Lecture -02: Introduction to Neural Networks

What are Neural Networks ?

Biological Neuron vs. Artificial Neuron

Biological Neuron	Artificial Neuron
Dendrites (input)	Input
Soma (cell body)	Node
Axon (output)	Output
Synapse	Interconnections
Adaptation-based learning	Model-based learning

Neural Network Learning Algorithms (1)

- Two main algorithms:
  - **Perceptron**: Initial algorithm for learning simple neural networks (with no hidden layer) developed in the 1950's.
  - **Backpropagation**: More complex algorithm for learning multi-layer neural networks developed in the 1980's.

Neural Network Learning Algorithms (2)

- Neural Netwoks are one the most important class of learning algorithms in ML.
- The learned classification model is an **algebraic function**.
- The function is *linear* for Perceptron algorithm, *non-linear* for Backpropagation algorithm
- Both features and the output classes are allowed to be real valued

Perceptron: The First Neural Network

Types of Artificial Neural Networks

- ANN can be categorized based on number of hidden layers contained in ANN architecture

01

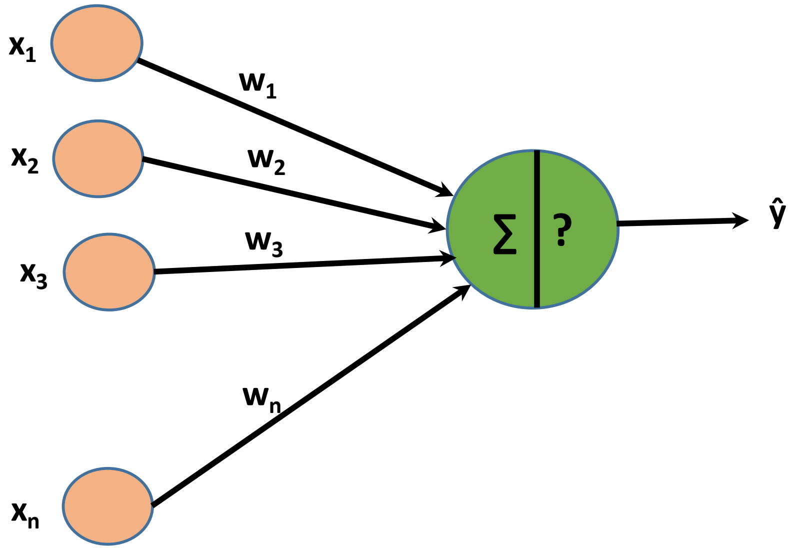
One Layer Neural Network (Perceptron)
- Contains 0 hidden layers

02

Multi Layer Neural Network
- Regular Neural Network
  - Contains 1 hidden layer
- Deep Neural Network
  - Contains >1 hidden layers

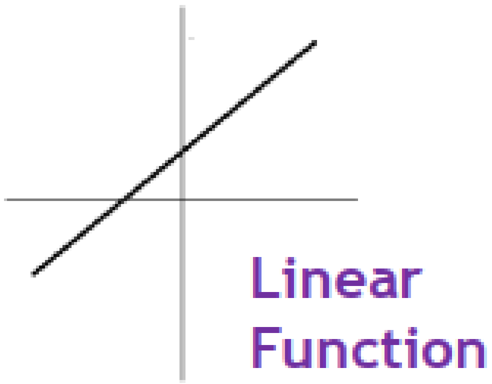
One layer Artificial Neural Network (Perceptron)

- Multiple input nodes
- Single output node
  - Takes weighted sum of the inputs
  - Unit function calculates the output for the network



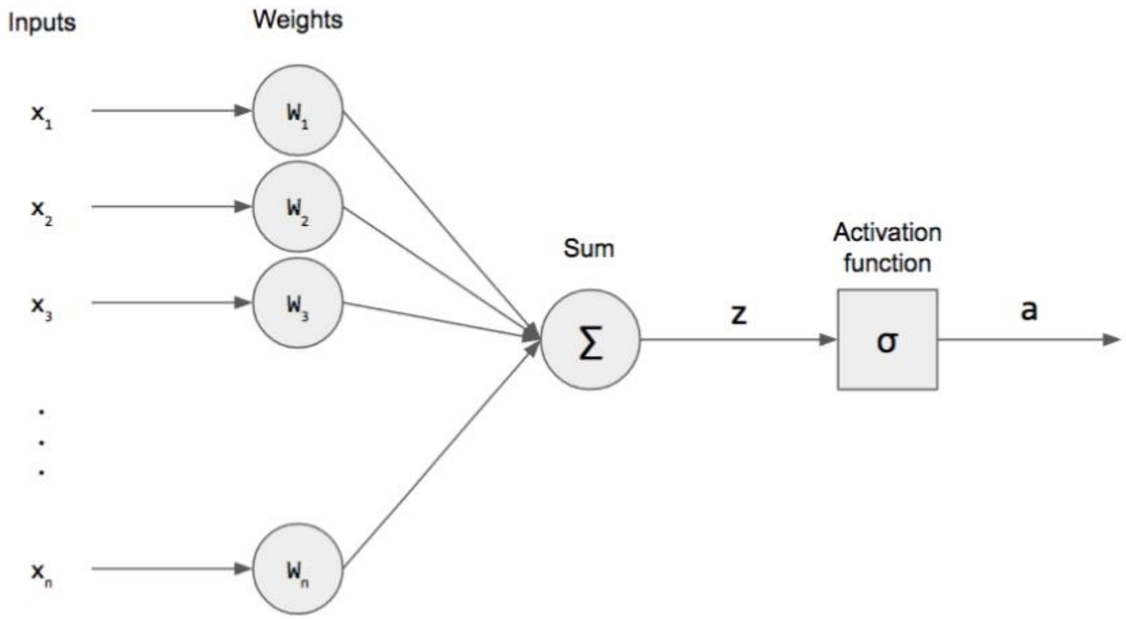
Unit Function

- Linear Function
  - Simply output the weighted sum



Unit Function

- Linear Function
  - Weighted sum followed by an activation function



Perceptron Example

- **To categorize a 2x2 pixel binary image to:**
  - “Bright” and “Dark”
- **The rule is:**
  - If it contains 2, 3 or 4 white pixels, it is “**bright**”
  - If it contains 0 or 1 white pixels, it is “**dark**”
- **Perceptron architecture**:
  - Four input units, one for each pixel
  - One output unit: +1 for bright, -1 for dark

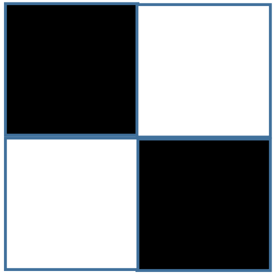
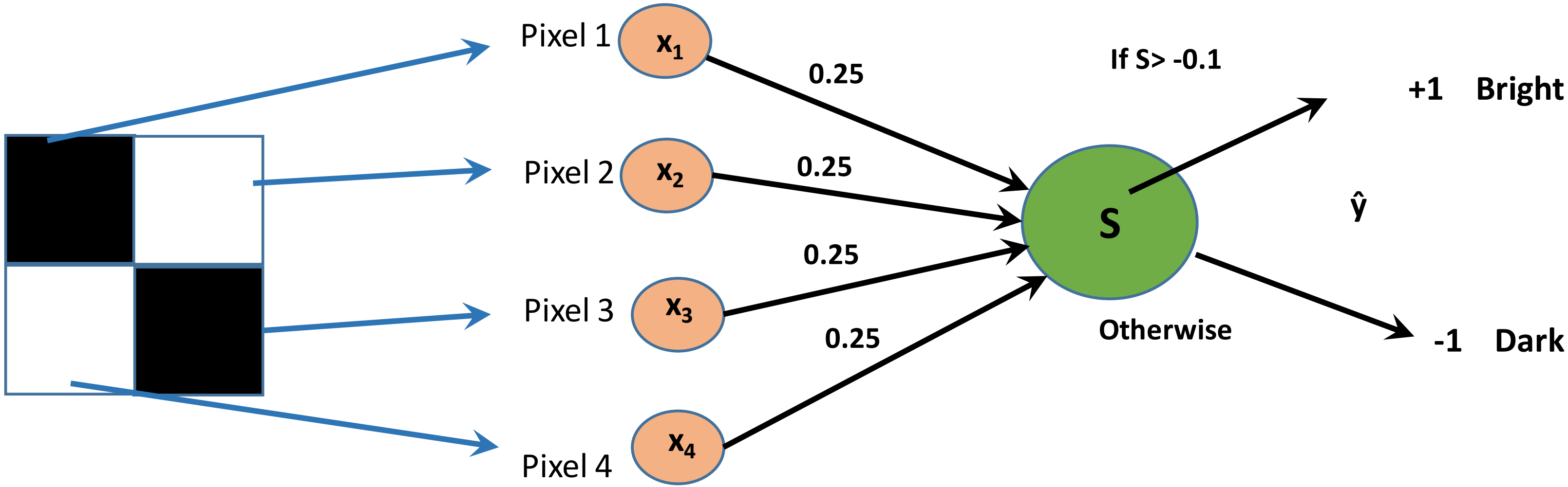


Image of 4 pixels

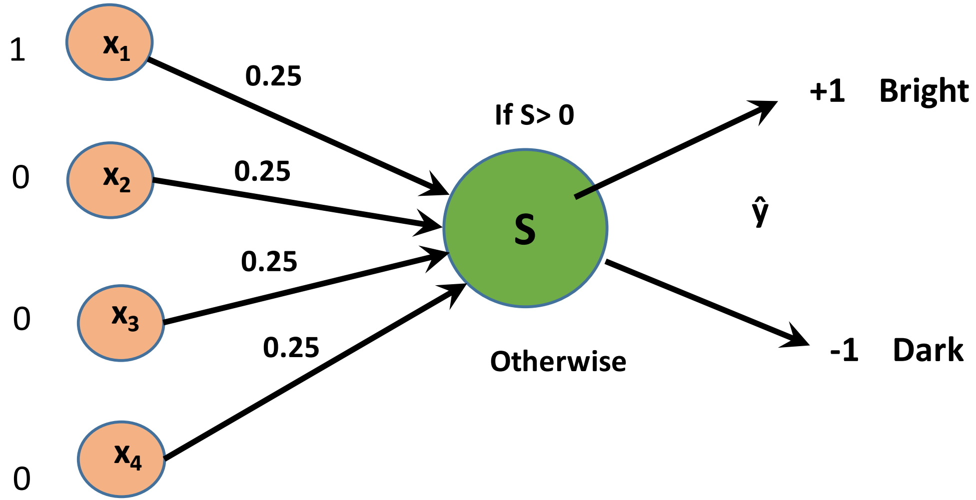
Perceptron Example



$$S = 0.25*x_1 + 0.25*x_2 + 0.25*x_3 + 0.25*x_4$$

Perceptron Example

- **Calculation (Step-1):**
  - $X_1 = 1$
  - $X_2 = 0$
  - $X_3 = 0$
  - $X_4 = 0$



S= 0.25*(1) + 0.25* (0) + 0.25*(0) + 0.25* (0) = 0.25

- **0.25 > 0, so the output of ANN is +1**
  - So the image is categorized as "Bright"
  - Target : "Dark"

Perceptron Training Rule (How to update weights)

- **When t(E) is different from o(E)**
  - Add Δ_i to weight w_i
  - Where $\Delta_i = \eta(t(E) - o(E))\, x_i$ → η is learning rate (Usually very small value)
  - Do this for every weight in the network
  - Let $\eta=0.1$

Calculating the error values

$\Delta_1 = \eta\,(t(E) - o(E))^* \, x_1$
$= 0.1\,(-1-1) * 1 = -0.2$

$\Delta_2 = \eta\,(t(E) - o(E))^* \, x_2$
$= 0.1\,(-1-1) * 0 = 0$

$\Delta_3 = \eta\,(t(E) - o(E))^* \, x_3$
$= 0.1\,(-1-1) * 0 = 0$

$\Delta_4 = \eta\,(t(E) - o(E))^* \, x_4$
$= 0.1\,(-1-1) * 0 = 0$

Calculating the New Weights

$w'_1 = w_1 + \Delta_1 = 0.25 - 0.2 = 0.05$

$w'_2 = w_2 + \Delta_2 = 0.25 + 0 = 0.25$

$w'_3 = w_3 + \Delta_3 = 0.25 + 0 = 0.25$

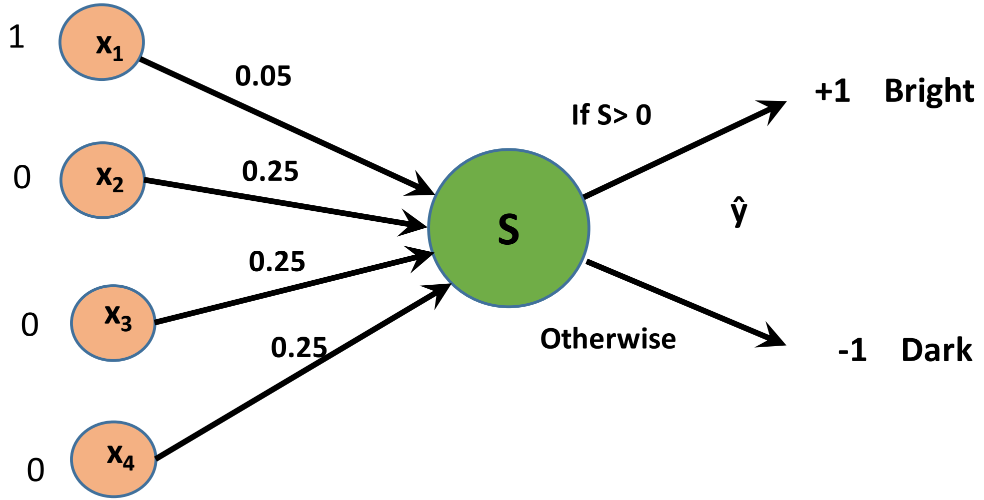
$w'_4 = w_4 + \Delta_4 = 0.25 + 0 = 0.25$

Perceptron Example

- **Calculation (Step-2):**
  - $X_1 = 1$
  - $X_2 = 0$
  - $X_3 = 0$
  - $X_4 = 0$

S= 0.05*(1) + 0.25* (0) + 0.25*(0) + 0.25* (0) = 0.05

- **0.05 > 0, so the output of ANN is +1**
  - So the image is categorized as "Bright"
  - Target : "Dark"



Perceptron Training Rule (How to update weights)

- **When t(E) is different from o(E)**
  - Add Δ_i to weight w_i
  - Where $\Delta_i = \eta(t(E) – o(E)\ )\ x_i$ → η is learning rate (Usually very small value)
  - Do this for every weight in the network
  - Let $\eta=0.1$

Calculating the error values

$\Delta_1 = \eta\ (t(E) – o(E)\ )^*\ x_1$
$= 0.1\ (\ -1\ -1\ )\ *\ 1 = -0.2$

$\Delta_2 = \eta\ (t(E) – o(E)\ )^*\ x_2$
$= 0.1\ (\ -1-1)\ *\ 0 = 0$

$\Delta_3 = \eta\ (t(E) – o(E)\ )^*\ x_3$
$= 0.1\ (\ -1-1)\ *\ 0 = 0$

$\Delta_4 = \eta\ (t(E) – o(E)\ )^*\ x_4$
$= 0.1\ (\ -1-1)\ *\ 0 = 0$

Calculating the New Weights

$w'_1 = w_1 + \Delta_1 = 0.05 - 0.2 = -0.15$

$w'_2 = w_2 + \Delta_2 = 0.25 + 0 = 0.25$

$w'_3 = w_3 + \Delta_3 = 0.25 + 0 = 0.25$

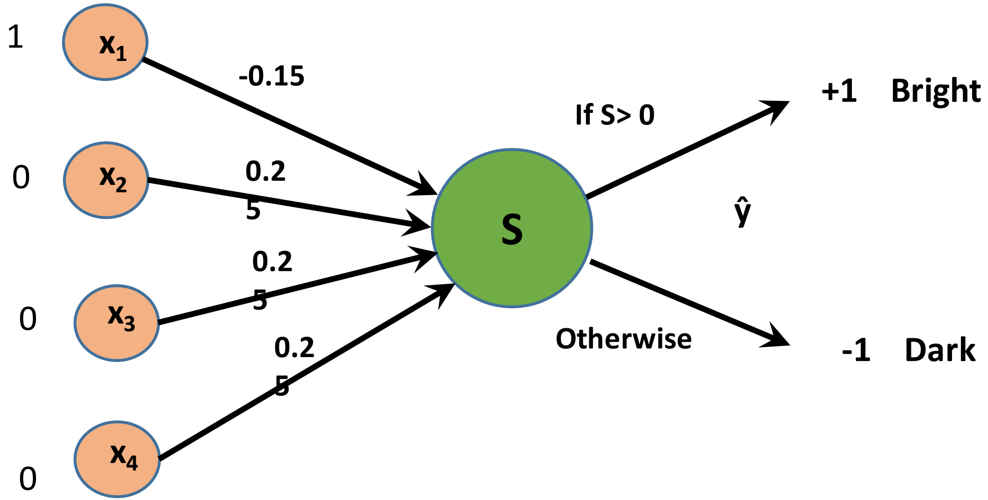
$w'_4 = w_4 + \Delta_4 = 0.25 + 0 = 0.25$

Perceptron Example

- **Calculation (Step-3):**
  - $X_1 = 1$
  - $X_2 = 0$
  - $X_3 = 0$
  - $X_4 = 0$

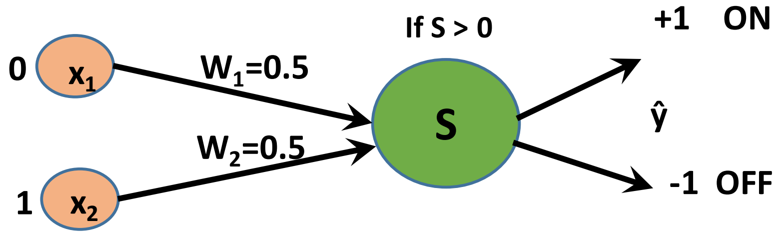
S= - 0.15*(1) + 0.25* (0) + 0.25*(0) + 0.25* (0) = - 0.15

- **- 0.15 < 0, so the output of ANN is -1**
  - So the image is categorized as "Dark"
  - Target : "Dark"



Another Example (AND)

X_1	X_2	X_1 AND X_2
0	0	0
0	1	0
1	0	0
1	1	1

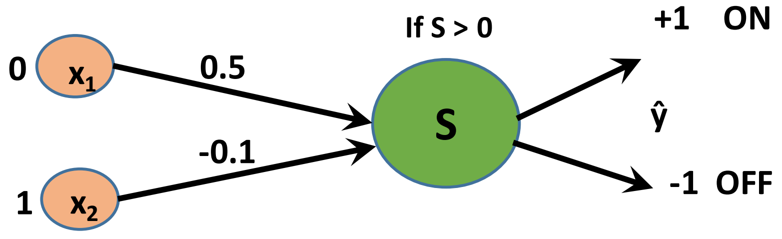


- X1 = 0
- X2 = 1,
- η = 0.1
- t(E) = -1

Weights	Step-1	Step-2	Step-3	Step-4
w1	0.5	0.5	0.5	0.5
w2	0.5	0.3	0.1	-0.1
Weighted Sum	0.5	0.3	0.1	-0.1
Observed Output	+1	+1	+1	-1

Another Example (AND)

X_1	X_2	X_1 AND X_2
0	0	0
0	1	0
1	0	0
1	1	1

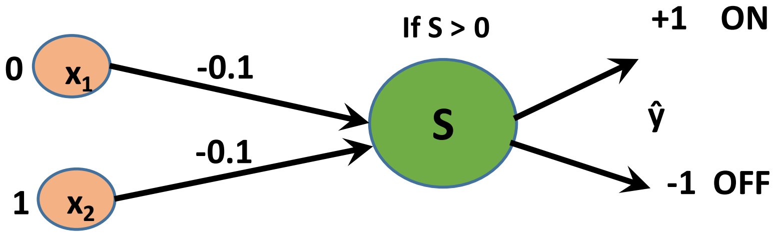


- X1 = 1
- X2 = 0,
- η = 0.1
- t(E) = -1

Weights	Step-1	Step-2	Step-3	Step-4
w1	0.5	0.3	0.1	-0.1
w2	-0.1	-0.1	-0.1	-0.1
Weighted Sum	0.5	0.3	0.1	-0.1
Observed Output	+1	+1	+1	-1

Another Example (AND)

X_1	X_2	X_1 AND X_2
0	0	0
0	1	0
1	0	0
1	1	1

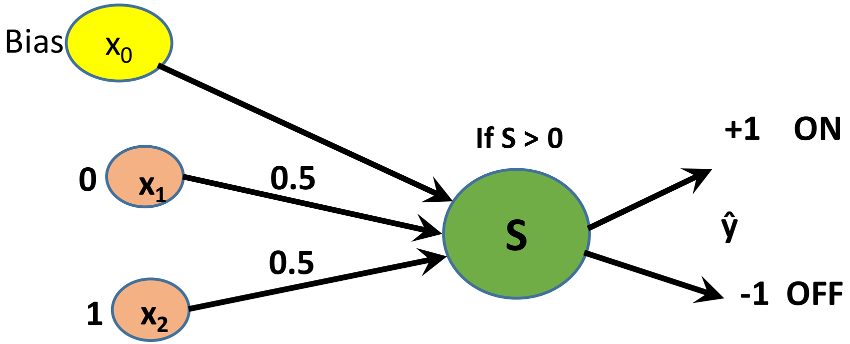


- X1 = 1
- X2 = 1,
- $\eta = 0.1$
- t(E) = +1

Weights	Step-1	Step-2
w1	-0.1	0.1
w2	-0.1	0.1
Weighted Sum	-0.2	0.2
Observed Output	- 1	+1

Use of Bias

- Bias is just like an intercept added in a linear equation.



output = sum (weights * inputs) + bias

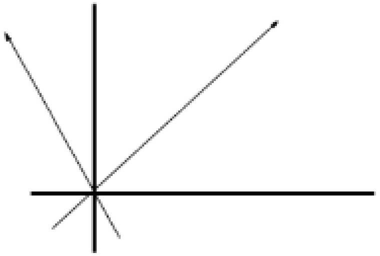
- The output is calculated by multiplying the inputs with their weights and then passing it through an activation function like the Sigmoid function, etc. Here, bias acts like a constant which helps the model to fit the given data.

Use of Bias

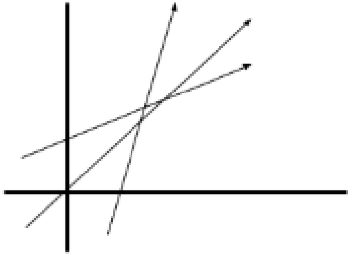
- A simpler way to understand bias is through a constant c of a linear function

$$y = mx + c$$

- It allows us to move the line down and up fitting the prediction with the data better. If the constant c is absent then the line will pass through the origin (0, 0) and we will get a poorer fit.



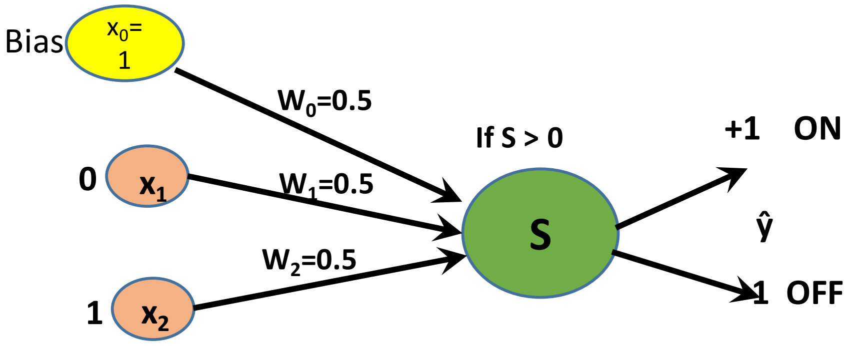
y = mx



y = mx + c

Example (AND) with Bias

X_1	X_2	X_1 AND X_2
0	0	0
0	1	0
1	0	0
1	1	1

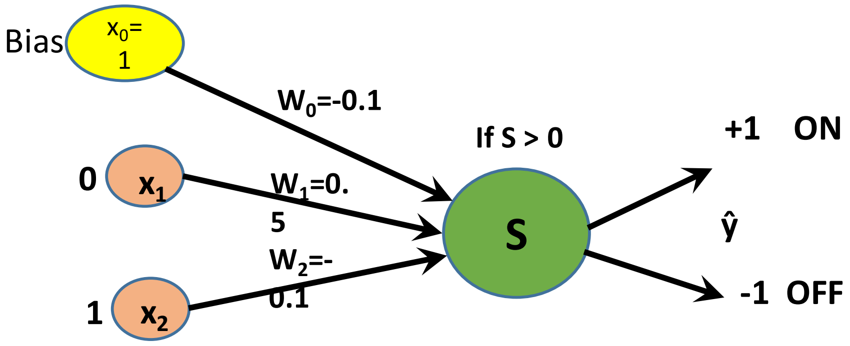


- X1 = 0
- X2 = 1,
- η = 0.1
- t(E) = -1

Weights	Step-1	Step-2	Step-3	Step-4
w0	0.5	0.3	0.1	-0.1
w1	0.5	0.5	0.5	0.5
w2	0.5	0.3	0.1	-0.1
Weighted Sum	1	0.6	0.2	-0.2
Observed Output	+1	+1	+1	-1

Example (AND) with Bias

X_1	X_2	X_1 AND X_2
0	0	0
0	1	0
1	0	0
1	1	1

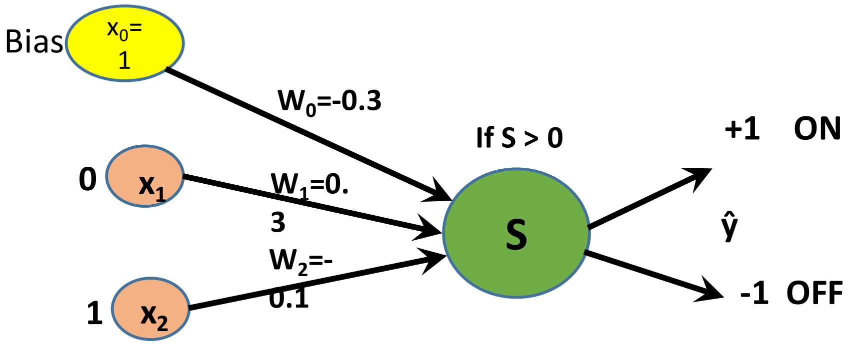


- X1 = 1
- X2 = 0,
- η = 0.1
- t(E) = -1

Weights	Step-1	Step-2
w0	-0.1	-0.3
w1	0.5	0.3
w2	-0.1	-0.1
Weighted Sum	0.4	0
Observed Output	+1	-1

Example (AND) with Bias

X_1	X_2	X_1 AND X_2
0	0	0
0	1	0
1	0	0
1	1	1



- $X1 = 1$
- $X2 = 1$,
- $\eta = 0.1$
- $t(E) = +1$

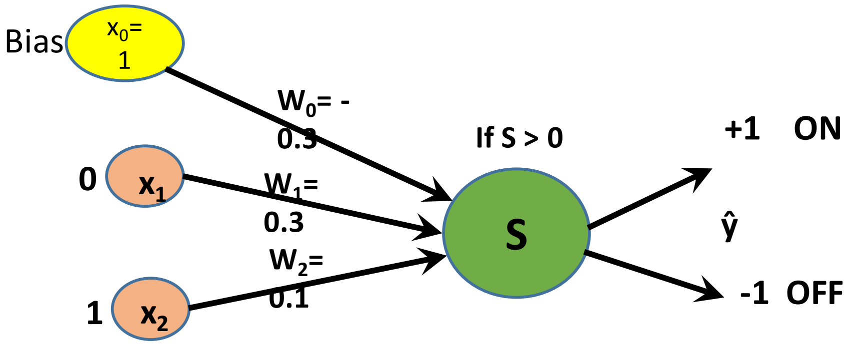
Weights	Step-1	Step-2
w0	-0.3	-0.1
w1	0.3	0.5
w2	-0.1	0.1
Weighted Sum	0	0.5
Observed Output	-1	+1

Example (AND) with Bias

After 2 Epochs

X_1	X_2	X_1 AND X_2
0	0	0
0	1	0
1	0	0
1	1	1

- X1 = 0
- X2 = 1,
- η = 0.1
- t(E) = -1



Final Weights

Weights	
w0	- 0.3
w1	0.3
w2	0.1

Learning in Perceptron

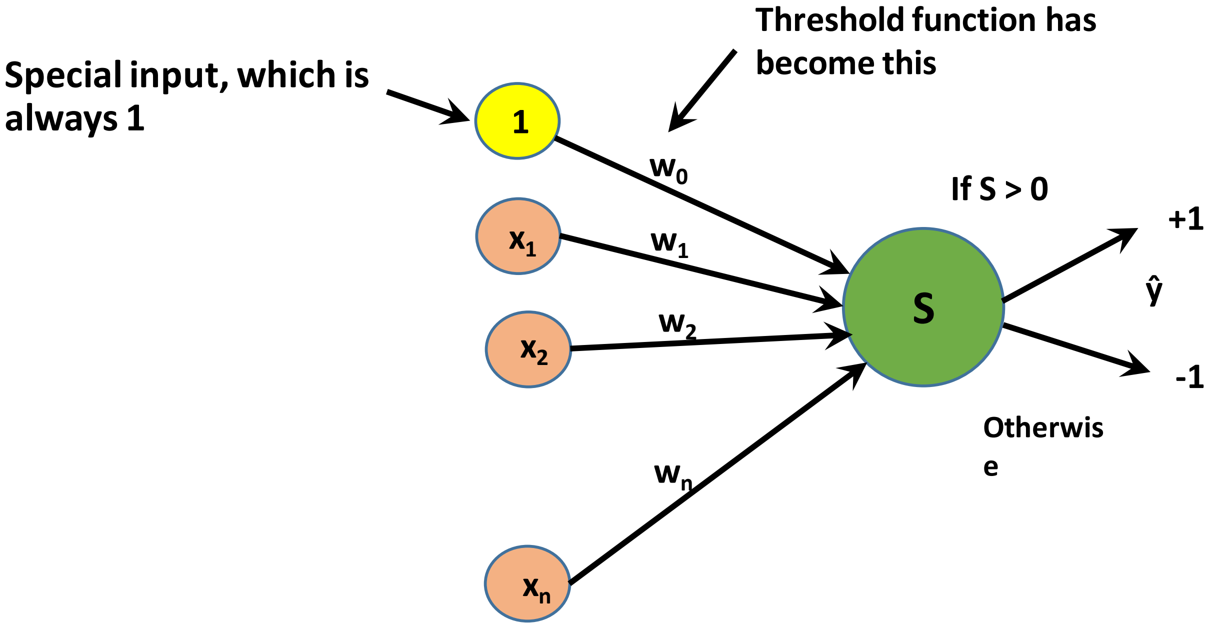
Need To Learn

- Both the **weights** between input and output units
- And the value for the bias

- **Make Calculations easier by:**
  - Thinking of the bias as a weight from a special input unit where the output from the unit is always 1

- **Exactly the same result:**
  - But we only have to worry about learning weights

New Representation for Perceptron



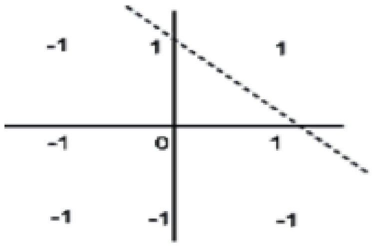
$$S = w_0 + w_1 * x_1 + w_2 * x_2 \ldots\ldots\ldots\; w_n * x_n$$

Learning Algorithm

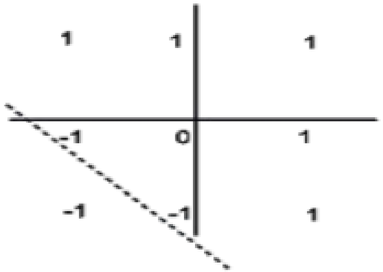
- Weights are randomly initialized.
- For each training example E
    - Calculate the observed output from Perceptron, o(E)
    - If the target output t(E) is different to o(E)
        - Then update all the weights so that o(E) becomes closer to t(E)
- This process is done for every example
- It is not necessary to stop when all examples are used.
    - Repeat the cycle again (an epoch) until network produces the correct output

Limitations of Perceptron

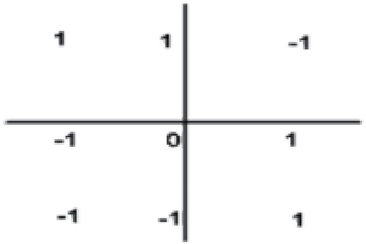
- The perceptron can only learn simple problems. this is only useful if the problem is linearly separable.
- A linearly separable problem is one in which the classes can be separated by a single hyperplane.



AND VALUES



OR VALUES



XOR VALUES

Any Questions?