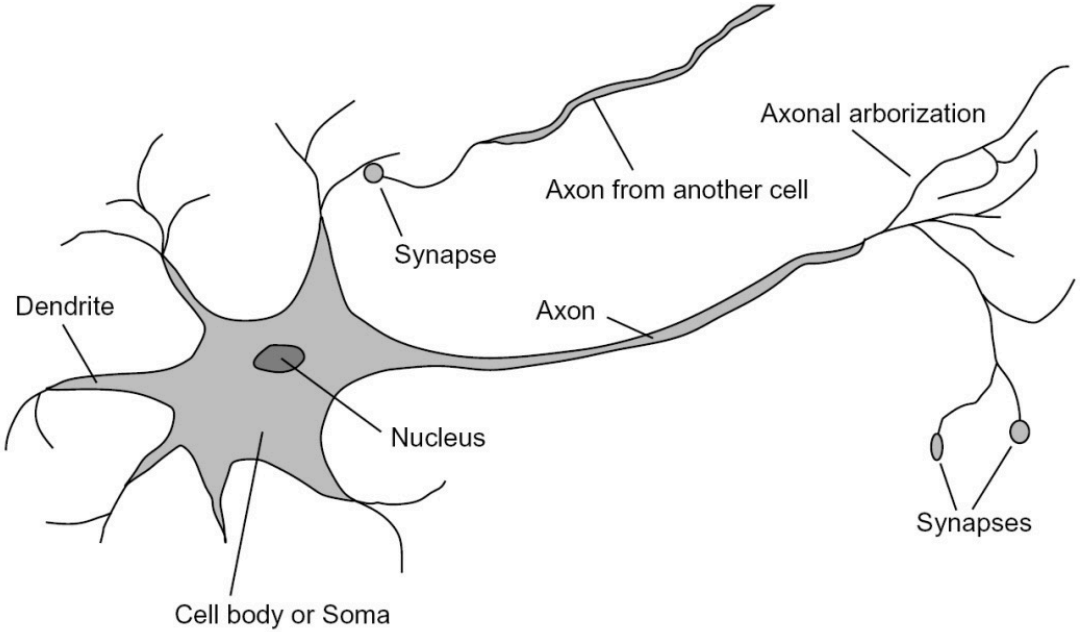
School of AI and Advanced Computing

Xi'an Jiaotong-Liverpool University
西交利物浦大学

03.03.2022

Table of Contents

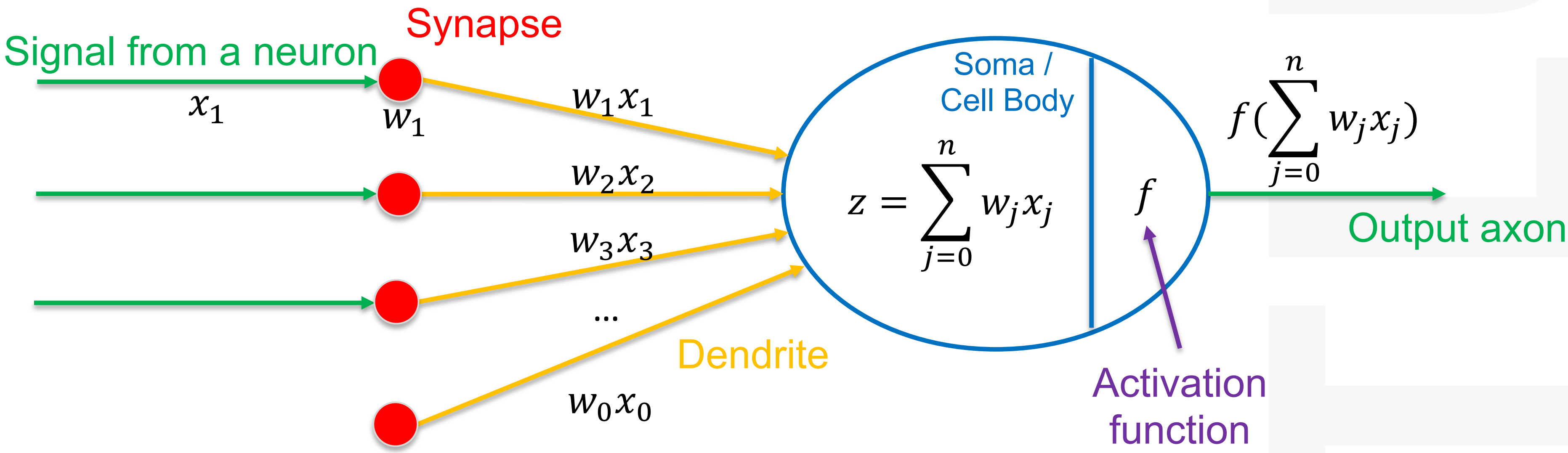
Perceptron – Biological Neuron



- The dendrite receives electrical signals from the axons of other neurons;
- At the synapses between the dendrite and axons, electrical signals are modulated in various amounts;
- An actual neuron fires an output signal through axon only when the total strength of the input signals exceed a certain threshold.

https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Neuron/index.html

Perceptron – Artificial Neuron

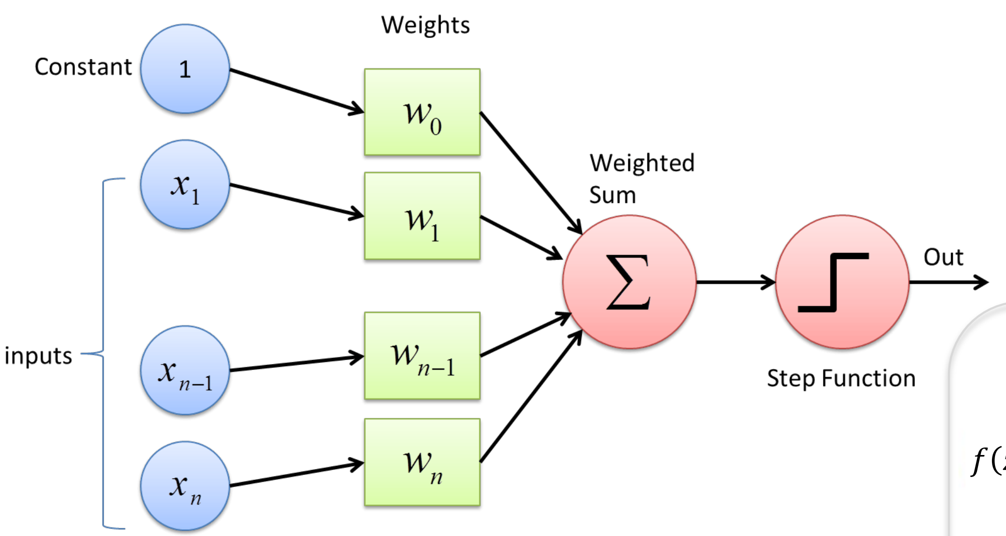


Biological Neuron	Artificial Neuron
Cell Body (Soma)	Node
Dendrites	Input
Synapse	Weights or interconnections
Axon	Output

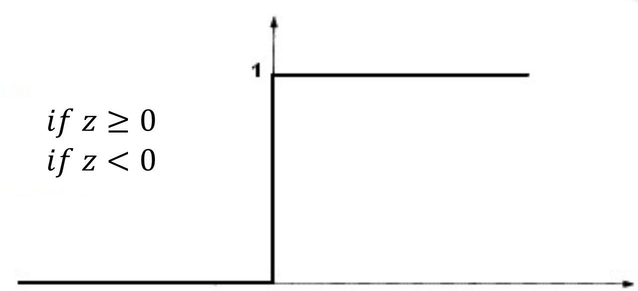
Perceptron

The perceptron is a mathematical model of a biological neuron.

A Perceptron accepts inputs, moderates them with certain weight values, then applies the activation function (**Step function**) to output the final result.



$$f(z) = \begin{cases} 1 & if\ z \geq 0 \\ 0 & if\ z < 0 \end{cases}$$



Step Function /
Heaviside Step Function

Perceptron

Mathematically,

j’th weight bias

$$\hat{y} = f\left(\sum_{j=1}^{n} w_j x_j + w_0\right) = f\left(\sum_{j=0}^{n} w_j x_j\right)$$

output i’th input

activation function

where,

$$f(z) = \begin{cases} 1 & if\ z \geq 0 \\ 0 & if\ z < 0 \end{cases}$$

The perceptron is an algorithm for *supervised learning* of binary linear classifiers.

Perceptron – Learning Rule

The idea:

For x^i in all training examples:

- If $y = 1$ and $f\left(x^i\right) = 1$ or
- If $y = 0$ and $f\left(x^i\right) = 0$

t: label

no need to change anything.

- If $y = 1$ and $f\left(x^i\right) = 0$ or — need to make $f\left(x^i\right)$ larger
- If $y = 0$ and $f\left(x^i\right) = 1$ — need to make $f\left(x^i\right)$ smaller

we need update w, based on

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \triangle\, \boldsymbol{w} \quad \text{, where} \quad \triangle\, \boldsymbol{w} = \overset{\text{learning rate}}{\alpha}\underbrace{\left(y^i - \hat{y}^i\right)}_{\begin{cases} 1 & \text{for positive example} \\ -1 & \text{for negative example} \end{cases}} x^i$$

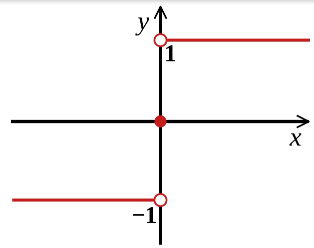
Perceptron – Learning Rule

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta\, \boldsymbol{w}$$

$$\Delta\, \boldsymbol{w} = \alpha\left(y^i - \hat{y}^i\right)x^i$$

$$\begin{cases} 1 & \text{for positive example} \\ -1 & \text{for negative example} \end{cases}$$

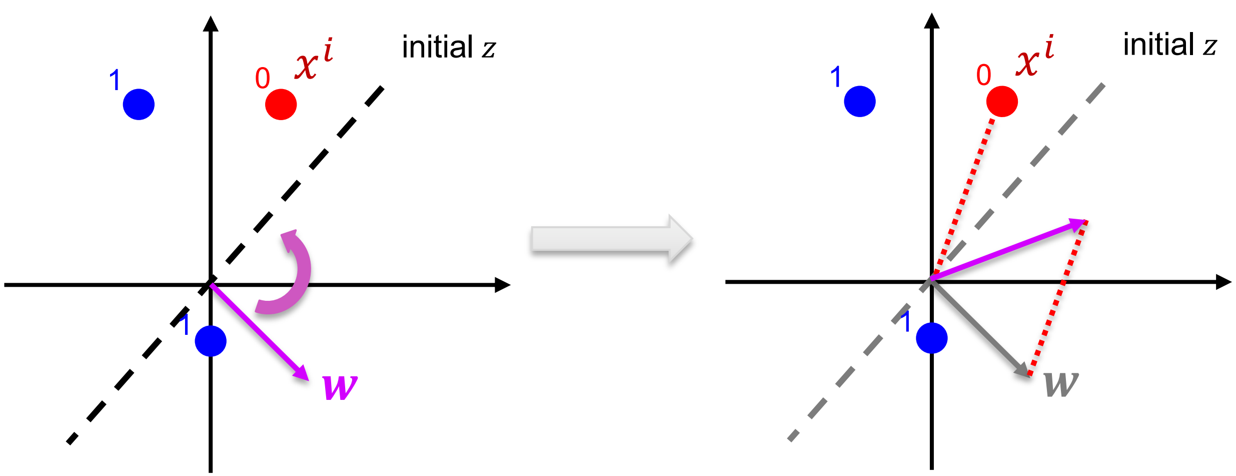
step function ↔ signum function



$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \cdot y^i x^i$$

$$\begin{cases} 1 & \text{for positive example} \\ -1 & \text{for negative example} \end{cases}$$

Why x^i ?



Perceptron – Learning Rule

Consequently,

Initialize the weights, $\boldsymbol{w}$, randomly

Repeat:

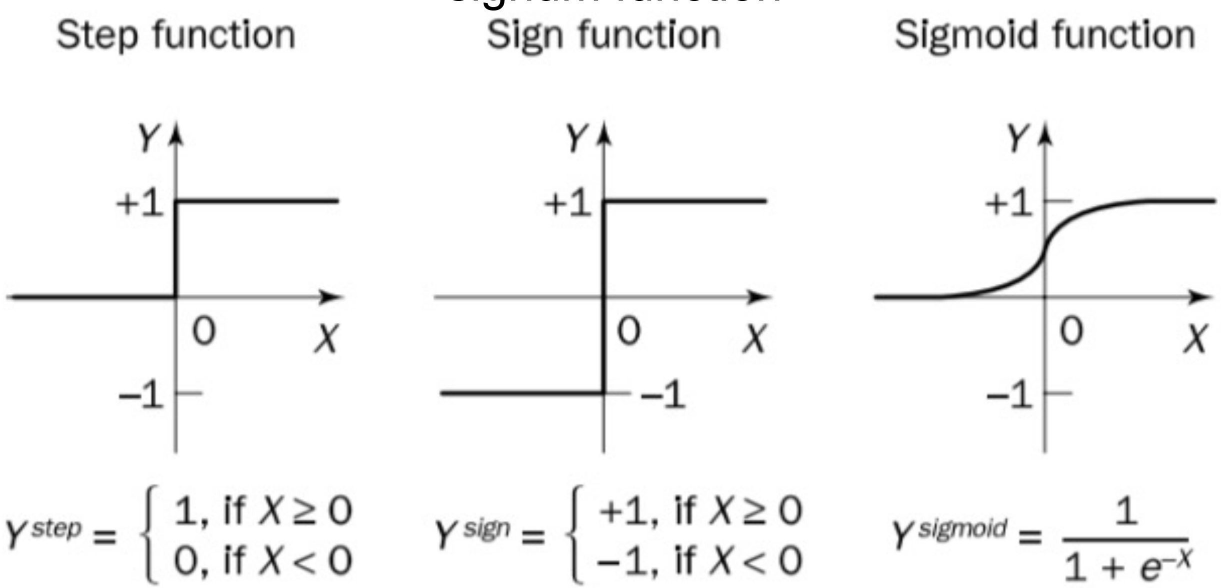
For each training example (x^i, y^i)

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\left(y^i - \hat{y}^i\right)x^i$$

Stop if the weights were not updated in this epoch.

Perceptron – Activation Function

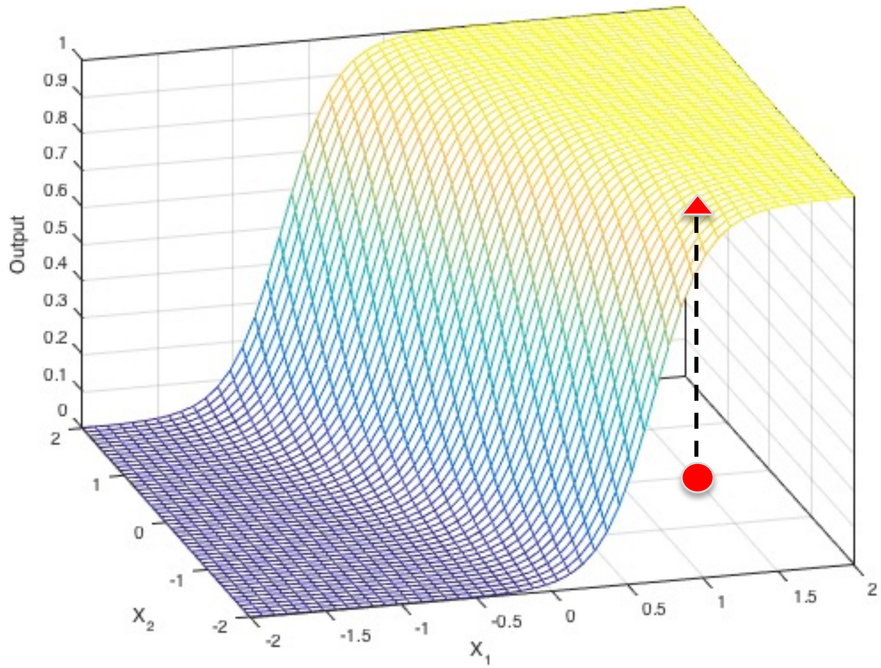
signum function

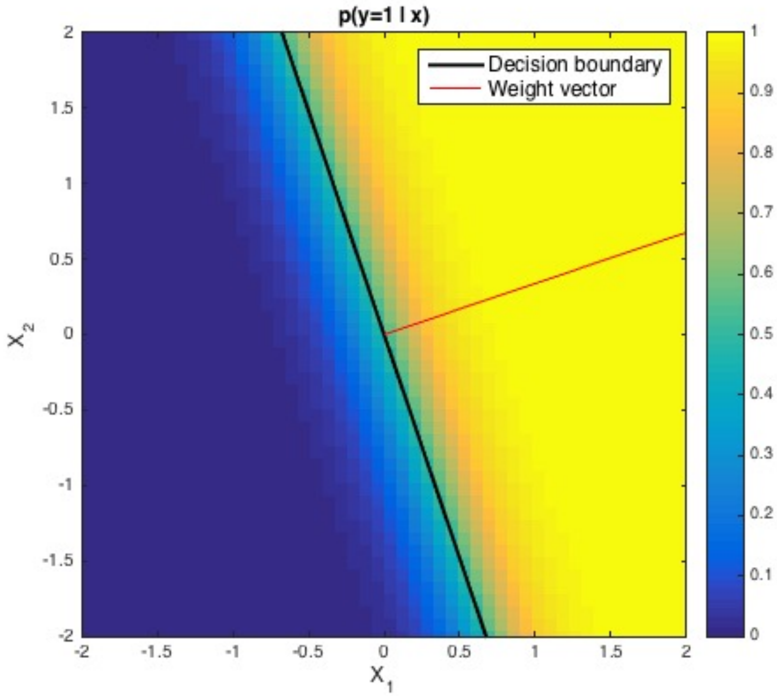


Sgn(.)

A single perceptron with a sigmoid(logistic) activation function is same as a logistic regression model. So we can use Gradient Descent again!

Perceptron – sigmoid in 3D space

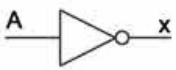
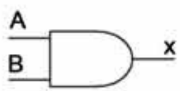




top view of the sigmoid function

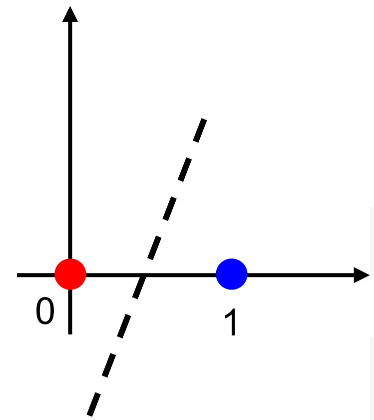
Perceptron – Logic Gates

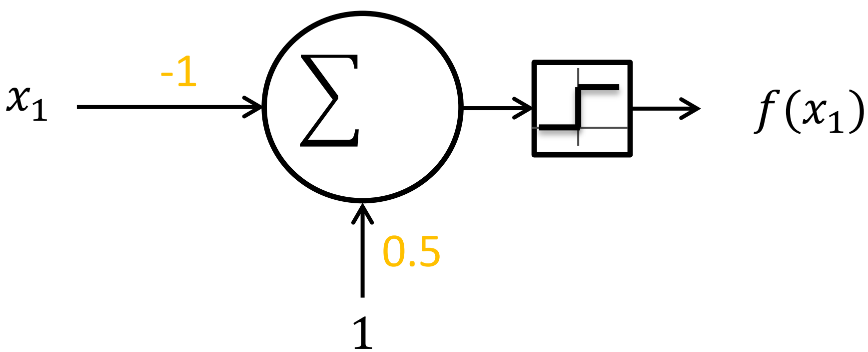
The perceptron naturally implements simple logic gates.

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A\oplus B$	$\overline{A\oplus B}$
Symbol							
Truth Table	A X 0 **1** 1 **0**	B A X 0 0 **0** 0 1 **0** 1 0 **0** 1 1 **1**	B A X 0 0 **1** 0 1 **1** 1 0 **1** 1 1 **0**	B A X 0 0 **0** 0 1 **1** 1 0 **1** 1 1 **1**	B A X 0 0 **1** 0 1 **0** 1 0 **0** 1 1 **0**	B A X 0 0 **0** 0 1 **1** 1 0 **1** 1 1 **0**	B A X 0 0 **1** 0 1 **0** 1 0 **0** 1 1 **1**

Perceptron – NOT

x_1	Output
0	1
1	0



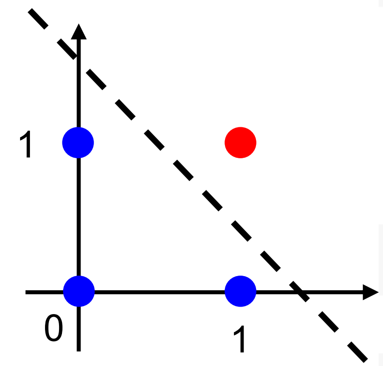


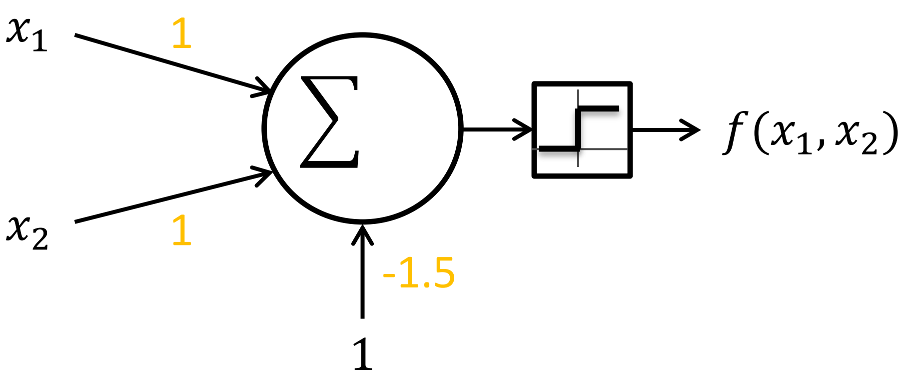
$$output = \begin{cases} 1 & if \;\; 0.5 - x_1 \geq 0 \\ 0 & otherwise \end{cases}$$

w=[0.5, -1]

Perceptron – AND

x_1	x_2	Output
0	0	0
0	1	0
1	0	0
1	1	1



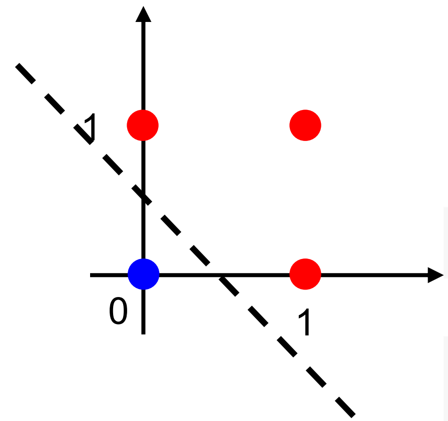


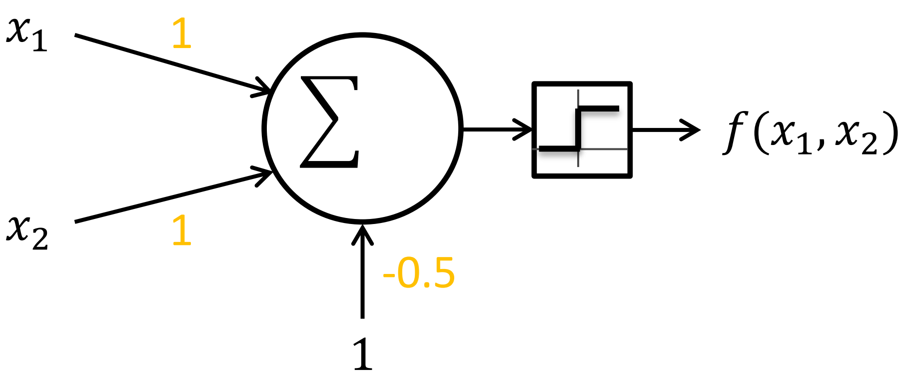
$$output = \begin{cases} 1 & if \ -1.5 + x_1 + x_2 \geq 0 \\ 0 & otherwise \end{cases}$$

w=[-1.5, 1, 1]

Perceptron – OR

x_1	x_2	Output
0	0	0
0	1	1
1	0	1
1	1	1



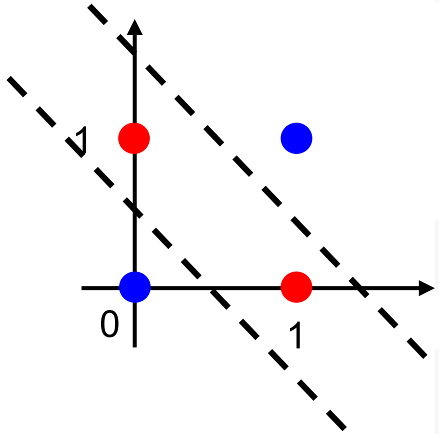


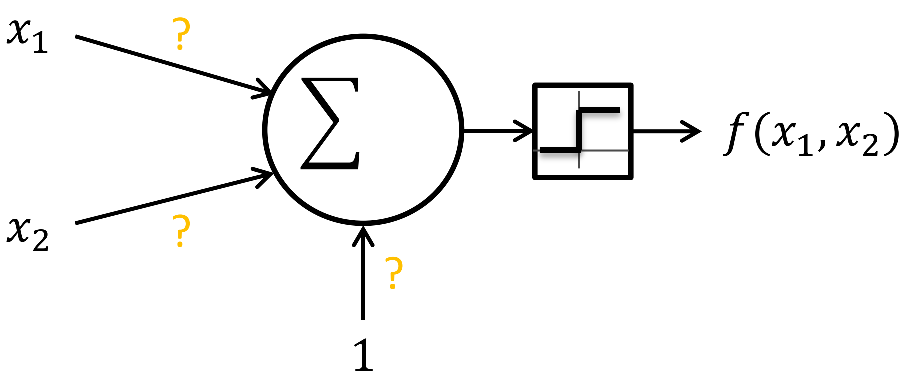
$$output = \begin{cases} 1 & if \; -0.5 + x_1 + x_2 \geq 0 \\ 0 & otherwise \end{cases}$$

w=[-0.5, 1, 1]

Perceptron – XOR

x_1	x_2	Output
0	0	0
0	1	1
1	0	1
1	1	0

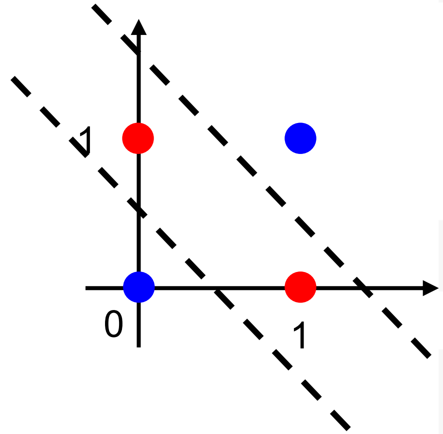


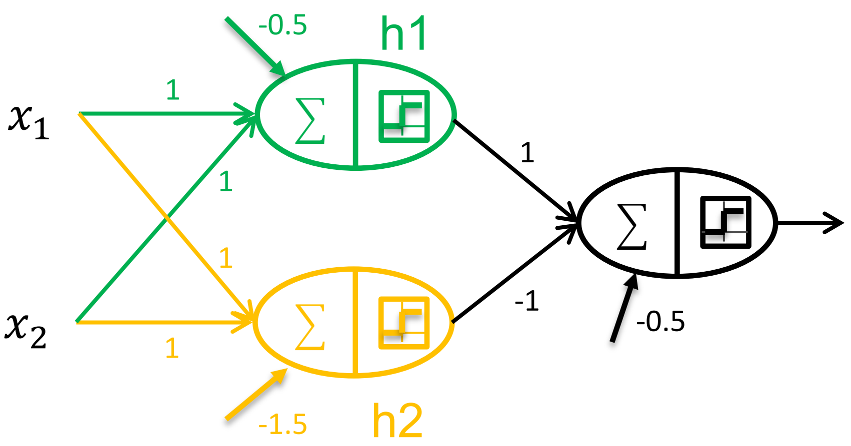


The XOR operator is not linearly separable and cannot be achieved by a single perceptron. ---- Limitation of Perceptron

Multi-layer Perceptron(MLP) – implementing XOR by MLP

x_1	x_2	Output
0	0	0
0	1	1
1	0	1
1	1	0

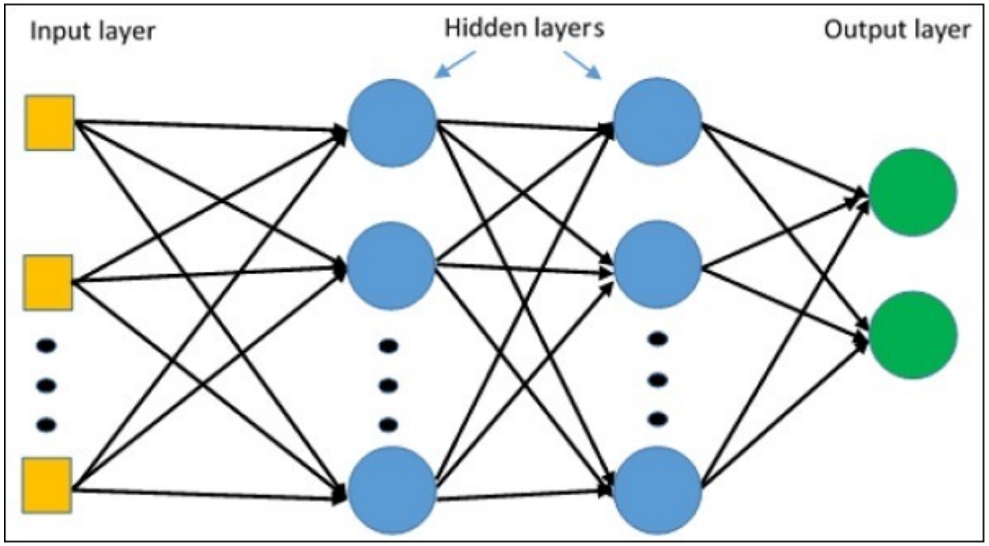


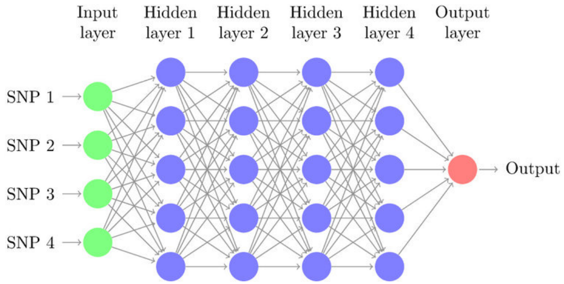


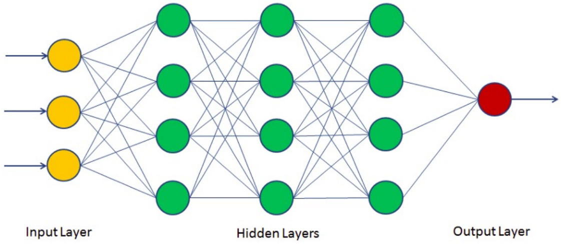
x1	x2	h1 sign(x_1+ x_2− 0.5)	h2 sign(x_1+ x_2− 1.5)	Final output sign(hu1 − hu2−0.5)
0	0	− 1	− 1	− 1
0	1	+ 1	− 1	+ 1
1	0	+ 1	− 1	+ 1
l	1	+ 1	+ 1	− 1

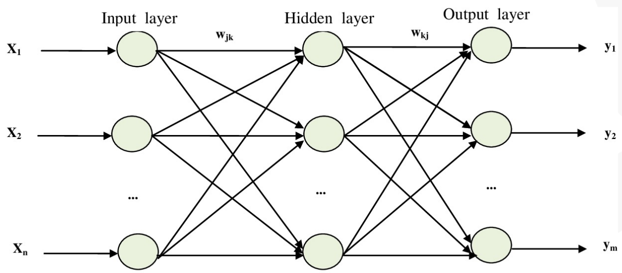
Multi-layer Perceptron(MLP)

An MLP consists of **at least three** layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function.









Multi-class Network – Review of Binary Classification

$$h(x) = sigmoid(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)$$ Sigmoid activation function

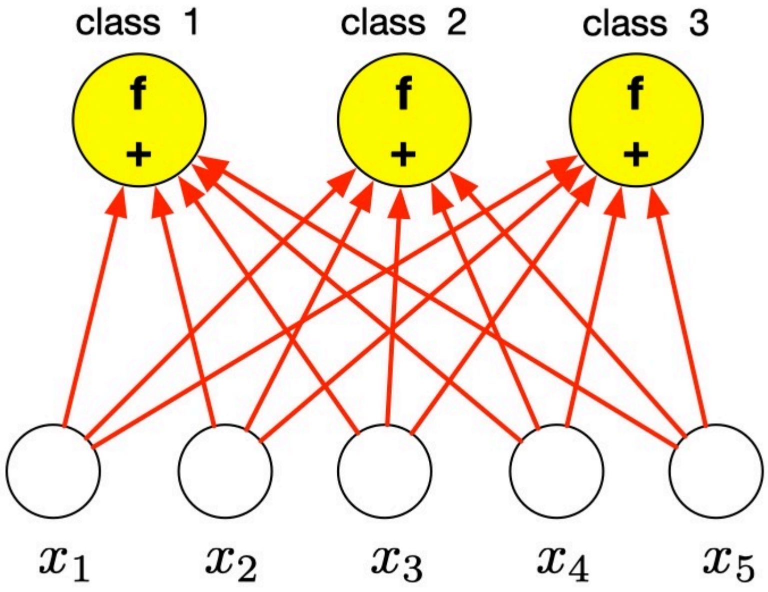
$$\mathcal{L} = -y \cdot \log\big(h(x)\big) - (1-y)\log(1 - h(x))$$ Cross-entropy loss function

$$\min \mathcal{J}(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}$$ Using Gradient Descent

Multi-class Network

We can transform the multi-class classification to several binary classification.
If we have K classes, set target of the correct class is **1**, all other targets are **0**

It is possible to have a multi-class net with sigmoids



Using multiple sigmoids for multiple classes means that the outputs of the network are not constrained to sum to one

Multi-class Network

We can do better…

To interpret the outputs of the net as classification probabilities, require

$$\sum_k \underline{P(C_k|x)} = 1$$

index of class   output of each neuron

Solution – use an output activation function with a sum-to-one constraint: **Softmax**

Multi-class Network – Notations

Notations:

K: number of classes

C_k: *k'* th class

y^i: the target label for the i' th training example.

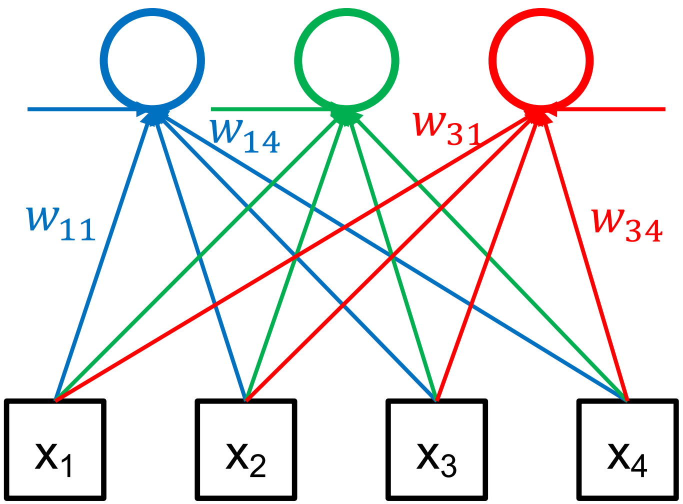
It is often more convenient to represent the target label as one-hot vector or *one-of-K* encoding

$$y^i = \underbrace{[0, \dots, 0,1,0, \dots, 0]}_{K \text{ dimensions}}$$

If y^i is belong to C_k class, the '1' should be at the k' th dim.

Multi-class Network – Notations

Notations:



w_{k0}: bias of the k'th neuron

w_{kj}: the weight for the connection from j'th input to k'th neuron

n: number of input dimensions

The weighted sum for the k'th neuron: $z_k = \sum_{j=0}^{n} w_{kj} x_j$

Multi-class Network – Softmax

Softmax function:
a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector.

$$softmax(z_1, \dots, z_K)_k = \frac{e^{z_k}}{\sum_p^K e^{z_p}}$$

$$\sigma(\boldsymbol{z})_k = \frac{\exp(z_k)}{\sum_p^K \exp(z_p)}$$

vector

Properties:

- Outputs are positive and sum to 1 (so they can be interpreted as probabilities)

- Exercise: how does the case of $K = 2$ relate to the sigmoid (logistic) function?

Multi-class Network – Softmax