Neural Networks

Lecture 13

Supervised Learning and Backpropagation Neural Networks (1)

Md. Mijanur Rahman, Prof. Dr.

Dept. of Computer Science and Engineering, Jatiya Kabi Kazi Nazrul Islam University, Bangladesh.

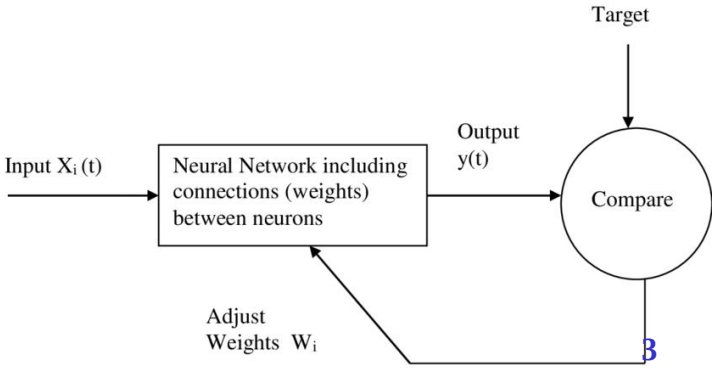
Web: www.mijanrahman.com | Email: mijanjkkniu@gmail.com; mijan@jkkniu.edu.bd

Supervised Learning and Backpropagation Neural Network

- **This chapter covers the following topics:**
  - Supervised Learning Network
  - Perceptron
  - Adaptive Linear Neuron (Adaline)
  - Multiple Adaptive Linear Neuron (Madaline)
  - Generalized Delta Learning Rule
  - Backpropagation Neural Networks
  - Backpropagation Algorithm

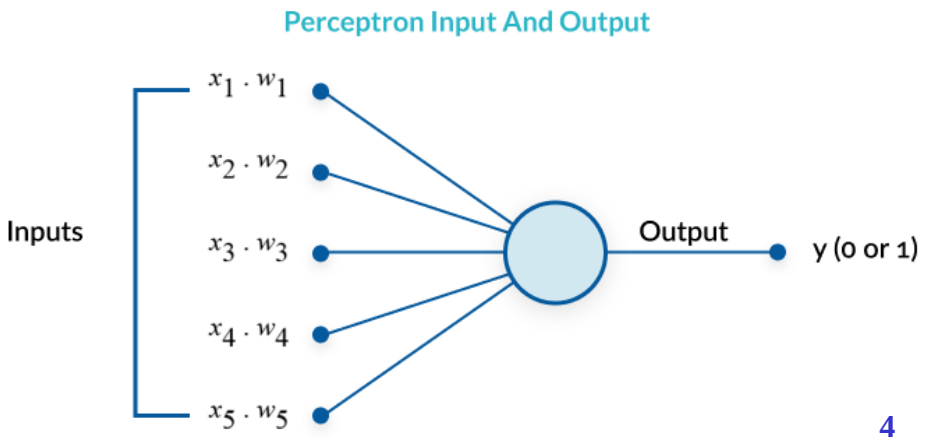
Supervised Learning Networks

- As the name suggests, **supervised learning** takes place under the supervision of a teacher. This learning process is dependent.
- During the training of ANN under supervised learning, the input vector is presented to the network, which will produce an output vector. This output vector is compared with the desired/target output vector.
- An error signal is generated if there is a difference between the actual output and the desired/target output vector.
- On the basis of this error signal, the weights would be adjusted until the actual output is matched with the desired output.



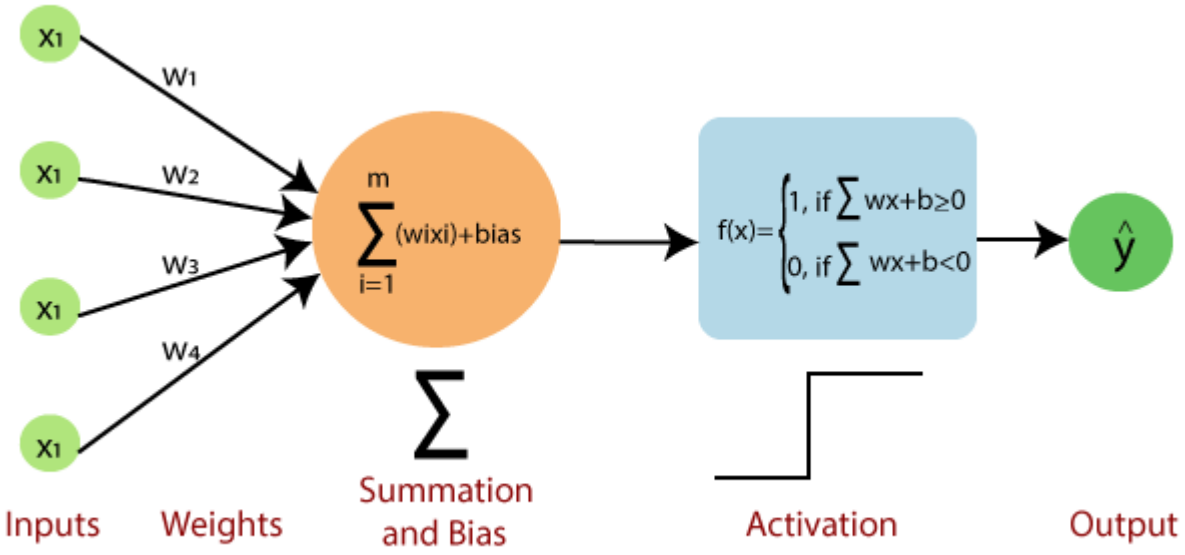
Perceptron…

- Developed by Frank Rosenblatt by using McCulloch and Pitts model, **perceptron** is the basic operational unit of artificial neural networks. It employs supervised learning rule and is able to classify the data into two classes.
- **Operational characteristics of the perceptron:**
  - It consists of a single neuron with an arbitrary number of inputs along with adjustable weights, but the output of the neuron is 1 or 0 depending upon the threshold. It also consists of a bias whose weight is always 1.



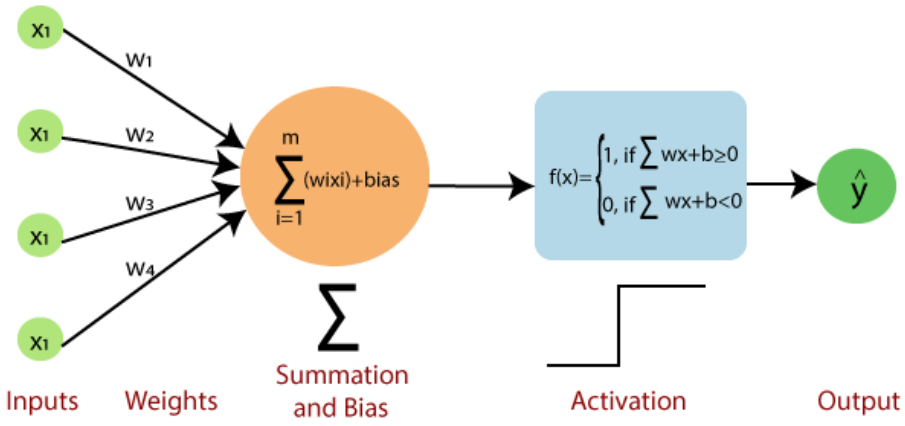
Perceptron…

- Following figure gives a schematic representation of the perceptron:



Perceptron…

- **Perceptron has the following three basic elements −**
  - **Links** − It would have a set of connection links, which carries a weight including a bias always having weight 1.
  - **Adder** − It adds the input after they are multiplied with their respective weights.
  - **Activation function** − It limits the output of neuron. The most basic activation function is a Heaviside step function that has two possible outputs. This function returns 1, if the input is positive, and 0 for any negative input.



Perceptron…

- **Training Algorithm:**
  - Perceptron network can be trained for **single output unit** as well as **multiple output units**.

1. Training Algorithm for Single Output Unit:

Step 1 − Initialize the following to start the training −

- Weights
- Bias
- Learning rate α

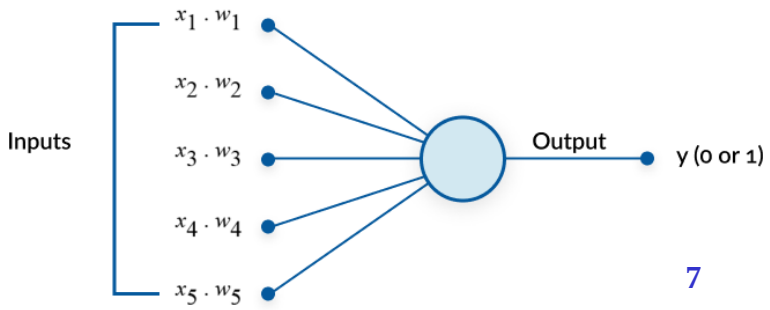
For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 − Continue step 3-8 when the stopping condition is not true.

Step 3 − Continue step 4-6 for every training vector **x**.

Step 4 − Activate each input unit as follows −

$$x_i = s_i \ (i = 1 \ to \ n)$$



Perceptron…

1. Training Algorithm for Single Output Unit:

Step 5 − Now obtain the net input with the following relation −

$$y_{in} = b + \sum_{i}^{n} x_i . w_i$$

Here '**b**' is bias and '**n**' is the total number of input neurons.

Step 6 − Apply the following activation function to obtain the final output.

$$f(y_{in}) = \begin{cases} 1 & if\ y_{in} > \theta \\ 0 & if\ -\theta \leqslant y_{in} \leqslant \theta \\ -1 & if\ y_{in} < -\theta \end{cases}$$

Perceptron…

1. Training Algorithm for Single Output Unit:

Step 7 − Adjust the weight and bias as follows −

Case 1 − if **y ≠ t** then,

$$w_i(new) = w_i(old) + \alpha\, t x_i$$

$$b(new) = b(old) + \alpha t$$

Case 2 − if **y = t** then,

$$w_i(new) = w_i(old)$$

$$b(new) = b(old)$$

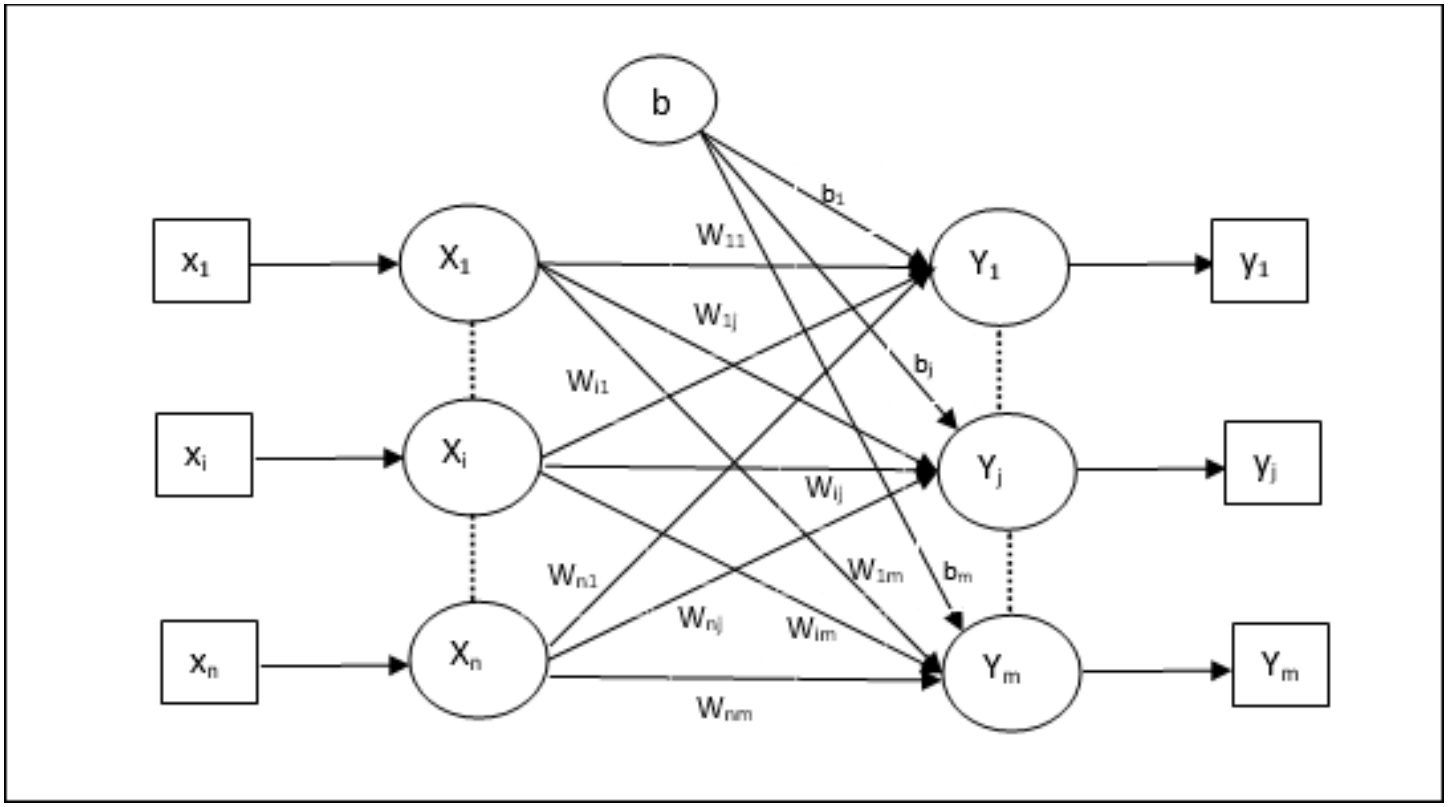
Here **'y'** is the actual output and **'t'** is the desired/target output.

Step 8 − Test for the stopping condition, which would happen when there is no change in weight.

Perceptron…

2. Training Algorithm for Multiple Output Units:

- The following diagram is the architecture of perceptron for multiple output classes.



Perceptron…

2. Training Algorithm for Multiple Output Units:

Step 1 − Initialize the following to start the training −

- Weights
- Bias
- Learning rate α

For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 − Continue step 3-8 when the stopping condition is not true.

Step 3 − Continue step 4-6 for every training vector **x**.

Step 4 − Activate each input unit as follows −

$$x_i = s_i \, (i = 1 \, to \, n)$$

Perceptron…

2. Training Algorithm for Multiple Output Units:

Step 5 – Obtain the net input with the following relation –

$$y_{in} = b + \sum_{i}^{n} x_i \, w_{ij}$$

Here '**b**' is bias and '**n**' is the total number of input neurons.

Step 6 – Apply the following activation function to obtain the final output for each output unit **j = 1 to m** –

$$f(y_{in}) = \begin{cases} 1 & if\ y_{inj} > \theta \\ 0 & if\ -\theta \leqslant y_{inj} \leqslant \theta \\ -1 & if\ y_{inj} < -\theta \end{cases}$$

Perceptron

2. Training Algorithm for Multiple Output Units:

Step 7 – Adjust the weight and bias for **x = 1 to n** and **j = 1 to m** as follows –

Case 1 – if $y_j \neq t_j$ then,

$$w_{ij}(new) = w_{ij}(old) + \alpha\, t_j x_i$$

$$b_j(new) = b_j(old) + \alpha t_j$$

Case 2 – if $y_j = t_j$ then,

$$w_{ij}(new) = w_{ij}(old)$$

$$b_j(new) = b_j(old)$$

Here **'y'** is the actual output and **'t'** is the desired/target output.

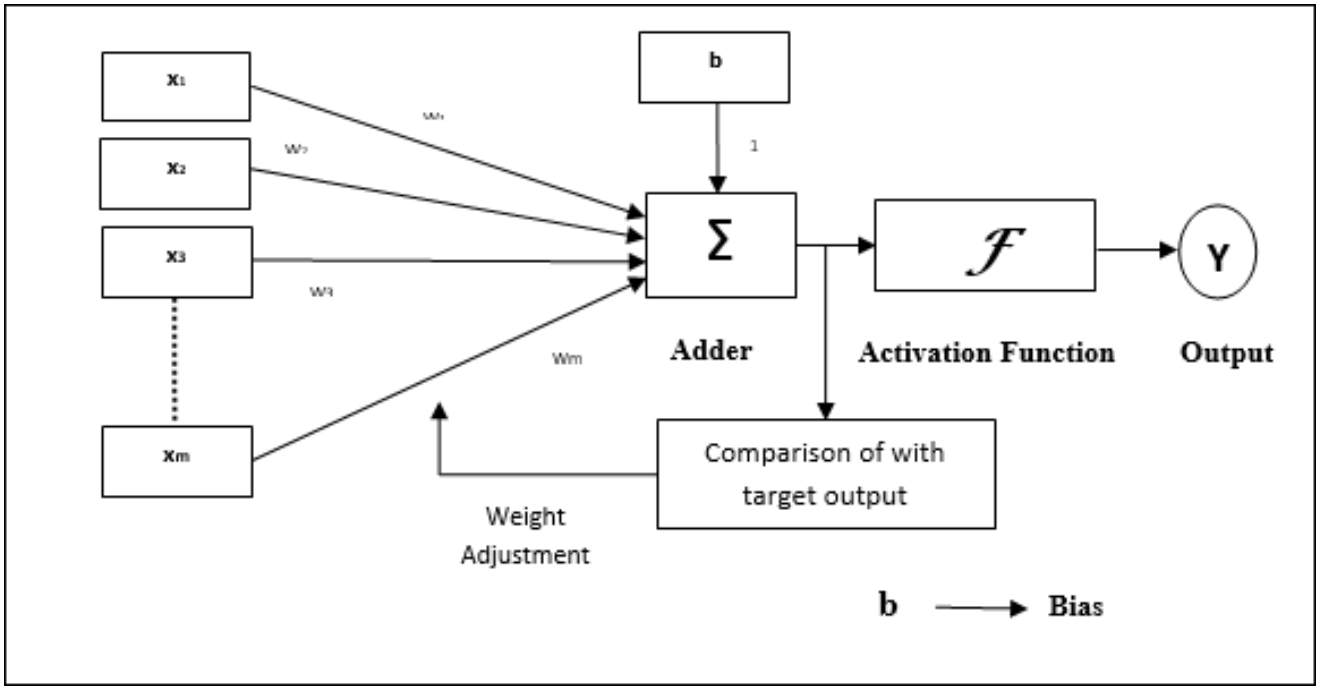
Step 8 – Test for the stopping condition, which will happen when there is no change in weight.

Adaptive Linear Neuron (Adaline)…

- **Adaline** which stands for **Adaptive Linear Neuron**, is a network having a single linear unit. It was developed by Widrow and Hoff in 1960.
- Some important points about **Adaline** are as follows −
  - It uses bipolar activation function.
  - It uses delta rule for training to minimize the Mean-Squared Error (MSE) between the actual output and the desired/target output.
  - The weights and the bias are adjustable.

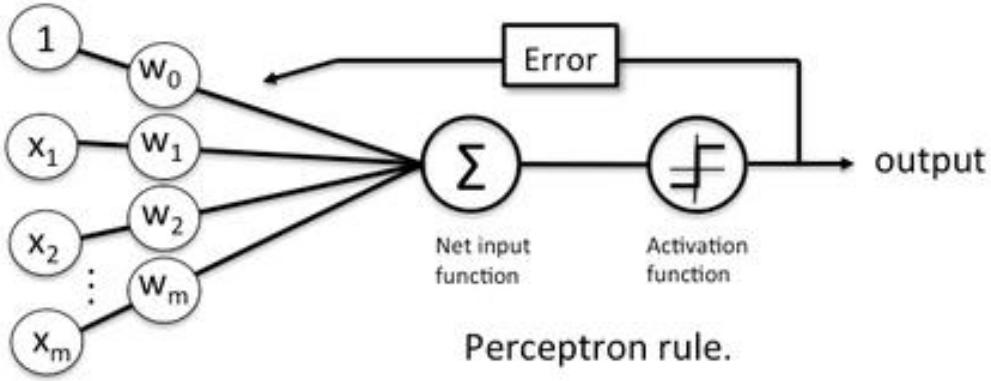
Adaptive Linear Neuron (Adaline)…

- **Adaline : Architecture**
- The basic structure of Adaline is similar to perceptron having an extra feedback loop with the help of which the actual output is compared with the desired/target output. After comparison on the basis of training algorithm, the weights and bias will be updated.

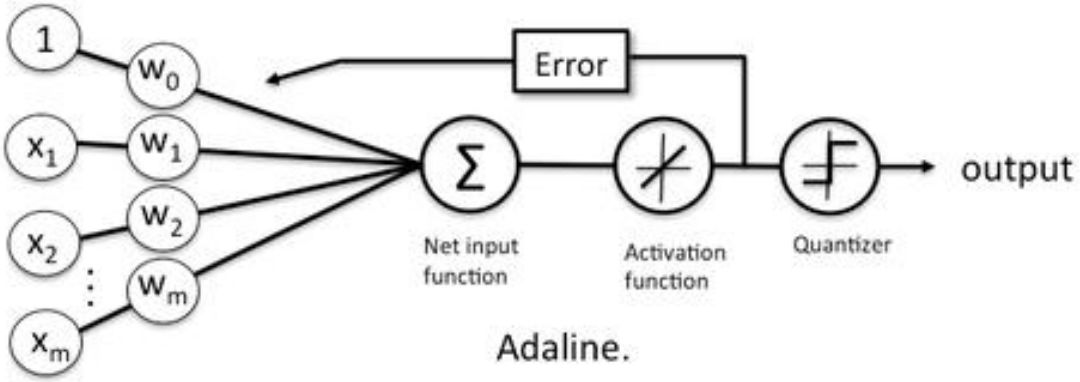


Adaptive Linear Neuron (Adaline)…

- **Perceptron vs Adaline**



Perceptron rule.



Adaline.

Adaptive Linear Neuron (Adaline)…

- **Adaline : Training Algorithm**

Step 1 − Initialize the following to start the training −

- Weights
- Bias
- Learning rate α

For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 − Continue step 3-8 when the stopping condition is not true.

Step 3 − Continue step 4-6 for every bipolar training pair **s:t**.

Step 4 − Activate each input unit as follows −

$$x_i = s_i \, (i = 1 \, to \, n)$$

Adaptive Linear Neuron (Adaline)…

- **Adaline : Training Algorithm**

Step 5 − Obtain the net input with the following relation −

$$y_{in} = b + \sum_{i}^{n} x_i \, w_i$$

Here **'b'** is bias and **'n'** is the total number of input neurons.

Step 6 − Apply the following activation function to obtain the final output −

$$f(y_{in}) = \begin{cases} 1 & if \; y_{in} \geqslant 0 \\ -1 & if \; y_{in} < 0 \end{cases}$$

Adaptive Linear Neuron (Adaline)

- ## Adaline : Training Algorithm

Step 7 – Adjust the weight and bias as follows –

Case 1 – if **y ≠ t** then,

$$w_i(new) = w_i(old) + \alpha(t - y_{in})x_i$$

$$b(new) = b(old) + \alpha(t - y_{in})$$

Case 2 – if **y = t** then,

$$w_i(new) = w_i(old)$$

$$b(new) = b(old)$$

Here **'y'** is the actual output and **'t'** is the desired/target output.

$(t - y_{in})$ is the computed error.

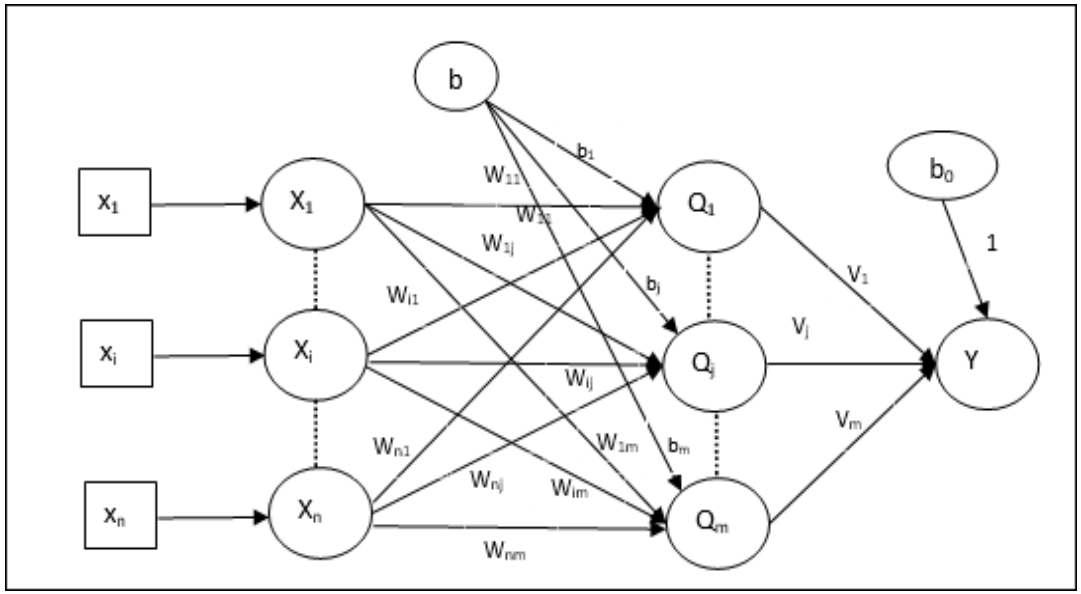
Step 8 – Test for the stopping condition, which will happen when there is no change in weight or the highest weight change occurred during training is smaller than the specified tolerance.

Multiple Adaptive Linear Neuron (Madaline)…

- **Madaline** which stands for **Multiple Adaptive Linear Neuron**, is a network which consists of many Adalines in parallel. It will have a single output unit.
- Some important points about Madaline are as follows −
  - It is just like a multilayer perceptron, where Adaline will act as a hidden unit between the input and the Madaline layer.
  - The weights and the bias between the input and Adaline layers, as in we see in the Adaline architecture, are adjustable.
  - The Adaline and Madaline layers have fixed weights and bias of 1.
  - Training can be done with the help of Delta rule.

Multiple Adaptive Linear Neuron (Madaline)…

- **Madaline** : **Architecture**
- The architecture of Madaline consists of **"n"** neurons of the input layer, **"m"** neurons of the Adaline layer, and 1 neuron of the Madaline layer.
- The Adaline layer can be considered as the hidden layer as it is between the input layer and the output layer, i.e. the Madaline layer.



Multiple Adaptive Linear Neuron (Madaline)…

- **Madaline : Training Algorithm**
- By now we know that **only the weights and bias between the input and the Adaline layer are to be adjusted**, and **the weights and bias between the Adaline and the Madaline layer are fixed**.

Step 1 − Initialize the following to start the training −

- Weights
- Bias
- Learning rate α

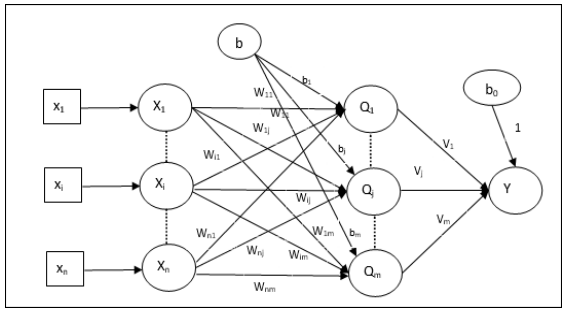
For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 − Continue step 3-8 when the stopping condition is not true.

Step 3 − Continue step 4-7 for every bipolar training pair **s:t**.

Step 4 − Activate each input unit as follows −

$$x_i = s_i \; (i = 1 \; to \; n)$$



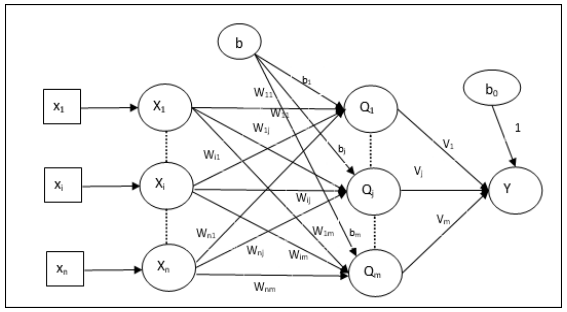
Multiple Adaptive Linear Neuron (Madaline)…

- **Madaline : Training Algorithm**

Step 5 − Obtain the net input at each hidden layer, i.e. the Adaline layer with the following relation −

$$Q_{inj} = b_j + \sum_{i}^{n} x_i \, w_{ij} \quad j = 1 \, to \, m$$

Here **'b'** is bias and **'n'** is the total number of input neurons.



Multiple Adaptive Linear Neuron (Madaline)…

- **Madaline : Training Algorithm**

  Step 6 − Apply the following activation function to obtain the final output at the Adaline and the Madaline layer −

$$f(x) = \begin{cases} 1 & if\ x \geqslant 0 \\ -1 & if\ x < 0 \end{cases}$$

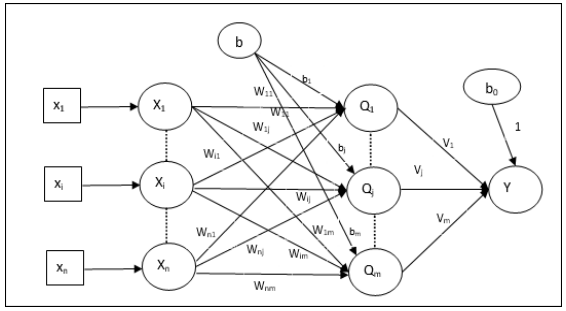
Output at the hidden (Adaline) unit

$$Q_j = f(Q_{inj})$$

Final output of the network

$$y = f(y_{in})$$

i.e. $y_{inj} = b_0 + \sum_{j=1}^{m} Q_j v_j$



Multiple Adaptive Linear Neuron (Madaline)…

- **Madaline : Training Algorithm**

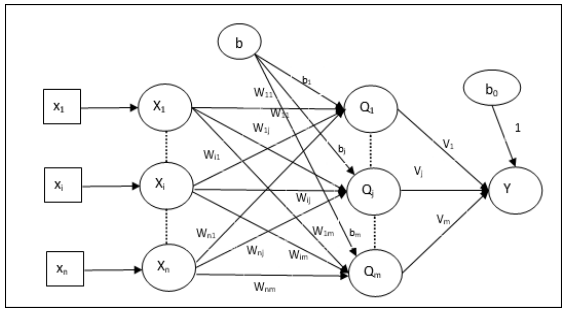
  Step 7 − Calculate the error and adjust the weights as follows −

  Case 1 − if **y ≠ t** and **t = 1** then,

$$w_{ij}(new) = w_{ij}(old) + \alpha(1 - Q_{inj})x_i$$

$$b_j(new) = b_j(old) + \alpha(1 - Q_{inj})$$

  In this case, the weights would be updated on $\mathbf{Q_j}$ where the net input is close to 0 because **t = 1**.



Multiple Adaptive Linear Neuron (Madaline)…

- **Madaline : Training Algorithm**

Case 2 − if **y ≠ t** and **t = -1** then,

$$w_{ik}(new) = w_{ik}(old) + \alpha(-1 - Q_{ink})x_i$$

$$b_k(new) = b_k(old) + \alpha(-1 - Q_{ink})$$

In this case, the weights would be updated on $\mathbf{Q_k}$ where the net input is positive because **t = -1**.

Here **'y'** is the actual output and **'t'** is the desired/target output.

Case 3 − if **y = t** then

There would be no change in weights.

Step 8 − Test for the stopping condition, which will happen when there is no change in weight or the highest weight change occurred during training is smaller than the specified tolerance.

SUPERVISED LEARNING AND BACKPROPAGATION NETWORK PARADIGMS

To be continued…