Artificial Intelligence (CS-323) Labs

Instructor Name: Ms. Hameeza Ahmed

Lab 4
Learning Algorithmic Design of Artificial Neural Network

Adaptive Linear Neuron (Adaline)

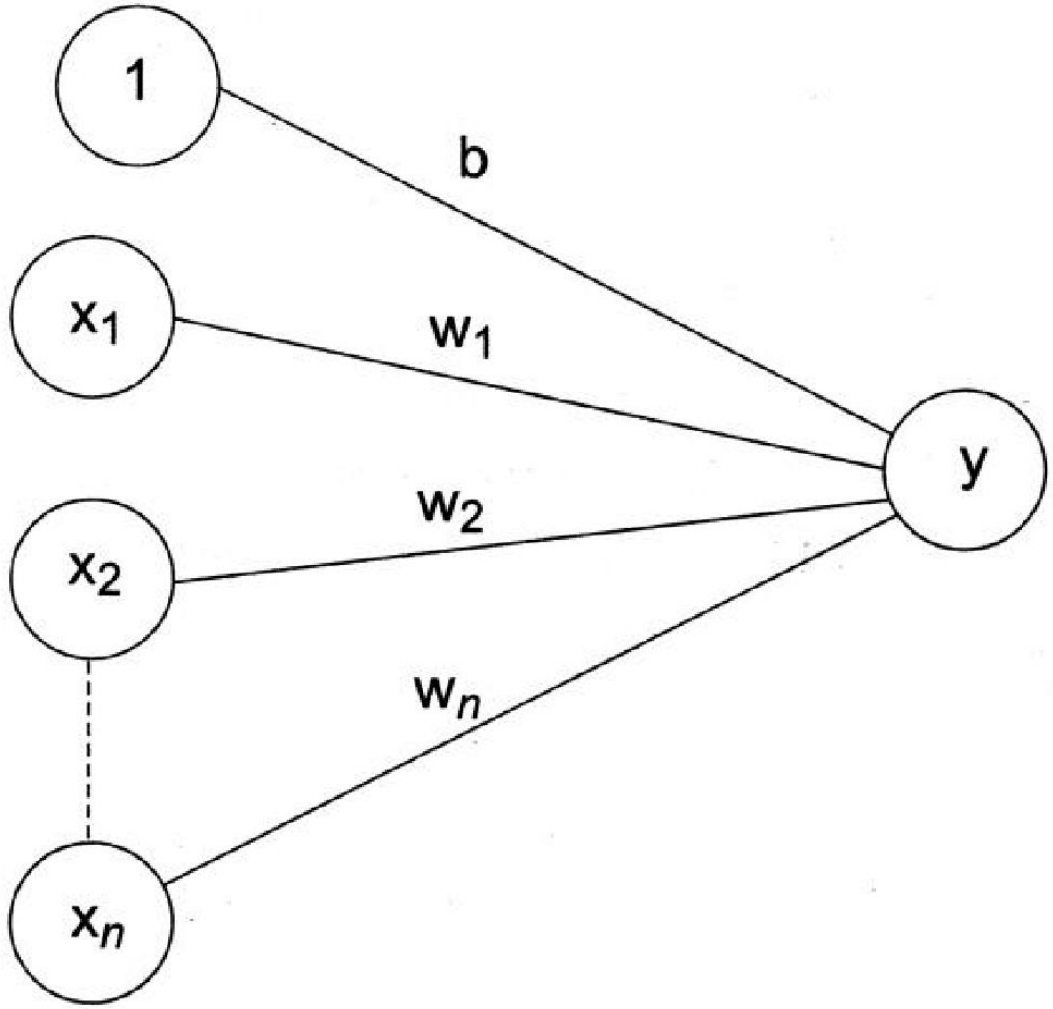
- Widrow and Hoff [1960] development developed the learning rule that is very closely related to the perceptron learning rule.
- The rule, called Delta rule, adjusts the weights to reduce the difference between the net input to the output unit, and the desired output, which results in the least mean squared (LMS error).
- Adaline (Adaptive Linear Neuron) and Madaline (Multilayered Adaline) networks use this LMS learning rule and are applied to various neural network applications.

Adaline

- Adaline is found to use bipolar activations for its input signals and target output. The weights and the bias of the Adaline are adjustable.
- The learning rule used can be called as Delta rule, Least Mean Square rule or Widrow-Hoff rule.
- The derivation of this rule with single output unit, several output units.
- Since the activation function is an identity function, the activation of the unit is its net input.
- When Adaline is to be used for pattern classification, then, after training, a threshold function is applied to the net input to obtain the activation.
- The Adaline unit can solve the problem with linear separability if it occurs.

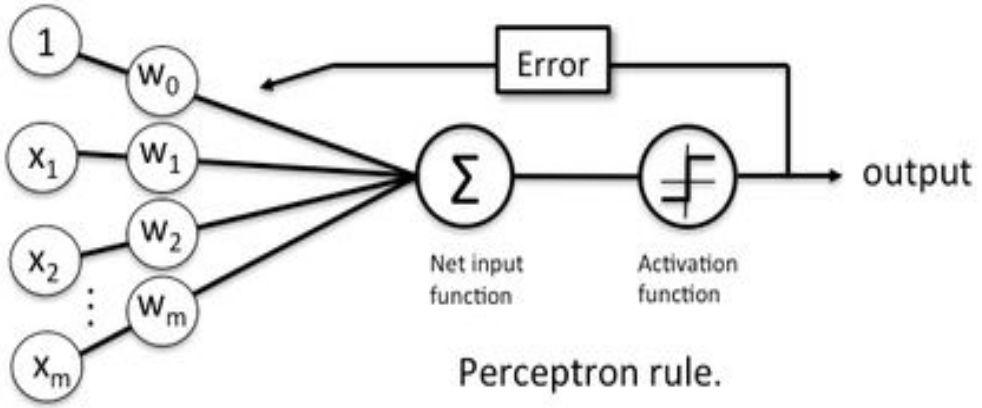
Adaline Architecture

- The Adaline has only one output unit.
- This output unit receives input from several units and also from bias; whose activation is always +1.
- The adaline also resembles a single layer network. It receives input from several neurons.
- It should be noted that it also receives input from the unit which is always _+1‘ called as bias.
- The bias weights are also trained in the same manner as the other weights.
- In Fig. 1, an input layer with *x1… xi…. xn* and bias, an output layer with only one output neuron is present.
- The link between the input and output neurons possess weighted interconnections.
- These weights get changed as the training progresses.

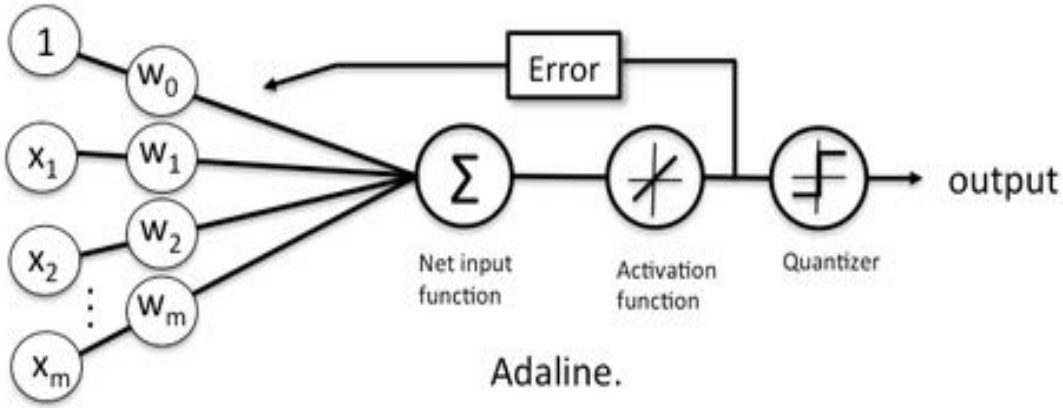


Adaline vs Perceptron

- The Adaline (Adaptive Linear Element) and the Perceptron are both linear classifiers when considered as individual units.
- They both take an input, and based on a threshold, output e.g. either a 0 or a 1.
- The main difference between the two, is that
  - a **Perceptron** takes that binary response (like a classification result) and computes an error used to update the weights,
  - The Perceptron uses the class labels to learn model coefficients.
  - whereas an **Adaline** uses a continuous response value to update the weights (so before the binarized output is produced).
  - Adaline uses continuous predicted values (from the net input) to learn the model coefficients, which is more "powerful" since it tells us by "how much" the model is right or wrong.
  - So, in the perceptron, as illustrated, we simply use the predicted class labels to update the weights, and in Adaline, we use a continuous response.
- The fact that the Adaline does this, allows its updates to be more representative of the actual error, before it is thresholded, which in turn allows a model to converge more quickly.



Perceptron rule.



Adaline.

Adaline vs Perceptron

- Both learning algorithms can actually be summarized by 4 simple steps:
- Initialize the weights to 0 or small random numbers.
- For each training sample:
  - Calculate the output value.
  - Update the weights.
- We write the weight update in each iteration as:
- $w_j := w_j + \Delta w_j$
- Where
- $\Delta w_j = \eta\left(target^{(i)} - \text{output}^{(\text{i})}\right)\text{x}_\text{j}^{(\text{i})}$
- The “output” is the continuous net input value in Adaline and the predicted class label in case of the perceptron; eta is the learning rate.

Adaline Algorithm

- Basically, the initial weights of Adaline network have to be set to small random values and not to zero as discussed in Hebb or perceptron networks, because this may influence the error factor to be considered.
- After the initial weights are assumed, the activations for the input unit are set.
- The net input is calculated based on the training input patterns and the weights.
- By applying delta learning rule, the weight updation is being carried out.
- The training process is continued until the error, which is the difference between the target and the net input becomes minimum.

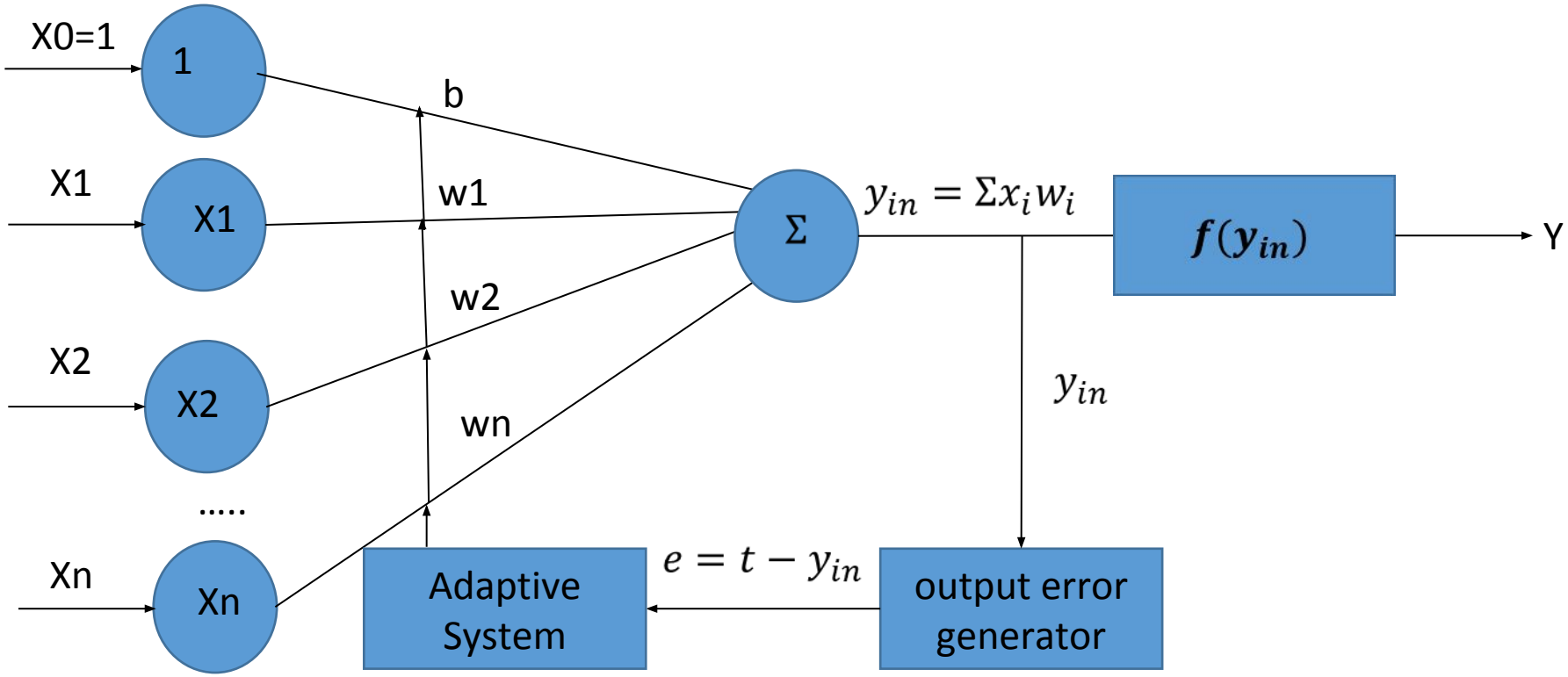
Pseudo Code for Adaline network

- The step based training algorithm for an adaline is as follows:
- **Training Algorithm**
- The following is the pseudo code for training an Adaline network.
- **Step 1:** Initialize weights and bias (not zero but small random values are used). Set learning rate α.
- **Step 2:** While stopping condition is false, do Step 3-7.
- **Step 3:** For each bipolar training pair $s:\ t$, perform Steps 4-6.
- **Step 4:** Set activations of input units $x_i\ =\ s_i\ for\ i\ 1\ to\ n.$
- **Step 5:** Compute net input to output unit $y_{in}\ =\ b\ +\sum_{i=1}^{n} x_i w_i$, **n is number of neurons**
- **Step 6**: Update bias and weights, $i\ =\ 1\ to\ n$
  - $w_i(new)\ =\ w_i\ (old)\ +\alpha(t-\ y_{in})x_i$, $(t-y_{in})$ is error value, t is desired output, yin is net input to output neuron, xi is input vector
  - $b(new)\ =\ b(old)+\ \alpha(t-\ y_{in})$
- **Step 7:** Test for stopping condition.
  - The stopping condition may be when the weight change reaches small level or number of iterations etc.

Pseudo Code for Adaline network

- **Testing Algorithm**
- The application procedure, which is used for testing the trained network is as follows. It is mainly based on the bipolar activation.
- **Step 1:** Initialize weights obtained from the training algorithm.
- **Step** 2: For each bipolar input vector x, perform Steps 3-5.
- **Step 3:** Set activations of input unit.
- **Step 4:** Calculate the net input to the output unit. $y_{in} = b + \sum_{i=1}^{n} x_i w_i$
- **Step 5:** Finally apply the activations to obtain the output y.
  - $y = f(y_{in}) = \begin{cases} 1, & if\ y_{in} \geq 0 \\ -1, & if\ y_{in} < 0 \end{cases}$

Adaline Model



Example

- **OR function with bipolar inputs and targets using Adaline network.**
- The truth table for the OR function with bipolar inputs and targets is given as,
- w1=w2=b=0.2, alpha=0.2
- The weights are calculated until least square error is obtained
- **For first input**
- X1=-1, X2=-1, t=-1
- $y_{in} = b + w_1x_1 + w_2x_2$
- $= 0.2 + 0.2 * -1 + 0.2 * -1$
- $= 0.2 - 0.2 - 0.2$
- $= -0.2$
- $t - y_{in} = -1 - (-0.2) = -1 + 0.2 = -0.8 \neq 0$
- **Update weights**
- $w_i(new) = w_i(old) + \alpha(t - y_{in})x_i$
- $w_1(new) = 0.2 + 0.2(-1 + 0.2)(-1) = 0.36$
- $w_2(new) = 0.2 + 0.2(-1 + 0.2)(-1) = 0.36$
- $b(new) = b(old) + \alpha(t - y_{in}) = 0.2 + 0.2(-1 + 0.2) = 0.04$
- $\Delta w_1 = \alpha(t - y_{in})x_i = 0.16, \Delta w_2 = \alpha(t - y_{in})x_2 = 0.16, \Delta b = \alpha(t - y_{in}) = -0.16$
- **Compute error**
- $E = (t - y_{in})^2 = (-0.8)^2 = 0.64$

X1	X2	Y
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

Example

- **For second input**
- X1=-1, X2=1, t=1
- w1=w2=0.36,b=0.04, alpha=0.2
- $y_{in} = b + w_1 x_1 + w_2 x_2$
- $= 0.04 + 0.36 * -1 + 0.36 * 1$
- $= 0.04 - 0.36 + 0.36$
- $= 0.04$
- $t - y_{in} = 1 - (0.04) = 0.96 \neq 0$
- **Update weights**
- $w_i(new) = w_i(old) + \alpha(t - y_{in})x_i$
- $w_1(new) = 0.36 + 0.2(1 - 0.04)(-1) = 0.168$
- $w_2(new) = 0.36 + 0.2(1 - 0.04)(1) = 0.552$
- $b(new) = b(old) + \alpha(t - y_{in}) = 0.04 + 0.2(1 - 0.04) = 0.232$
- $\Delta w_1 = \alpha(t - y_{in})x_i = -0.008$, $\Delta w_2 = \alpha(t - y_{in})x_2 = 0.008$, $\Delta b = \alpha(t - y_{in}) = 0.008$
- **Compute error**
- $E = (t - y_{in})^2 = (0.96)^2 = 0.9216$

X1	X2	Y
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

Tasks

- Develop a MATLAB program for OR function with bipolar inputs and targets using Adaline network. The truth table for the OR function with bipolar inputs and targets is given as,
- Stop the training after 10 iterations.
- Stop the training when total squared error becomes 0.5.

X1	X2	Y
-1	-1	-1
-1	1	1
1	-1	1
1	1	1