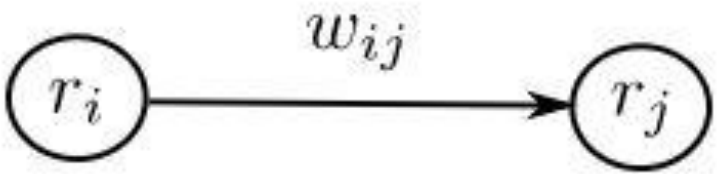
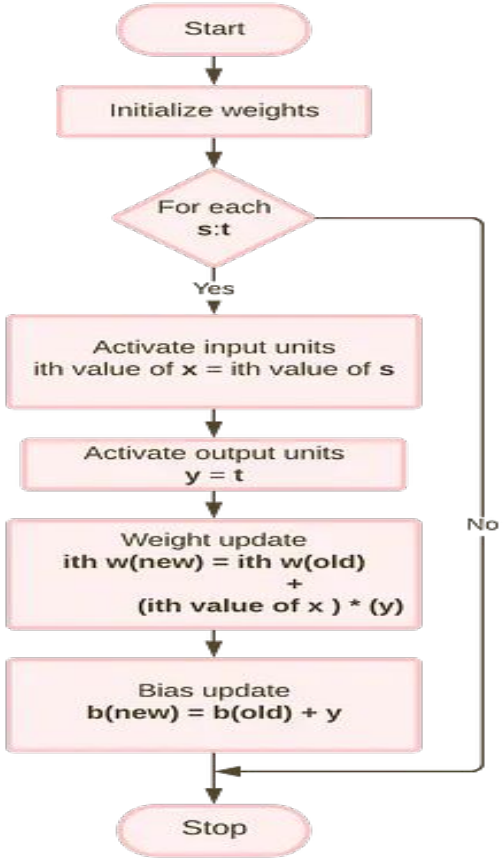
Learning in ANN

Hebb Network

Hebb or Hebbian learning rule comes under **Artificial Neural Network** (ANN) which is an architecture of a large number of interconnected elements called neurons. These neurons process the input received to give the desired output. The nodes or neurons are linked by **inputs**(x1,x2,x3...xn), **connection weights**(w1,w2,w3...wn), and **activation functions**(a function that defines the output of a node).

This network is suitable for bipolar data. The Hebbian learning rule is generally applied to logic gates.

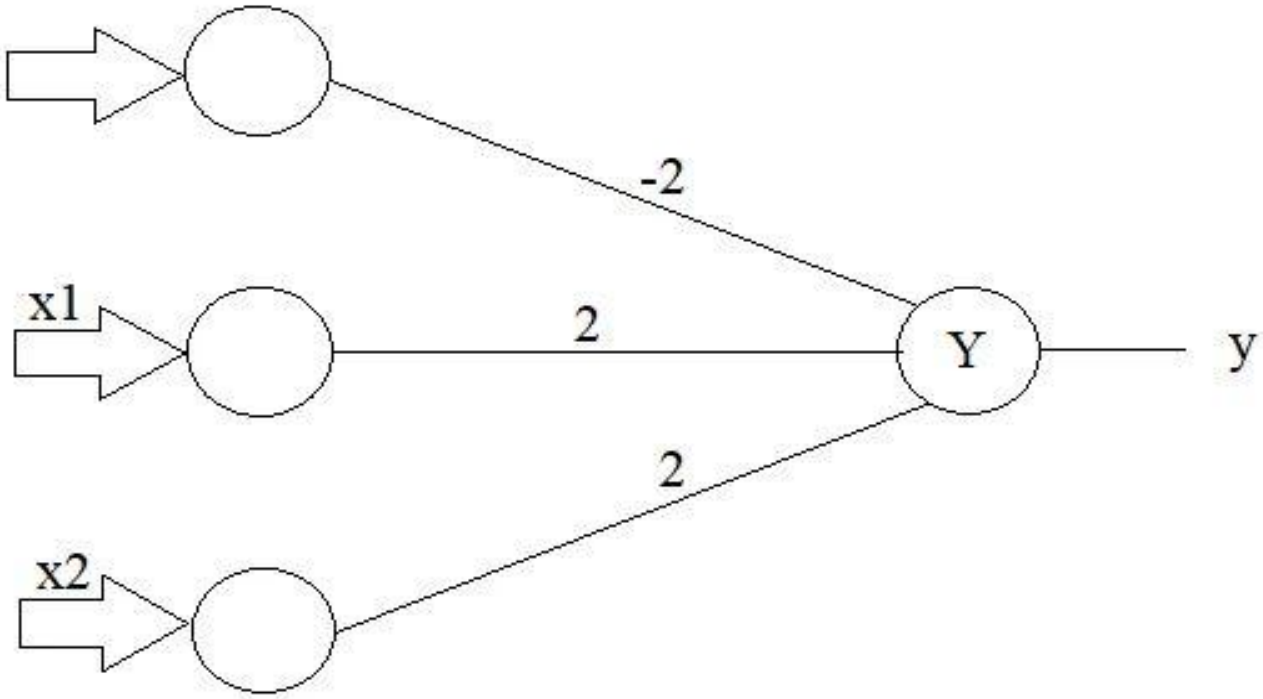



Start
Initialize weights
For each
s:t
Yes
No
Activate input units
ith value of x = ith value of s
Activate output units
y = t
Weight update
ith w(new) = ith w(old)
+
(ith value of x) * (y)
Bias update
b(new) = b(old) + y
Stop

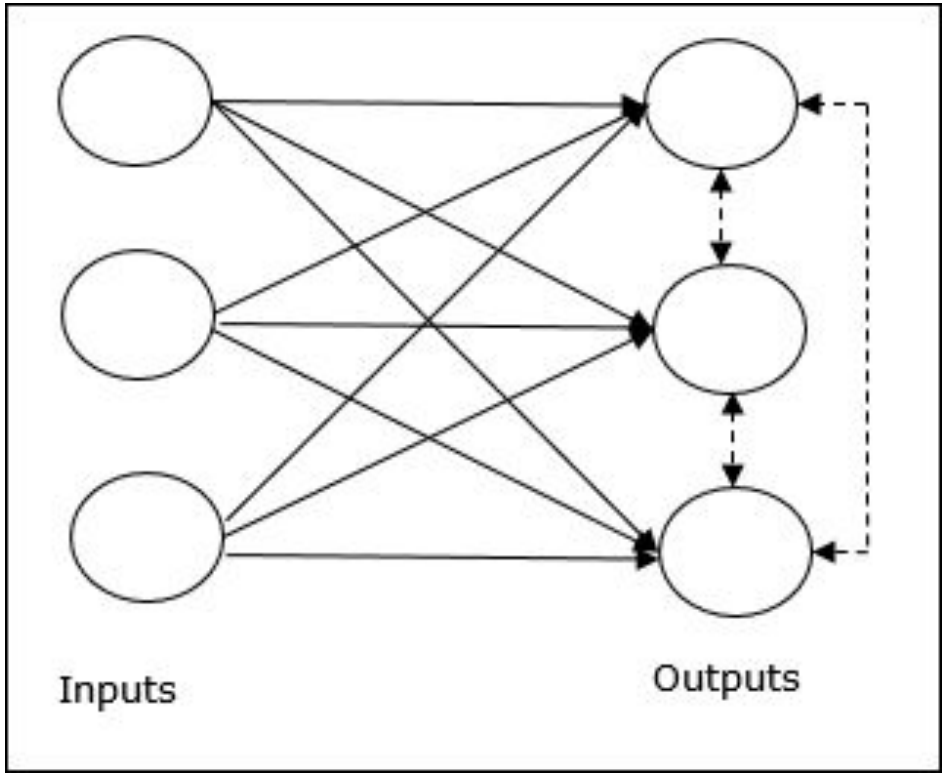
Inputs			Target
x1	x2	b	y
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

Inputs				Weight Changes			Weights		
x1	x2	b	y	Δw1	Δw2	Δb	w1 (0	w2 0	b 0)
1	1	1	1	1	1	1	1	1	1
1	-1	1	-1	-1	1	-1	0	2	0
-1	1	1	-1	1	-1	-1	1	1	-1
-1	-1	1	-1	1	1	-1	2	2	-2

Here the final weights we get are w1=2, w2=2, b=-2



Competitive Learning in ANN



Mathematical Formulation cont.

- Change of weight for the winner
  If a neuron does not respond to the input pattern, then no learning takes place in that neuron. However, if a particular neuron wins, then the corresponding weights are adjusted as follows –

$$\Delta w_{kj} = \begin{cases} -\alpha(x_j - w_{kj}), & if\ neuron\ k\ wins \\ 0 & if\ neuron\ k\ losses \end{cases}$$

  Here α is the learning rate.
  This clearly shows that we are favoring the winning neuron by adjusting its weight and if a neuron is lost, then we need not bother to re-adjust its weight.

Mathematical Formulation

- Condition to be a winner
  Suppose if a neuron $\mathbf{y_k}$ wants to be the winner, then there would be the following condition

$$y_k = \begin{cases} 1 & if\ v_k > v_j\ for\ all\ j, j \neq k \\ 0 & otherwise \end{cases}$$

  It means that if any neuron, say, $\mathbf{y_k}$ wants to win, then its induced local field $theoutputofthesummationunit$, say $\mathbf{v_k}$, must be the largest among all the other neurons in the network.

- Condition of the sum total of weight
  Another constraint over the competitive learning rule is the sum total of weights to a particular output neuron is going to be 1. For example, if we consider neuron **k** then

$$\sum_k w_{kj} = 1 \quad for\ all\ k$$

Step 1: Initialization

We start by initializing the weights of the two neurons to random values. Let's assume:

- Neuron 1 weight: 2
- Neuron 2 weight: 8

Step 2: Presenting the input vector

Now, we present an input vector to the network. Let's say our input vector is '5'.

Step 3: Calculating distance

We calculate the distance between the input vector and the weights of the two neurons. The neuron with the weight closest to the input vector 'wins.' This could be calculated using any distance metric, for example, the absolute difference:

- Neuron 1 distance: $|5-2| = 3$
- Neuron 2 distance: $|5-8| = 3$

Since both distances are equal, we can choose the winner randomly. Let's say Neuron 1 is the winner.

Step 4: Updating weights

We adjust the winning neuron's weight to bring it closer to the input vector. If our learning rate (a tuning parameter in an optimization algorithm that determines the step size at each iteration) is 0.5, the weight update would be:

- Neuron 1 weight: 2 + 0.5*(5-2) = 3.5
- Neuron 2 weight: 8 (unchanged)

Step 5: Iteration

We repeat the process with all the other input vectors in the dataset, updating the weights after each presentation.

Step 6: Convergence

After several iterations (also known as epochs), the neurons' weights will start to converge to the centers of their corresponding input clusters. In this case, with 1-dimensional data ranging from 1 to 10, we could expect one neuron to converge around the lower range (1 to 5) and the other around the higher range (6 to 10).

This process exemplifies how competitive learning works. Over time, each neuron specializes in a different cluster of the data, enabling the system to identify and represent the inherent groupings in the dataset.