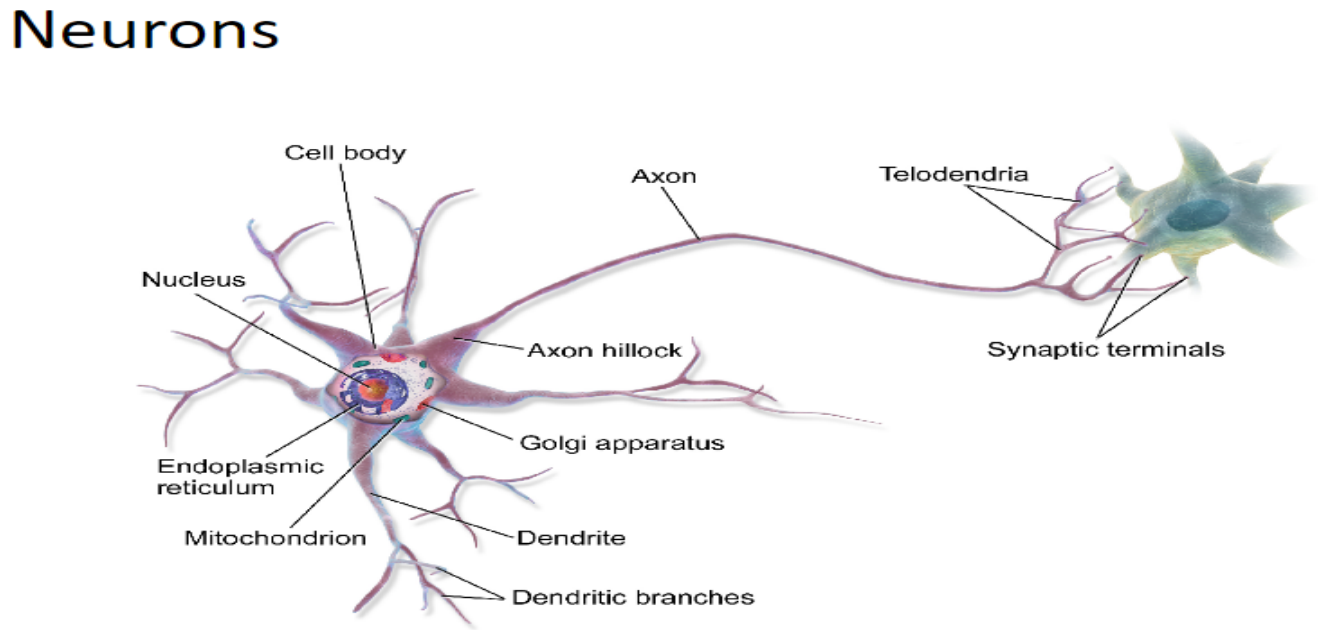
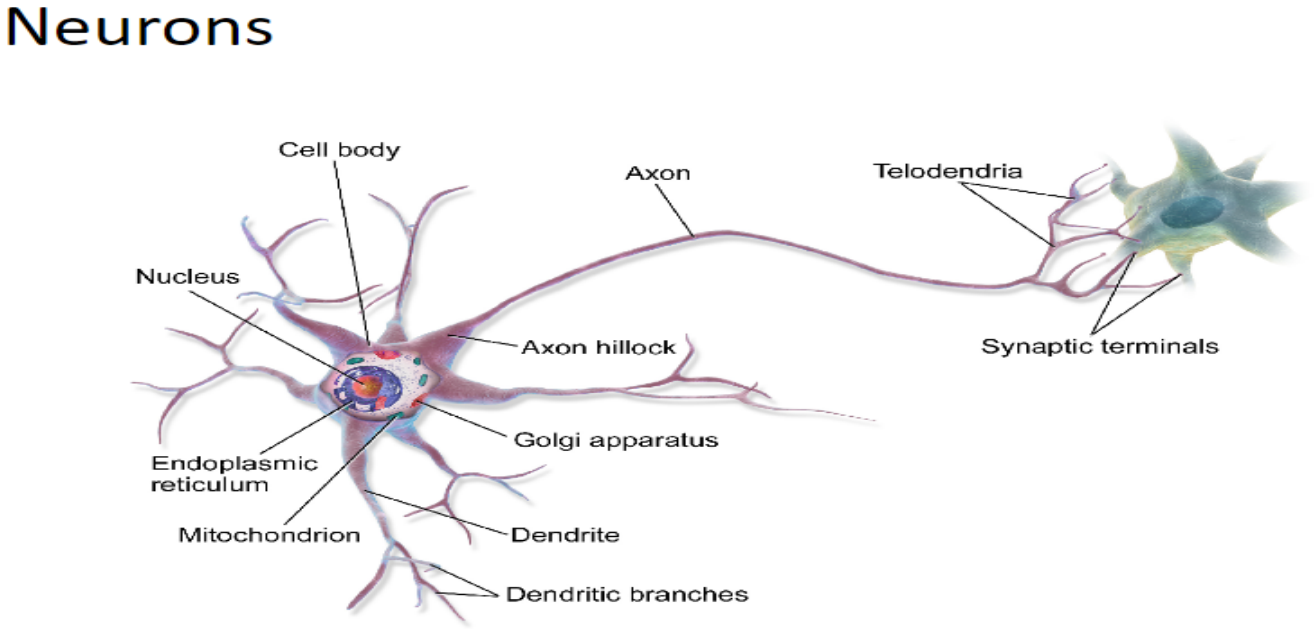
Biological Neuron

- The Nervous system helps us to sense and respond to our environment and it is divided into two parts: Central nervous system and Peripheral nervous system.
- Central nervous system consists of the brain and spinal cord.
- Peripheral nervous system consists of sensory and motor nerve cells that runs throughout the rest of the body.
- All cells of the nervous system are comprised of neurons. Neurons are the basic unit of the nervous system and nervous tissue.



Biological Neuron

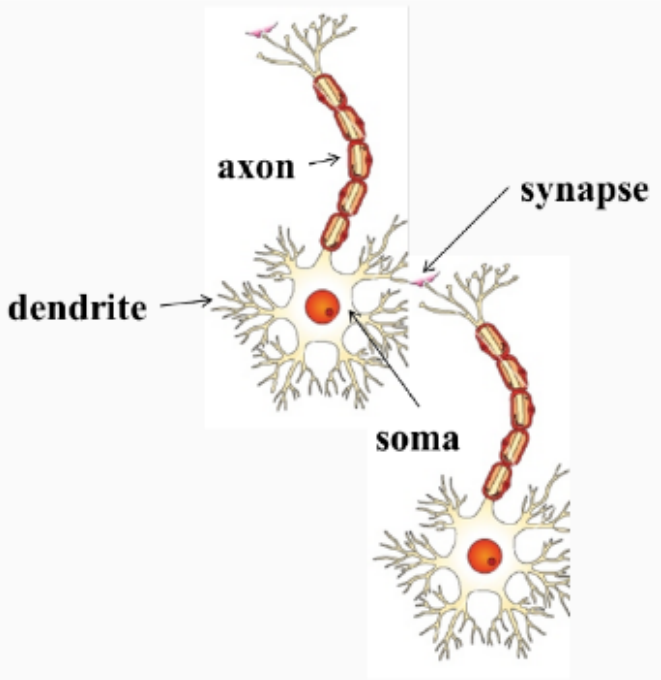
- A human brain consists of approximately 10^{11} neurons communicating with each other with the help of electrical impulses. Each neuron is approximately 10 micron in length.
- The neurons in brain help the human being to think, remember and solve the problem.
- There is a chemical called neurotransmitter in each neuron by which signal (sense) transmitted.



Biological Neuron

- Neurons communicate with each other with the help of electrical impulses.
- Neurons carry electrical impulses from one part to another part of the body.
- Neurons are responsible for sending, receiving, and interpreting information from all parts of the body.

Neurons

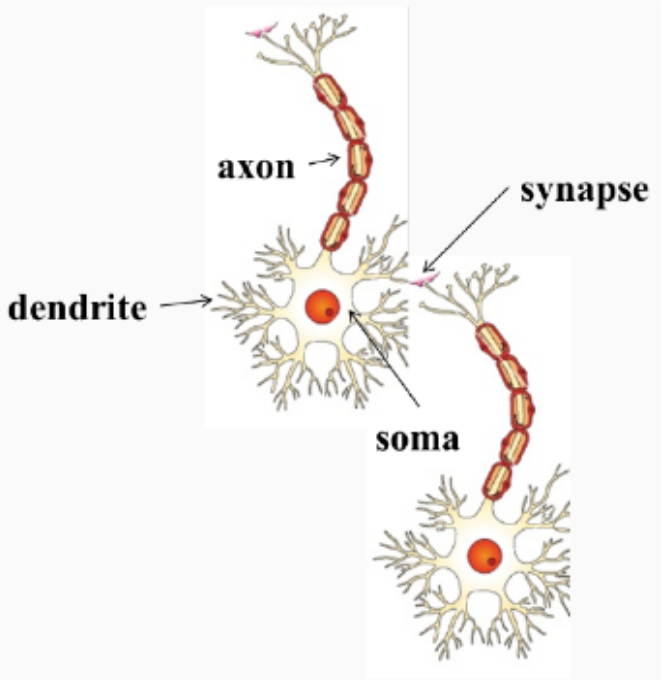


- **dendrite:** receives signals from other neurons
- **synapse:** point of connection to other neurons
- **soma:** processes the information
- **axon:** transmits the output of this neuron

Biological Neuron

- All inputs (signal) from one neuron arrive to another neuron through dendrites.
- Then the input signals are accumulated and processed at soma.
- The processed signal is transmitted as output from the axon to the other neuron through synapse.

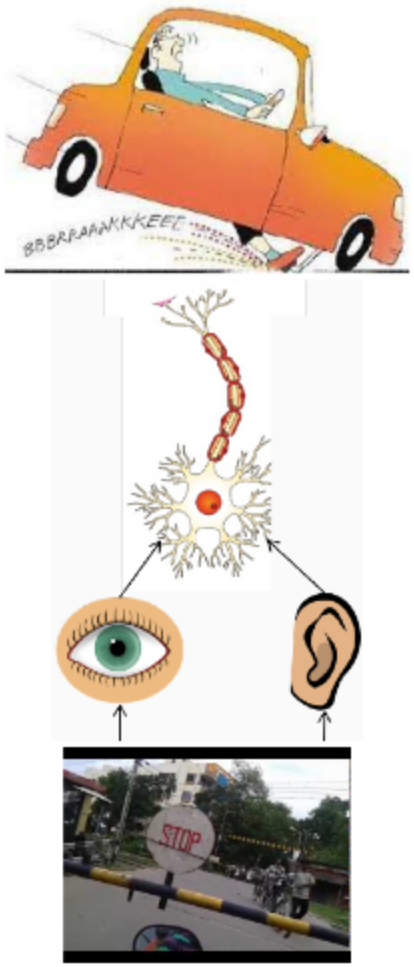
Neurons



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

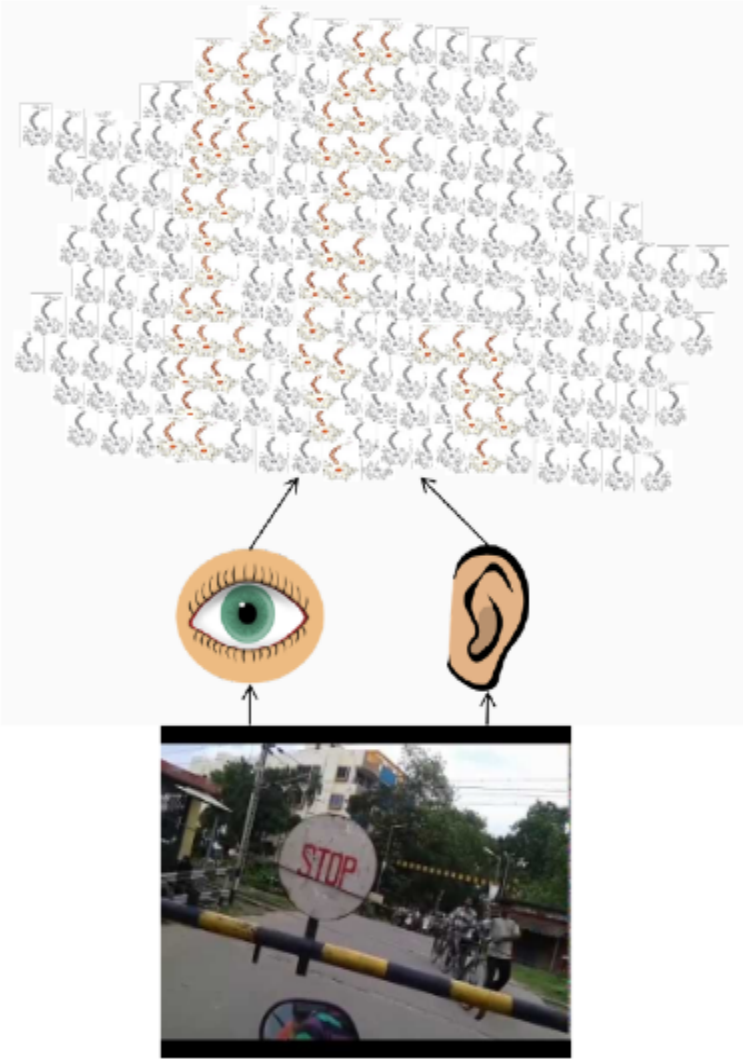
Biological Neuron

- Our sense organs interact with the outside world
- They relay information to the neurons
- The neurons (may) get activated and produces a response (applying car brake in this case)



Biological Neuron

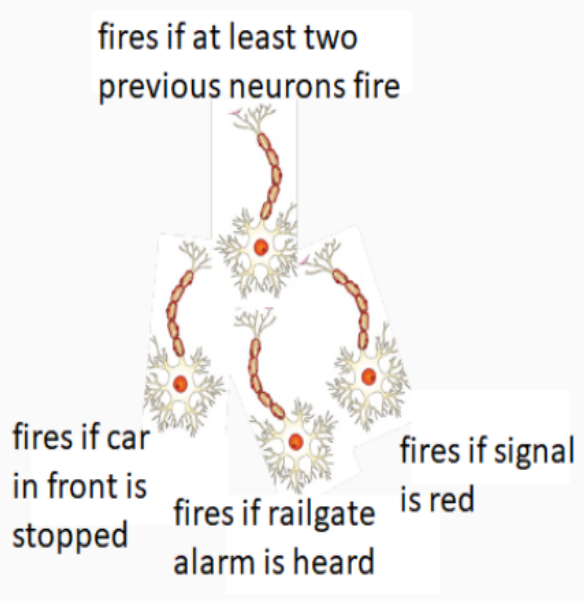
- Of course, in reality, it is not just a single neuron which does all this
- There is a massively parallel interconnected network of neurons
- The sense organs relay information to the lowest layer of neurons
- Some of these neurons may fire (in red) in response to this information and in turn relay information to other neurons they are connected to
- These neurons may also fire (again, in red) and the process continues eventually resulting in a response (applying car brake in this case)
- An average human brain has around 10^{11} (100 billion) neurons!



Biological Neuron

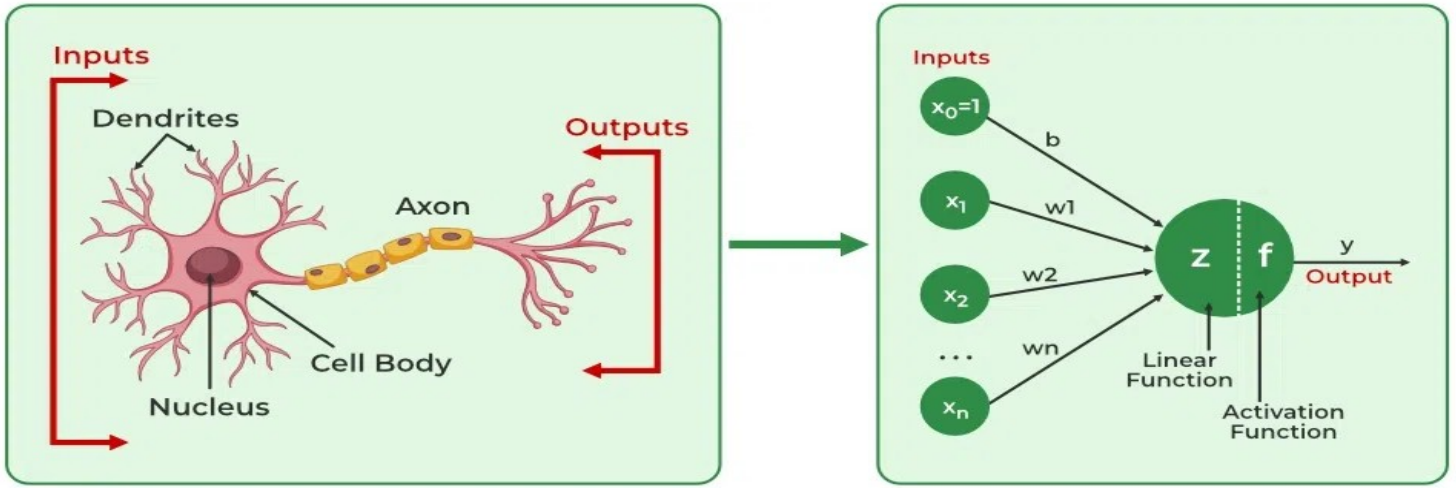
- An action (like when a mosquito bites, car comes in front of your car) may produce an electrical impulses, which usually lasts for about a millisecond.
- This pulse is created due to an incoming signal and all signals may not produce pulses in axon unless it crosses a threshold value (i.e., signal strength > threshold).
- The impulse in Axon of a neuron is collective signals arrive at dendrites which summed up at Soma (nucleus).

- This massively parallel network also ensures that there is division of work
- Each neuron may perform a certain role or respond to a certain stimulus



Biological Neuron to Artificial Neuron

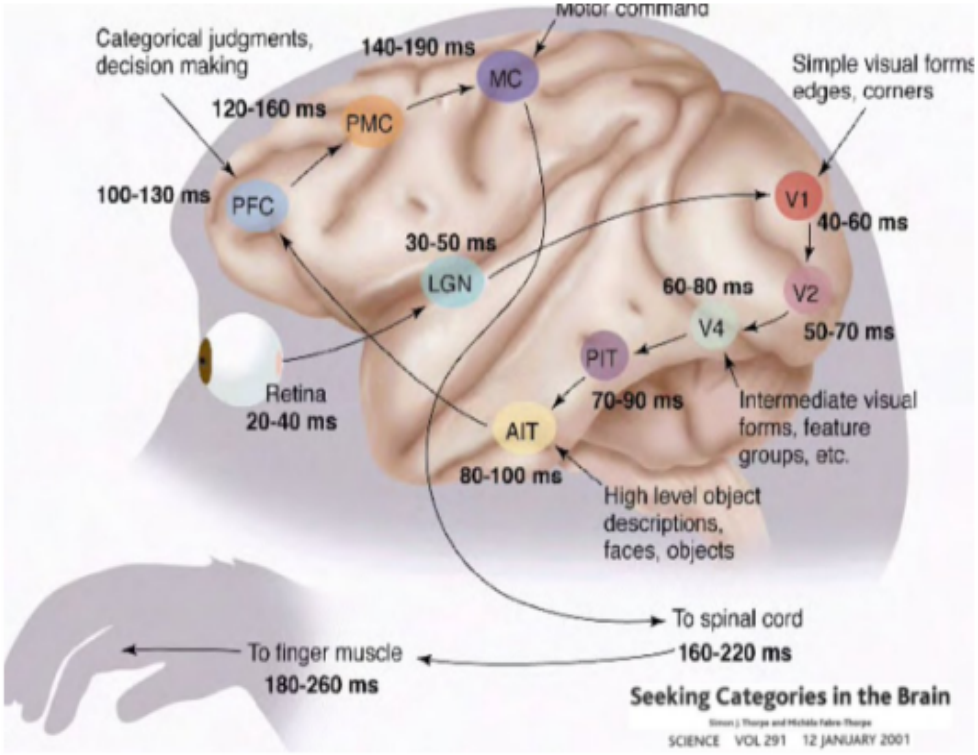
- An artificial neural network is inspired by the way the human brain processes information.
- It is just the mimic or simulation of our biological nervous system.



Biological Neuron	Dendrite	Cell nucleus or Soma	Synapses	Axon
Artificial Neuron	Inputs	Nodes	Weights	Output

Biological Neural Network

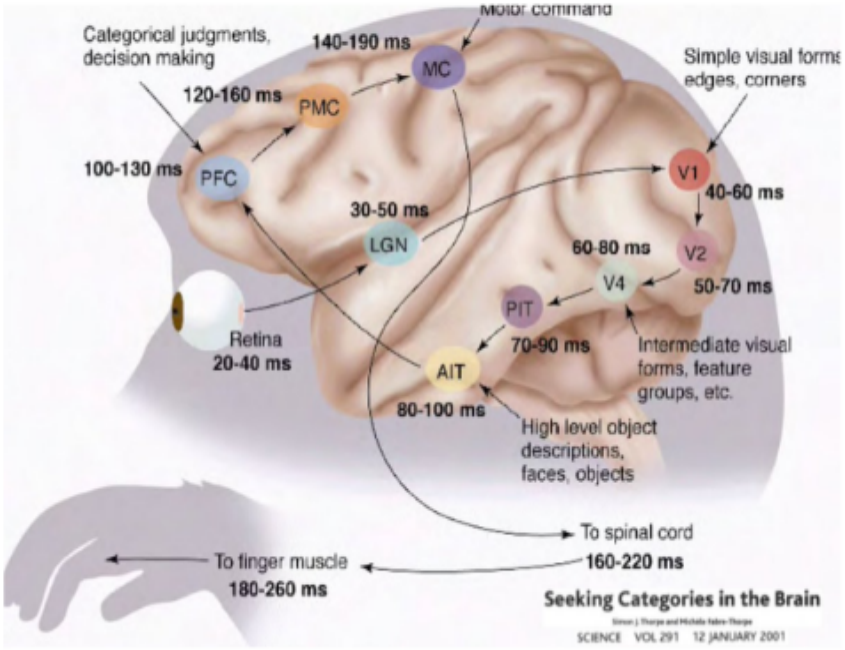
Hierarchical processing



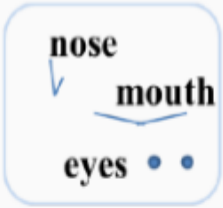
- The neurons in the brain are arranged in a hierarchy
- We illustrate this with the help of visual cortex (part of the brain) which deals with processing visual information
- Starting from the retina, the information is relayed to several layers (follow the arrows)

Biological Neural Network

Hierarchical processing (Example)



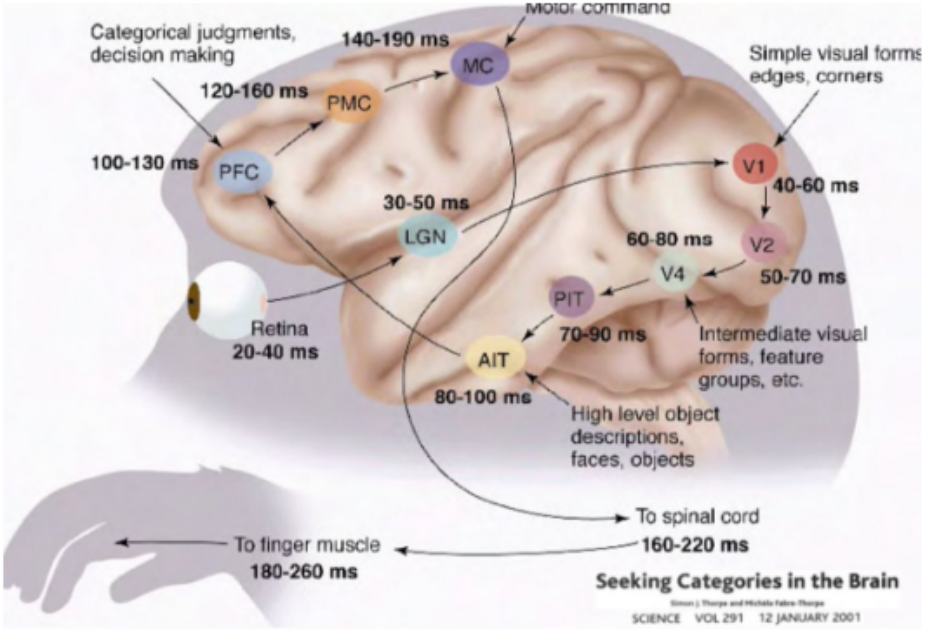
Layer 1: detect edges & corners

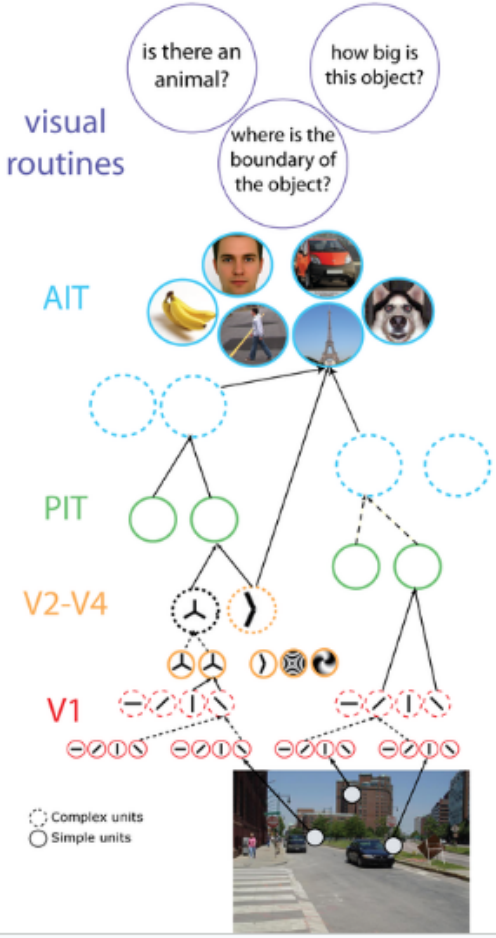


Layer 2: form feature groups

Layer 3: detect high level objects, faces, etc.

Biological Neural Network

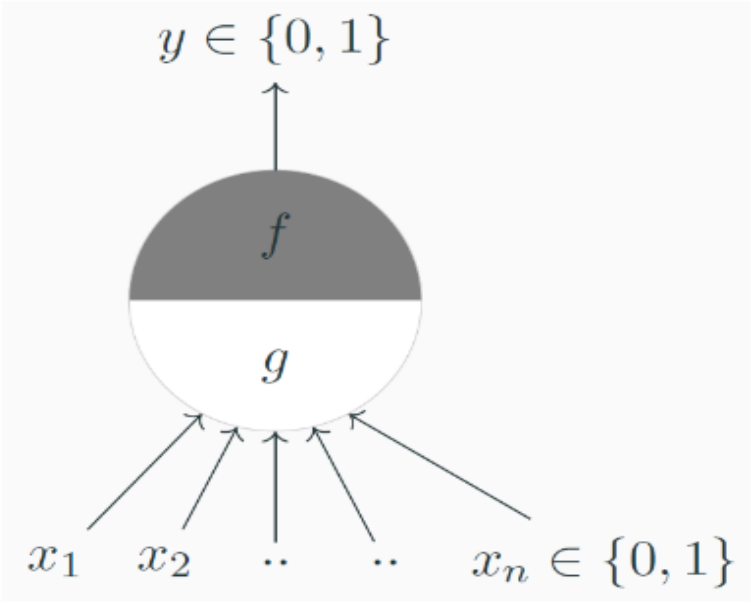




http://www.scholarpedia.org/article/Models_of_visual_cortex

Artificial Neuron

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
- Inputs can be excitatory or inhibitory
- $y = 0$ if any $x_i = 1$ is inhibitory else

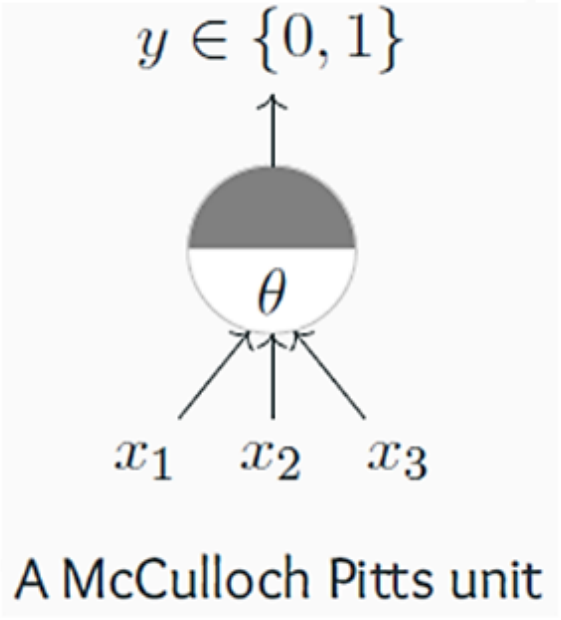
$$g(x_1, x_2, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$y = f(g(\mathbf{x})) = 1 \quad if \quad g(\mathbf{x}) \geq \theta$$
$$= 0 \quad if \quad g(\mathbf{x}) < \theta$$

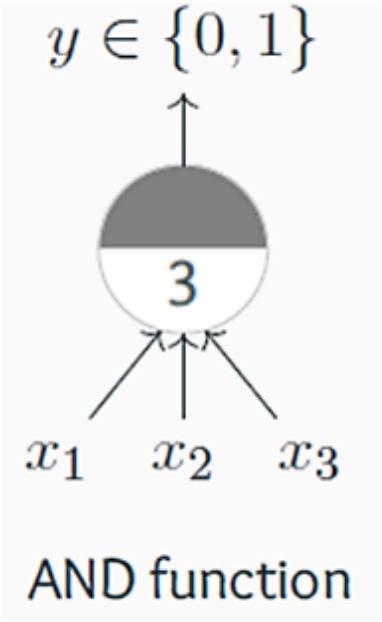
- θ is called thresholding parameter

Boolean Function

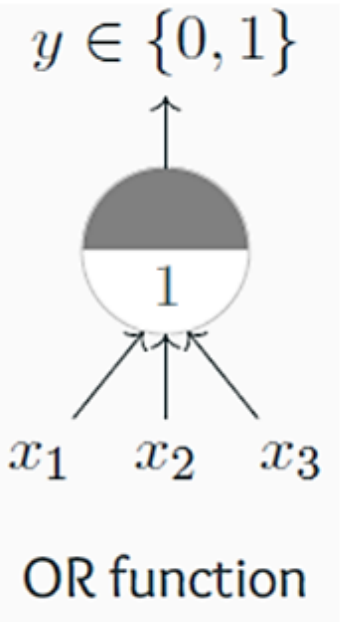
Learning Boolean Functions



A McCulloch Pitts unit



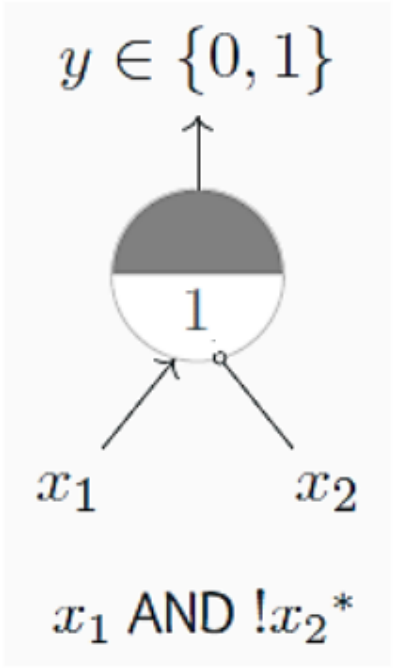
AND function



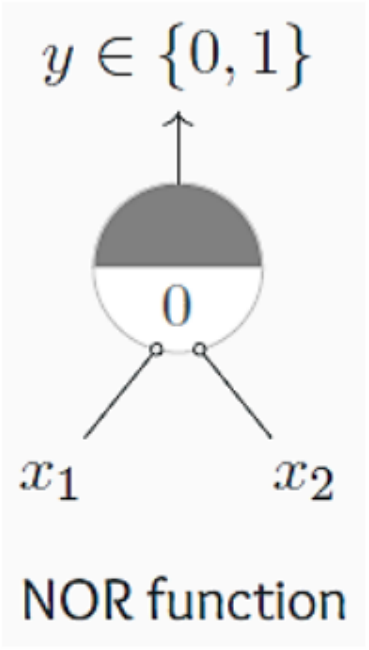
OR function

Boolean Function

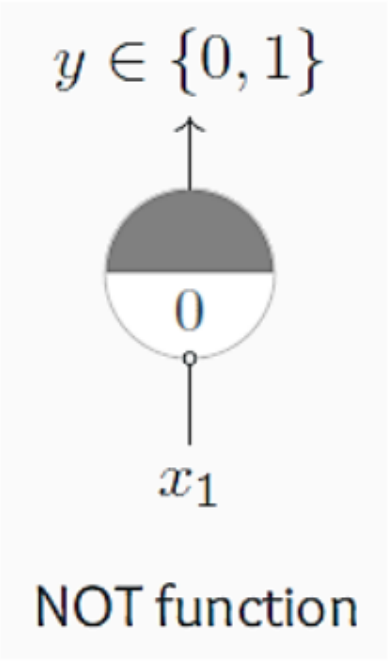
Learning Boolean Functions



x_1 AND $!x_2$*

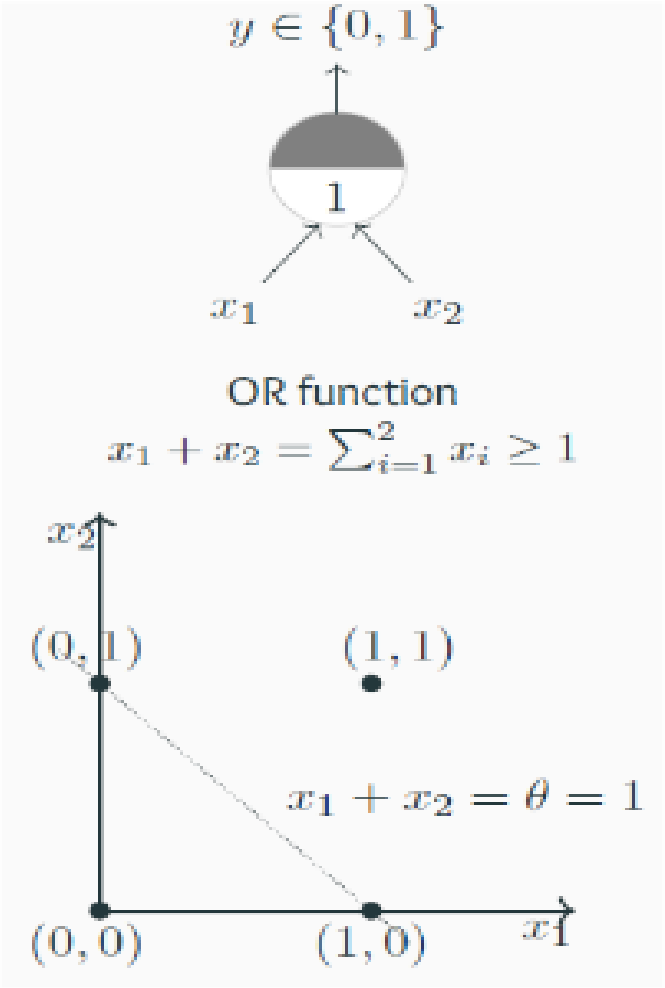


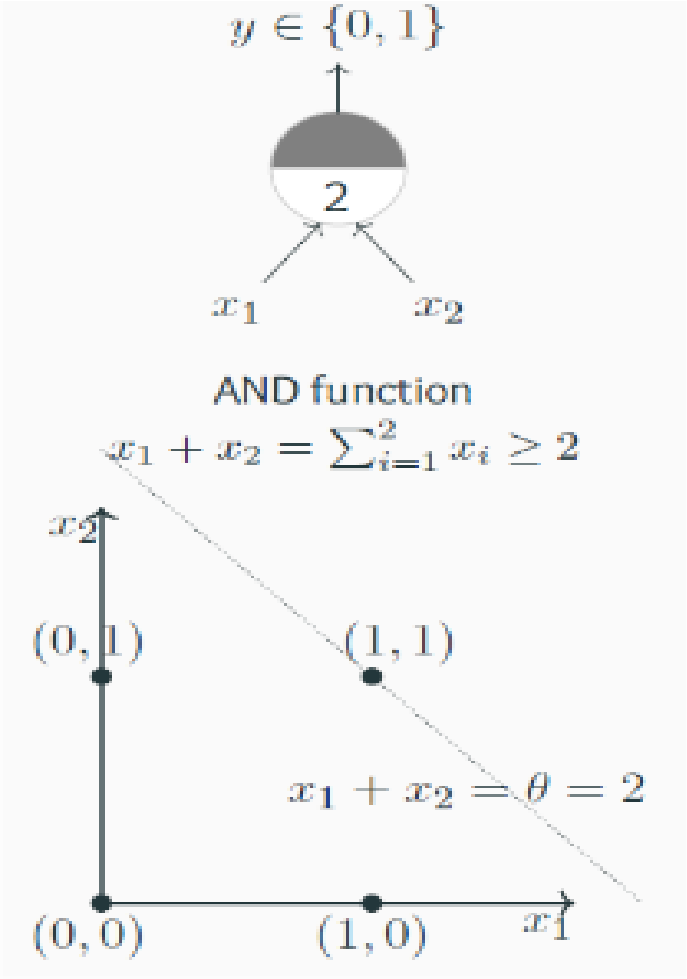
NOR function



NOT function

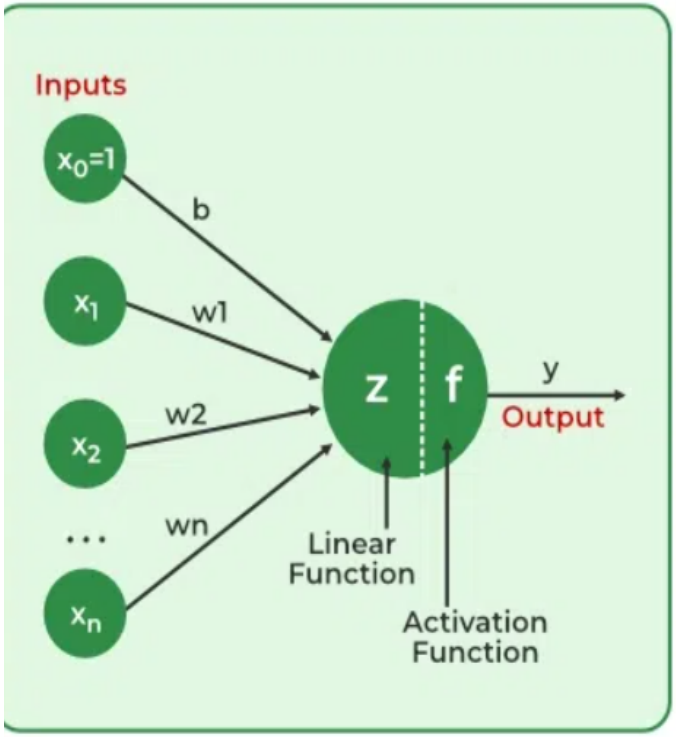
Geometric Interpretation





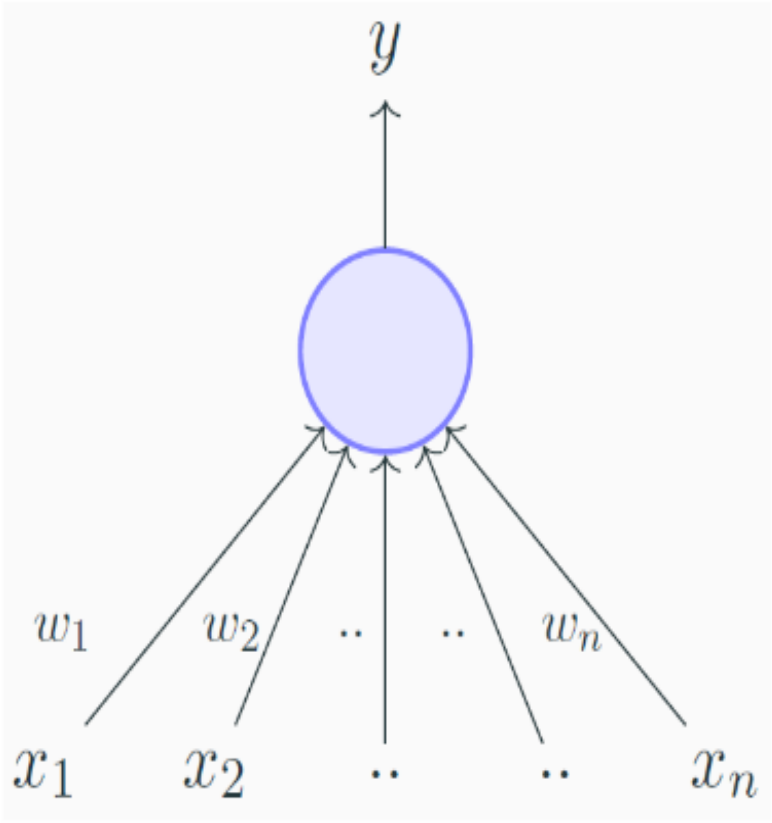
Perceptron

- 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



Perceptron

Perceptron



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

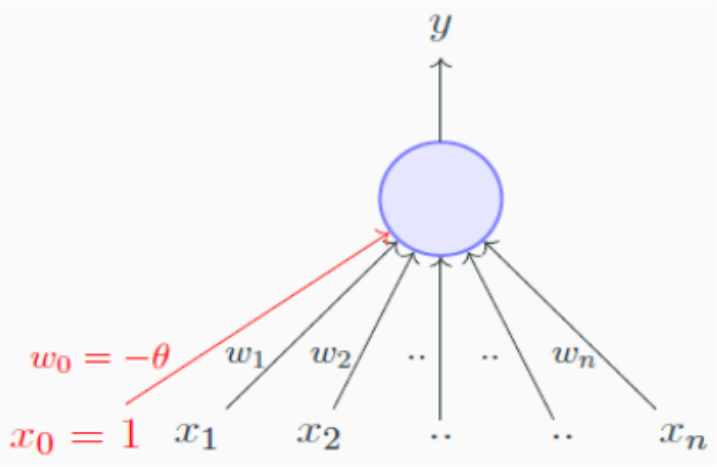
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

Rewriting the same

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

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

Perceptron



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

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

- Bias, b is used to adjust the output along with the weighted sum of the inputs to the neuron. It allows to shift the activation function to either left or right to fit the model better.
- High bias means the model is not fitting well on the training set => training error is large.
- High variance means model is not able to make accurate prediction on the validation set => validation error is large.

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$
$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$
$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

Learning Boolean Functions

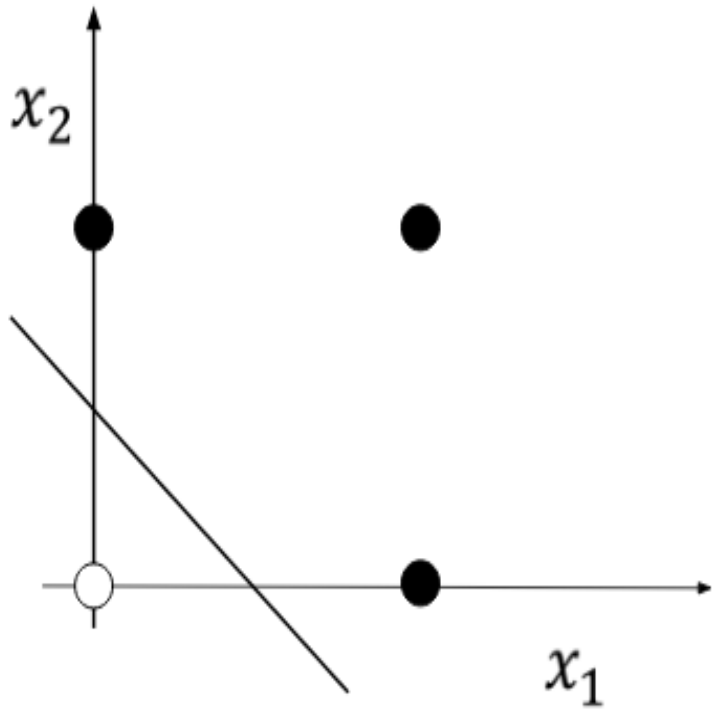
x_1	x_2	OR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$

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

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

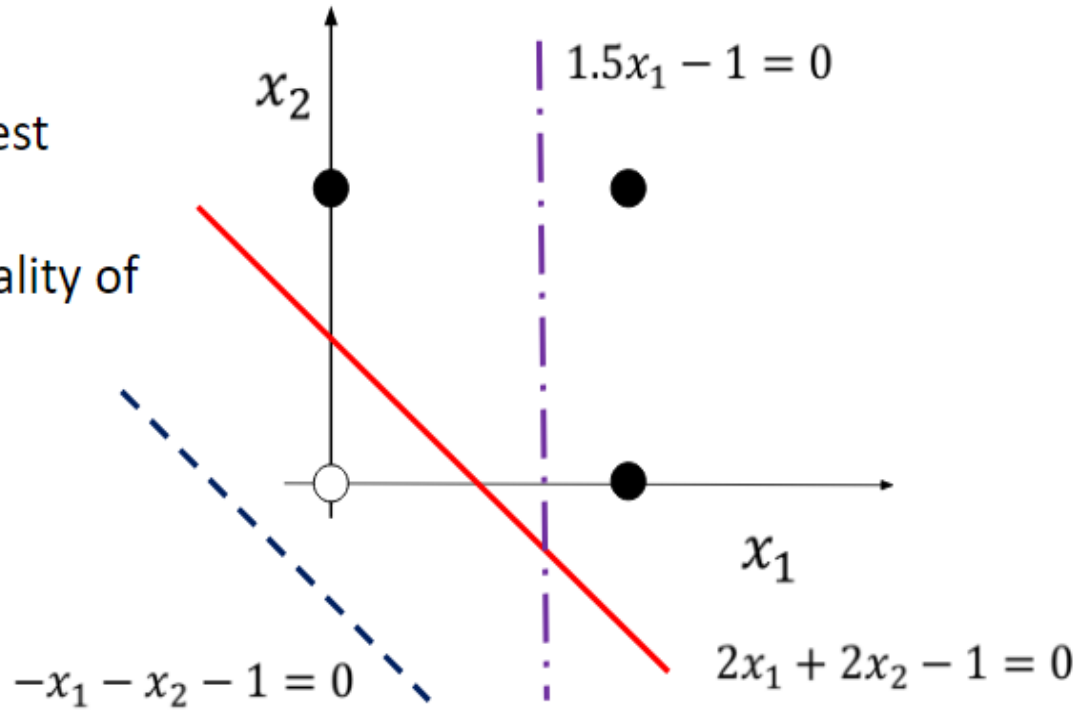
$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 \geq -w_0$$



Learning Boolean Function

Perceptrons

- Which one of these is best solution?
- How do we measure quality of solution?

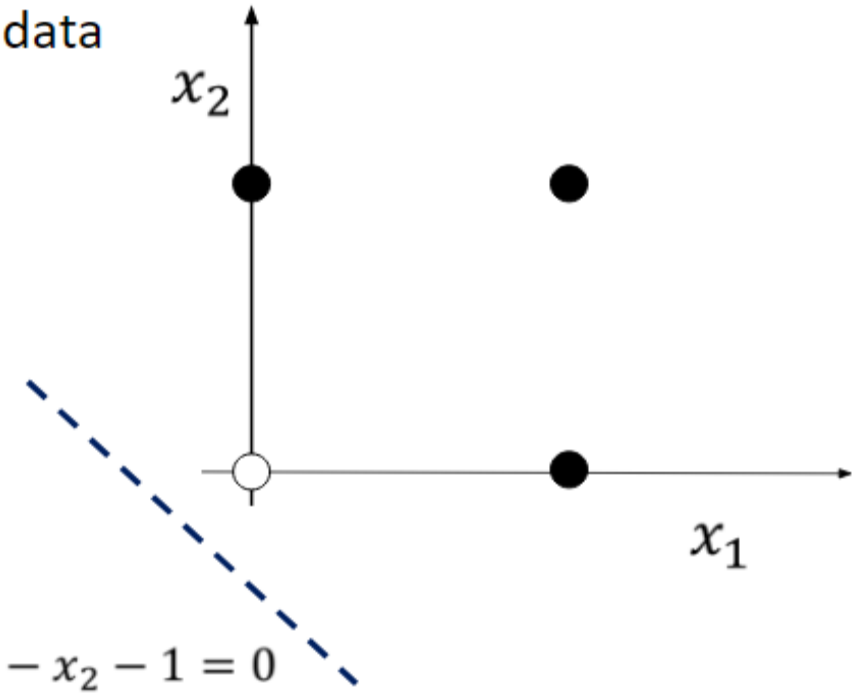


Learning Boolean Function

Error

- We measure error by counting the number of misclassified data
- Can we find errors for these solutions?

w_1	w_2	Error

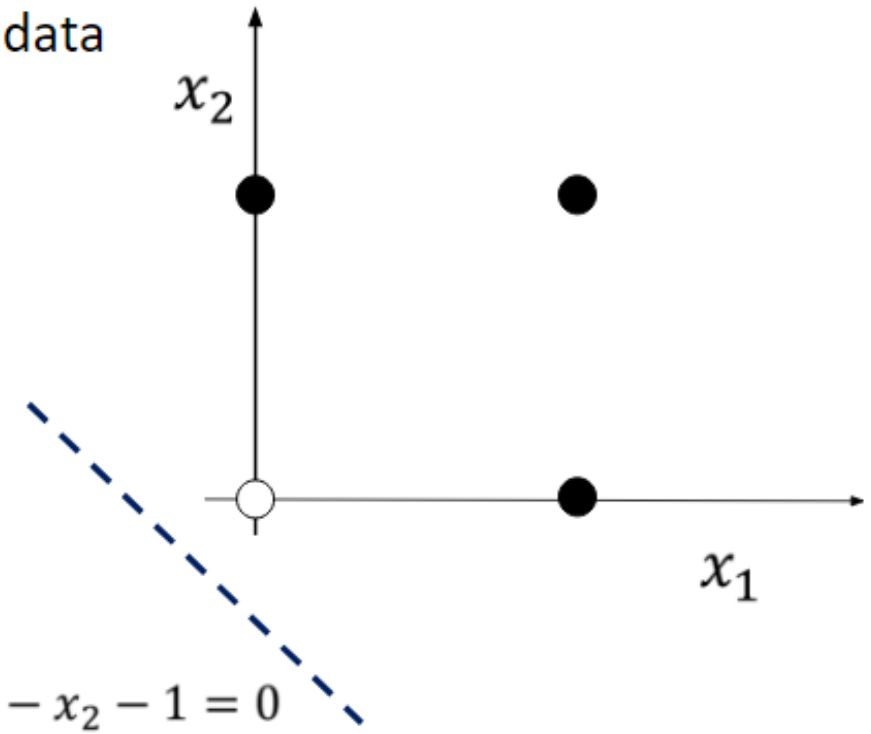


Learning Boolean Function

Error

- We measure error by counting the number of misclassified data
- Can we find errors for these solutions?

w_1	w_2	Error
-1	-1	3

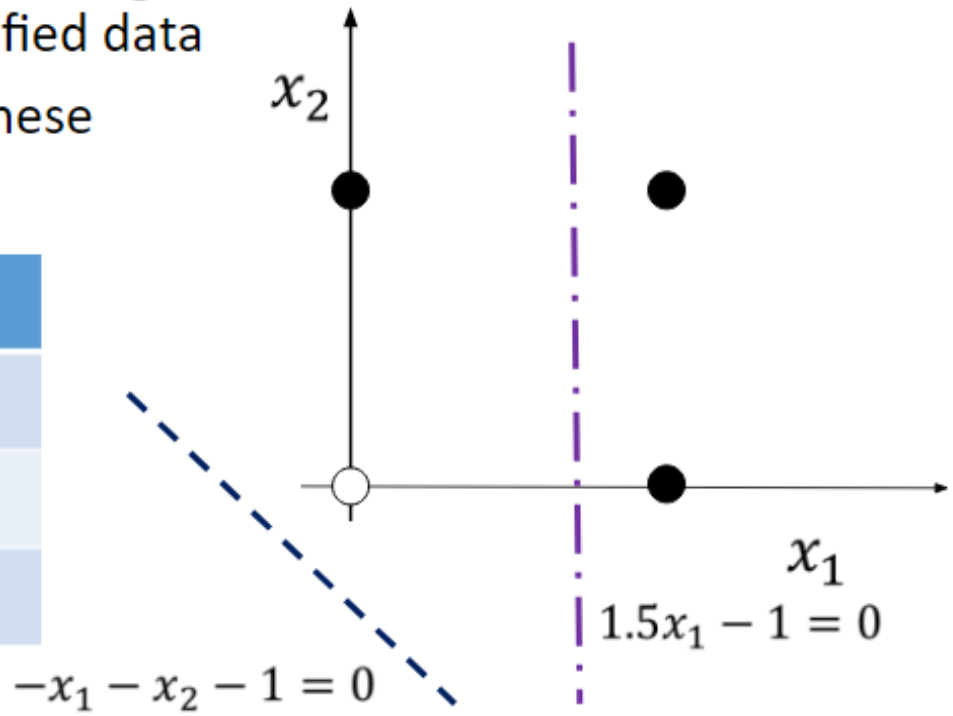


Learning Boolean Function

Error

- We measure error by counting the number of misclassified data
- Can we find errors for these solutions?

w_1	w_2	Error
-1	-1	3

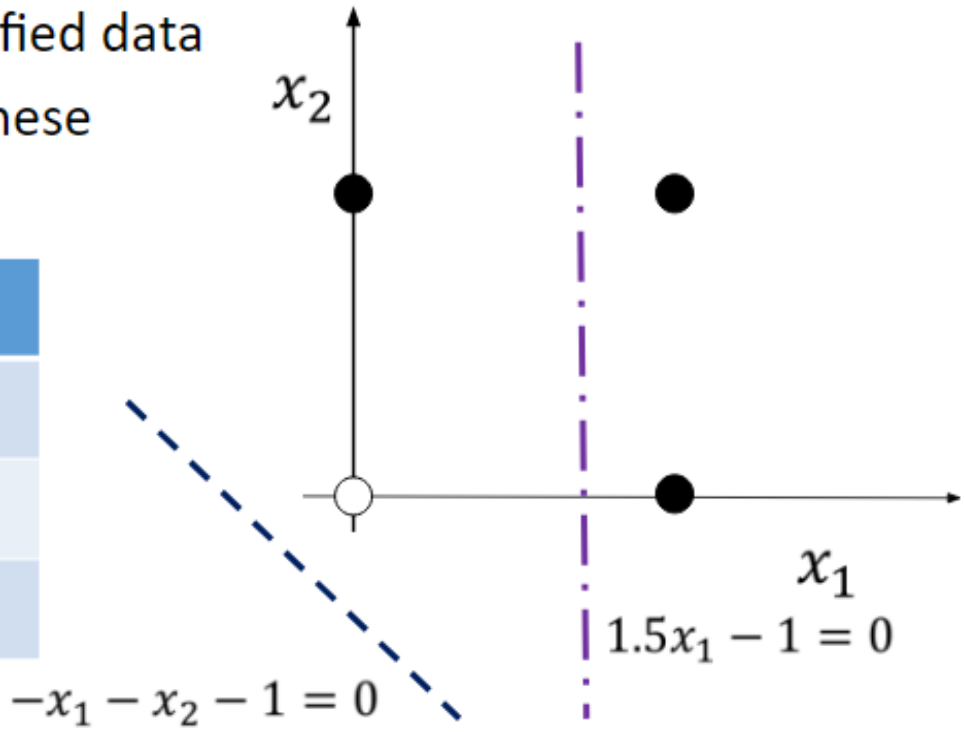


Learning Boolean Function

Error

- We measure error by counting the number of misclassified data
- Can we find errors for these solutions?

w_1	w_2	Error
-1	-1	3
1.5	0	1

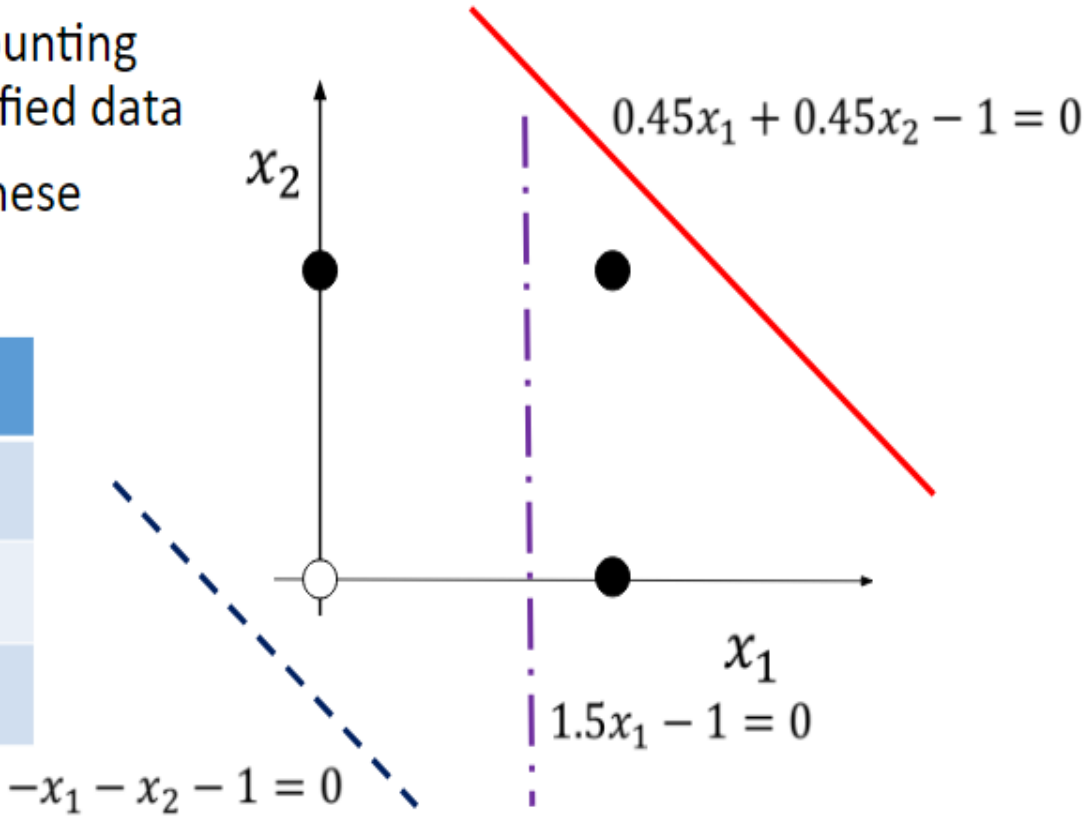


Learning Boolean Function

Error

- We measure error by counting the number of misclassified data
- Can we find errors for these solutions?

w_1	w_2	Error
-1	-1	3
1.5	0	1

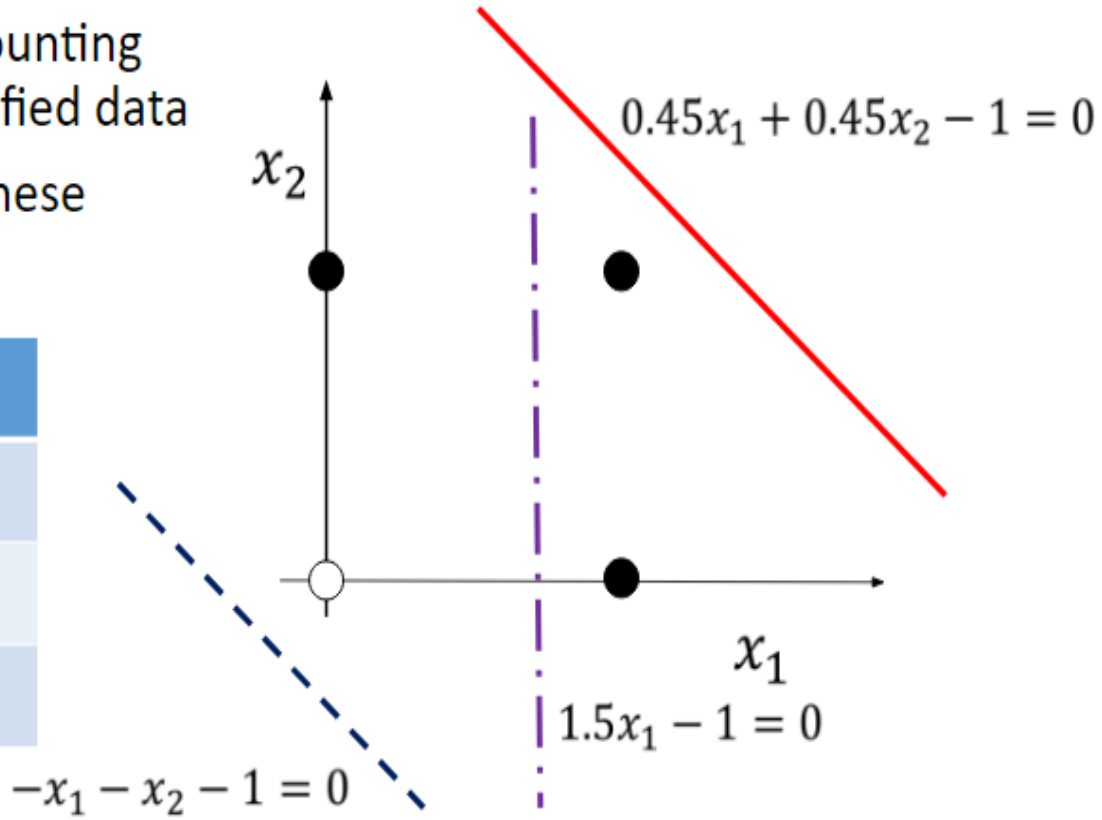


Learning Boolean Function

Error

- We measure error by counting the number of misclassified data
- Can we find errors for these solutions?

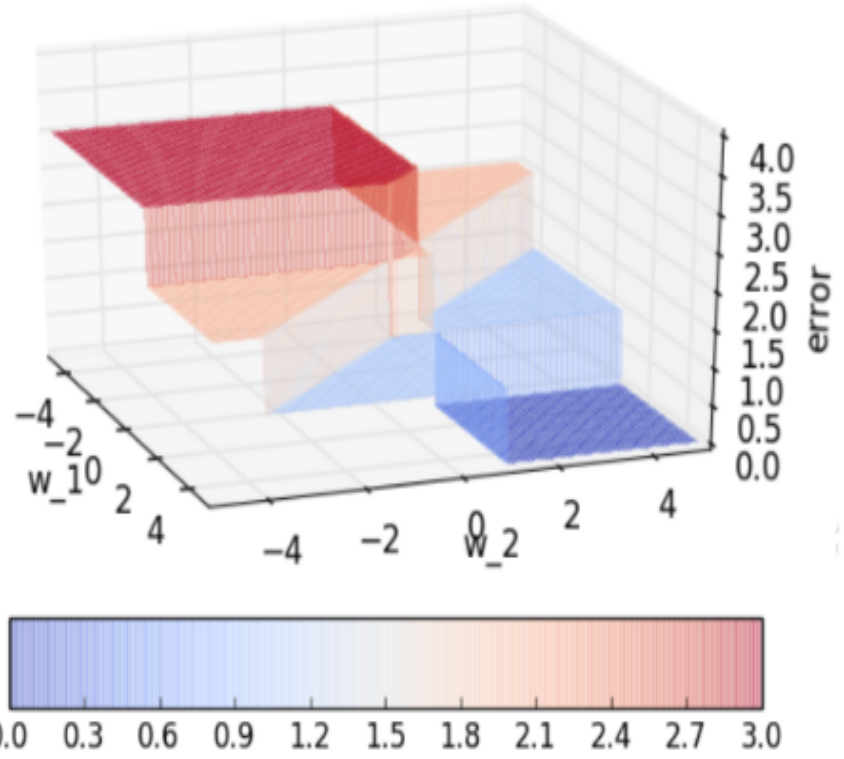
w_1	w_2	Error
-1	-1	3
1.5	0	1
0.45	0.45	3



Error Minimization Algorithm

Error Surface

- Lets fix $w_0 = -1$
- Plot errors for various values of w_1 and w_2
- We want to find parameter (w_1, w_2) values such that error is 0
- We want to find an algorithm that finds values of parameters such that error is minimized.



Implementation of AND gate using Neural Network

1. Problem Definition

An AND gate has two inputs and one output. The output is 1 if both inputs are 1; otherwise, the output is 0. The truth table for an AND gate is:

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

AND gate using Neural Network

2. Neural Network Structure

For this simple problem, a single-layer neural network (also known as a perceptron) is sufficient. The network will have:

- **2 Input Neurons**: Corresponding to Input 1 and Input 2.
- **1 Output Neuron**: Produces the final output.
- **Weights**: To be learned during training.
- **Bias**: To adjust the decision boundary.

AND gate using Neural Network

3. Network Architecture

- **Input Layer**: 2 neurons (one for each input).
- **Output Layer**: 1 neuron.

4. Activation Function

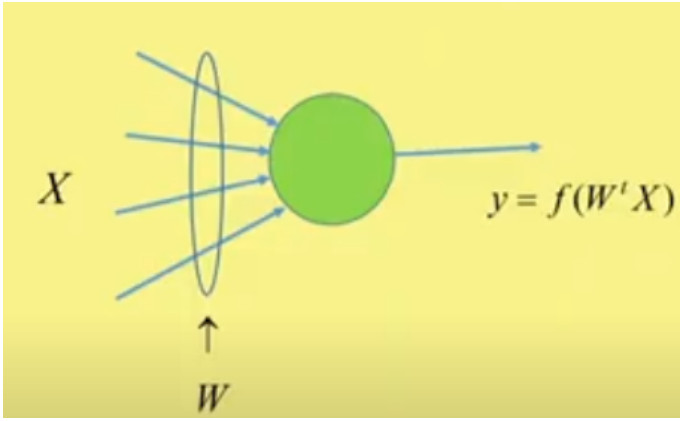
We use the step function (also known as the Heaviside step function) for the output neuron. This function outputs 1 if the input is greater than or equal to a certain threshold, and 0 otherwise.

AND gate using Neural Network

5. Mathematical Formulation

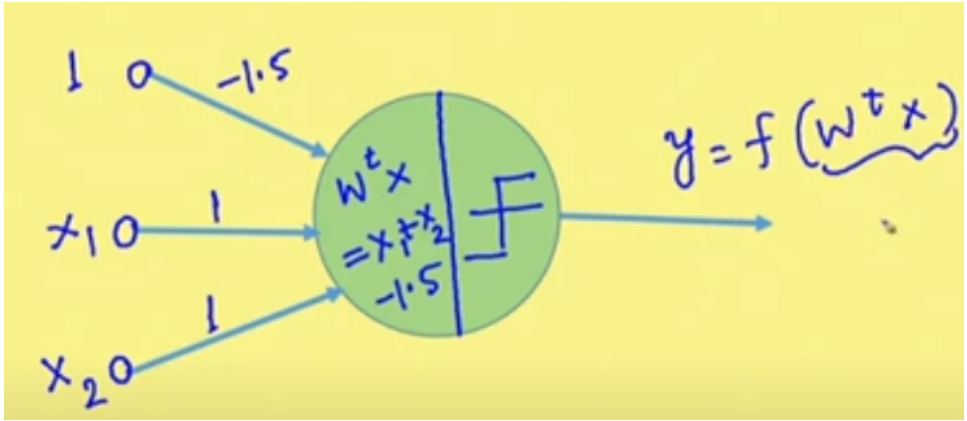
The output y of the neuron can be represented as:

$$y = \text{Step}(w_1 \cdot x_1 + w_2 \cdot x_2 + b)$$



Where:

- x_1 and x_2 are the inputs.
- w_1 and w_2 are the weights.
- b is the bias.



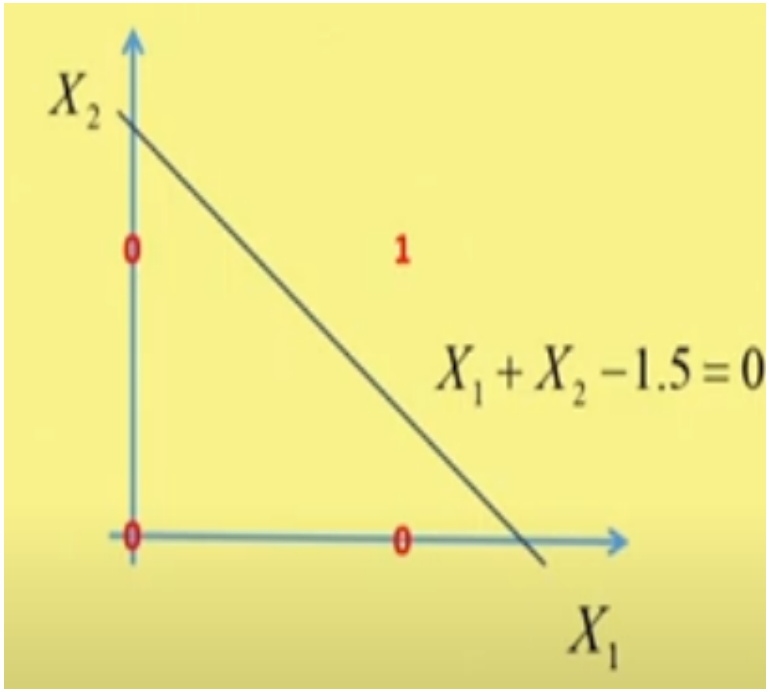
AND gate using Neural Network

- The step function can be defined as:
- If y =1 if value of step function is ≥ 0

  = 0 otherwise.
- Training For the AND gate, we need to find suitable values for w_1, w_2, and b such that the network's output matches the AND gate's truth table.

$$W = \begin{bmatrix} b \\ w_1 \\ w_2 \end{bmatrix} \qquad X = \begin{bmatrix} 1 \\ X_1 \\ X_2 \end{bmatrix}$$

After training: $W = \begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix}$



Matrix for of AND gate implementation

- Including the bias, the feature vectors are represented by matrix X, and weight vector is W

$$W = \begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix} \qquad X^{t} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$X'W = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix} \Rightarrow \Rightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

To capture Nonlinearity, use step function

Implementation of OR gate using Neural Network

1. Problem Definition

The OR gate truth table is as follows:

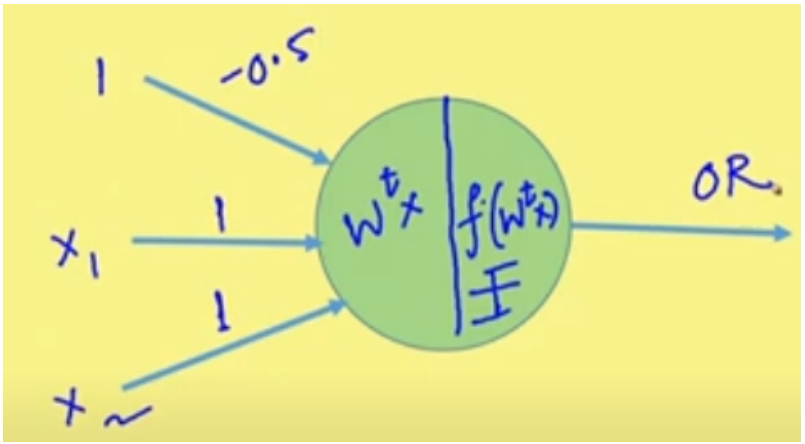
Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

OR gate using Neural Network

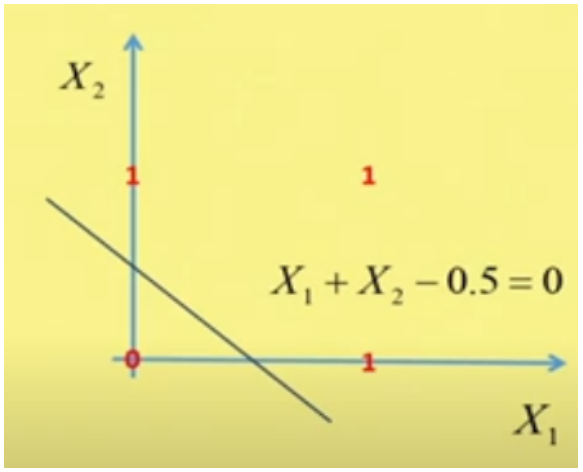
2. Neural Network Structure

A single-layer neural network (perceptron) is sufficient for this task. The network will have:

- **2 Input Neurons**: One for each input.
- **1 Output Neuron**: To produce the output.
- **Weights**: To be learned.
- **Bias**: To adjust the decision boundary.

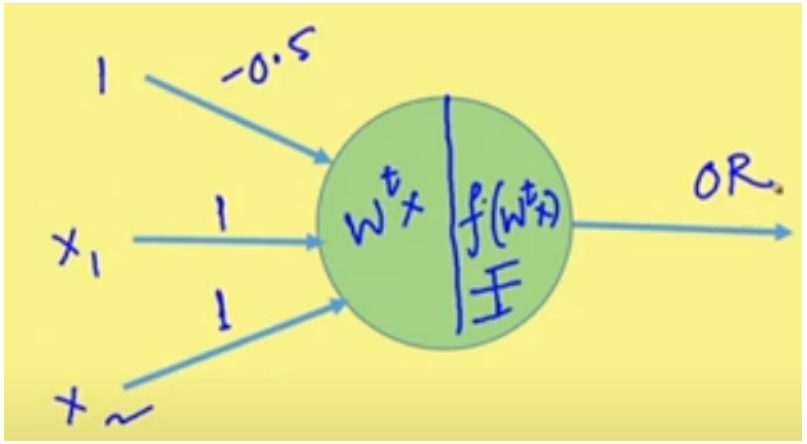


- Here, $W = \begin{bmatrix} b \\ w_1 \\ w_2 \end{bmatrix}$, $X = \begin{bmatrix} 1 \\ X_1 \\ X_2 \end{bmatrix}$ and after training : $W = \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix}$



OR Gate Implementation

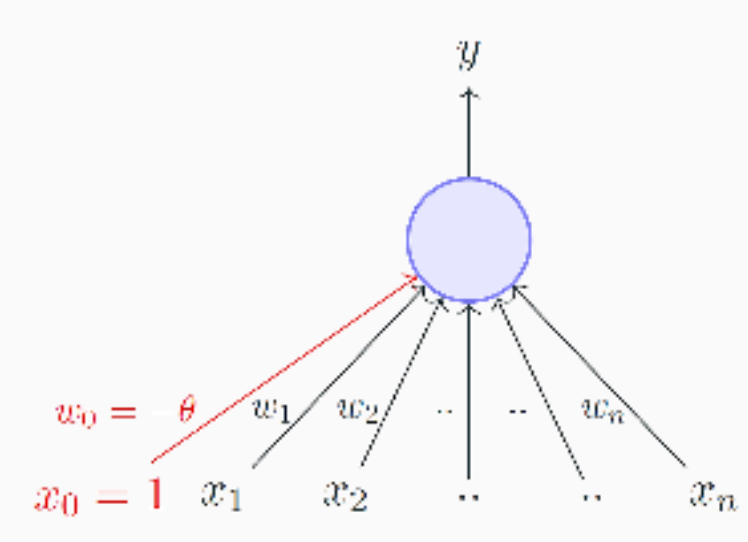
$$X'W = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0.5 \\ 0.5 \\ 1.5 \end{bmatrix} \Rightarrow \Rightarrow \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$



Perceptron Learning Algorithm

- Both AND and OR gates are implemented by a single neuron or perceptron, because the input features of two different classes are linearly separable.
- Here, output layer is the only layer and the information pass in forward direction only, i.e., from inputs (also called Input layer) to the output layer.
- If there are multiple classes (i.e., more than two) than there will be m neurons in the output layer to classify m classes. Here also information pass in forward direction only, i.e., from input layer to the output layer.
- This network is called Single Layer Feed Forward Neural Network (SLFFNN).
- We estimate the parameters of the perceptron by using perceptron learning algorithm.

Perceptron Learning Algorithm



- If there are 10 neurons in the SLFFNN, we need to iterate the algorithm 10 times.

Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

while $!convergence$ **do**

  Pick random $\mathbf{x} \in P \cup N$;

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

    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;

  end

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

    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;

  end

end

//the algorithm converges when all the inputs are classified correctly

Perceptron Learning contd.

Considering two vectors $\boldsymbol{w}$ and $\boldsymbol{x}$

$$\boldsymbol{w} = [w_0, w_1, w_2, \dots, w_n]$$

$$\boldsymbol{x} = [1, x_1, x_2, \dots, x_n]$$

$$\boldsymbol{w} \cdot \boldsymbol{x} = \boldsymbol{w}^T \boldsymbol{x} = \sum_{i=0}^{n} w_i \times x_i$$

$$y = 1 \; if \; \boldsymbol{w}^T \boldsymbol{x} \geq \boldsymbol{0}$$
$$= 0 \; if \; \boldsymbol{w}^T \boldsymbol{x} < \boldsymbol{0}$$

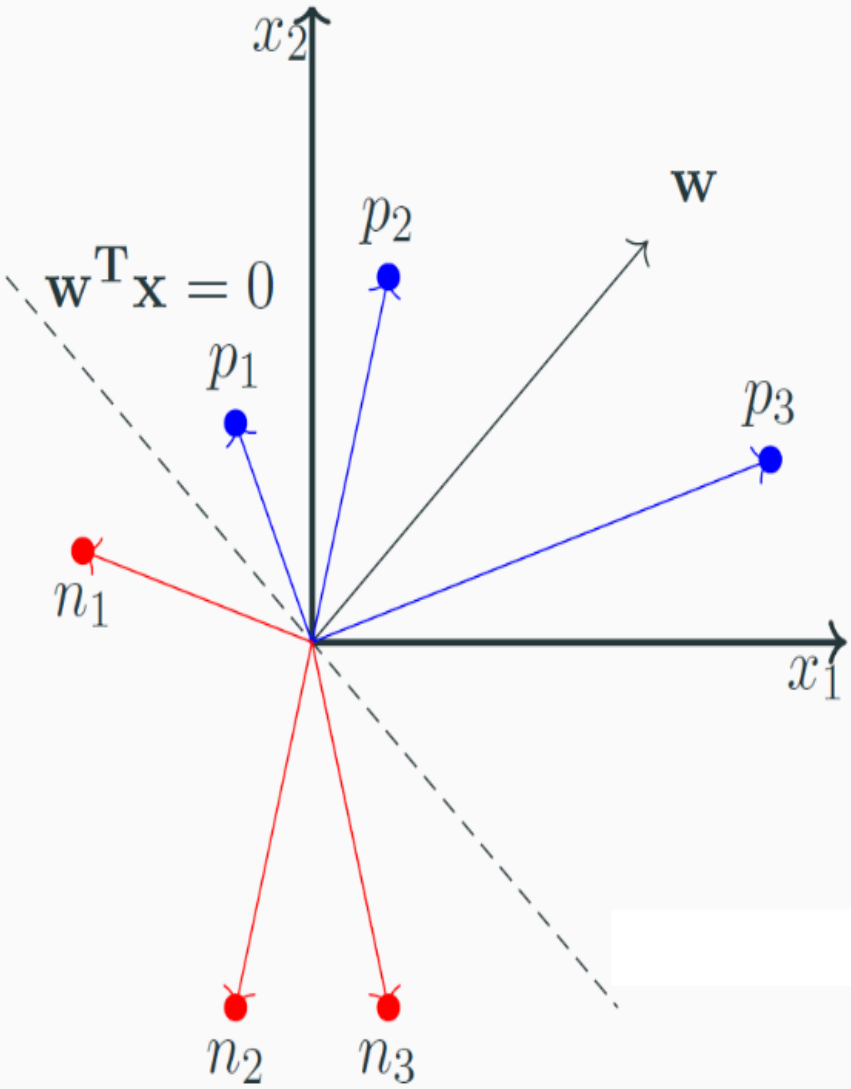
- The algorithm needs to find the line $\boldsymbol{w}^T \boldsymbol{x} = 0$ which divides the space into two regions
- What is the angle (α) between $\boldsymbol{w}$ and data point $\boldsymbol{x}$ which lies on that line?
- The angle is 90°

$$(cos\alpha = \frac{\boldsymbol{w}^T \boldsymbol{x}}{\|w\| \|x\|} = 0)$$

Thus the vector $\boldsymbol{w}$ is perpendicular to all points on the line. Hence, perpendicular to the line itself.

Perceptron Learning contd.

- Consider some points (vectors) which lie in the positive half space of this line ($\boldsymbol{w}^T\boldsymbol{x} \geq \mathbf{0}$)
- What will be the angle between any such vector and $\boldsymbol{w}$?
  - Less than 90°
- What about points (vectors) which lie in the negative half space of this line ($\boldsymbol{w}^T\boldsymbol{x} < \mathbf{0}$)
  - Greater than 90°



Perceptron Learning Algorithm

Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

while $!convergence$ **do**

  Pick random $\mathbf{x} \in P \cup N$;

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

    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;

  end

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

    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;

  end

end

//the algorithm converges when all the inputs are classified correctly

- If $x \in P$ and $\boldsymbol{w} \cdot \boldsymbol{x} < 0$ then
  - The angle (α) between $\boldsymbol{x}$ and current $\boldsymbol{w}$ is greater $90°$
  - However, expected value of α is less than $90°$
- Lets inspect the new angle (α_{new}) when $\boldsymbol{w_{new}} = \boldsymbol{w} + \boldsymbol{x}$

$$\cos(\alpha_{new}) \propto \boldsymbol{w_{new}}^T \boldsymbol{x}$$
$$\cos(\alpha_{new}) \propto (\boldsymbol{w} + \boldsymbol{x})^T \boldsymbol{x}$$
$$\cos(\alpha_{new}) \propto \boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{x}^T \boldsymbol{x}$$
$$\cos(\alpha_{new}) \propto cos\alpha + \boldsymbol{x}^T \boldsymbol{x}$$
$$\cos(\alpha_{new}) > cos\alpha$$
$$\alpha_{new} < \alpha$$

Perceptron Learning Algorithm

Perceptron Learning Algorithm Convergence Proof

- If $x \in N$ then $-x \in P$
- So, $\boldsymbol{w}^T \boldsymbol{x} < 0 \Rightarrow \boldsymbol{w}^T(-\boldsymbol{x}) \geq 0$
- Consider a single set of data points $P' = P \cup N'$
- Thus, for every p in P', $\boldsymbol{w}^T \boldsymbol{p} \geq 0$
- We normalize all data points (p) and consider normalized solution vector as $\boldsymbol{w}^*$

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
N^- contains negations of all points in N;
$P' \leftarrow P \cup N^-;$
Initialize w randomly;
while !$convergence$ **do**
  Pick random $\mathbf{p} \in P'$;
  $\mathbf{p} \leftarrow \frac{p}{||p||} \quad (so\ now, ||p|| = 1)$;
  if $\mathbf{w}.\mathbf{p} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{p}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

Perceptron Learning Algorithm

Perceptron Learning Algorithm Convergence Proof

- We want to find optimal solution w^*
- At any time step / iteration, we make correction iff $w^T p_i < 0$
- So, at time step t we can make k $(k \leq t)$ many corrections only.

Now suppose at time step t we inspected the point p_i and found that $w^T p_i < 0$

We update $w_{t+1} = w_t + p_i$

Let β be the angle between w^* and w_{t+1}

$cos\beta = \frac{w^* \cdot w_{t+1}}{\|w_{t+1}\|}$ (as, $\|w^*\| = 1$)

Numerator = $w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i)$

$= w^* \cdot w_t + w^* \cdot p_i$

$\geq w^* \cdot w_t + \delta$ $(\delta = \min(w^* \cdot p_i) \forall i)$

$\geq w^* \cdot (w_{t-1} + p_j) + \delta$

$\geq w^* \cdot w_{t-1} + w^* \cdot p_j + \delta$

$\geq w^* \cdot w_{t-1} + 2\delta$

$\geq w^* \cdot w_0 + k\delta$ (by Induction)

Perceptron Learning Algorithm

Perceptron Learning Algorithm Convergence Proof

We have, $cos\beta = \frac{w^* \cdot w_{t+1}}{\|w_{t+1}\|}$

Numerator $\geq w^* \cdot w_0 + k\delta$

Denominator^2 = $\|w_{t+1}\|^2$

$= (w_t + p_i) \cdot (w_t + p_i)$

$= \|w_t\|^2 + 2w_t \cdot p_i + \|p_i\|^2$

$\leq \|w_t\|^2 + \|p_i\|^2$ (as, $w_t \cdot p_i < 0$)

$\leq \|w_t\|^2 + 1$ (as, $\|p_i\| = 1$)

$\leq \|w_{t-1}\|^2 + 1 + 1$

$\leq \|w_{t-1}\|^2 + 2$

$\leq \|w_0\|^2 + k$ (By induction)

Perceptron Learning Algorithm Convergence Proof

We have, $cos\beta = \frac{w^* \cdot w_{t+1}}{\|w_{t+1}\|}$

Numerator $\geq w^* \cdot w_0 + k\delta$

Denominator$^2 \leq \|w_0\|^2 + k$

Hence, $cos\beta \geq \frac{w^* \cdot w_0 + k\delta}{\sqrt{\|w_0\|^2 + k}}$

Thus, $cos\beta$ grows proportional to $\sqrt{k}$

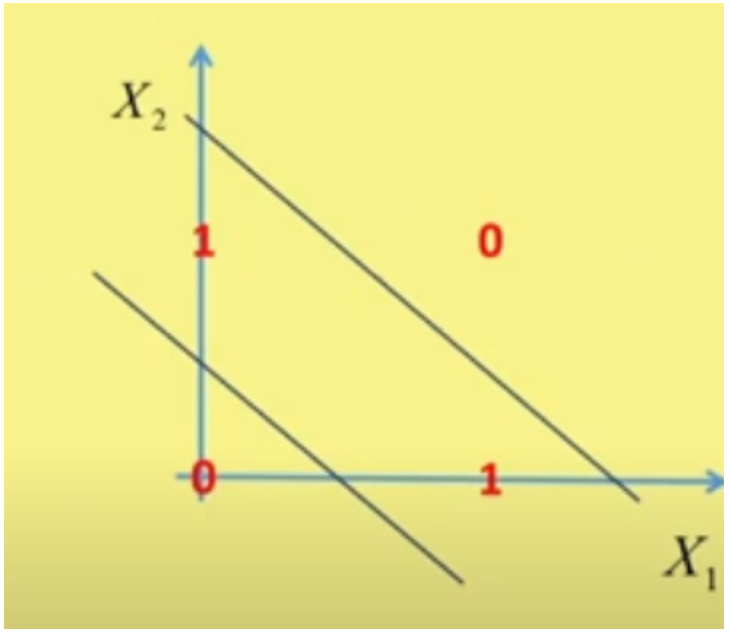
But, $cos\beta \leq 1$, hence k must be bounded by a maximum number

So, there can only be finite number of updates (iterations) k and the algorithm will converge.

Implementation of XOR gate using Neural Network

- The outputs of AND and OR gates are linearly separable. So single layer perceptron can implement them.
- But the outputs of XOR is not linearly separable.
- This means a single-layer perceptron cannot solve it.
- Instead, we need a neural network with at least one hidden layer to solve the XOR problem.

X_1	X_2	y
0	0	0
0	1	1
1	0	1
1	1	0



XOR gate using Neural Network

1. Neural Network Structure

To solve the XOR problem, we can use a neural network with:

- **2 Input Neurons**: For the two input values.
- **1 Hidden Layer**: Typically with 2 neurons, which can capture the XOR pattern.
- **1 Output Neuron**: To produce the final output.

2. Activation Functions

- **Hidden Layer**: Use a nonlinear activation function like the sigmoid or ReLU.
- **Output Layer**: Use the sigmoid function to get a probability-like output between 0 and 1.

XOR gate using Neural Network

3. Mathematical Formulation

- For a neural network with one hidden layer, the computations are as follows:

i) Hidden Layer:

- Compute the weighted sum for each hidden neuron.
- Apply the activation function (e.g., sigmoid, or ReLU) to capture non-linearity.

ii) Output Layer:

- Compute the weighted sum from hidden layer outputs.
- Apply the activation function (e.g., sigmoid).

XOR gate using Neural Network

- **Loss Function:** Binary crossentropy may be used for binary classification.
- **Optimizer:** Stochastic Gradient Descent (SGD) may be used to optimize the weights.
- **Training**:
  - **Epochs**: Number of iterations over the entire dataset.
- **Evaluation and Prediction**:
  - After training, the model's accuracy is evaluated, and predictions are made on the XOR inputs.

- This simple neural network will be able to learn the XOR function effectively, demonstrating the power of neural networks to solve problems that are not linearly separable.

XOR Gate

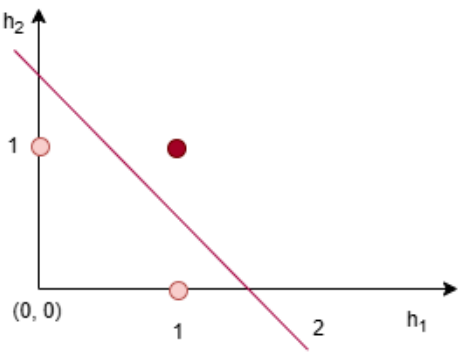
$$X_1 \oplus X_2 = (X_1 + X_2).(\overline{X}_1 + \overline{X}_2)$$

X_1	X_2	$h_1 = X_1 + X_2$	$h_2 = \overline{X}_1 + \overline{X}_2$	$h_1.h_2 = X_1 \oplus X_2$
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

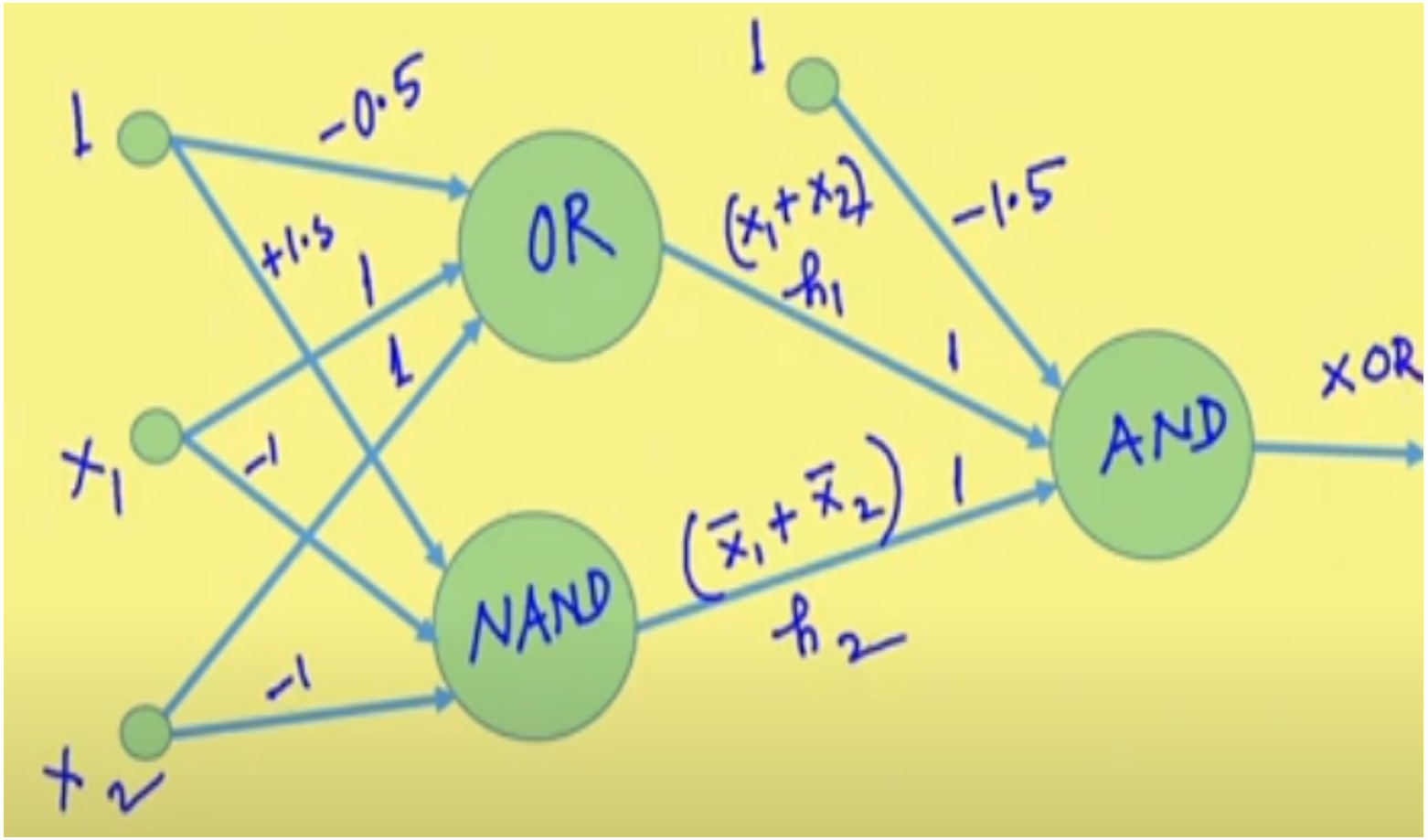
- We compute OR operation and NAND operation on X_1 and X_2, and finally AND operation on h_1 and h_2 gives the XOR operation.
- As for AND operation, the weight vector is [-1.5, 1,1] so for NAND gate it is: [1.5, -1, -1].

XOR Gate

- So, two operations OR and NAND transform the input vectors to an intermediate vectors.
- Initial feature space was 2-D in form of $[X_1, X_2]$ and the intermediate feature space is $[h_1, h_2]$.
- In intermediate feature space, the feature vectors are linearly separable.
- The XOR gate is implemented by the AND operation of h_1 h_2.



XOR gate using Neural Network



XOR Gate

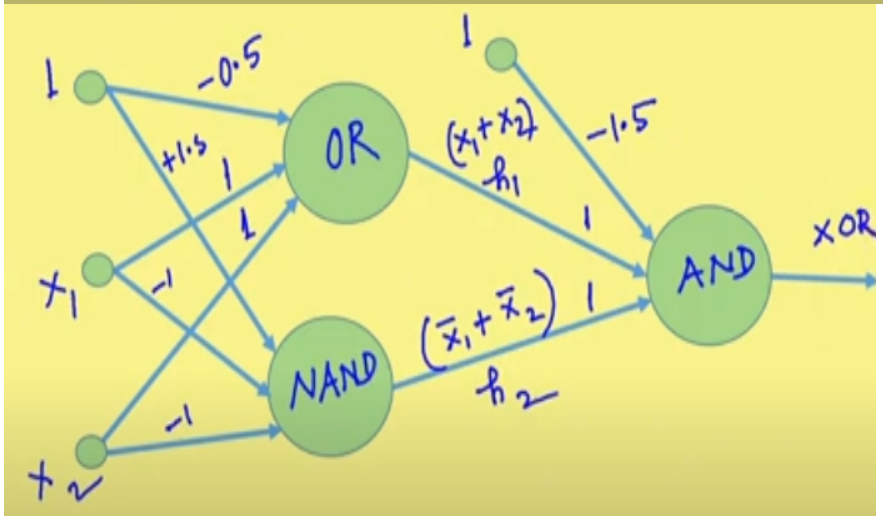
$$\begin{bmatrix} -0.5 & 1 & 1 \\ 1.5 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.5 & 0.5 & 0.5 & 1.5 \\ 1.5 & 0.5 & 0.5 & -0.5 \end{bmatrix} \Rightarrow \text{step} \Rightarrow \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

W_1^t X h

$$h^t W_2 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0.5 \\ 0.5 \\ -0.5 \end{bmatrix} \Rightarrow \text{step} \Rightarrow \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$X_1 \oplus X_2$

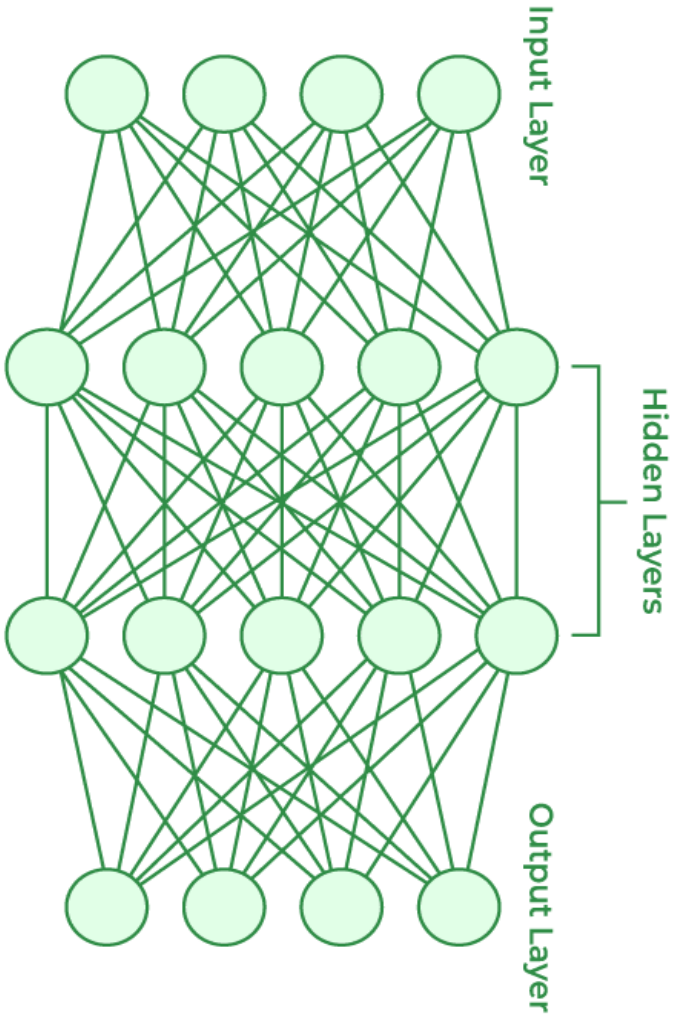
- It is an example of Multi Layer Feed Forward Neural Network.



- For XOR, we need 3 neurons of which two neurons are at hidden layer and one neuron at output layer.
- One neuron at hidden layer computes OR operation and other computes NAND operation.

Some key points of neural networks

- **Structure**: A neural network consists of layers of interconnected nodes (or neurons).
- These layers include:
  - **Input Layer**: Receives the initial data.
  - **Hidden Layers**: Perform intermediate processing. There can be one or more hidden layers.
  - **Output Layer**: Produces the final prediction or classification.
- **Weights and Biases**: Each connection between nodes has an associated weight that is adjusted during training. Biases are added to the weighted sum of inputs to help the model fit the data better.
- **Activation Functions**: These functions determine if a neuron should be activated or not. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh.



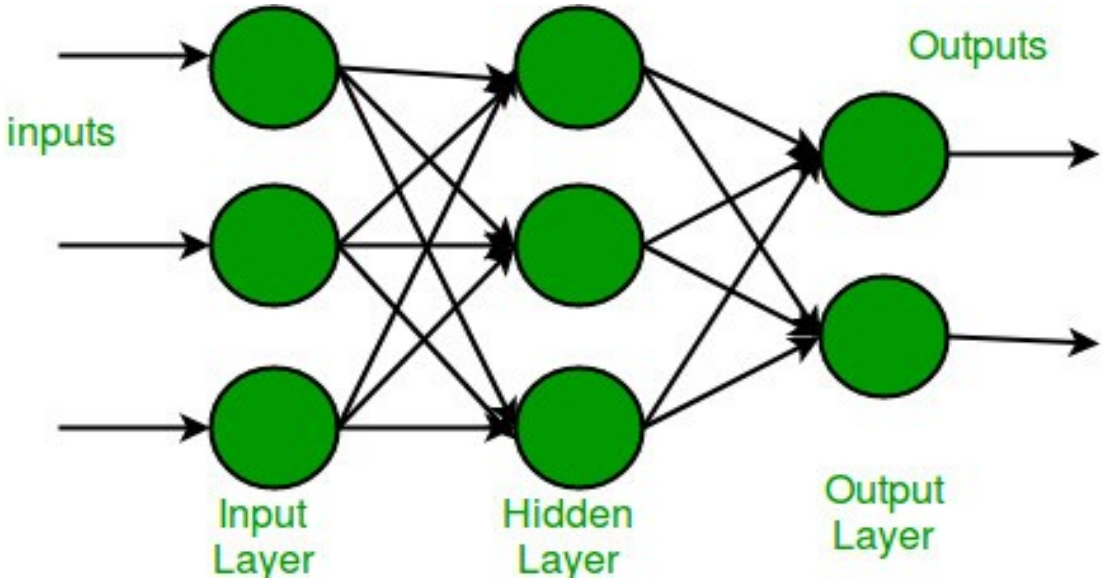
Some key points of neural networks

- **Training**: Neural networks learn by adjusting weights and biases based on the errors of their predictions. This process is often done using algorithms like backpropagation combined with optimization techniques like gradient descent.
- **Applications**: Neural networks are used in various applications such as image and speech recognition, natural language processing, and game playing. They are especially powerful in tasks where patterns and relationships in data are complex and not easily defined by traditional algorithms.

Some key points of neural networks

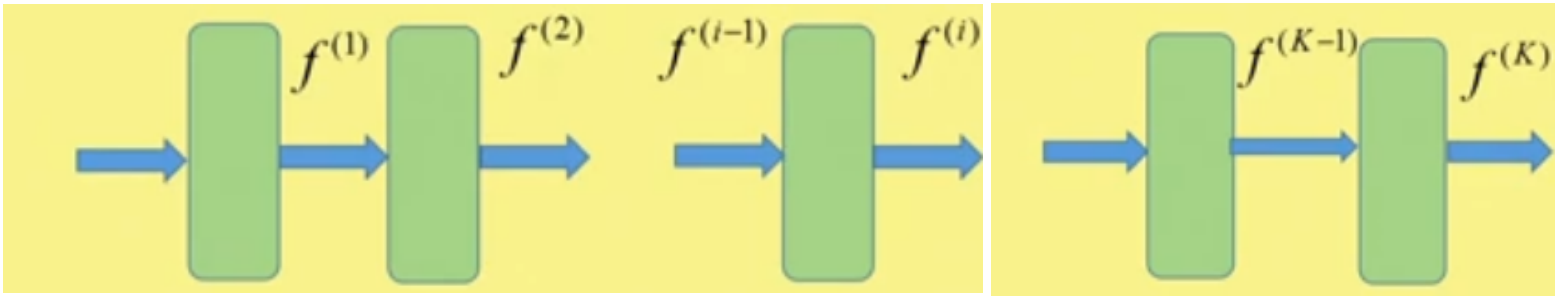
- **Types**: There are different types of neural networks tailored to specific tasks:
  - **Feedforward Neural Networks**: The simplest type, where information moves in one direction—from input to output.
  - **Convolutional Neural Networks (CNNs)**: Effective for image processing tasks.
  - **Recurrent Neural Networks (RNNs)**: Suitable for sequential data like time series or text.
- Neural networks have become a foundational technology in modern AI, enabling many of the advanced applications we see today.

Multi Layer Feed Forward Neural Network



- Here, for 2 classes, we have two neurons at the output layer. For m-class problem, the output layer consists of m number of neurons.
- There will be at least one hidden layer.

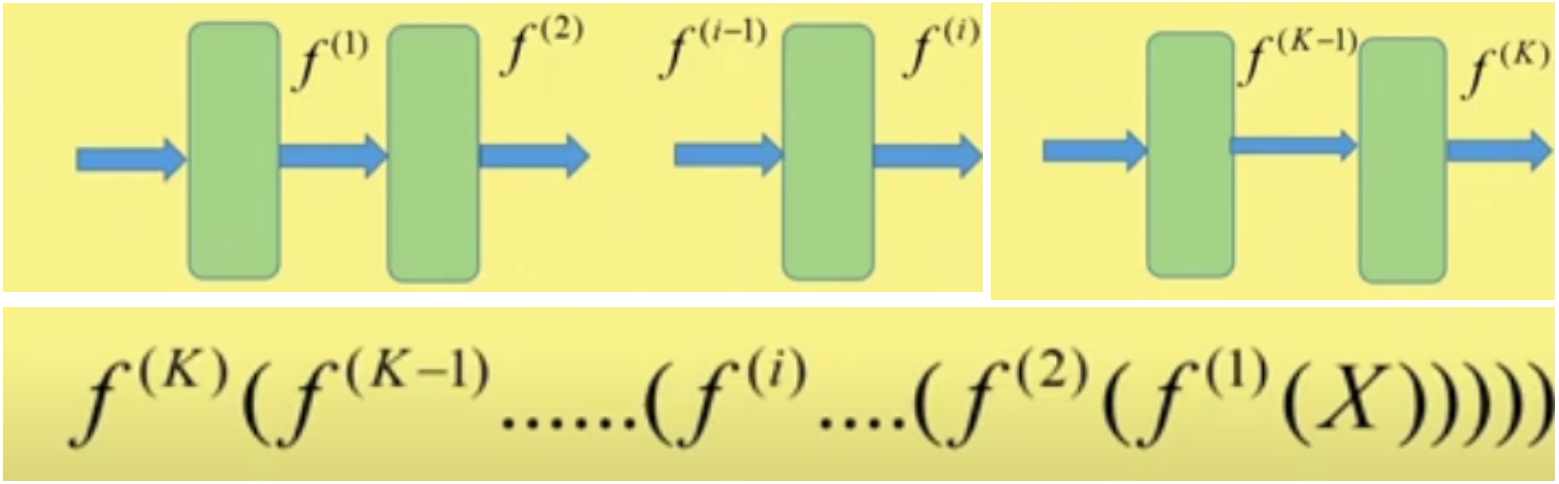
Neural Network Function



$$f^{(K)}(f^{(K-1)}.....(f^{(i)}....(f^{(2)}(f^{(1)}(X)))))$$

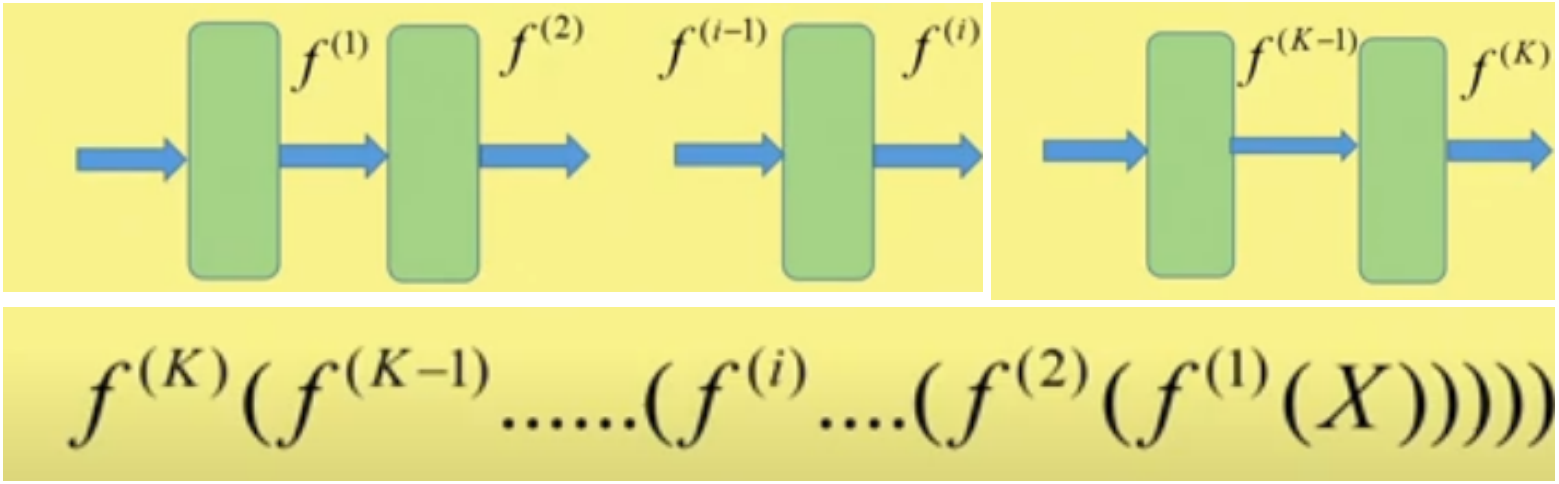
- Each layer of the neural network computes a non-linear function of the input feature vector inputted to that particular layer of neurons.
- The output of that layer of neurons in an intermediate feature vector (may be of same or different dimension depending on how many neurons we have considered).
- In every layer, the neurons compute a non-linear function.

Neural Network Function



- The i-th layer neurons computes a non-linear function, say f^i on the feature vectors which are computed by the previous layer given by the function f^{i-1} and each of these are non-linear function.
- If input vector is X, the first layer computes $f^1(X)$, which is inputted to the next layer. It computes $f^2(f^1(X))$, and this way it continues and finally, we get output f^k.

Neural Network Function



- Thus the input feature vector is non-linearly mapped to intermediate feature vector, which is again non-linearly mapped to another representation, and so on. And finally, the last representation of the feature vectors are separable by the linear classifier.
- This is the working principle of multi layer feed forward neural network. It is feed forward because information is always flowing from input to output. The information does not flow in the reverse direction.

- K indicates the number of hidden layers. If K is very high then the neural network is called Deep neural Network.

Non-linear Classifier

- To implement a non-linear classifier, we try to map the input vectors to intermediate feature space using non-linear mapping.
- The intermediate feature vectors may be again mapped to another intermediate feature space using non-linear mapping, and so on.
- Final intermediate feature vectors are linearly separable. So, at the, say, K-th layer we may use a linear classifier.
- Here, we observe that information always flows in the forward direction, and so called feed forward.
- There are some networks where information may feed back to the previous layer. Example, Recurrent Neural Network (RNN).
- Training of the neural network means estimation of the parameters, like weights and bias.
- We give learning or training to the neural networks using back propagation algorithm, where error is propagated in the backward direction. This is because, only at the output layer, we can compute the error or loss function.
- Before discussing the training of the neural networks, we will discuss different types of non-linear functions (in the next class).