- Artificial Neural Network -

ADALINE and MADALINE
ADALINE

- ADALINE (Adaptive Linear Neuron) is a network model proposed by Bernard Widrow in 1959.
Training Rule

• The activation function used is
  \[ y = 1 \text{ if } y_{\text{in}} \geq 0 \]
  \[ y = -1 \text{ if } y_{\text{in}} < 0. \]

• The training rule is called the Widrow-Hoff rule or the Delta Rule.

• It can be theoretically shown that the rule minimizes the root mean square error between the activation value and the target value.

• That’s why it’s called the the Least Mean Square (LMS) rule as well.
The δ Rule

The δ rule works also works for more than one output unit.
The δ Rule

Consider one single output unit.
The delta rule changes the weights of the neural connections so as to minimize the difference between the net input to the output unit $y_{in}$ and the target value $t$.
The goal is to minimize the error over all training patterns.
However, this is accomplished by reducing the error to each pattern one at a time.
Weight corrections can also be accumulated over a number of training patterns (called *batch updating*) if desired.
The Training Algorithm

Initialize weights to small random values
Set learning rate $\alpha$ to a value between 0 and 1
while (the largest weight change $\leq$ threshold) do
  for each bipolar training pair $s:t$ do
    {Set activation of input units $i=1..n \{x_i = s_i\}$
    Compute net input to the output unit:
    \[ y_{\text{in}} = b + \sum x_i w_i \]
    Update bias and weights:
    for $i=1..n$ {
      \[
      b(\text{new}) = b(\text{old}) + \alpha(t - y_{\text{in}}) \\
      w_i(\text{new}) = w_i(\text{old}) + \alpha(t - y_{\text{in}})x_i
      \]
    }
  }
//end while
Setting Learning Parameter $\alpha$

• Usually, just use a small value for $\alpha$, something like 0.1.

• If the value is too large, the learning process will not converge.

• If the value of $\alpha$ is too small, learning will be extremely slow (Hecht-Nielsen 1990).

• For a single neuron, a practical range for $\alpha$ is $0.1 \leq n \times \alpha \leq 1.0$, where $n$ is the number of input units (Widrow, Winger and Baxter 1988).
MADALINE

• When several ADALINE units are arranged in a single layer so that there are several output units, there is no change in how ADALINEs are trained from that of a single ADALINE.
• A MADALINE consists of many ADALINEs arranged in a multi-layer net.
• We can think of a MADALINE as having a hidden layer of ADALINEs.
MADALINE (Many Adalines)

- A Madaline is composed of several Adalines
- Each ADALINE unit has a bias. There are two hidden ADALINEs, z1 and z2. There is a single output ADALINE Y.
- Each ADALINE simply applies a threshold function to the unit’s net input.
Y is a non-linear function of the input vector (x1, x2).
The use of hidden units Z1 and Z2 gives the net additional power, but makes training more complicated.
There are two training algorithms for a MADALINE with one hidden layer.

Algorithm *MR-I* is the original MADALINE training algorithm (Widrow and Hoff 1960).

*MR-I changes the weights on to the hidden ADALINEs only*. The weights for the output unit are fixed. It assumes that the output unit is an OR unit.

*MR-II (Widrow, Winter and Baxter 1987) adjusts all weights in the net*. It doesn’t make the assumption that the output unit is an OR unit.
MR-I Training Algorithm

Determine the weights of units (here, v1, v2 and bias b3) such that the output unit Y behaves like an OR unit.

In other words, Y is 1 if the Z1 or Z2 (or both) is (are) 1; Y is -1 if both Z1 and Z2 are -1.

Here a weight of $\frac{1}{2}$ on each of v1, v2 and v3 works.

The weights on the hidden ADALINEs are adjusted according to MR-I algorithm.

In this example, weights on the first ADALINE (w11 and w21) and weights on the second ADALINE (w12 and w22) are adjusted according to MR-I algorithm.

Remember the activation function is

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$
MR-I Training Algorithm

Set learning parameter $\alpha$  //Assume bipolar units and outputs. Only 1 hidden layer.

**while** stopping condition is false **do**

  **for each** bipolar training pair $s:t$ **do**

  Set activation of input units $i = 1$ to $n$ \{ $x_i = s_i$ \}

  Compute net input to hidden units, e.g., $z_{in1} = b_1 + x_1 w_{11} + x_2 w_{21}$

  Determine output of each hidden ADALINE, e.g., $z_1 = f(z_{in1})$

  Determine output of net: $y_{in} = b_3 + z_1 v_{1} + z_2 v_{2}; \ y = f(y_{in})$

    //Determine error and update weights

  **if** $t = y$, **then** no updates are performed //no error

  **if** $t = 1$, //error, the expected output is 1, the computed output is -1; at least one of the Z’s should be 1

    **then** update weights on $Z_J$, the unit whose net input is closest to 1 (or closest 0, both are the same)

      $b_J (\text{new}) = b_J (\text{old}) + \alpha (1 - z_{inJ})$

      $w_{iJ} (\text{new}) = w_{iJ} (\text{old}) + \alpha (1 - z_{inJ}) x_i$

  **if** $t = -1$, **then** update weights on all units $Z_k$ that have positive net input//error

  **endfor**

**endwhile**

**Stopping criterion:** Weight changes have stopped or reached an acceptable level or after a certain number of iterations.
MR-I Training Algorithm

**Motivation for performing updates:** Update weights only if an error has occurred.

Update weights in such a way that it is more likely for the net to produce the desired response.

**If t=1 and error has occurred** (i.e., $y=-1$, or the OR unit is off when it should actually be on): It means that all Z units had value -1 and at least one Z unit needs to have value +1. Therefore, we consider $Z_J$ to be the unit whose net input is closest to 0 and adjust its weights.

**If t=-1 and error has occurred** (i.e., $y=1$ or the OR unit is on when it should actually be off): It means that at least one Z unit had value +1 and all Z units must have value -1. Therefore, we adjust the weights on all of the Z units with positive net input.
Example of Use of MRI

• Solving the XOR problem using MRI
• The training patterns are:

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{Step 0.} & \\
\text{The weights into } Z_1 \text{ and into } Z_2 \text{ are small random values; the weights into } Y \text{ are those found in Example 2.19. The learning rate, } \alpha, \text{ is } .5. \\
\text{Weights into } Z_1 & \quad \text{Weights into } Z_2 & \quad \text{Weights into } Y \\
\begin{array}{ccc}
  w_{11} & w_{21} & b_1 \\
  .05 & .2 & .3 \\
\end{array} & \begin{array}{ccc}
  w_{12} & w_{22} & b_2 \\
  .1 & .2 & .15 \\
\end{array} & \begin{array}{ccc}
  v_1 & v_2 & b_3 \\
  .5 & .5 & .5 \\
\end{array}
\end{align*}
\]

\text{Step 1.} \quad \text{Begin training.}

\text{Step 2.} \quad \text{For the first training pair, } (1, 1): \quad t = -1

\text{Step 3.} \quad x_1 = 1, \quad x_2 = 1

\text{Step 4.} \quad z_{in_1} = .3 + .05 + .2 = .55,
\quad z_{in_2} = .15 + .1 + .2 = .45.
Madaline Training for XOR Using MR1 Algorithm

Step 5. \[ z_1 = 1, \]
\[ z_2 = 1. \]

Step 6. \[ y_{in} = .5 + .5 + .5; \]
\[ y = 1. \]

Step 7. \[ t - y = -1 - 1 = -2 \neq 0, \] so an error occurred. Since \( t = -1, \) and both \( Z \) units have positive net input, update the weights on unit \( Z_1 \) as follows:

\[ b_1(\text{new}) = b_1(\text{old}) + \alpha(-1 - z_{in_1}) \]
\[ = .3 + (.5)(-1.55) \]
\[ = -.475 \]

\[ w_{11}(\text{new}) = w_{11}(\text{old}) + \alpha(-1 - z_{in_1})x_1 \]
\[ = .05 + (.5)(-1.55) \]
\[ = -.725 \]

\[ w_{21}(\text{new}) = w_{21}(\text{old}) + \alpha(-1 - z_{in_1})x_2 \]
\[ = .2 + (.5)(-1.55) \]
\[ = -.575 \]

update the weights on unit \( Z_2 \) as follows:

\[ b_2(\text{new}) = b_2(\text{old}) + \alpha(-1 - z_{in_2}) \]
\[ = .15 + (.5)(-1.45) \]
\[ = -.575 \]

\[ w_{12}(\text{new}) = w_{12}(\text{old}) + \alpha(-1 - z_{in_2})x_1 \]
\[ = .1 + (.5)(-1.45) \]
\[ = -.625 \]

\[ w_{22}(\text{new}) = w_{22}(\text{old}) + \alpha(-1 - z_{in_2})x_2 \]
\[ = .2 + (.5)(-1.45) \]
\[ = -.525 \]

After four epochs of training, the final weights are found to be:

\[ w_{11} = -0.73 \quad w_{12} = 1.27 \]
\[ w_{21} = 1.53 \quad w_{22} = -1.33 \]
\[ b_1 = -0.99 \quad b_2 = -1.09 \]
Geometric Interpretation of Madaline MR1 weights

• The positive response region for the Madaline trained in the previous example is the union of the regions where each of the hidden units have a positive response.

• The decision boundary for each hidden unit can be calculated as described in Section 2.1.3 of Faussette’s book.

For hidden unit $Z_1$, the boundary line is

$$x_2 = -\frac{w_{11}}{w_{21}} x_1 - \frac{b_1}{w_{21}}$$

$$= \frac{0.73}{1.53} x_1 + \frac{0.99}{1.53}$$

$$= 0.48 x_1 + 0.65$$

For hidden unit $Z_2$, the boundary line is

$$x_2 = -\frac{w_{12}}{w_{22}} x_1 - \frac{b_2}{w_{22}}$$

$$= \frac{1.27}{1.33} x_1 + \frac{1.09}{1.33}$$

$$= 0.96 x_1 - 0.82$$
• We see the positive response regions for Z1 and Z2, and then the positive response region for the output Y unit which is the intersection of the two Z1 and Z2 regions.
There is no assumption that the output unit acts as a logical OR. The goal is to change weights in all layers of the net, i.e., in all hidden layers when we have several hidden layers + output layer. But, we also want to cause the least disturbance in the net so that it remains stable from iteration to iteration. This causes least “unlearning” of the patterns for which the net has been trained previously. This is sometimes called the “don’t rock the boat” principle. Several output nodes may be used; the total error for any input pattern is the sum of the squares of the errors at each output unit.
The MR-II algorithm is considerably different from the back-propagation algorithm we will learn later.

The weights are initialized to small random values and training patterns are presented repeatedly in epochs.

The algorithm modifies the weights for the nodes in hidden layer=1, then layer=2, .. up to the output layer.

The training algorithm is a trial-and-error procedure following the minimum disturbance principle.

Nodes that can affect the output error incurring the least change in their weights have precedence in the learning process.
MR-II Training Algorithm

Set learning rate $\alpha$

while stopping condition is false do
  for each bipolar training pair $s:t$ do
    Compute output of the net based on current weights and activation function
    if $t \neq y$, then for each unit whose net input is sufficiently close to 0
      (say, between $-\alpha$ and $\alpha$, with $\alpha=0.25$) do
        {Sort all such units in the network at all levels based on their net input values.
        Start with the unit whose net is closest to 0, then for the next closest, etc.
        Change the unit’s output from +1 to -1, or vice versa
        If modifying the output of this node improves network performance
          (i.e., reduces error on test set)
          then //if the error is not reduced, undo the reversal
            adjust the weights on this unit to achieve the output reversal} //how to do is not given
  endfor
endwhile

Stopping criterion: Weight changes have stopped or reached an acceptable level or after a certain number of iterations.
MR-II Training Algorithm

Algorithm MRII;
repeat
  Present a training pattern \( f \) to the network;
  Compute outputs of all hidden nodes and the output node;
  Let \( h = 1 \);
  while the pattern is misclassified
    and \( h \leq \) the number of hidden layers, do
      Sort the Adalines in layer \( h \), in the order of
      increasing net input magnitude (\( |\sum_j w_{ij}a_j| \)),
      but omitting nodes for which \( |\sum_j w_{ij}a_j| > \theta \),
      where \( \theta \) is a predetermined threshold;
    Let \( S = (A_1, \ldots, A_k) \) be the sorted sequence;
    while network output differs from desired output,
      and \( S \) contains nodes not yet examined
      in this iteration, do
        if reversing output of the next element \( A_j \in S \)
        can improve network performance,
        then modify connection weights leading into \( A_j \)
        to accomplish the output reversal;
        end-if;
      end-while
    \( h := h + 1 \);
  end-while
until performance is considered satisfactory or the upper
bound on the number of iterations has been reached.