since the weight w_{IJ} influences the error only at output unit Y_J. Furthermore, using the fact that

$$y\_in_J = \sum_{i=1}^{n} x_i w_{iJ},$$

we obtain

$$\begin{aligned}\frac{\partial E}{\partial w_{IJ}} &= -2(t_J - y\_in_J)\frac{\partial y\_in_J}{\partial w_{IJ}} \\ &= -2(t_J - y\_in_J)x_I.\end{aligned}$$

Thus, the local error will be reduced most rapidly (for a given learning rate) by adjusting the weights according to the delta rule,

$$\Delta w_{IJ} = \alpha(t_J - y\_in_J)x_I.$$

The preceding two derivations of the delta rule can be generalized to the case where the training data are only samples from a larger data set, or probability distribution. Minimizing the error for the training set will also minimize the expected value for the error of the underlying probability distribution. (See Widrow & Lehr, 1990 or Hecht-Nielsen, 1990 for a further discussion of the matter.)

2.4.5 Madaline

As mentioned earlier, a MADALINE consists of Many ADaptive LInear NEurons arranged in a multilayer net. The examples given for the perceptron and the derivation of the delta rule for several output units both indicate there is essentially no change in the process of training if several ADALINE units are combined in a single-layer net. In this section we will discuss a MADALINE with one hidden layer (composed of two hidden ADALINE units) and one output ADALINE unit. Generalizations to more hidden units, more output units, and more hidden layers, are straightforward.

Architecture

A simple MADALINE net is illustrated in Figure 2.24. The outputs of the two hidden ADALINES, z_1 and z_2, are determined by signals from the same input units X_1 and X_2. As with the ADALINES discussed previously, each output signal is the result of applying a threshold function to the unit's net input. Thus, y is a nonlinear function of the input vector (x_1, x_2). The use of the hidden units, Z_1 and Z_2, give the net computational capabilities not found in single layer nets, but also complicate the training process. In the next section we consider two training algorithms for a MADALINE with one hidden layer.

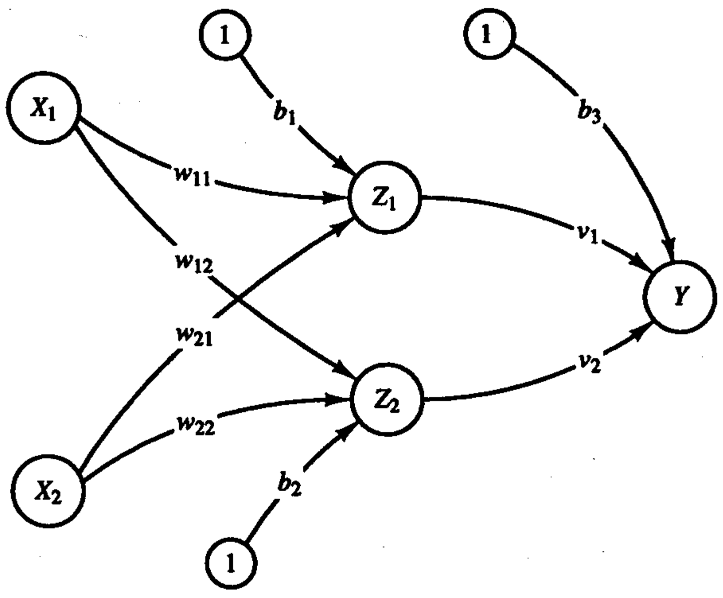

Figure 2.24 A MADALINE with two hidden ADALINES and one output ADALINE.

Algorithm

In the MRI algorithm (the original form of MADALINE training) [Widrow and Hoff, 1960], only the weights for the hidden ADALINES are adjusted; the weights for the output unit are fixed. The MRII algorithm [Widrow, Winter, & Baxter, 1987] provides a method for adjusting all weights in the net.

We consider first the MRI algorithm; the weights v_1 and v_2 and the bias b_3 that feed into the output unit Y are determined so that the response of unit Y is 1 if the signal it receives from either Z_1 or Z_2 (or both) is 1 and is -1 if both Z_1 and Z_2 send a signal of -1. In other words, the unit Y performs the logic function OR on the signals it receives from Z_1 and Z_2. The weights into Y are

$$v_1 = \frac{1}{2}$$

and

$$v_2 = \frac{1}{2},$$

with the bias

$$b_3 = \frac{1}{2}$$

(see Example 2.19). The weights on the first hidden ADALINE (w_{11} and w_{21}) and the weights on the second hidden ADALINE (w_{12} and w_{22}) are adjusted according to the algorithm.

Training Algorithm for MADALINE (MRI). The activation function for units Z_1, Z_2, and Y is

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0; \\ -1 & \text{if } x < 0. \end{cases}$$

```
Step 0.  Initialize weights:
             Weights $v_1$ and $v_2$ and the bias $b_3$ are set as described;
             small random values are usually used for ADALINE weights.
         Set the learning rate $\alpha$ as in the ADALINE training algorithm (a small
         value).
Step 1.  While stopping condition is false, do Steps 2–8.
         Step 2.  For each bipolar training pair, s:t, do Steps 3–7.
                  Step 3.  Set activations of input units:
                               $x_i = s_i$.
                  Step 4.  Compute net input to each hidden ADALINE
                           unit:
                               $z\_in_1 = b_1 + x_1 w_{11} + x_2 w_{21}$,
                               $z\_in_2 = b_2 + x_1 w_{12} + x_2 w_{22}$.
                  Step 5.  Determine output of each hidden ADALINE
                           unit:
                               $z_1 = f(z\_in_1)$,
                               $z_2 = f(z\_in_2)$.
                  Step 6.  Determine output of net:
                               $y\_in = b_3 + z_1 v_1 + z_2 v_2$;
                                  $y = f(y\_in)$.
                  Step 7.  Determine error and update weights:
                           If $t = y$, no weight updates are performed.
                           Otherwise:
                           If $t = 1$, then update weights on $Z_J$,
                           the unit whose net input is closest to 0,
                               $b_J(\text{new}) = b_J(\text{old}) + \alpha(1 - z\_in_J)$,
                               $w_{iJ}(\text{new}) = w_{iJ}(\text{old}) + \alpha(1 - z\_in_J)x_i$;
                           If $t = -1$, then update weights on all units
                           $Z_k$ that have positive net input,
                               $b_k(\text{new}) = b_k(\text{old}) + \alpha(-1 - z\_in_k)$,
                               $w_{ik}(\text{new}) = w_{ik}(\text{old}) + \alpha(-1 - z\_in_k)x_i$.
```

```
        Step 8.   Test stopping condition.
                  If weight changes have stopped (or reached an acceptable
                  level), or if a specified maximum number of weight update
                  iterations (Step 2) have been performed, then stop; other-
                  wise continue.
```

Step 7 is motivated by the desire to (1) update the weights only if an error occurred and (2) update the 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, it means that all Z units had value -1 and at least one Z unit needs to have a value of $+1$. Therefore, we take Z_J to be the Z unit whose net input is closest to 0 and adjust its weights (using ADALINE training with a target value of $+1$):

$$b_J(\text{new}) = b_J(\text{old}) + \alpha(1 - z\_in_J),$$

$$w_{iJ}(\text{new}) = w_{iJ}(\text{old}) + \alpha(1 - z\_in_J)x_i.$$

If $t = -1$ and error has occurred, 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, (using ADALINE training with a target of -1):

$$b_k(\text{new}) = b_k(\text{old}) + \alpha(-1 - z\_in_k),$$

$$w_{ik}(\text{new}) = w_{ik}(\text{old}) + \alpha(-1 - z\_in_k)x_i.$$

MADALINES can also be formed with the weights on the output unit set to perform some other logic function such as AND or, if there are more than two hidden units, the "majority rule" function. The weight update rules would be modified to reflect the logic function being used for the output unit [Widrow & Lehr, 1990].

A more recent MADALINE training rule, called MRII [Widrow, Winter, & Baxter, 1987], allows training for weights in all layers of the net. As in earlier MADALINE training, the aim is to cause the least disturbance to the net at any step of the learning process, in order to cause as little "unlearning" of patterns for which the net had been trained previously. This is sometimes called the "don't rock the boat" principle. Several output units may be used; the total error for any input pattern (used in Step 7b) is the sum of the squares of the errors at each output unit.

Training Algorithm for MADALINE (MRII).

```
Step 0.   Initialize weights:
          Set the learning rate $\alpha$.
Step 1.   While stopping condition is false, do Steps 2–8.
          Step 2.     For each bipolar training pair, s:t, do Steps 3–7.
          Step 3–6.   Compute output of net as in the MRI algorithm.
          Step 7.     Determine error and update weights if necessary:
            If $t \neq y$, do Steps 7a–b for each hidden unit whose net
            input is sufficiently close to 0 (say, between $-.25$ and
            .25). Start with the unit whose net input is closest to 0,
            then for the next closest, etc.
            Step 7a.   Change the unit's ouput
                       (from $+1$ to $-1$, or vice versa).
            Step 7b.   Recompute the response of the net.
                       If the error is reduced:
                           adjust the weights on this unit
                           (use its newly assigned output value
                           as target and apply the Delta Rule).
    Step 8.     Test stopping condition.
                If weight changes have stopped (or reached an accept-
                able level), or if a specified maximum number of weight
                update iterations (Step 2) have been performed, then
                stop; otherwise continue.
```

A further modification is the possibility of attempting to modify pairs of units at the first layer after all of the individual modifications have been attempted. Similarly adaptation could then be attempted for triplets of units.

Application

Example 2.20 Training a MADALINE for the XOR function

This example illustrates the use of the MRI algorithm to train a MADALINE to solve the XOR problem. Only the computations for the first weight updates are shown.

The training patterns are:

x_1	x_2	t
1	1	-1
1	-1	1
-1	1	1
-1	-1	-1

Step 0.
The weights into Z_1 and into Z_2 are small random values; the weights into Y are those found in Example 2.19. The learning rate, α, is .5.

Weights into Z_1			**Weights into Z_2**			**Weights into Y**		
w_{11}	w_{21}	b_1	w_{12}	w_{22}	b_2	v_1	v_2	b_3
.05	.2	.3	.1	.2	.15	.5	.5	.5

```
Step 1.   Begin training.
          Step 2.   For the first training pair, (1, 1): $-1$
                    Step 3.   $x_1 = 1, \quad x_2 = 1$
                    Step 4.   $z\_in_1 = .3 + .05 + .2 = .55,$
                              $z\_in_2 = .15 + .1 + .2 = .45.$
```