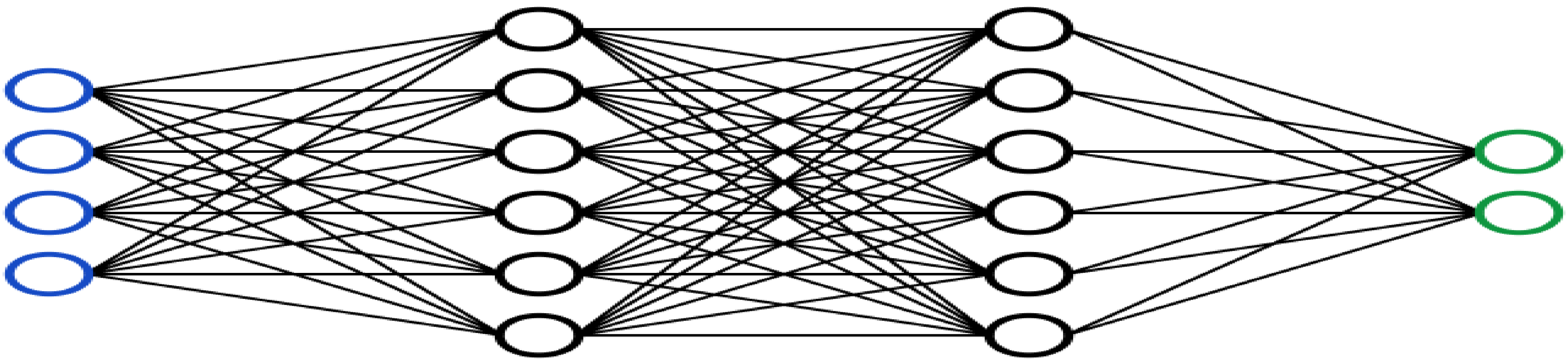


Artificial Neural Network (ANN)

Lecture4

Dina El-Manakhly, Ph. D.

dina_almnakhly@science.suez.edu.eg

What is bias in a neural network?

- Neural network bias can be defined as the constant which is added to the product of features and weights. It is used to offset the result. It helps the models to shift the activation function towards the positive or negative side.

Consider a sigmoid activation function which is represented by the equation below:

$$sigmoid\ function = \frac{1}{1 + e^{-x}}$$

On replacing the variable 'x' with the equation of line, we get the following:

$$sigmoid\ function = \frac{1}{1 + e^{-(w*x+b)}}$$

In the above equation, 'w' is weights, 'x' is the feature vector, and 'b' is defined as the bias.

Vary the values of the weight ‘w’, keeping bias ‘b’=0

```python
# Define the sigmoid function with adjustable steepness
def sigmoid(x, k=1, b=0):
    return 1 / (1 + np.exp(-(k*x+b)))

# Generate input values
x = np.linspace(-10, 10, 200)

# Calculate sigmoid values
# with different steepness values
y_normal = sigmoid(x)
y_steep1 = sigmoid(x, k=2)
y_steep2 = sigmoid(x, k=3)
y_steep3 = sigmoid(x, k=4)
```

1

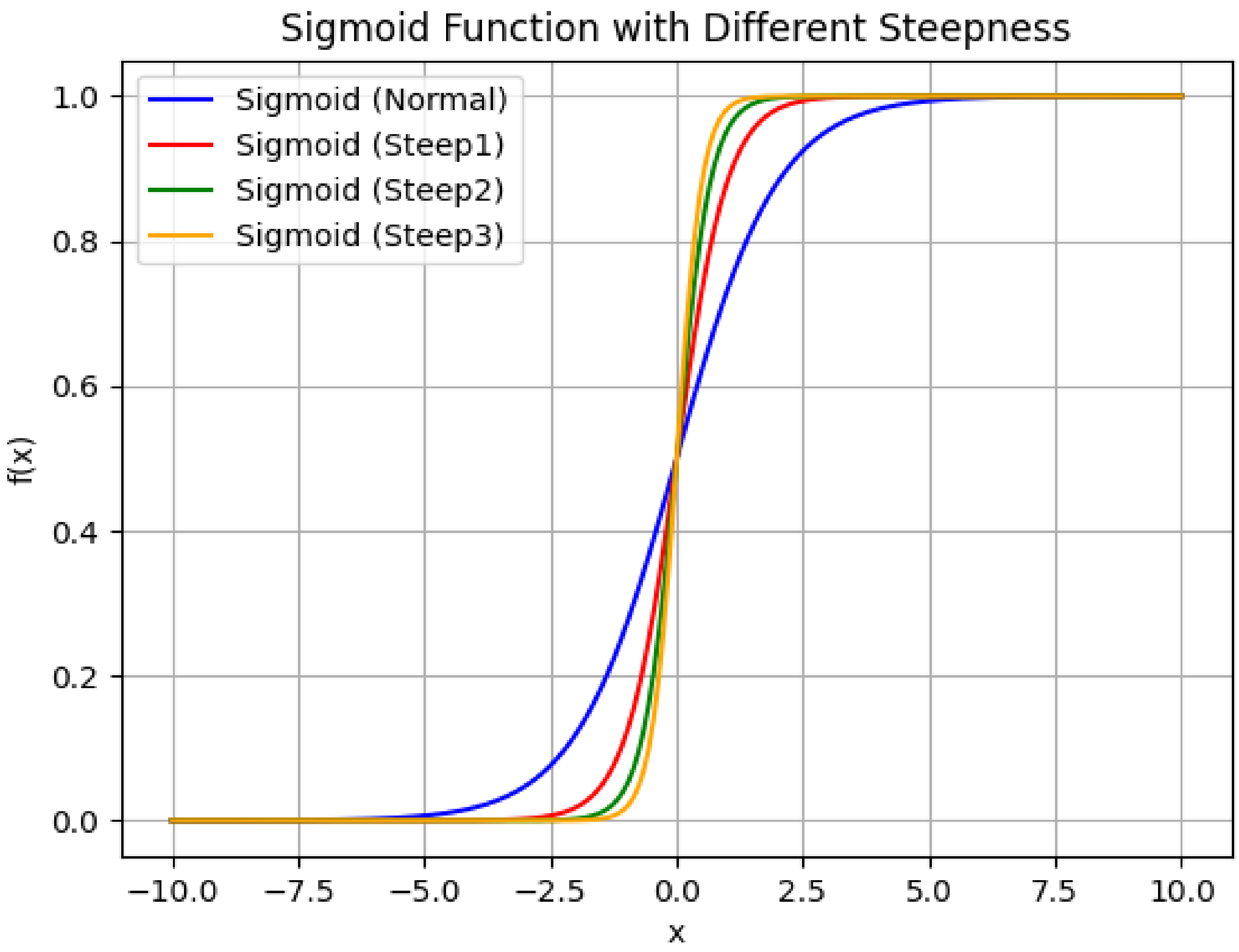
```python
plt.plot(x, y_normal, label='Sigmoid (Normal)', color='blue')
plt.plot(x, y_steep1, label='Sigmoid (Steep1)', color='red')
plt.plot(x, y_steep2, label='Sigmoid (Steep2)', color='green')
plt.plot(x, y_steep3, label='Sigmoid (Steep3)', color='orange')

plt.title('Sigmoid Function with Different Steepness')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.legend()
plt.grid(True)
plt.show()
```

2

Conclusion

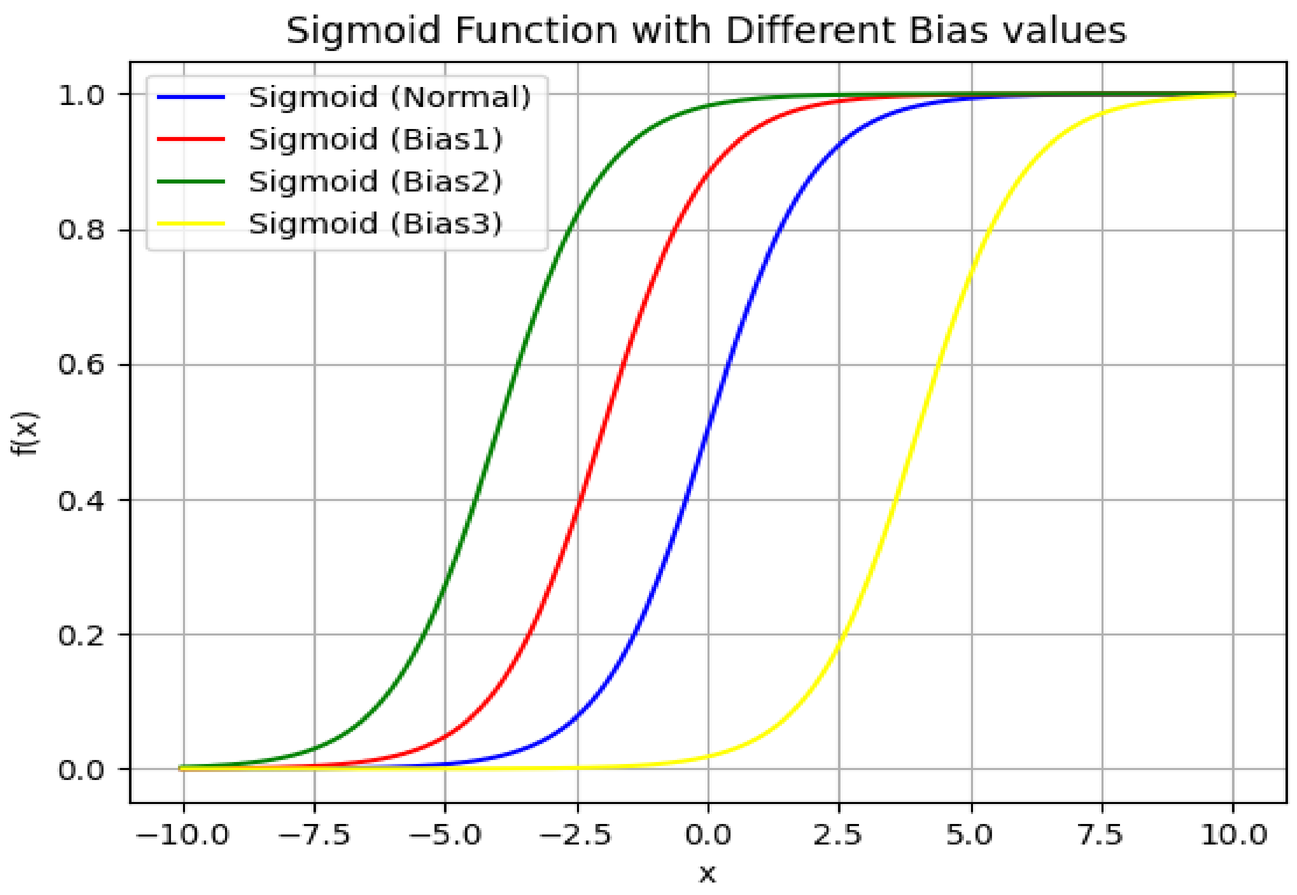
Vary the values of the weight ‘w’, keeping bias ‘b’=0

- While changing the values of ‘w’, there is no way we can shift the origin of the activation function, i.e., the sigmoid function.
- On changing the values of ‘w’, only the steepness of the curve will change.
- There is only one way to shift the origin and that is to include bias ‘b’.



Keep the value of weight ‘w’ fixed and varying the value of bias ‘b’

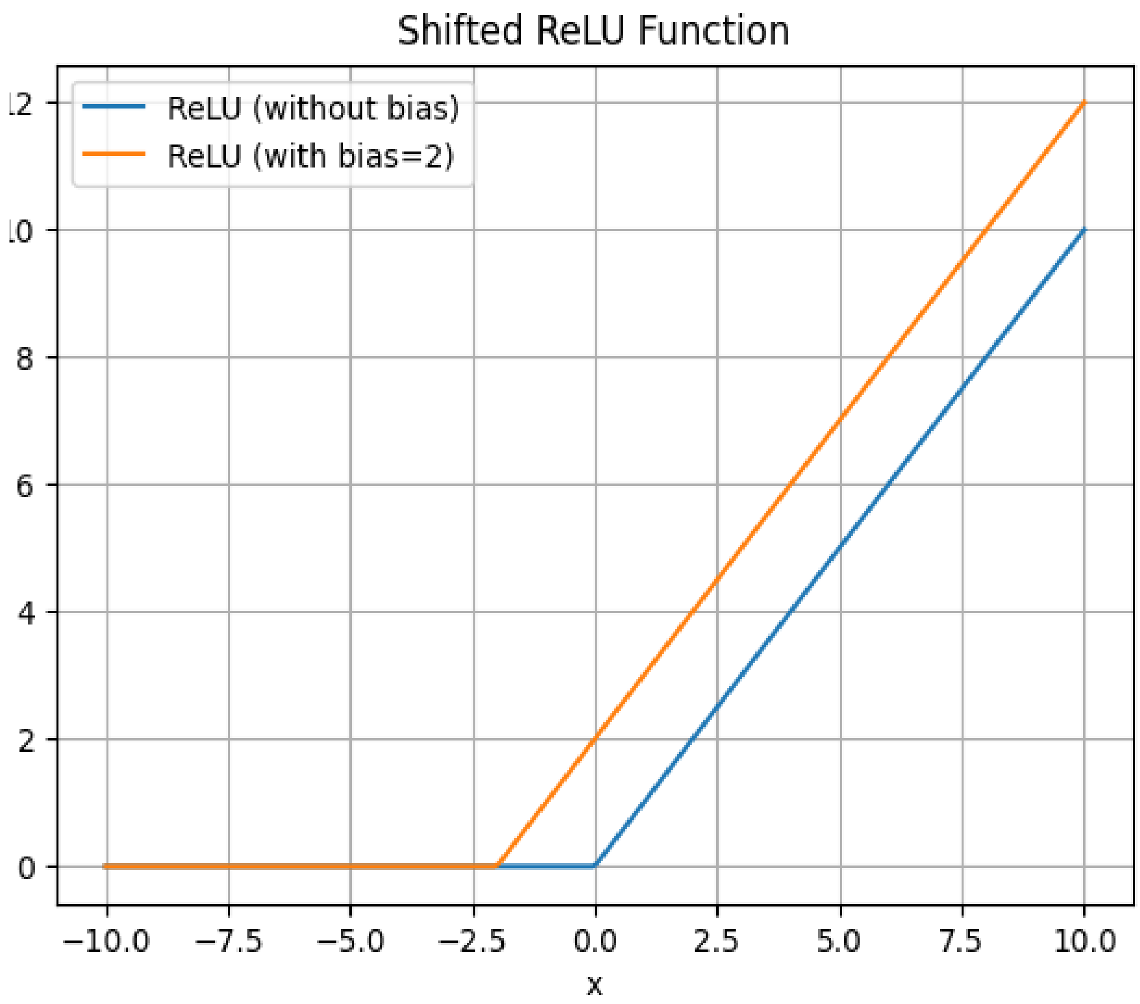
```
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x, k=1,b=0):
    return 1 / (1 + np.exp(-(k*x+b)))

# Generate input values
x = np.linspace(-10, 10, 200)

y_normal = sigmoid(x)
y_Bias1 = sigmoid(x, b=2)
y_Bias2 = sigmoid(x, b=4)
y_Bias3 = sigmoid(x, b=-4)
```



From the graph, it can be concluded that the bias is required for shifting the origin of the curve to the left or right.

Another Example: Including Bias Within the Relu Activation Function

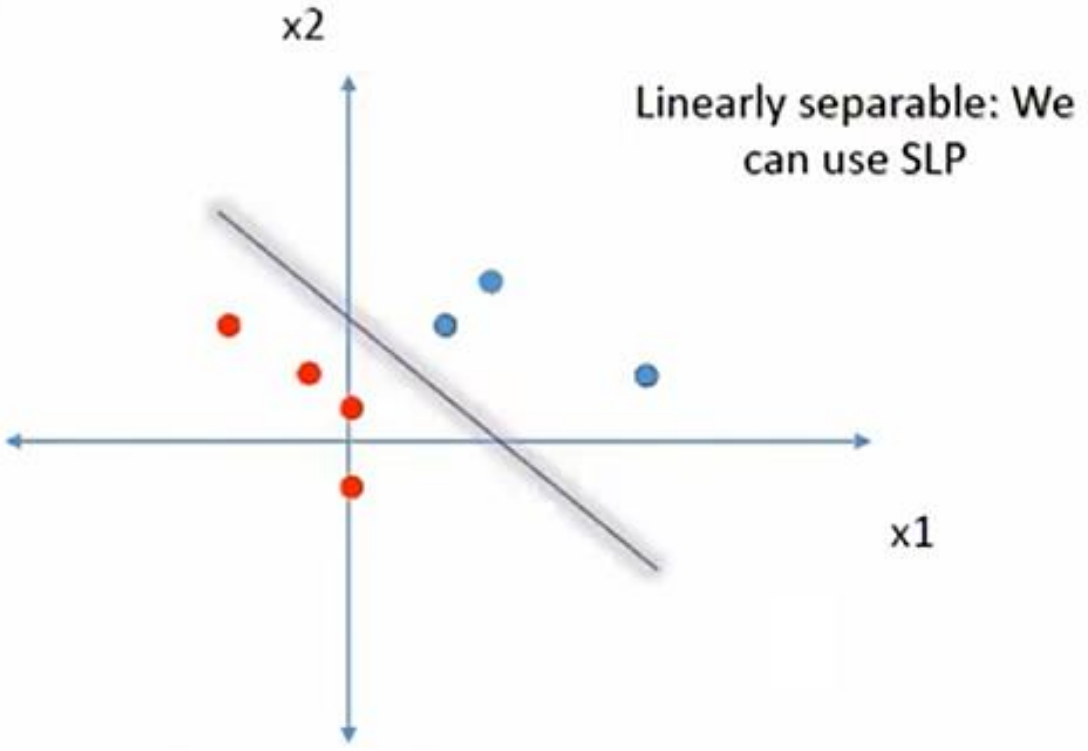
```
def shifted_relu(x, bias):
    return np.maximum(0, x + bias)

x = np.linspace(-10, 10, 200)
bias = 2
y = shifted_relu(x, bias)

# Plot the original ReLU function (without bias)
plt.plot(x, np.maximum(0, x), label='ReLU (without bias)')
# Plot the shifted ReLU function with the chosen bias
plt.plot(x, y, label='ReLU (with bias={})'.format(bias))
```



Single Layer Perceptron
What can perceptron represents?

- Visualization example (2 features)

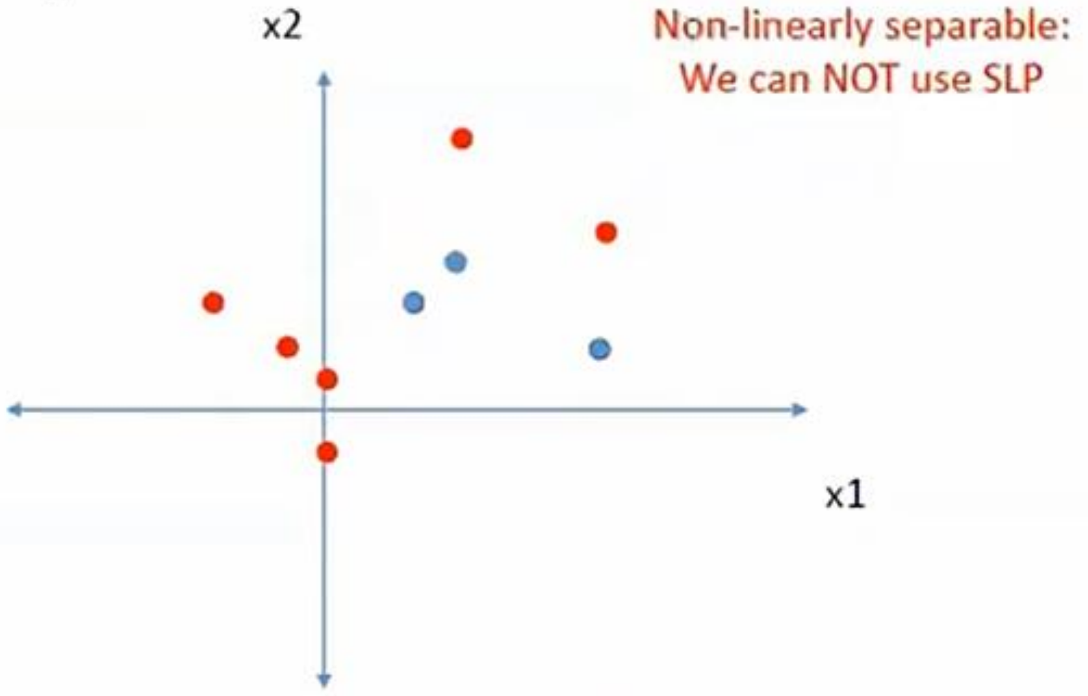
x1	x2	t
2	3	0
-3	3	1
3	4	0
-1	2	1
7	2	0
0	1	1
0	-2	1



What can perceptron represents? (cont'd)

- Visualization example (2 features)

x1	x2	t
2	3	0
-3	3	1
3	4	0
-1	2	1
7	2	0
0	1	1
0	-2	1
3	8	1
7	5	1

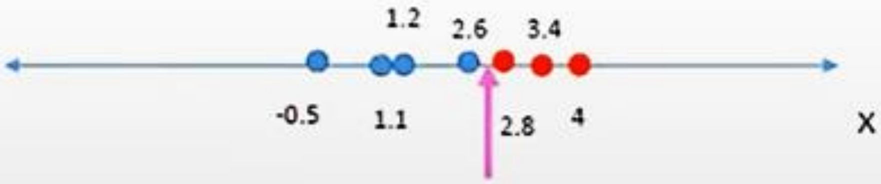


What can perceptron represents? (cont'd)

- Visualization example (1 feature)

x	t
1.1	0
2.8	1
-0.5	0
1.2	0
4	1
2.6	0
3.4	1

Note: in 1d, SLP is a point separator

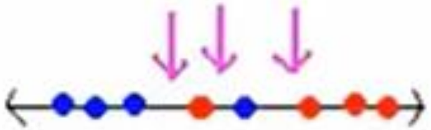


We can separate
the two classes
with one point:
We can use SLP

What can perceptron represents? (cont'd)

- Visualization example (1 feature)

Need at least 2 points:
Can't use SLP



Need at least 3 points:
Can't use SLP

Boolean Functions

AND

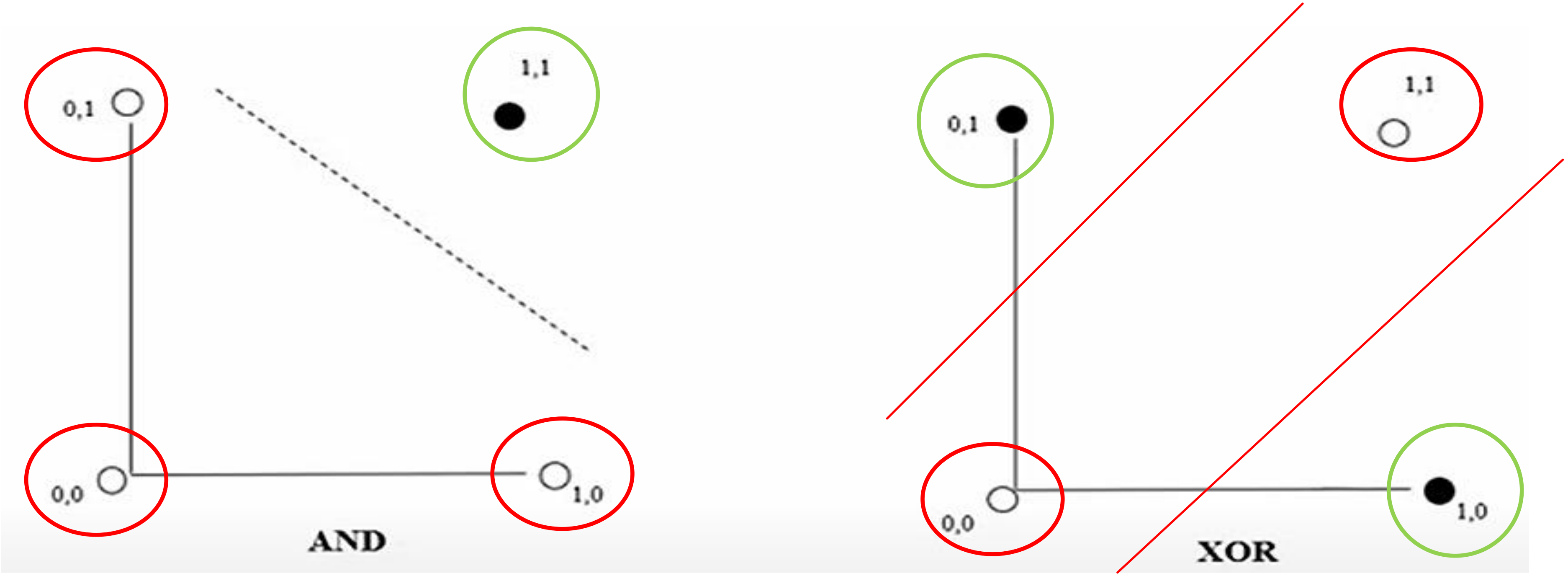
Input		Output
A	B	F = A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR

Input		Output
A	B	F = A+B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

What can perceptron represents?



- Functions which can be separated in this way are called linearly separable.
- Only linearly separable functions can be represented by a perceptron.

First neural network

<u>Steps of perceptron</u>

1) Multiply all input values with corresponding weight values and then add to calculate the weighted sum

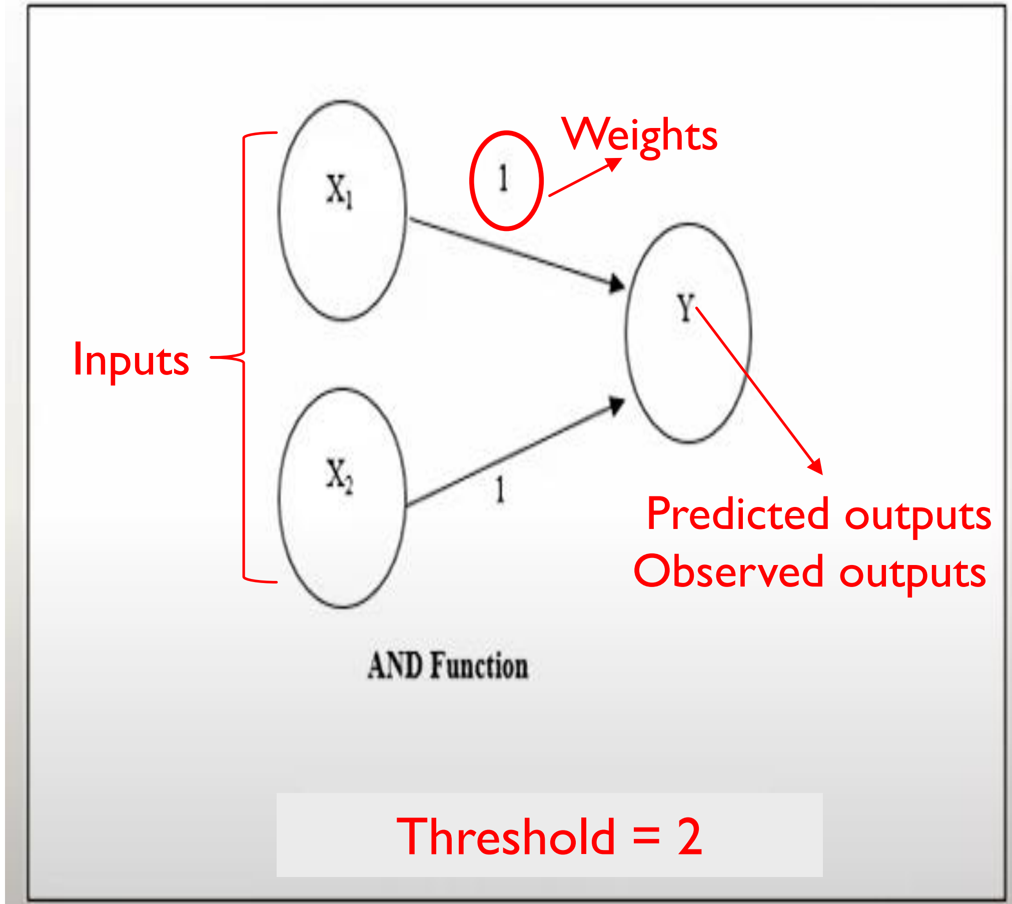
$$\Sigma w_i * x_i = x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + x_4 * w_4$$

2) An activation function is applied with the above-mentioned weighted sum giving us an output as follows:

$$Y = f\,(\Sigma w_i * x_i)$$

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j < \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j \geq \text{threshold} \end{cases}$$

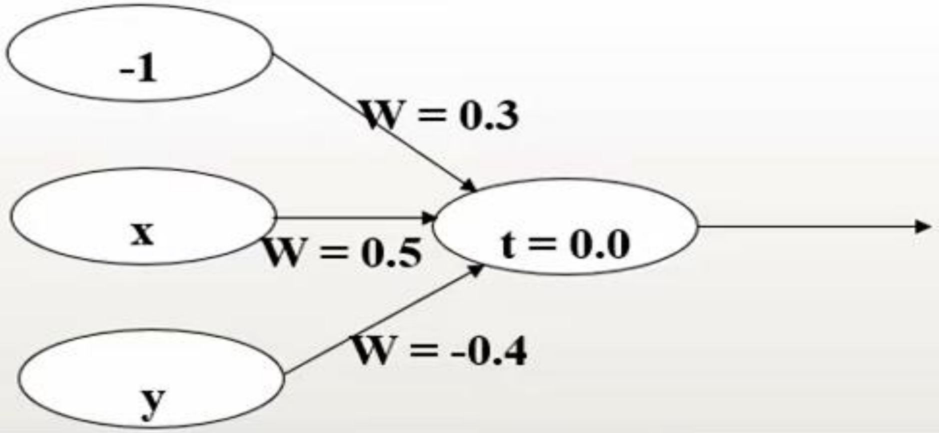
$f\,(\Sigma w_i * x_i)$



Actual outputs
True outputs

AND		
X1	X2	Y
1	1	1
1	0	0
0	1	0
0	0	0

Training a perceptron (Simple Example)



I_1	I_2	I_3	Summation	Output
-1	0	0	(-1*0.3) + (0*0.5) + (0*-0.4) = -0.3	0
-1	0	1	(-1*0.3) + (0*0.5) + (1*-0.4) = -0.7	0
-1	1	0	(-1*0.3) + (1*0.5) + (0*-0.4) = 0.2	1
-1	1	1	(-1*0.3) + (1*0.5) + (1*-0.4) = -0.2	0

AND		
X1	X2	Y
1	1	1
1	0	0
0	1	0
0	0	0

If the predicted output is incorrect, the weights are adjusted. This update process continues until the predicted output matches the target output for all training examples.

How to update weights in SLP ?

- Single layer perceptron
  - Using perceptron learning algorithm
  - Using delta rule

Perceptron Learning Algorithm

- **An epoch in neural network means training the neural network with all the training data for one cycle. In an epoch, we use all of the data exactly once.**

- **Learning rule**

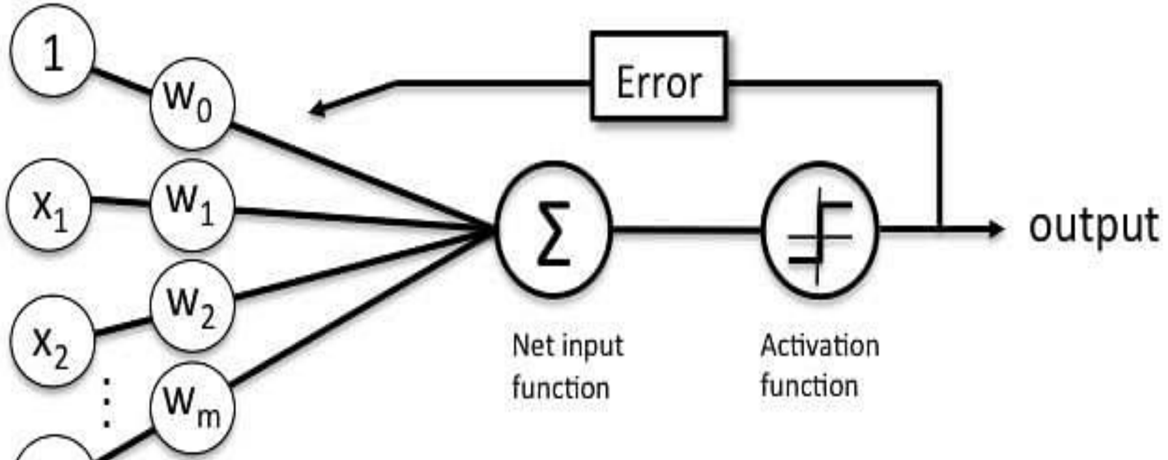
  While epoch produces an error
  Present network with next inputs from epoch
  Error = T – O → True output – observed output
  If Error <> 0 then

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \underbrace{t}_{\text{target value}} - \underbrace{o}_{\text{perceptron output}}) \underbrace{x_i}_{\text{input value}}$$

  End if
  End While



Perceptron rule.

Perceptron learning algorithm

x1	x2	t
0	0	0
0	1	1
1	0	1
1	1	1

Perceptron learning algorithm

x1	x2	bias	t
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

Perceptron learning algorithm

Initial weights

Error=t-y

x1	x2	bias	w1	w2	w_bias	net	y	t	
0	0	1	0.1	0.2	-0.2	-0.2	0	0	Error=0
0	1	1	0.1	0.2	-0.2	0	0	1	Error=1
1	0	1						1	
1	1	1						1	

Calculate net = x1*w1 + x2*w2 + bias*w_bias

Calculate y =
1 if net >= threshold,
0 if net < threshold

Threshold should be given. If not, assume random threshold

Here we assume threshold = 0.1 → net < threshold

Perceptron learning algorithm

Error=t-y

x1	x2	bias	w1	w2	w_bias	net	y	t	
0	0	1	0.1	0.2	-0.2	-0.2	0	0	Error=0
0	1	1	0.1	0.2	-0.2	0	0	1	Error=1
1	0	1	0.1	1.2	0.8	0.9	1	1	Error=0
1	1	1	0.1	1.2	0.8	2.1	1	1	Error=0

$w\_new = w\_old + (t-y)*x$

Use these as initial weights for next epoch

Perceptron learning algorithm

Next epoch:

x1	x2	bias	w1	w2	w_bias	net	y	t	
0	0	1	0.1	1.2	0.8	0.8	1	0	Error= -1
0	1	1	0.1	1.2	-0.2	1	1	1	Error= 0
1	0	1	0.1	1.2	-0.2	-0.1	0	1	Error= 1
1	1	1	1.1	1.2	0.8	3.1	1	1	Error= 0

w_new = w_old + (t-y)*x

Use these as initial weights for next epoch

Perceptron learning algorithm

Next epoch:

x1	x2	bias	w1	w2	w_bias	net	y	t	
0	0	1	1.1	1.2	0.8	0.8	1	0	Error= -1
0	1	1	1.1	1.2	-0.2	1	1	1	Error= 0
1	0	1	1.1	1.2	-0.2	0.9	1	1	Error= 0
1	1	1	1.1	1.2	-0.2	2.1	1	1	Error= 0

w_new = w_old + (t-y)*x

Use these as initial weights for next epoch

Perceptron learning algorithm

Next epoch:

x1	x2	bias	w1	w2	w_bias	net	y	t	
0	0	1	1.1	1.2	-0.2	-0.2	0	0	**Error= 0**
0	1	1	1.1	1.2	-0.2	1	1	1	**Error= 0**
1	0	1	1.1	1.2	-0.2	0.9	1	1	**Error= 0**
1	1	1	1.1	1.2	-0.2	2.1	1	1	**Error= 0**

w_new = w_old + (t-y)*x