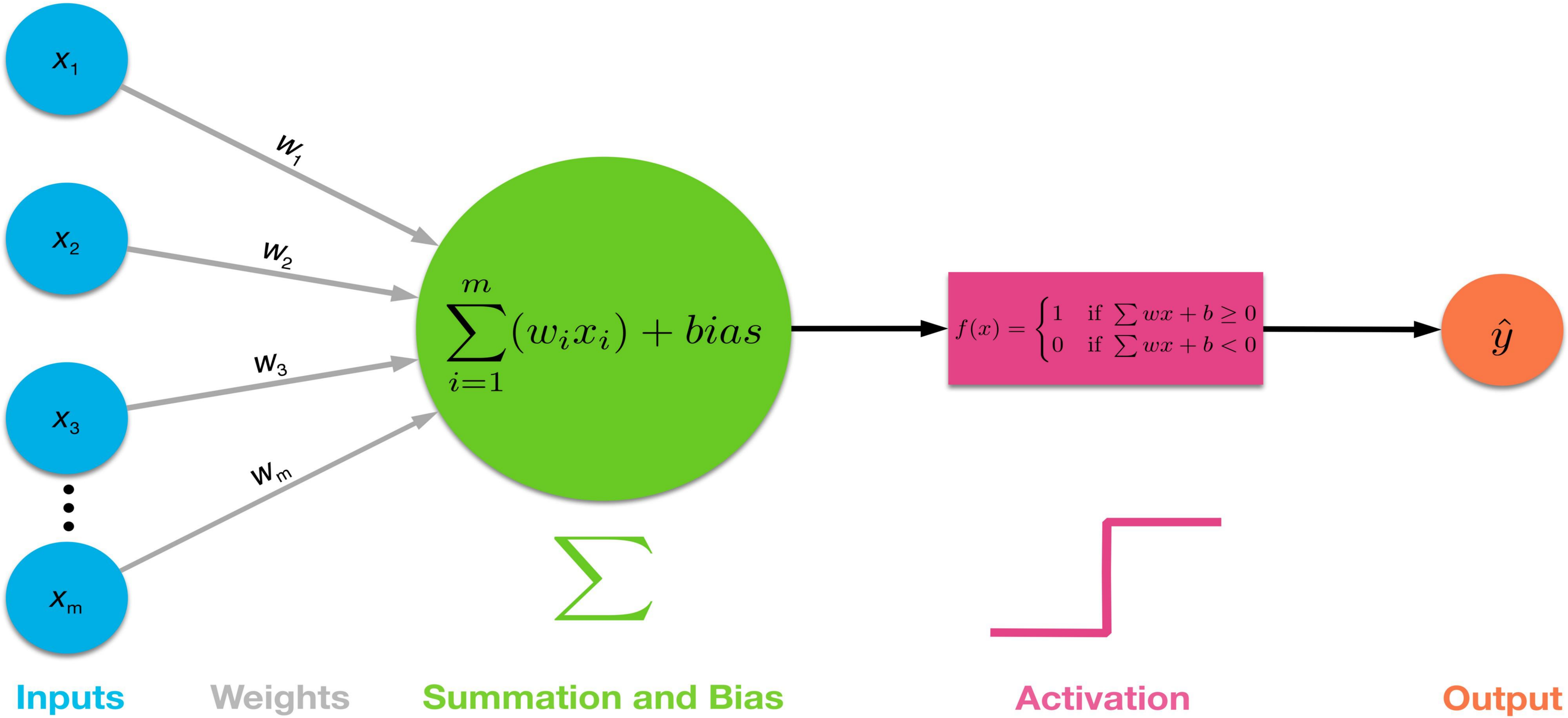
Artificial Neural Network (ANN)

Perceptron Learning
CI (CS-3030)

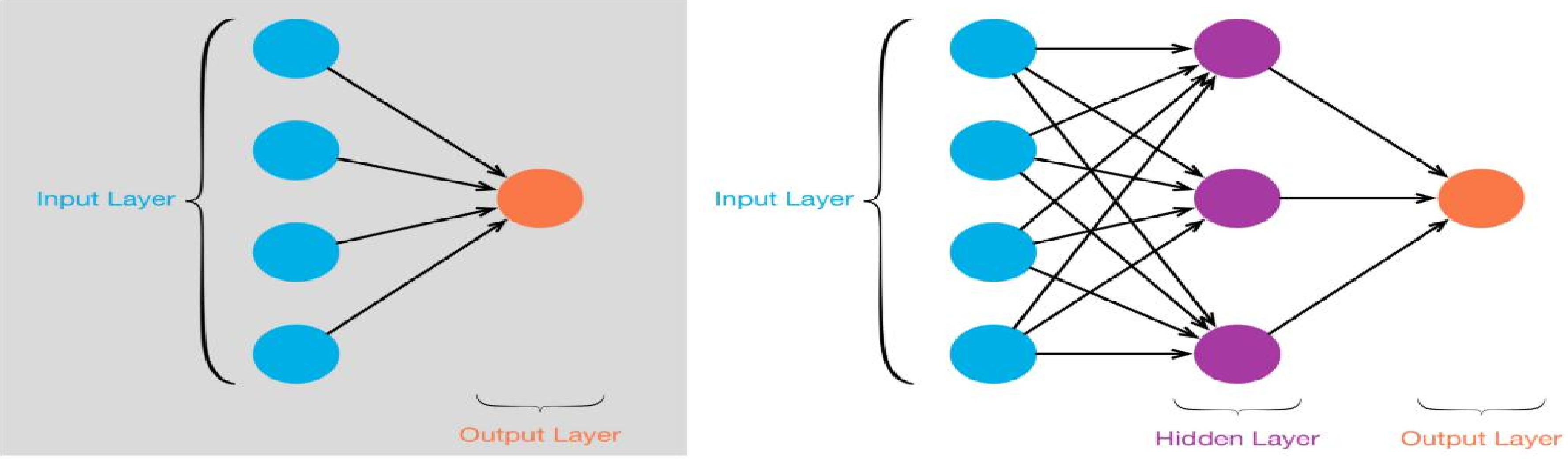
Rajdeep Chatterjee
Assistant Professor
School of Computer Engineering
KIIT Deemed to be University,
Bhubaneswar-751024, Odisha, India

Agenda

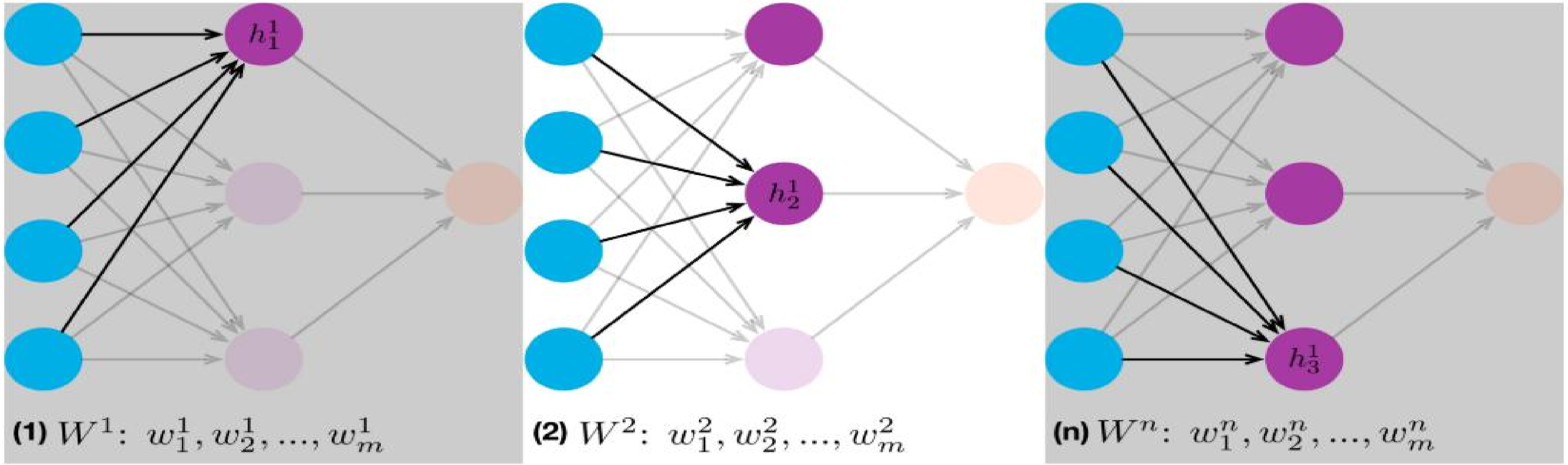
- Quick Recap
- ANN working models
- Backpropagation
- Perceptrons learning
- OR Perceptron classifier
- Python 3.6 example



x_1
x_2
x_3
x_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + bias$
$f(x) = \begin{cases} 1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0 \end{cases}$
$\hat{y}$
$\sum$
Inputs
Weights
Summation and Bias
Activation
Output



Left: Single-Layer Perceptron; Right: Perceptron with Hidden Layer



Procedure to Hidden Layer Outputs

Breakthrough: Multi-Layer Perceptron

Fast forward almost two decades to 1986, Geoffrey Hinton, David Rumelhart, and Ronald Williams published a paper "*Learning representations by back-propagating errors*", which introduced:

1. **Backpropagation**, a procedure to ***repeatedly adjust the weights*** so as to minimize the difference between actual output and desired output
2. **Hidden Layers**, which are ***neuron nodes stacked in between inputs and outputs***, allowing neural networks to learn more complicated features (such as XOR logic)

Pseudo-codes

Supervised Training

1. Generate a training pair or pattern:
   - an input $\mathbf{x} = [\, x_1\ x_2 \dots x_n]$
   - a target output y_{target} (known/given)
2. Then, present the network with **x** and allow it to generate an output **y**
3. Compare y with y_{target} to compute the error
4. Adjust weights, **w**, to reduce error
5. Repeat 2-4 multiple times

Perceptron Learning Rule

1. Initialize weights at random
2. For each training pair/pattern ($\mathbf{x}$, $\mathbf{y_{target}}$)
   - Compute output y
   - Compute error, $\delta = (y_{target} - y)$
   - Use the error to update weights as follows:

     $\Delta w = w - w_{old} = \eta * \delta * x$ or $w_{new} = w_{old} + \eta * \delta * x$

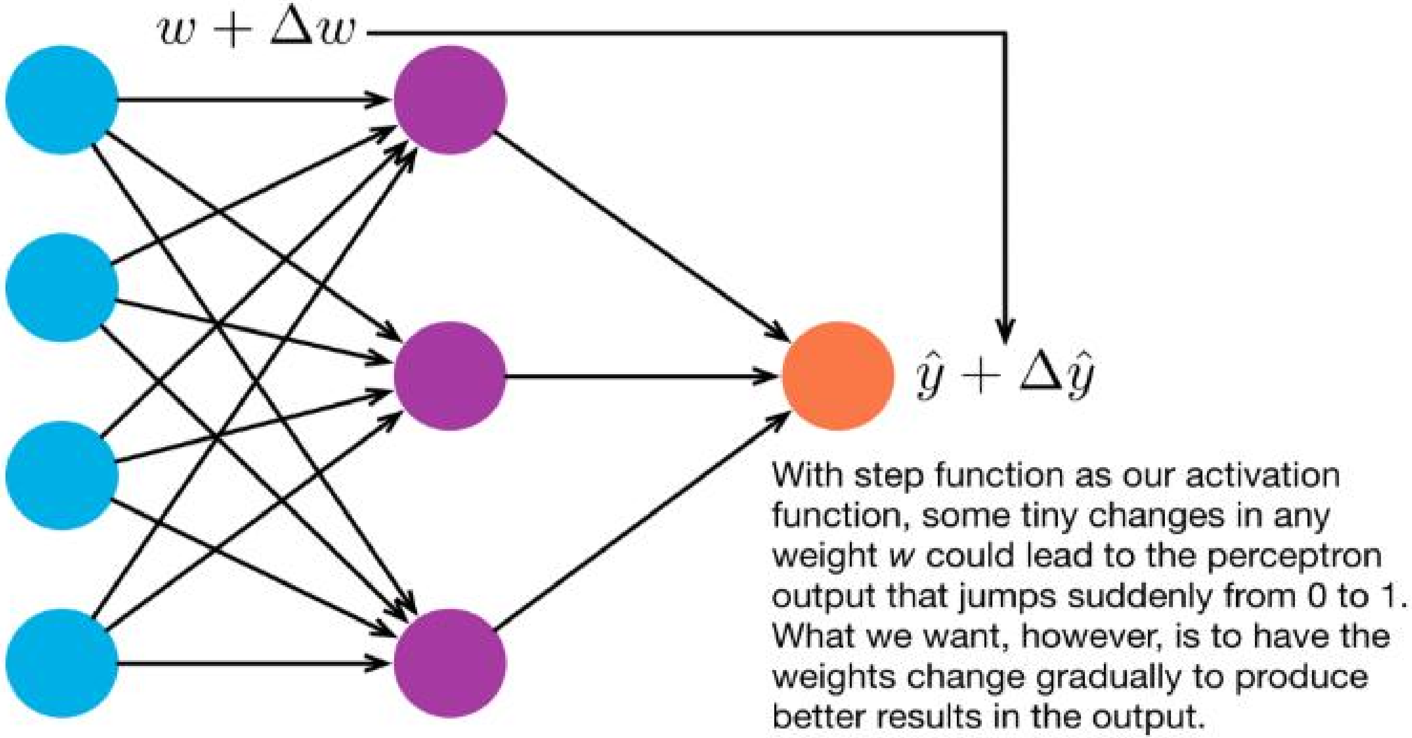
   where η is called the **learning rate** or **step size** and it determines how smoothly the learning process is taking place.
3. Repeat 2 until convergence (i.e. error δ is zero)

The **Perceptron Learning Rule** is then given by

$$w_{new} = w_{old} + \eta * \delta * x$$

where

$$\delta = (y_{target} - y)$$



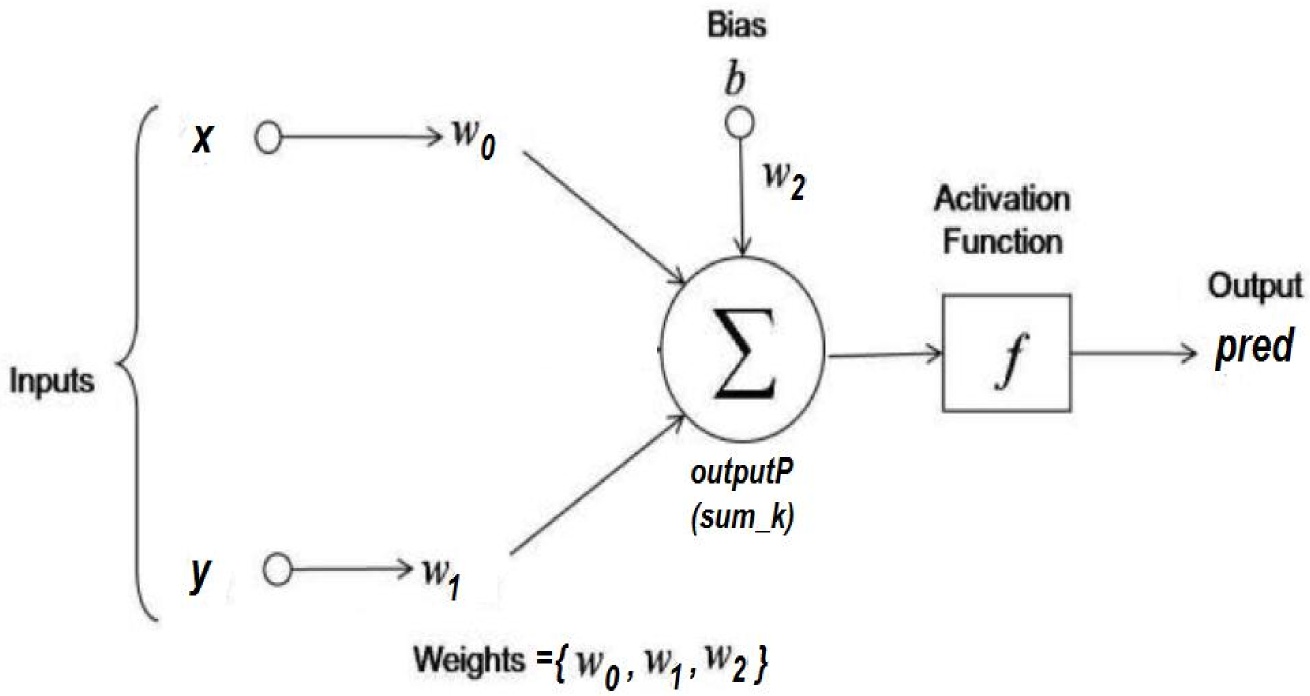
We Want Gradual Change in Weights to Gradually Change Outputs

$$z = \sum_{i=1}^{m} w_i x_i + bias$$

Sigmoid Function is: $\sigma(z) = \frac{1}{1+e^{-z}}$

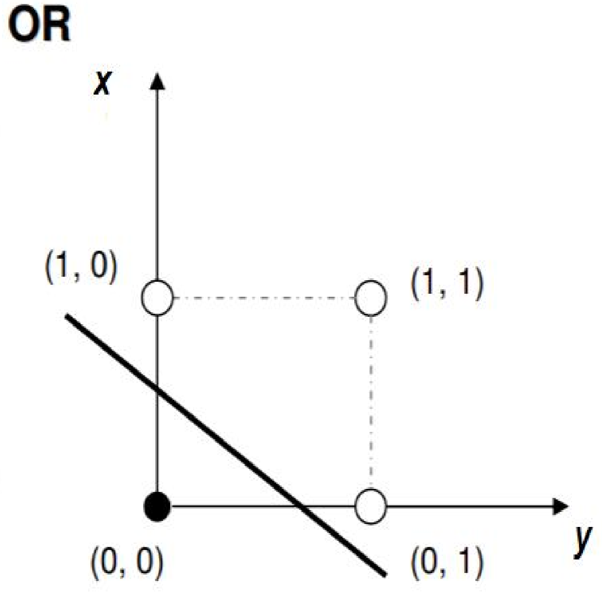
Sigmoid Function

Perceptron Example - OR Classifier

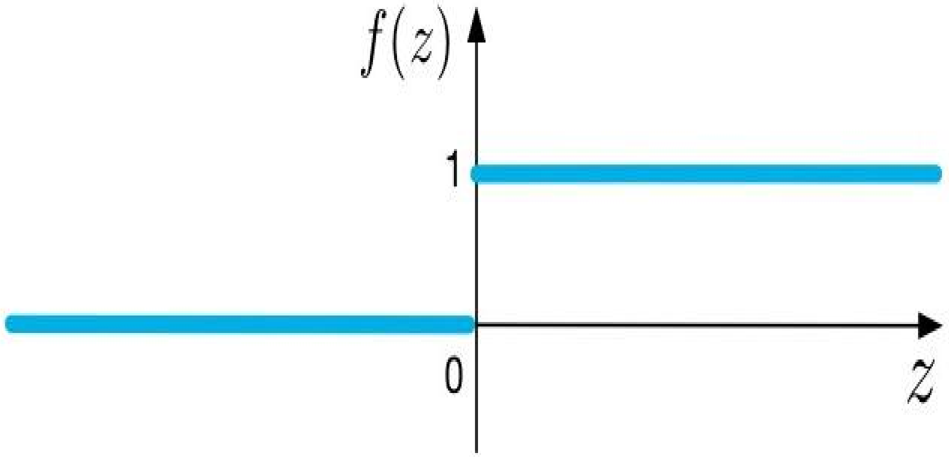


Operations done by a neuron

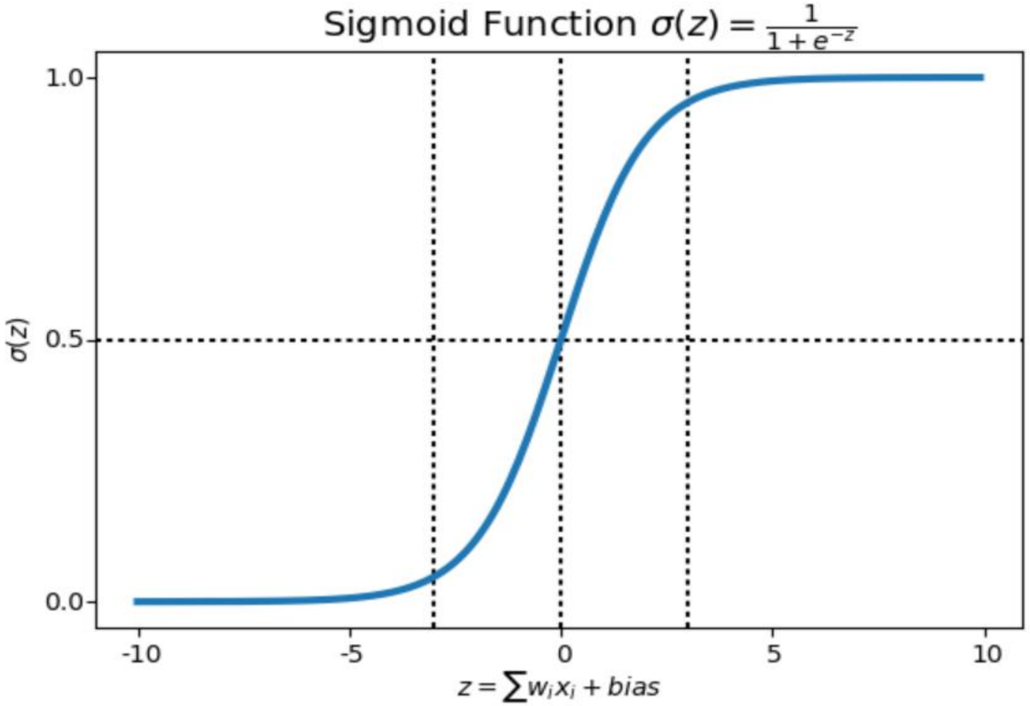
OR		
x	*y*	out
0	0	0
0	1	1
1	0	1
1	1	1



Example (1): **Step** Activation Function
Example (2): **Sigmoid** Activation Function



Step Function



Sigmoid Function