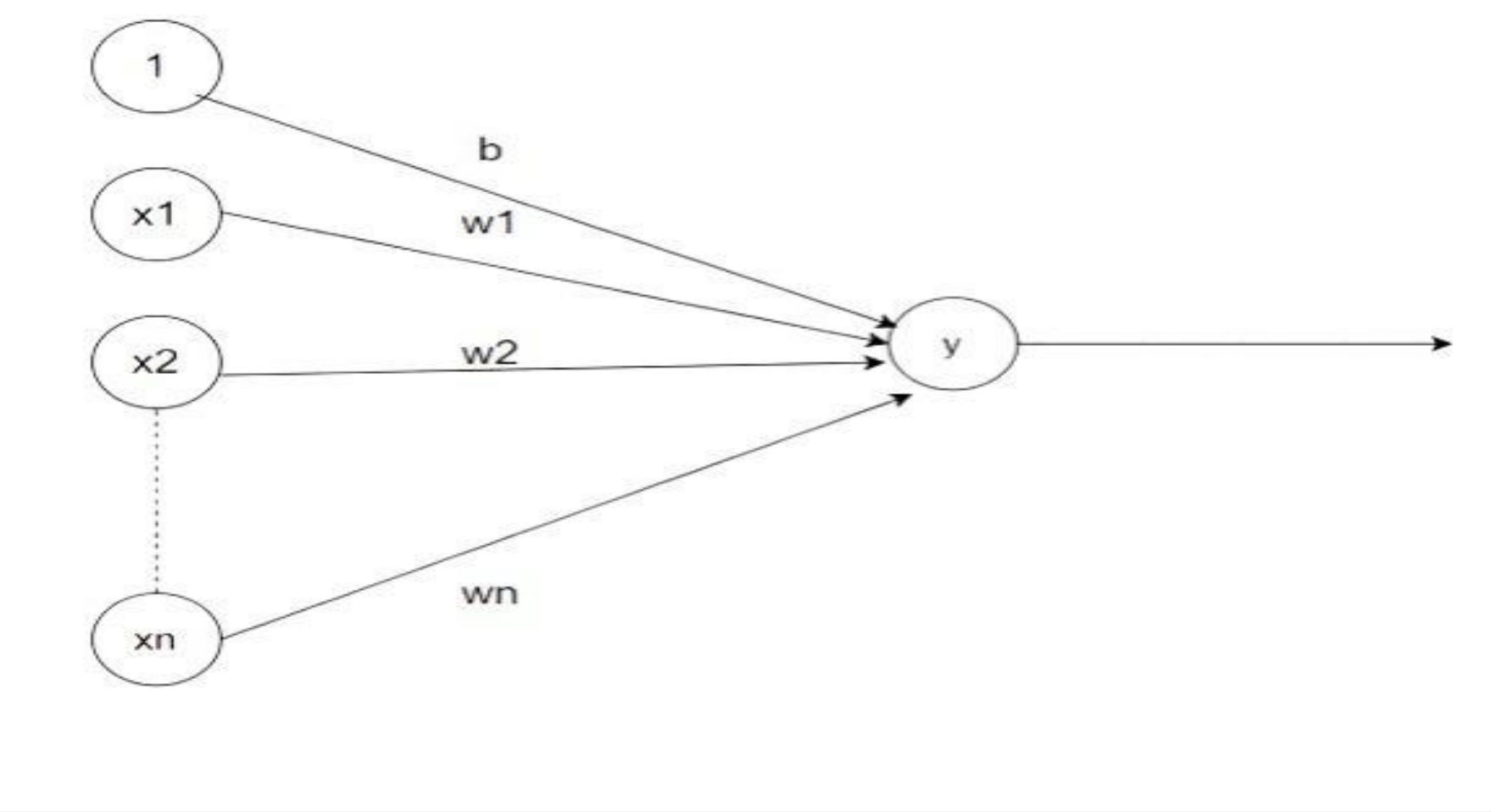
ADALINE and MADALINE

ADALINE

- Bernard Widrow and Ted Hoff developed the adaptive linear neuron (ADALINE) model shortly after the perceptron network in 1960.
- ADALINE is a network with a single linear unit that means only one output.
- ADALINE is limited to linearly separable problems.
- Adaline is a binary classifier as the perceptron.
- Weights between the input unit and output unit are adjustable.

ADALINE Network



Continued (ADALINE)

- The core idea of ADALINE is to minimize the difference between the actual output and the desired output by adjusting the weights in the network.
- This difference is often quantified using the Instantaneous Squared Error and the weight adjustment is done using the Widrow-Hoff or LMS (Least Mean Squares) rule.
- Unlike perceptron, ADALINE uses a linear activation function, i.e., the output is the same as the net input.

Least Mean Squares (LMS)

- The LMS algorithm adjusts the model's parameters (like weights in a neural network) to minimize the mean of the squared errors across all training examples. Hence, it is called the Least Mean Squares algorithm.
- In essence, the LMS algorithm works by iteratively updating the model parameters to reduce the average of the squared differences between the actual output and the predicted output, aiming to find the best fit for the data.
- LMS algorithm is also known as delta or Widrow-Hoff learning rule.

The *least-mean-square (LMS) algorithm* is based on the use of *instantaneous values* for the cost function, namely,

$$\mathscr{E}(\mathbf{w}) = \frac{1}{2} e^2(n)$$

where $e(n)$ is the error signal measured at time n. Differentiating $\mathscr{E}(\mathbf{w})$ with respect to the weight vector $\mathbf{w}$ yields

$$\frac{\partial \mathscr{E}(\mathbf{w})}{\partial \mathbf{w}} = e(n) \frac{\partial e(n)}{\partial \mathbf{w}}$$

As with the linear least-squares filter, the LMS algorithm operates with a linear neuron so we may express the error signal as

$$e(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n)$$

Hence,

$$\frac{\partial e(n)}{\partial \mathbf{w}(n)} = -\mathbf{x}(n)$$

and

$$\frac{\partial \mathscr{E}(\mathbf{w})}{\partial \mathbf{w}(n)} = -\mathbf{x}(n)e(n)$$

Using this latter result as an *estimate* for the gradient vector, we may write

$$\hat{\mathbf{g}}(n) = -\mathbf{x}(n)e(n)$$

- Using above two equations, for the method of gradient descent, we may formulate the LMS algorithm as follow:

$$\hat{\boldsymbol{w}}(\boldsymbol{n}+\mathbf{1}) = \hat{\boldsymbol{w}}(\boldsymbol{n}) + \boldsymbol{\eta}\boldsymbol{x}(\boldsymbol{n})\boldsymbol{e}(\boldsymbol{n})$$

where η is the learning rate. Unlike Gradient descent algorithm, in the LMS algorithm the weight vector $\hat{w}(n)$ traces a random trajectory. For this reason, LMS algorithm is referred to as "stochastic gradient algorithm". The LMS algorithm as follows:

Training Sample: Input signal vector $= \mathbf{x}(n)$
Desired response $= d(n)$

User-selected parameter: η

Initialization. Set $\hat{\mathbf{w}}(0) = \mathbf{0}$.

Computation. For $n = 1, 2, \ldots$, compute

$$e(n) = d(n) - \hat{\mathbf{w}}^T(n)\mathbf{x}(n)$$

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\,\mathbf{x}(n)e(n)$$

Limitations of ADALINE:

- Linearly Separable Problems: ADALINE can only solve linearly separable problems because it uses a linear activation function. Complex non-linear problems cannot be handled by a single ADALINE neuron.
- Slow Convergence: For some problems, the convergence can be slow depending on the learning rate and the initial weights.

Multiple Adaptive Linear Neurons (MADALINE)

- The multiple adaptive linear neurons (Madaline) model consists of many Adaline's in parallel with a single output unit whose value is based on certain selection rules. It may use majority vote rule.
- On using this rule, the output would have as answer either true or false.
- The weights between the input layer and the Adaline layer are adjusted during the training process.
- The training process for a Madaline system is similar to that of an Adaline.

Architecture of Madaline layer

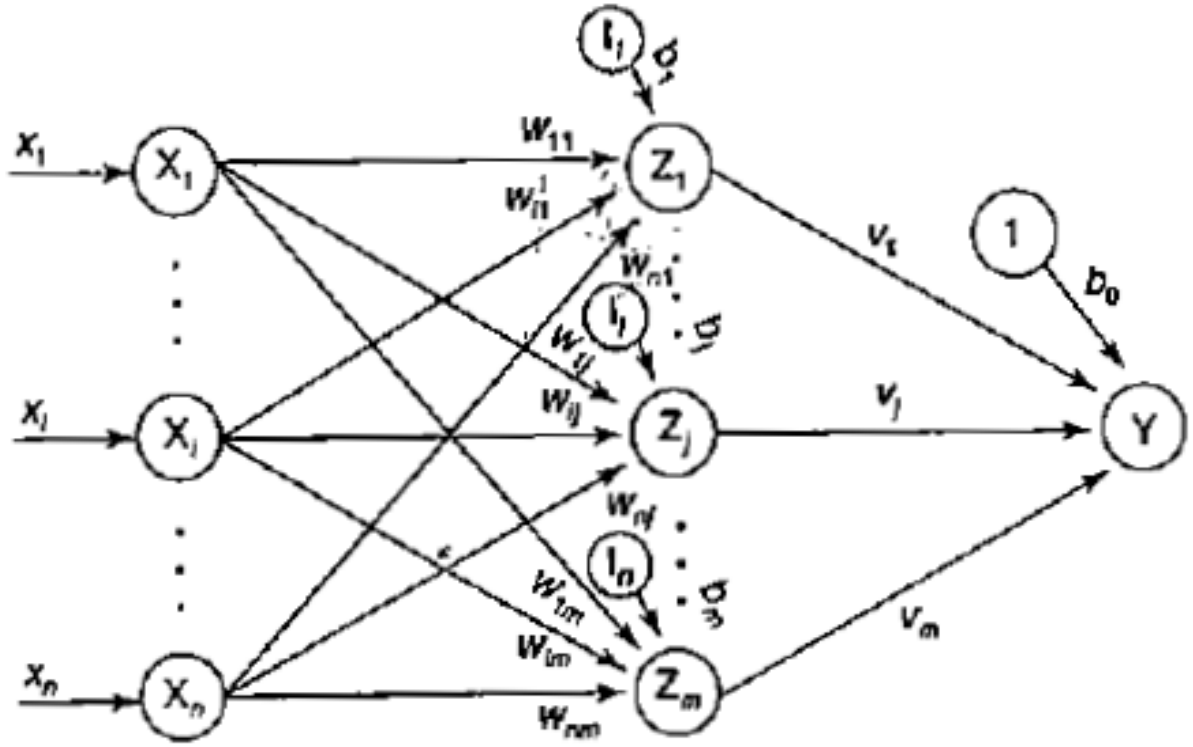


Figure 3-7 Architecture of Madaline layer.

- A simple Madaline architecture is shown in the above Figure, which consists of "n" unit of input layer,"m" units of Adaline layer and "1" unit of the Madaline layer. Each neuron in the Adaline and Madaline layers has a bias of excitation 1.
- The Adaline layer is present between the input layer and the Madaline (output) layer; hence, the Adaline layer can be considered a hidden layer.
- In case of training, the weight between the input layer and the hidden layer are adjusted, and the weights between the hidden layer and the output layer are fixed.

MADALINE Algorithm

- In this training algorithm, only the weights between the hidden layer and the input layer are adjusted, and the weight for the output units are fixed. The weights $\boldsymbol{v_1, v_2, \ldots, v_m}$ and the bias $\boldsymbol{b_0}$ that enter into output unit **Y** are determined so that the response of unit **Y** is 1. Thus, the weights entering **Y** unit may be taken as $v_1 = v_2, = \cdots = v_m = \frac{1}{2}$ *and the bias* b_0 *is taken as* $1/2$.
- The activation for the Adaline (hidden) and Madaline (output) units is given by: $f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$

Step 0: Initialize the weights. The weights entering the output unit are set as above. Set initial small random values for Adaline weights. Also set initial learning rate α.

Step 1: When stopping condition is false, perform Steps 2–3.

Step 2: For each bipolar training pair $s:t$, perform Steps 3–7.

Step 3: Activate input layer units. For $i = 1$ to n,

$$x_i = s_i$$

Step 4: Calculate net input to each hidden Adaline unit:

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

Step 5: Calculate output of each hidden unit:

$$z_j = f(z_{inj})$$

Step 6: Find the output of the net:

$$y_{in} = b_0 + \sum_{j=1}^{m} z_j v_j$$

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

Step 7: Calculate the error and update the weights.

1. If $t = y$, no weight updation is required.
2. If $t \neq y$ and $t = +1$, update weights on z_j, where net input is closest to 0 (zero):

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

3. If $t \neq y$ and $t = -1$, update weights on units z_k whose net input is positive:

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

Step 8: Test for the stopping condition. (If there is no weight change or weight reaches a satisfactory level, or if a specified maximum number of iterations of weight updation have been performed then stop, or else continue).

- There are two training algorithms for MADALINE, viz., MR-I and MR-II. In MR-I algorithm, only the weights of the hidden units are modified during the training and the weights for the inter-connections from the hidden units to the output unit are kept unaltered. However, in case of MR-II, all weights are adjusted, if required.