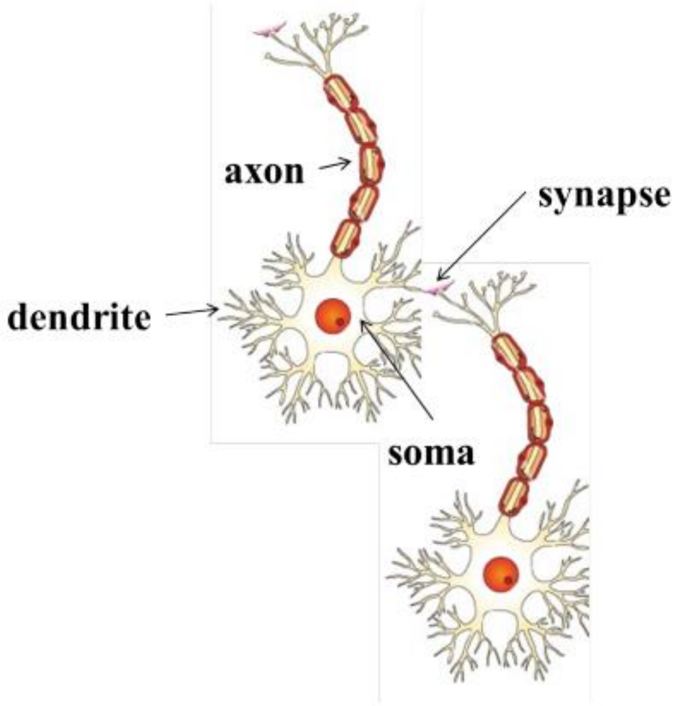
Neural Network

Perceptron and Activation

Neural Network

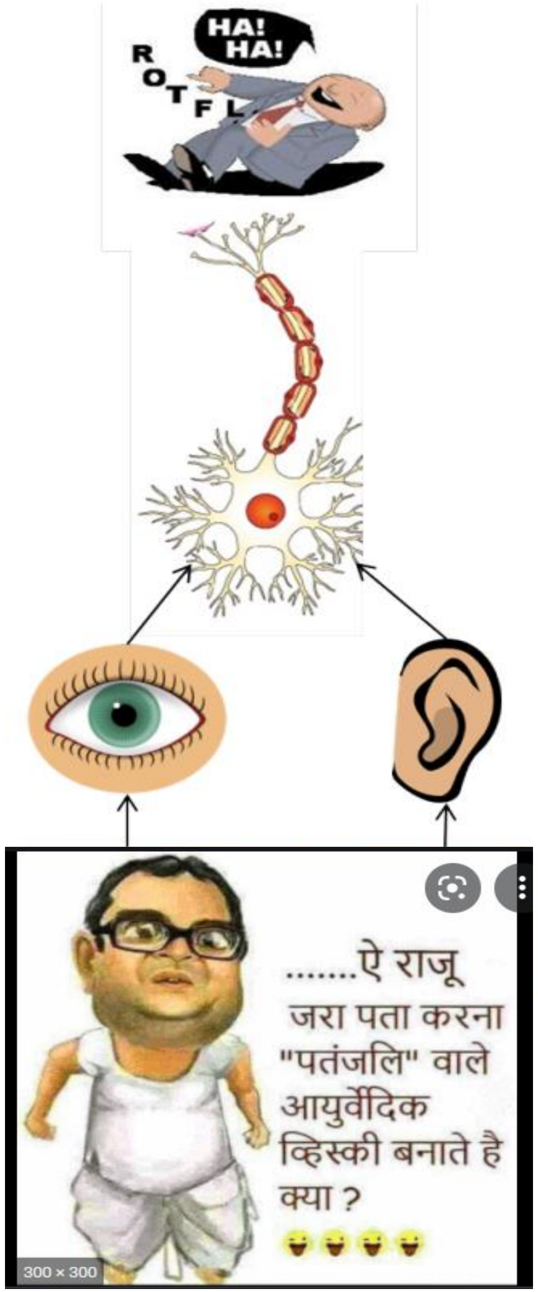
- Most fundamental unit of a neural network is an artificial neuron
- Inspiration comes from biology
- $biological\ neurons = neural\ cells = neural\ processing\ units$

- Terminology:
- This network contains 3 layer
- The layer containing the inputs (x_1, x_2) is called **input layer**.
- The middle layer containing 4 perceptron is called the **hidden layer**.
- The final layer containing one output neuron is called the **output layer**.

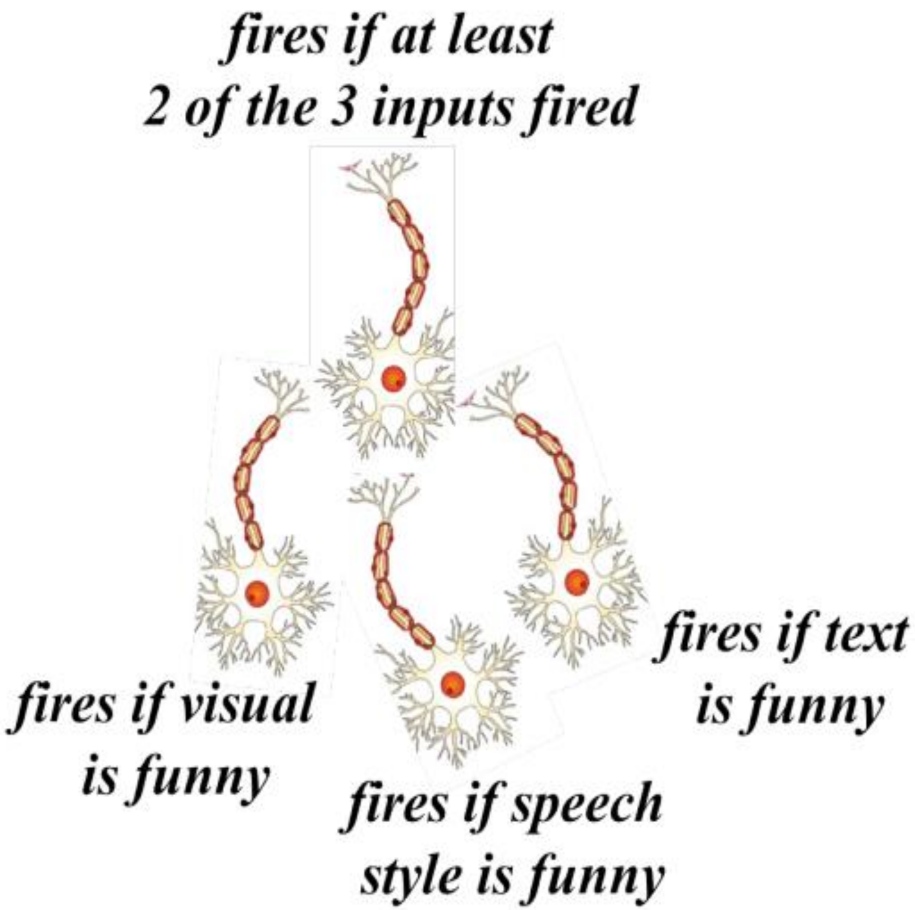


Biological Neurons*

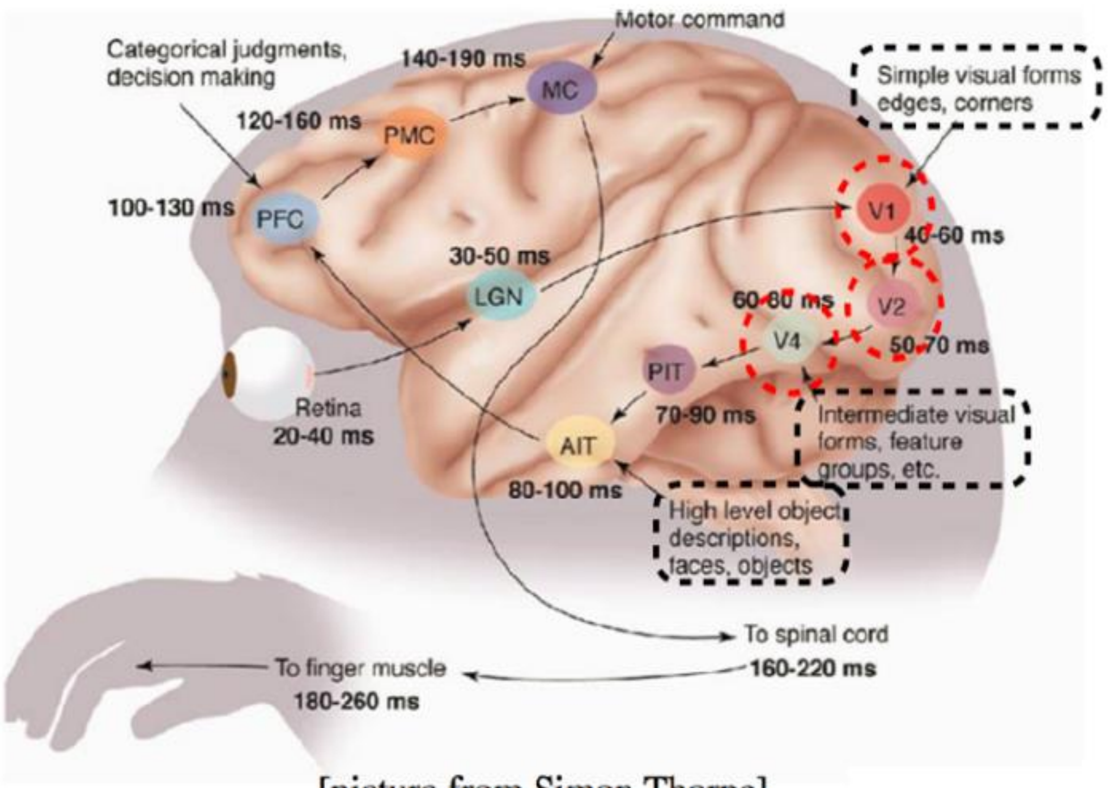
- dendrite: receives signals from other neurons
- synapse: point of connection to other neurons
- soma: processes the information
- axon: transmits the output of this neuron



- Our sense organs interact with the outside world
- They relay information to the neurons
- The neurons (may) get activated and produces a response (laughter in this case)
- Of course, in reality, it is not just a single neuron which does all this
- There is a massively parallel interconnected network of neurons
- The sense organs relay information to the lowest layer of neurons

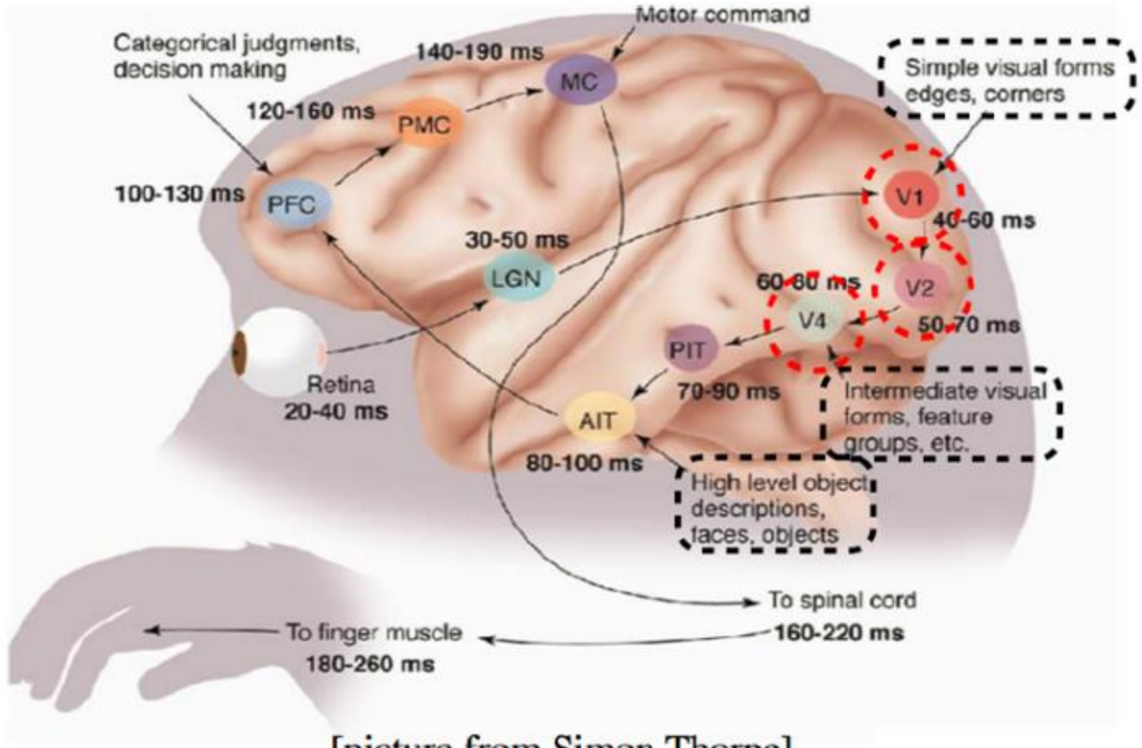


- An average human brain has around 10^11 (100 billion) neurons!
- Massively parallel network also ensures that there is division of work
- Each neuron may perform a certain role or respond to a certain stimulus

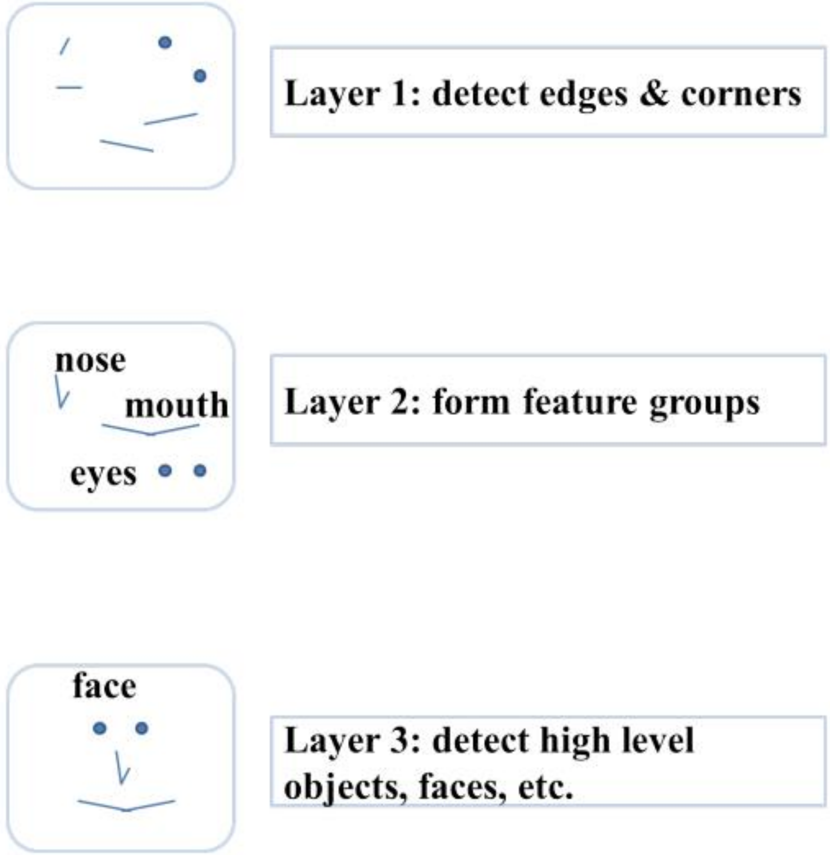


[picture from Simon Thorpe]

- The neurons in the brain are arranged in a hierarchy
- We illustrate this with the help of visual cortex (part of the brain) which deals with processing visual information
- Starting from the retina, the information is relayed to several layers (follow the arrows)
- We observe that the layers V1, V2 to AIT form a hierarchy (from identifying simple visual forms to high level objects)

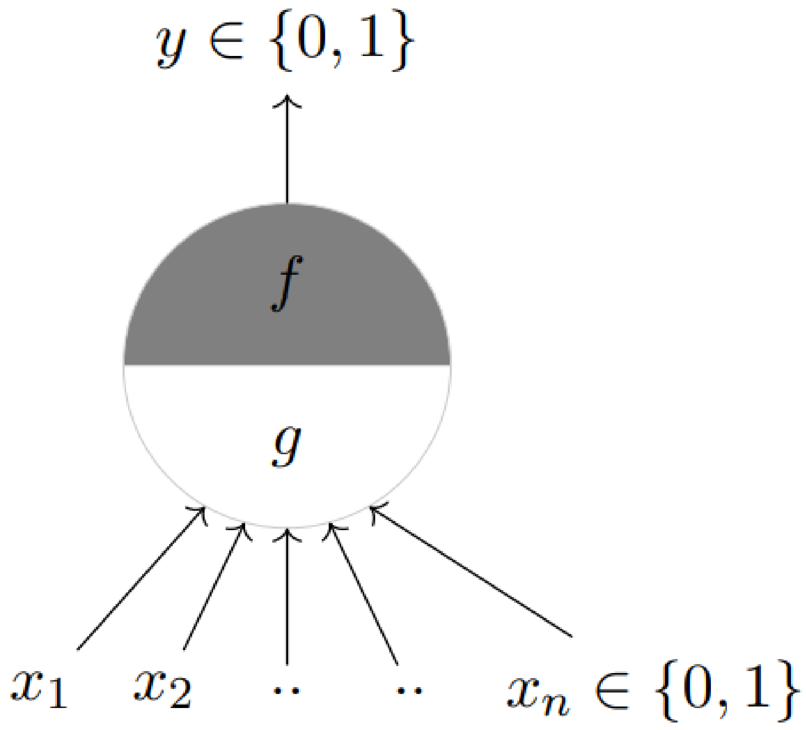


[picture from Simon Thorpe]



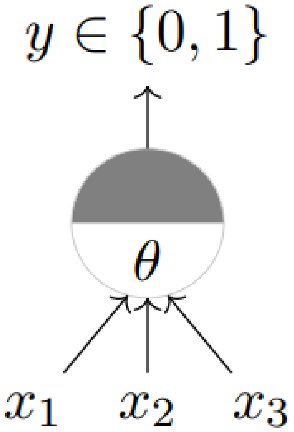
Sample illustration of hierarchical processing*

McCulloch Pitts Neuron

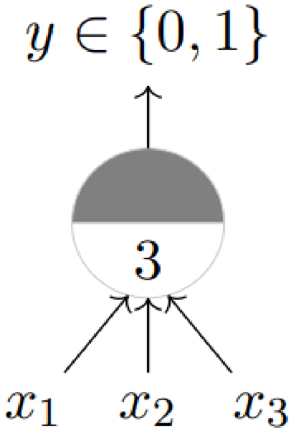


- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified computational model of the neuron (1943)
- g aggregates the inputs and the function f takes a decision based on this aggregation
- The inputs can be excitatory or inhibitory
- y = 0 if any xi is inhibitory, else
- $g(x_1, x_2, \ldots, x_n) = g(x) = \sum_{\{i=1\}}^{n} x_i$
- $y = f\big(g(x)\big) = 1 \; if \; g(x) \geq \theta,$
  $\qquad\qquad\quad = 0 \; if \; g(x) < \theta$
- θ is thresholding parameter

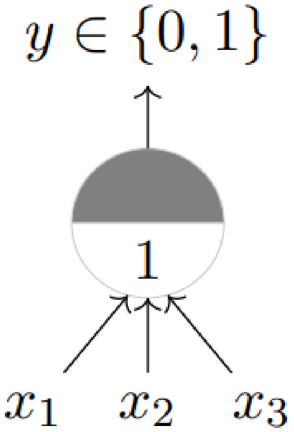
Implementation Boolean function



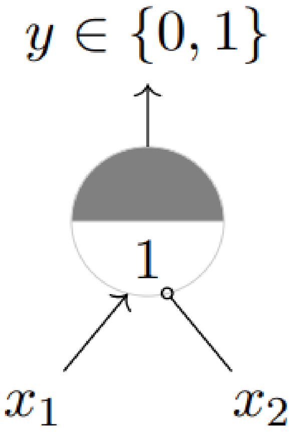
A McCulloch Pitts unit



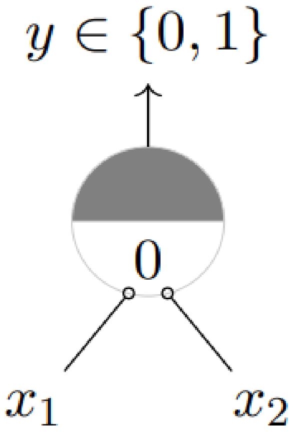
AND function



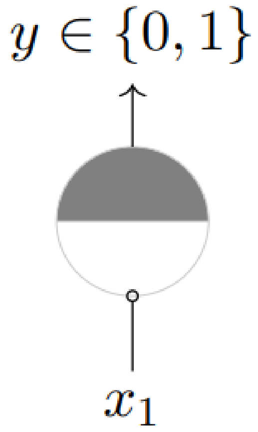
OR function



x_1 AND $!x_2$*



NOR function

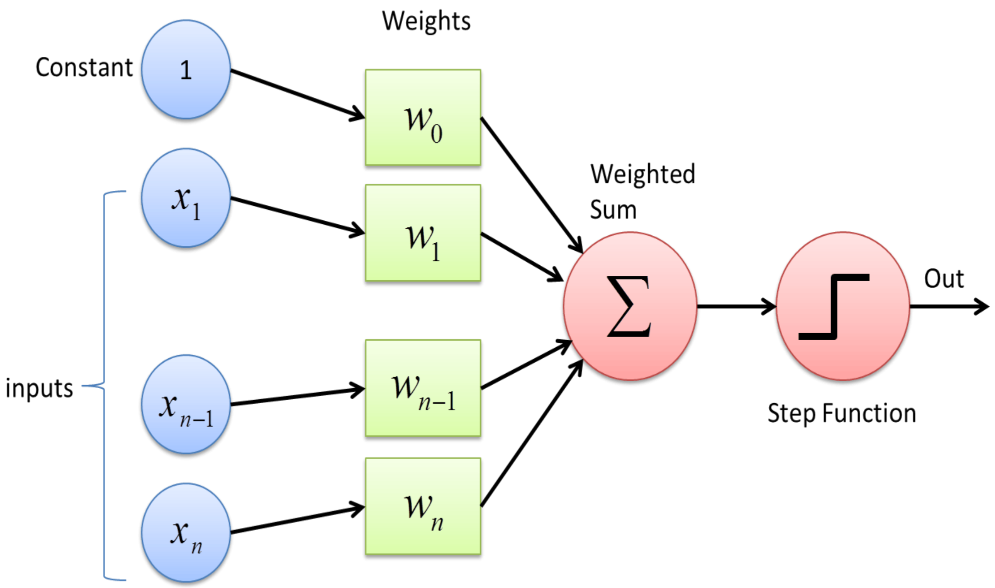


NOT function

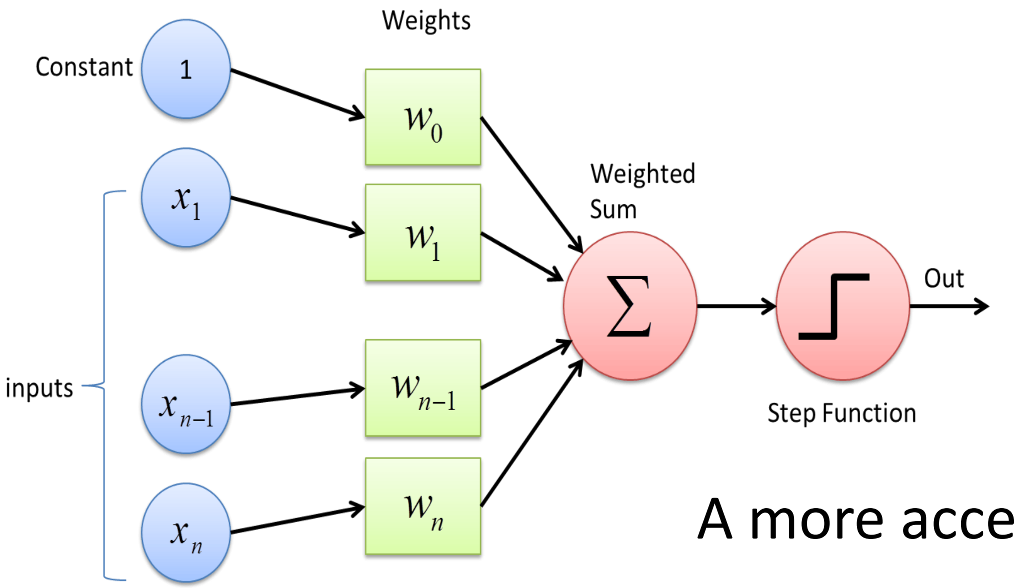
- A single MP neuron splits the inputs into two halves
- A single MP neuron can be used to represent Boolean functions which are linearly separable

Perceptron

- Inputs can not be boolean always
- Threshold calculation is not easy
- Functions can be linearly nonseparable
- We may need to assign weight(important) for some inputs.



- Frank Rosenblatt, an American psychologist, proposed the classical perceptron model(1958)
- A more general computational model than McCulloch–Pitts neurons
- Main differences: Introduction of numerical weights for inputs and a mechanism for learning these weights
- Inputs are no longer limited to boolean values
- Refined and carefully analyzed by Minsky and Papert (1969) - their model is referred to as the perceptron model here



$$y = 1\ if \sum_{\{i=1\}}^{n} w_i * x_i \geq \theta$$

$$= 0\ if \sum_{\{i=1\}}^{n} w_i * x_i < \theta$$

Rewriting the above

$$y = 1\ if \sum_{\{i=1\}}^{n} w_i * x_i - \theta \geq 0$$

$$= 0\ if \ \textstyle\sum_{\{i=1\}}^{n} w_i * x_i - \theta < 0$$

A more accepted convention

$$y = 1\ if \sum_{\{i=0\}}^{n} w_i * x_i - \theta \geq 0$$

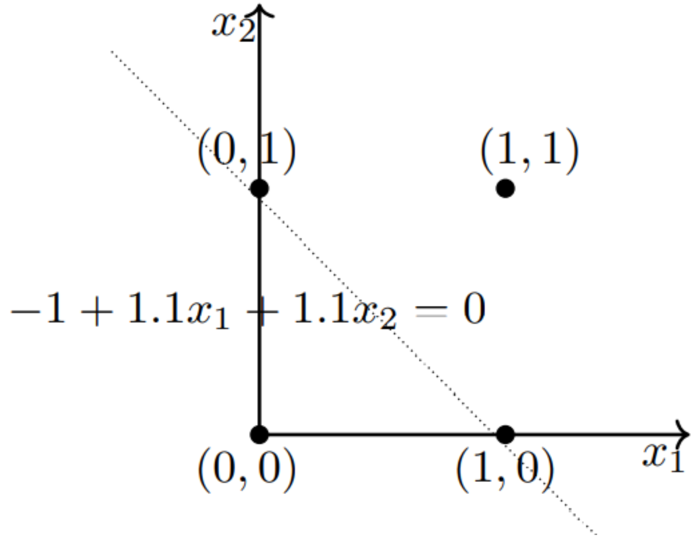
$$= 0\ if \ \textstyle\sum_{\{i=0\}}^{n} w_i * x_i - \theta < 0$$

Where $x_0 = 1\ and\ w_0 = -\theta$

x_1	x_2	OR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

- $w_0 + w_1.0 + w_2.0 < 0 \rightarrow w_0 < 0$
- $w_0 + w_1.1 + w_2.0 \geq 0 \rightarrow w_1 \geq -\mathrm{w}_0$
- $w_0 + w_1.0 + w_2.1 \geq 0 \rightarrow w_2 \geq -\mathrm{w}_0$
- $w_0 + w_1.1 + w_2.1 \geq 0 \rightarrow w_1 + w_2 \geq -\mathrm{w}_0$

- One possible solution to these set of inequalities is $w_0 = -2, w_1 = 2.2, w_2 = 2.2$ *and others*



Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs\ with\ label\ 1;$

N $\leftarrow inputs\ with\ label\ 0;$

Initialize w randomly

While ! $convergence$ do

Pick x random $x \in P \cup N$

If $x \in P$ and $\sum_{i=0}^{n} w_i * x_i < 0$ then

$$w = w + x$$

End

$$if\ x \in N\ and \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$w = w - x$$

End

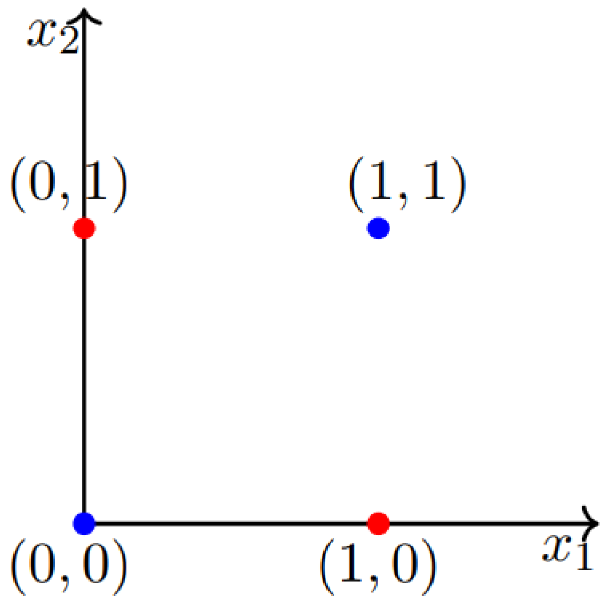
End

Network of Perceptron

XOR function

x_1	x_2	XOR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$

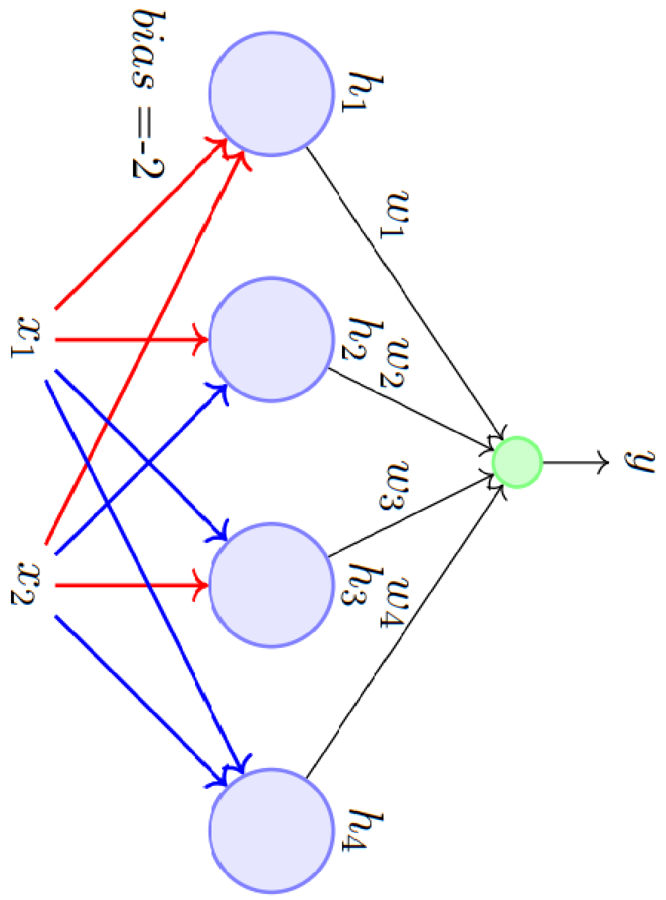
$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$
$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 \geq -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 \geq -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 < 0 \implies w_1 + w_2 < -w_0$$



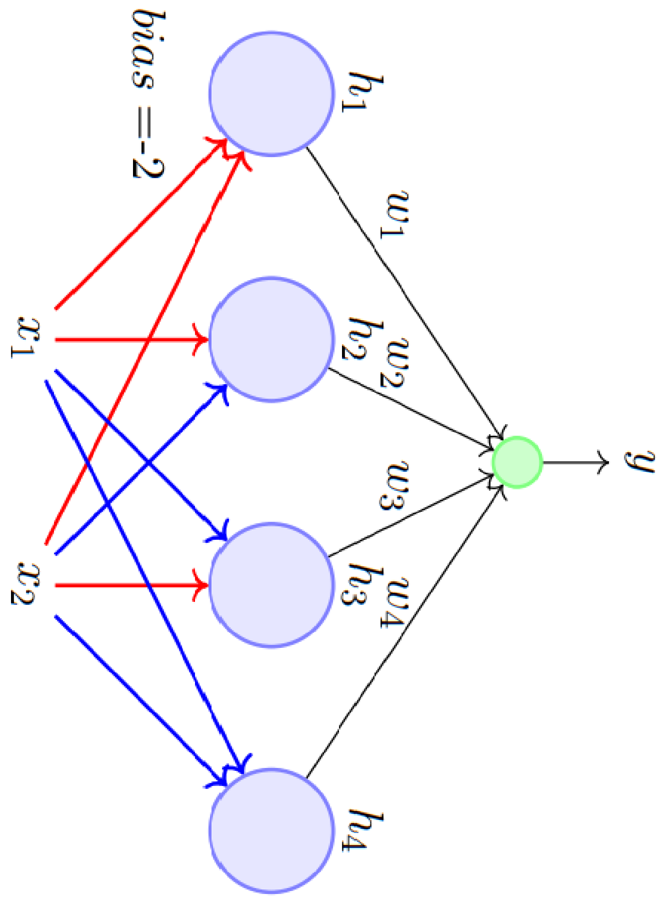
- Impossible to draw a line which separates the red and blue points.

- Condition 4 contradicts 2 and 3
- Hence we cant have solution to this set of inequalities

- Consider True=+1 and False =-1
- Consider 2 inputs and four perceptron
- Each input is connected to all 4 perceptron with specific weight.
- Red edges indicate w= -1, blue edges indicate w=+1

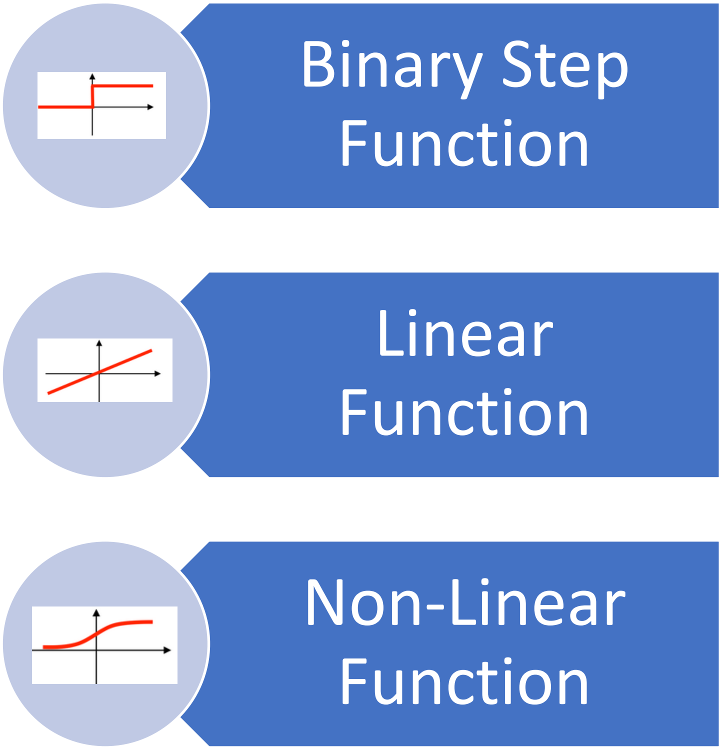


- Terminology:
- This network contains 3 layer
- The layer containing the inputs (x_1, x_2) is called **input layer**.
- The middle layer containing 4 perceptron is called the **hidden layer**.
- The final layer containing one output neuron is called the **output layer**.



Activation Function

- An activation function decides whether a should fire or not
- Derives output from a set of input values fed to a neuron
- Add non-linearity to the network
- A neuron without activation function performs a linear transformation on the inputs using weights and biases



Binary Step Function

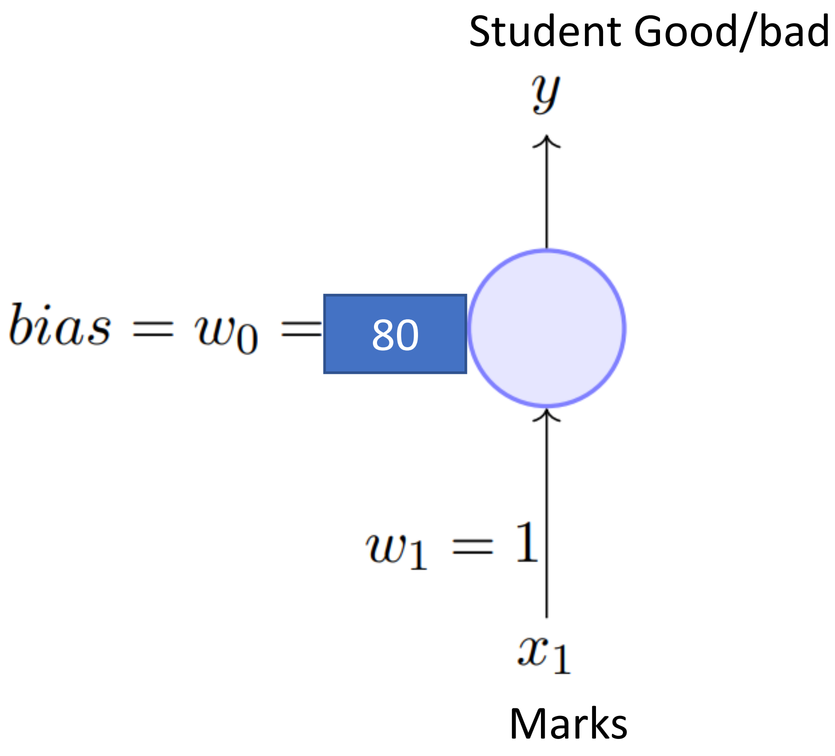
- Depends on a threshold value that decide whether a neuron should be activated or not
- Can not be used for multiclass classification
- Gradient of step function is zero

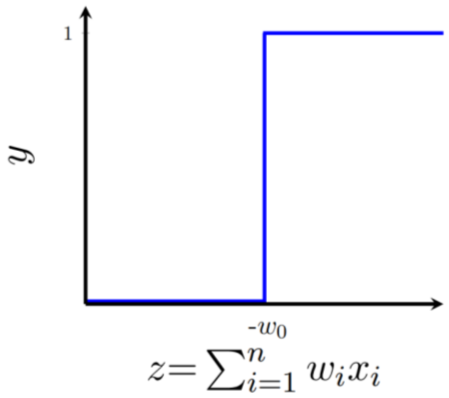
$$f(x) = \begin{cases} 0 & for\, x < 0 \\ 1 & for\, x \geqslant 0 \end{cases}$$

Linear Function

- Output function will be confined between any range
- $f(x) = x$
- Range $+\infty\ to\ -\infty$

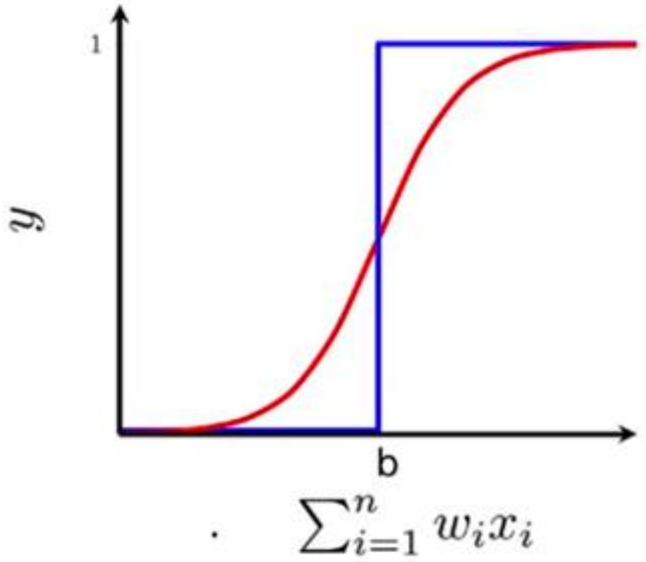
- Thresholding logic in perceptron is very rigid
- Consider the problem of deciding the student quality
- Student is good for marks 80 but bad for marks 79
- The perceptron function behaves like a step function
- Real world applications anticipate smoother functions which gradually changes from 0 to 1





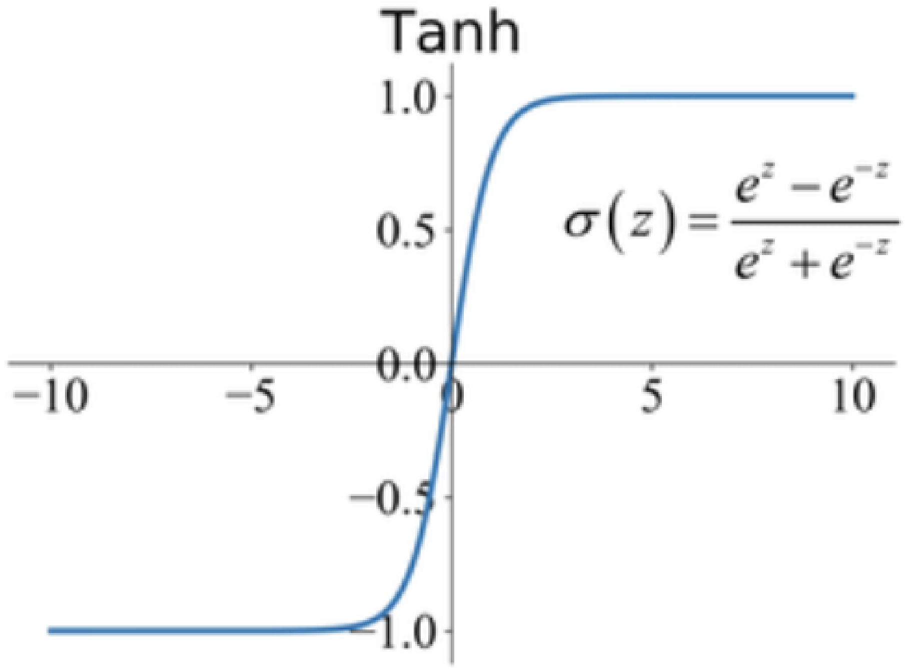
Sigmoid Neuron

- Sigmoid neurons adjust the abruptness of the function
- No sharp transition around threshold $-w_0$
- Logistic function:
- $y = \frac{1}{1+e^{-w_0+\sum w_{i^*} x_i}}$
- y is real value between 0 and 1 which can be interpreted as probability



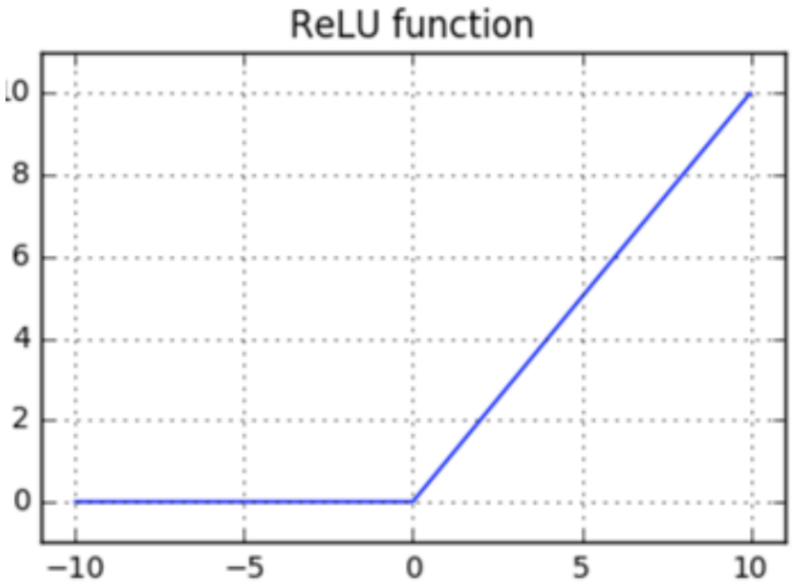
Tanh function

- Is sigmoidal(-s shape)
- Range of tanh from $-1\ to + 1$
- The negative inputs will be mapped strongly negative and zero inputs will be mapped near zero
- The function is differentiable



Relu Function

- Rectified Linear Unit
- Mostly used activation function
- $f(x) = \max(0, x)$
- Computationally less expensive



Objective

- Data: $\{x_i, y_i\}_{i=1}^{n}$
- Model: Objective is to build relation between x and y.
- $\hat{y} = w^T x$ or any other function
- Parameters: w is parameter which needs to learn from the data
- Learning algorithm: Gradient descent
- Objective/Loss/Error Function: minimize the loss function

- Parameter update rule:
- $w_{t+1} = w_t - \eta \nabla w_t \quad b_{t+1} = b_t - \eta \nabla b_t$
- Where $\nabla w_t = \frac{\partial \mathcal{L}(w,b)}{\partial w}_{at\ w=w_t and\ b=b_t}$
- and $\nabla b_t = \frac{\partial \mathcal{L}(w,b)}{\partial b}_{at\ w=w_t and\ b=b_t}$

Algorithm: gradient_descent()

$t \leftarrow 0;$
$max\_iterations \leftarrow 1000;$
while $t < max\_iterations$ **do**
  $w_{t+1} \leftarrow w_t - \eta \nabla w_t;$
  $b_{t+1} \leftarrow b_t - \eta \nabla b_t;$
  $t \leftarrow t + 1;$
end