# 10 Widrow-Hoff Learning


| | |
|---|---|
| Objectives | 10-1 |
| Theory and Examples | 10-2 |
| ADALINE Network | 10-2 |
| Single ADALINE | 10-3 |
| Mean Square Error | 10-4 |
| LMS Algorithm | 10-7 |
| Analysis of Convergence | 10-9 |
| Adaptive Filtering | 10-13 |
| Adaptive Noise Cancellation | 10-15 |
| Echo Cancellation | 10-21 |
| Summary of Results | 10-22 |
| Solved Problems | 10-24 |
| Epilogue | 10-40 |
| Further Reading | 10-41 |
| Exercises | 10-42 |


## Objectives

In the previous two chapters we laid the foundation for *performance learning*, in which a network is trained to optimize its performance. In this chapter we apply the principles of performance learning to a single-layer linear neural network.

Widrow-Hoff learning is an approximate steepest descent algorithm, in which the performance index is mean square error. This algorithm is important to our discussion for two reasons. First, it is widely used today in many signal processing applications, several of which we will discuss in this chapter. In addition, it is the precursor to the backpropagation algorithm for multilayer networks, which is presented in Chapter 11

# Theory and Examples

Bernard Widrow began working in neural networks in the late 1950s, at about the same time that Frank Rosenblatt developed the perceptron learning rule. In 1960 Widrow, and his graduate student Marcian Hoff, introduced the ADALINE (ADAptive LInear NEuron) network, and a learning rule which they called the LMS (Least Mean Square) algorithm [WiHo60].

Their ADALINE network is very similar to the perceptron, except that its transfer function is linear, instead of hard-limiting. Both the ADALINE and the perceptron suffer from the same inherent limitation: they can only solve linearly separable problems (recall our discussion in Chapters 3 and 4). The LMS algorithm, however, is more powerful than the perceptron learning rule. While the perceptron rule is guaranteed to converge to a solution that correctly categorizes the training patterns, the resulting network can be sensitive to noise, since patterns often lie close to the decision boundaries. The LMS algorithm minimizes mean square error, and therefore tries to move the decision boundaries as far from the training patterns as possible.

The LMS algorithm has found many more practical uses than the perceptron learning rule. This is especially true in the area of digital signal processing. For example, most long distance phone lines use ADALINE networks for echo cancellation. We will discuss these applications in detail later in the chapter.

Because of the great success of the LMS algorithm in signal processing applications, and because of the lack of success in adapting the algorithm to multilayer networks, Widrow stopped work on neural networks in the early 1960s and began to work full time on adaptive signal processing. He returned to the neural network field in the 1980s and began research on the use of neural networks in adaptive control, using temporal backpropagation, a descendant of his original LMS algorithm.

## ADALINE Network

The ADALINE network is shown in Figure 10.1. Notice that it has the same basic structure as the perceptron network we discussed in Chapter 4. The only difference is that it has a linear transfer function.

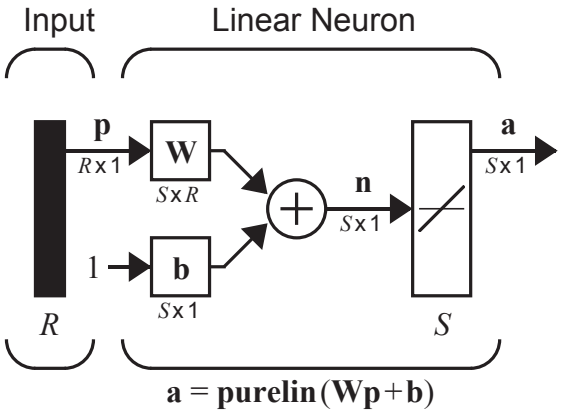



Figure 10.1 ADALINE Network

The output of the network is given by

$$\mathbf{a} = \mathbf{purelin}(\mathbf{Wp} + \mathbf{b}) = \mathbf{Wp} + \mathbf{b}. \tag{10.1}$$

Recall from our discussion of the perceptron network that the $i$th element of the network output vector can be written

$$a_i = purelin(n_i) = purelin({}_i\mathbf{w}^T\mathbf{p} + b_i) = {}_i\mathbf{w}^T\mathbf{p} + b_i, \tag{10.2}$$

where ${}_i\mathbf{w}$ is made up of the elements of the $i$th row of $\mathbf{W}$:

$${}_i\mathbf{w} = \begin{bmatrix} w_{i,1} \\ w_{i,2} \\ \vdots \\ w_{i,R} \end{bmatrix}. \tag{10.3}$$

## Single ADALINE

To simplify our discussion, let's consider a single ADALINE with two inputs. The diagram for this network is shown in Figure 10.2.

The output of the network is given by

$$\begin{aligned} a &= purelin(n) = purelin({}_1\mathbf{w}^T\mathbf{p} + b) = {}_1\mathbf{w}^T\mathbf{p} + b \\ &= {}_1\mathbf{w}^T\mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b. \end{aligned} \tag{10.4}$$

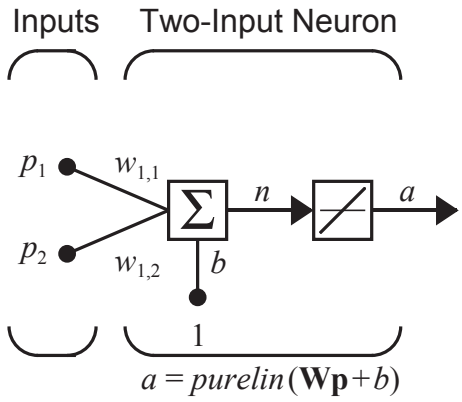



Figure 10.2 Two-Input Linear Neuron

You may recall from Chapter 4 that the perceptron has a *decision boundary*, which is determined by the input vectors for which the net input $n$ is zero. Now, does the ADALINE also have such a boundary? Clearly it does. If we set $n = 0$ then ${}_1\mathbf{w}^T\mathbf{p} + b = 0$ specifies such a line, as shown in Figure 10.3.

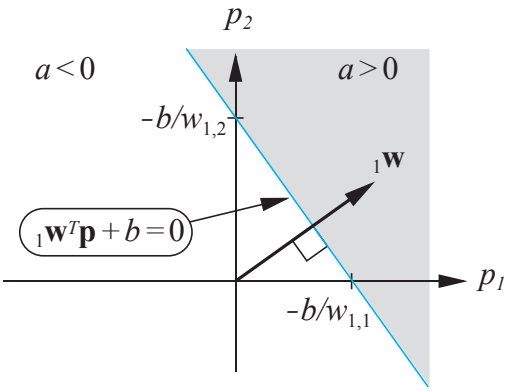



Figure 10.3 Decision Boundary for Two-Input ADALINE

The neuron output is greater than 0 in the gray area. In the white area the output is less than zero. Now, what does this imply about the ADALINE? It says that the ADALINE can be used to classify objects into two categories. However, it can do so only if the objects are linearly separable. Thus, in this respect, the ADALINE has the same limitation as the perceptron.

## Mean Square Error

Now that we have examined the characteristics of the ADALINE network, we are ready to begin our development of the LMS algorithm. As with the perceptron rule, the LMS algorithm is an example of supervised training, in which the learning rule is provided with a set of examples of proper network behavior:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}, \tag{10.5}$$

where $\mathbf{p}_q$ is an input to the network, and $\mathbf{t}_q$ is the corresponding target output. As each input is applied to the network, the network output is compared to the target.

The LMS algorithm will adjust the weights and biases of the ADALINE in order to minimize the mean square error, where the error is the difference between the target output and the network output. In this section we want to discuss this performance index. We will consider first the single-neuron case.

To simply our development, we will lump all of the parameters we are adjusting, including the bias, into one vector:

$$\mathbf{x} = \begin{bmatrix} {}_1\mathbf{w} \\ b \end{bmatrix}. \tag{10.6}$$

Similarly, we include the bias input "1" as a component of the input vector

$$\mathbf{z} = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}. \tag{10.7}$$

Now the network output, which we usually write in the form

$$a = {}_1\mathbf{w}^T\mathbf{p} + b, \tag{10.8}$$

can be written as

$$a = \mathbf{x}^T\mathbf{z}. \tag{10.9}$$

Mean Square Error

This allows us to conveniently write out an expression for the ADALINE network *mean square error*:

$$F(\mathbf{x}) = E[e^2] = E[(t-a)^2] = E[(t-\mathbf{x}^T\mathbf{z})^2], \tag{10.10}$$

where the expectation is taken over all sets of input/target pairs. (Here we use $E[\ ]$ to denote expected value. We use a generalized definition of expectation, which becomes a time-average for deterministic signals. See [WiSt85].) We can expand this expression as follows:

$$\begin{aligned} F(\mathbf{x}) &= E[t^2 - 2t\mathbf{x}^T\mathbf{z} + \mathbf{x}^T\mathbf{z}\mathbf{z}^T\mathbf{x}] \\ &= E[t^2] - 2\mathbf{x}^T E[t\mathbf{z}] + \mathbf{x}^T E[\mathbf{z}\mathbf{z}^T]\mathbf{x}. \end{aligned} \tag{10.11}$$

This can be written in the following convenient form:

$$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\mathbf{x}, \tag{10.12}$$

where

$$c = E[t^2], \ \mathbf{h} = E[t\mathbf{z}] \text{ and } \mathbf{R} = E[\mathbf{z}\mathbf{z}^T]. \tag{10.13}$$

Correlation Matrix

Here the vector $\mathbf{h}$ gives the cross-correlation between the input vector and its associated target, while $\mathbf{R}$ is the input *correlation matrix*. The diagonal elements of this matrix are equal to the mean square values of the elements of the input vectors.

Take a close look at Eq. (10.12), and compare it with the general form of the quadratic function given in Eq. (8.35) and repeated here:

$$F(\mathbf{x}) = c + \mathbf{d}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}. \tag{10.14}$$

We can see that the mean square error performance index for the ADALINE network is a quadratic function, where

$$\mathbf{d} = -2\mathbf{h} \text{ and } \mathbf{A} = 2\mathbf{R}. \tag{10.15}$$

This is a very important result, because we know from Chapter 8 that the characteristics of the quadratic function depend primarily on the Hessian matrix $\mathbf{A}$. For example, if the eigenvalues of the Hessian are all positive, then the function will have one unique global minimum.

In this case the Hessian matrix is twice the correlation matrix $\mathbf{R}$, and it can be shown that all correlation matrices are either positive definite or positive semidefinite, which means that they can never have negative eigenvalues. We are left with two possibilities. If the correlation matrix has only positive eigenvalues, the performance index will have one unique global minimum (see Figure 8.7). If the correlation matrix has some zero eigenvalues, the performance index will either have a weak minimum (see Figure 8.9) or no minimum (see Problem P8.7), depending on the vector $\mathbf{d} = -2\mathbf{h}$.

Now let's locate the stationary point of the performance index. From our previous discussion of quadratic functions we know that the gradient is

$$\nabla F(\mathbf{x}) = \nabla\left(c + \mathbf{d}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}\right) = \mathbf{d} + \mathbf{A}\mathbf{x} = -2\mathbf{h} + 2\mathbf{R}\mathbf{x}. \tag{10.16}$$

The stationary point of $F(\mathbf{x})$ can be found by setting the gradient equal to zero:

$$-2\mathbf{h} + 2\mathbf{R}\mathbf{x} = 0. \tag{10.17}$$

Therefore, if the correlation matrix is positive definite there will be a unique stationary point, which will be a strong minimum:

$$\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}. \tag{10.18}$$

It is worth noting here that the existence of a unique solution depends only on the correlation matrix $\mathbf{R}$. Therefore the characteristics of the input vectors determine whether or not a unique solution exists.

## LMS Algorithm

Now that we have analyzed our performance index, the next step is to design an algorithm to locate the minimum point. If we could calculate the statistical quantities $\mathbf{h}$ and $\mathbf{R}$, we could find the minimum point directly from Eq. (10.18). If we did not want to calculate the inverse of $\mathbf{R}$, we could use the steepest descent algorithm, with the gradient calculated from Eq. (10.16). In general, however, it is not desirable or convenient to calculate $\mathbf{h}$ and $\mathbf{R}$. For this reason we will use an approximate steepest descent algorithm, in which we use an estimated gradient.

The key insight of Widrow and Hoff was that they could estimate the mean square error $F(\mathbf{x})$ by

$$\hat{F}(\mathbf{x}) = (t(k) - a(k))^2 = e^2(k), \tag{10.19}$$

where the expectation of the squared error has been replaced by the squared error at iteration $k$. Then, at each iteration we have a gradient estimate of the form:

$$\hat{\nabla} F(\mathbf{x}) = \nabla e^2(k). \tag{10.20}$$

Stochastic Gradient

This is sometimes referred to as the *stochastic gradient*. When this is used in a gradient descent algorithm, it is referred to as "on-line" or incremental learning, since the weights are updated as each input is presented to the network.

The first $R$ elements of $\nabla e^2(k)$ are derivatives with respect to the network weights, while the $(R+1)$st element is the derivative with respect to the bias. Thus we have

$$[\nabla e^2(k)]_j = \frac{\partial e^2(k)}{\partial w_{1,j}} = 2e(k)\frac{\partial e(k)}{\partial w_{1,j}} \text{ for } j = 1, 2, \ldots, R, \tag{10.21}$$

and

$$[\nabla e^2(k)]_{R+1} = \frac{\partial e^2(k)}{\partial b} = 2e(k)\frac{\partial e(k)}{\partial b}. \tag{10.22}$$

Now consider the partial derivative terms at the ends of these equations. First evaluate the partial derivative of $e(k)$ with respect to the weight $w_{1,j}$:

$$\frac{\partial e(k)}{\partial w_{1,j}} = \frac{\partial [t(k) - a(k)]}{\partial w_{1,j}} = \frac{\partial}{\partial w_{1,j}} [t(k) - ({}_1\mathbf{w}^T \mathbf{p}(k) + b)]$$

$$= \frac{\partial}{\partial w_{1,j}} \left[ t(k) - \left( \sum_{i=1}^{R} w_{1,i} p_i(k) + b \right) \right] \tag{10.23}$$

where $p_i(k)$ is the $i$th element of the input vector at the $k$th iteration. This simplifies to

$$\frac{\partial e(k)}{\partial w_{1,j}} = -p_j(k) . \tag{10.24}$$

In a similar way we can obtain the final element of the gradient:

$$\frac{\partial e(k)}{\partial b} = -1 . \tag{10.25}$$

Note that $p_j(k)$ and $1$ are the elements of the input vector $\mathbf{z}$, so the gradient of the squared error at iteration $k$ can be written

$$\hat{\nabla} F(\mathbf{x}) = \nabla e^2(k) = -2e(k)\mathbf{z}(k) . \tag{10.26}$$

Now we can see the beauty of approximating the mean square error by the single error at iteration $k$, as in Eq. (10.19). To calculate this approximate gradient we need only multiply the error times the input.

This approximation to $\nabla F(\mathbf{x})$ can now be used in the steepest descent algorithm. From Eq. (9.10) the steepest descent algorithm, with constant learning rate, is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla F(\mathbf{x})\big|_{\mathbf{x} = \mathbf{x}_k} . \tag{10.27}$$

If we substitute $\hat{\nabla} F(\mathbf{x})$, from Eq. (10.26), for $\nabla F(\mathbf{x})$ we find

$$\mathbf{x}_{k+1} = \mathbf{x}_k + 2\alpha e(k)\mathbf{z}(k) , \tag{10.28}$$

or

$${}_1\mathbf{w}(k+1) = {}_1\mathbf{w}(k) + 2\alpha e(k)\mathbf{p}(k) , \tag{10.29}$$

and

$$b(k+1) = b(k) + 2\alpha e(k) . \tag{10.30}$$

These last two equations make up the least mean square (LMS) algorithm. This is also referred to as the delta rule or the Widrow-Hoff learning algorithm.

The preceding results can be modified to handle the case where we have multiple outputs, and therefore multiple neurons, as in Figure 10.1. To update the $i$th row of the weight matrix use

$$ {}_i\mathbf{w}(k+1) = {}_i\mathbf{w}(k) + 2\alpha e_i(k)\mathbf{p}(k), \tag{10.31} $$

where $e_i(k)$ is the $i$th element of the error at iteration $k$. To update the $i$th element of the bias we use

$$ b_i(k+1) = b_i(k) + 2\alpha e_i(k). \tag{10.32} $$

LMS Algorithm

The *LMS algorithm* can be written conveniently in matrix notation:

$$ \mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha \mathbf{e}(k)\mathbf{p}^T(k), \tag{10.33} $$

and

$$ \mathbf{b}(k+1) = \mathbf{b}(k) + 2\alpha \mathbf{e}(k). \tag{10.34} $$

Note that the error $\mathbf{e}$ and the bias $\mathbf{b}$ are now vectors.

## Analysis of Convergence

The stability of the steepest descent algorithm was investigated in Chapter 9. There we found that the maximum stable learning rate for quadratic functions is $\alpha < 2/\lambda_{max}$, where $\lambda_{max}$ is the largest eigenvalue of the Hessian matrix. Now we want to investigate the convergence of the LMS algorithm, which is approximate steepest descent. We will find that the result is the same.

To begin, note that in the LMS algorithm, Eq. (10.28), $\mathbf{x}_k$ is a function only of $\mathbf{z}(k-1), \mathbf{z}(k-2), \ldots, \mathbf{z}(0)$. If we assume that successive input vectors are statistically independent, then $\mathbf{x}_k$ is independent of $\mathbf{z}(k)$. We will show in the following development that for stationary input processes meeting this condition, the expected value of the weight vector will converge to

$$ \mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}. \tag{10.35} $$

This is the minimum mean square error $\{E[e_k^2]\}$ solution, as we saw in Eq. (10.18).

Recall the LMS algorithm (Eq. (10.28)):

$$ \mathbf{x}_{k+1} = \mathbf{x}_k + 2\alpha e(k)\mathbf{z}(k). \tag{10.36} $$

Now take the expectation of both sides:

$$E[\mathbf{x}_{k+1}] = E[\mathbf{x}_k] + 2\alpha E[e(k)\mathbf{z}(k)]. \tag{10.37}$$

Substitute $t(k) - \mathbf{x}_k^T\mathbf{z}(k)$ for the error to give

$$E[\mathbf{x}_{k+1}] = E[\mathbf{x}_k] + 2\alpha\{E[t(k)\mathbf{z}(k)] - E[(\mathbf{x}_k^T\mathbf{z}(k))\mathbf{z}(k)]\}. \tag{10.38}$$

Finally, substitute $\mathbf{z}^T(k)\mathbf{x}_k$ for $\mathbf{x}_k^T\mathbf{z}(k)$ and rearrange terms to give

$$E[\mathbf{x}_{k+1}] = E[\mathbf{x}_k] + 2\alpha\{E[t_k\mathbf{z}(k)] - E[(\mathbf{z}(k)\mathbf{z}^T(k))\mathbf{x}_k]\}. \tag{10.39}$$

Since $\mathbf{x}_k$ is independent of $\mathbf{z}(k)$:

$$E[\mathbf{x}_{k+1}] = E[\mathbf{x}_k] + 2\alpha\{\mathbf{h} - \mathbf{R}E[\mathbf{x}_k]\}. \tag{10.40}$$

This can be written as

$$E[\mathbf{x}_{k+1}] = [\mathbf{I} - 2\alpha\mathbf{R}]E[\mathbf{x}_k] + 2\alpha\mathbf{h}. \tag{10.41}$$

This dynamic system will be stable if all of the eigenvalues of $[\mathbf{I} - 2\alpha\mathbf{R}]$ fall inside the unit circle (see [Brog91]). Recall from Chapter 9 that the eigenvalues of $[\mathbf{I} - 2\alpha\mathbf{R}]$ will be $1 - 2\alpha\lambda_i$, where the $\lambda_i$ are the eigenvalues of $\mathbf{R}$. Therefore, the system will be stable if

$$1 - 2\alpha\lambda_i > -1. \tag{10.42}$$

Since $\lambda_i > 0$, $1 - 2\alpha\lambda_i$ is always less than 1. The condition on stability is therefore

$$\alpha < 1/\lambda_i \quad \text{for all } i, \tag{10.43}$$

or

$$0 < \alpha < 1/\lambda_{max}. \tag{10.44}$$

Note that this condition is equivalent to the condition we derived in Chapter 9 for the steepest descent algorithm, although in that case we were using the eigenvalues of the Hessian matrix $\mathbf{A}$. Now we are using the eigenvalues of the input correlation matrix $\mathbf{R}$. (Recall that $\mathbf{A} = 2\mathbf{R}$.)

If this condition on stability is satisfied, the steady state solution is

$$E[\mathbf{x}_{ss}] = [\mathbf{I} - 2\alpha\mathbf{R}]E[\mathbf{x}_{ss}] + 2\alpha\mathbf{h}, \tag{10.45}$$

or

$$E[\mathbf{x}_{ss}] = \mathbf{R}^{-1}\mathbf{h} = \mathbf{x}^*. \tag{10.46}$$

Thus the LMS solution, obtained by applying one input vector at a time, is the same as the minimum mean square error solution of Eq. (10.18).

To test the ADALINE network and the LMS algorithm consider again the apple/orange recognition problem originally discussed in Chapter 3. For simplicity we will assume that the ADALINE network has a zero bias.

The LMS weight update algorithm of Eq. (10.29) will be used to calculate the new weights at each step in the network training:

$$\mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha e(k)\mathbf{p}^T(k). \tag{10.47}$$

First let's compute the maximum stable learning rate $\alpha$. We can get such a value by finding the eigenvalues of the input correlation matrix. Recall that the orange and apple vectors and their associated targets are

$$\left\{\mathbf{p}_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}, t_1 = \begin{bmatrix} -1 \end{bmatrix}\right\} \qquad \left\{\mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, t_2 = \begin{bmatrix} 1 \end{bmatrix}\right\}. \tag{10.48}$$

If we assume that the input vectors are generated randomly with equal probability, we can compute the input correlation matrix:

$$\mathbf{R} = E[\mathbf{p}\mathbf{p}^T] = \frac{1}{2}\mathbf{p}_1\mathbf{p}_1^T + \frac{1}{2}\mathbf{p}_2\mathbf{p}_2^T$$

$$= \frac{1}{2}\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}\begin{bmatrix} 1 & -1 & -1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}\begin{bmatrix} 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}. \tag{10.49}$$

The eigenvalues of $\mathbf{R}$ are

$$\lambda_1 = 1.0, \qquad \lambda_2 = 0.0, \qquad \lambda_3 = 2.0. \tag{10.50}$$

Thus, the maximum stable learning rate is

$$\alpha < \frac{1}{\lambda_{max}} = \frac{1}{2.0} = 0.5. \tag{10.51}$$

To be conservative we will pick $\alpha = 0.2$. (Note that in practical applications it might not be practical to calculate $\mathbf{R}$, and $\alpha$ could be selected by trial and error. Other techniques for choosing $\alpha$ are given in [WiSt85].)

We will start, arbitrarily, with all the weights set to zero, and then will apply inputs $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_1$, $\mathbf{p}_2$, etc., in that order, calculating the new weights after each input is presented. (The presentation of the weights in alternat-

ing order is not necessary. A random sequence would be fine.) Presenting $\mathbf{p}_1$, the orange, and using its target of -1 we get

$$a(0) = \mathbf{W}(0)\mathbf{p}(0) = \mathbf{W}(0)\mathbf{p}_1 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} = 0, \tag{10.52}$$

and

$$e(0) = t(0) - a(0) = t_1 - a(0) = -1 - 0 = -1. \tag{10.53}$$

Now we can calculate the new weight matrix:

$$\mathbf{W}(1) = \mathbf{W}(0) + 2\alpha e(0)\mathbf{p}^T(0)$$

$$= \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} + 2(0.2)(-1) \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}^T = \begin{bmatrix} -0.4 & 0.4 & 0.4 \end{bmatrix}. \tag{10.54}$$

According to plan, we will next present the apple, $\mathbf{p}_2$, and its target of 1:

$$a(1) = \mathbf{W}(1)\mathbf{p}(1) = \mathbf{W}(1)\mathbf{p}_2 = \begin{bmatrix} -0.4 & 0.4 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = -0.4, \tag{10.55}$$

and so the error is

$$e(1) = t(1) - a(1) = t_2 - a(1) = 1 - (-0.4) = 1.4. \tag{10.56}$$

Now we calculate the new weights:

$$\mathbf{W}(2) = \mathbf{W}(1) + 2\alpha e(1)\mathbf{p}^T(1)$$

$$= \begin{bmatrix} -0.4 & 0.4 & 0.4 \end{bmatrix} + 2(0.2)(1.4) \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}^T = \begin{bmatrix} 0.16 & 0.96 & -0.16 \end{bmatrix}. \tag{10.57}$$

Next we present the orange again:

$$a(2) = \mathbf{W}(2)\mathbf{p}(2) = \mathbf{W}(2)\mathbf{p}_1 = \begin{bmatrix} 0.16 & 0.96 & -0.16 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} = -0.64. \tag{10.58}$$

The error is

$$e(2) = t(2) - a(2) = t_1 - a(2) = -1 - (-0.64) = -0.36 . \tag{10.59}$$

The new weights are

$$\mathbf{W}(3) = \mathbf{W}(2) + 2\alpha e(2)\mathbf{p}^T(2) = \begin{bmatrix} 0.016 & 1.1040 & -0.0160 \end{bmatrix} . \tag{10.60}$$

If we continue this procedure, the algorithm converges to

$$\mathbf{W}(\infty) = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} . \tag{10.61}$$

Compare this result with the result of the perceptron learning rule in Chapter 4. You will notice that the ADALINE has produced the same decision boundary that we designed in Chapter 3 for the apple/orange problem. This boundary falls halfway between the two reference patterns. The perceptron rule did not produce such a boundary. This is because the perceptron rule stops as soon as the patterns are correctly classified, even though some patterns may be close to the boundaries. The LMS algorithm minimizes the mean square error. Therefore it tries to move the decision boundaries as far from the reference patterns as possible.

## Adaptive Filtering

As we mentioned at the beginning of this chapter, the ADALINE network has the same major limitation as the perceptron network; it can only solve linearly separable problems. In spite of this, the ADALINE has been much more widely used than the perceptron network. In fact, it is safe to say that it is one of the most widely used neural networks in practical applications. One of the major application areas of the ADALINE has been adaptive filtering, where it is still used extensively. In this section we will demonstrate an adaptive filtering example.

Tapped Delay Line

In order to use the ADALINE network as an adaptive filter, we need to introduce a new building block, the tapped delay line. A *tapped delay line* with $R$ outputs is shown in Figure 10.4.

The input signal enters from the left. At the output of the tapped delay line we have an $R$-dimensional vector, consisting of the input signal at the current time and at delays of from $1$ to $R-1$ time steps.

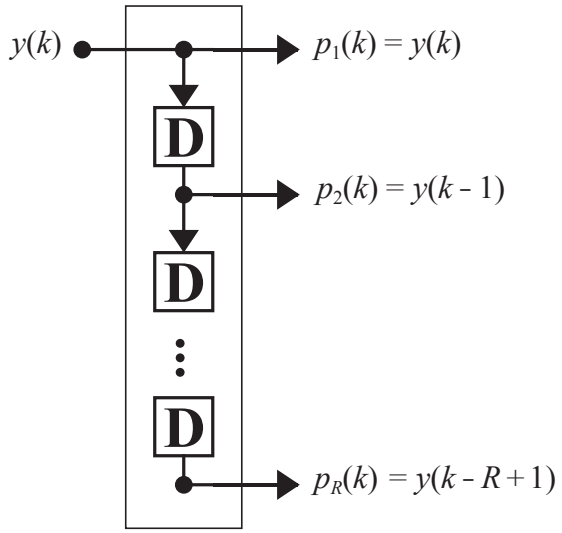



Figure 10.4 Tapped Delay Line

Adaptive Filter

If we combine a tapped delay line with an ADALINE network, we can create an *adaptive filter*, as is shown in Figure 10.5. The output of the filter is given by

$$a(k) = purelin(\mathbf{W}\mathbf{p} + b) = \sum_{i=1}^{R} w_{1,i}y(k-i+1) + b\,. \tag{10.62}$$

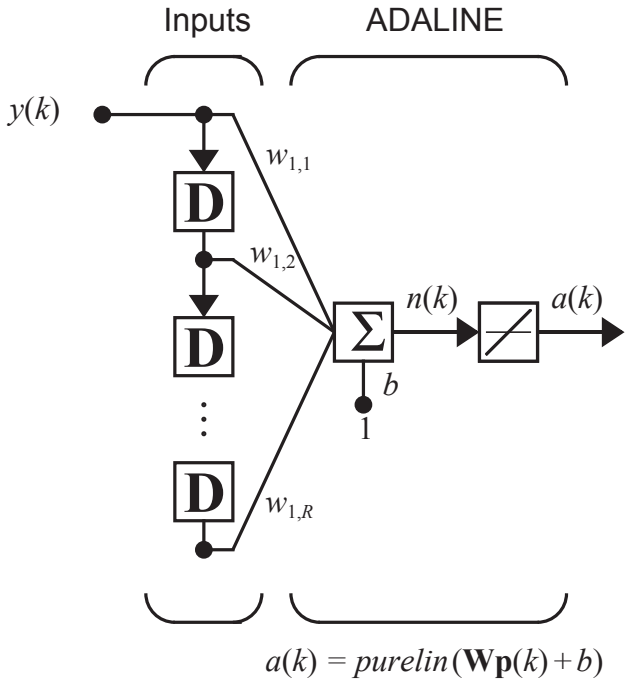



Figure 10.5 Adaptive Filter ADALINE

If you are familiar with digital signal processing, you will recognize the network of Figure 10.5 as a finite impulse response (FIR) filter [WiSt85]. It is beyond the scope of this text to review the field of digital signal processing, but we can demonstrate the usefulness of this adaptive filter through a simple, but practical, example.

## Adaptive Noise Cancellation


An adaptive filter can be used in a variety of novel ways. In the following example we will use it for noise cancellation. Take some time to look at this example, for it is a little different from what you might expect. For instance, the output "error" that the network tries to minimize is actually an approximation to the signal we are trying to recover!

Let's suppose that a doctor, in trying to review the electroencephalogram (EEG) of a distracted graduate student, finds that the signal he would like to see has been contaminated by a 60-Hz noise source. He is examining the patient on-line and wants to view the best signal that can be obtained. Figure 10.6 shows how an adaptive filter can be used to remove the contaminating signal.

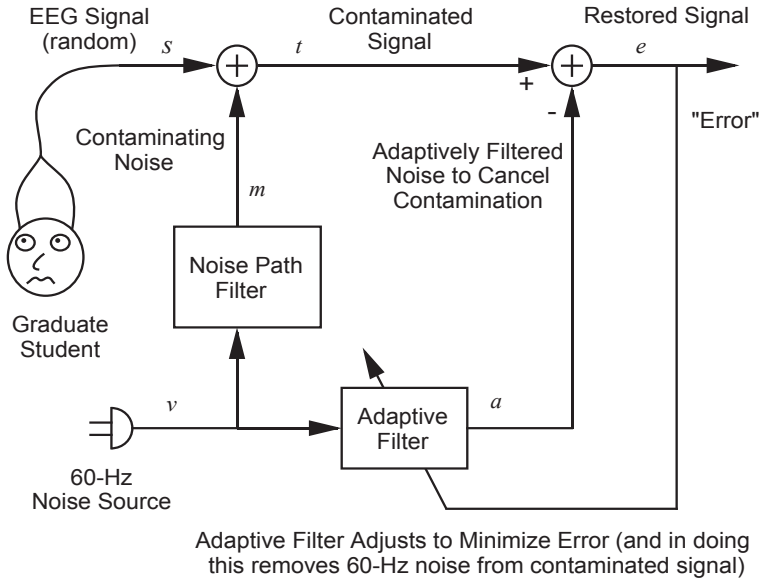



Figure 10.6 Noise Cancellation System

As shown, a sample of the original 60-Hz signal is fed to an adaptive filter, whose elements are adjusted so as to minimize the "error" $e$. The desired output of the filter is the contaminated EEG signal $t$. The adaptive filter will do its best to reproduce this contaminated signal, but it only knows about the original noise source, $v$. Thus, it can only reproduce the part of $t$ that is linearly correlated with $v$, which is $m$. In effect, the adaptive filter will attempt to mimic the noise path filter, so that the output of the filter

$a$ will be close to the contaminating noise $m$. In this way the error $e$ will be close to the original uncontaminated EEG signal $s$.

In this simple case of a single sine wave noise source, a neuron with two weights and no bias is sufficient to implement the filter. The inputs to the filter are the current and previous values of the noise source. Such a two-input filter can attenuate and phase-shift the noise $v$ in the desired way. The filter is shown in Figure 10.7.

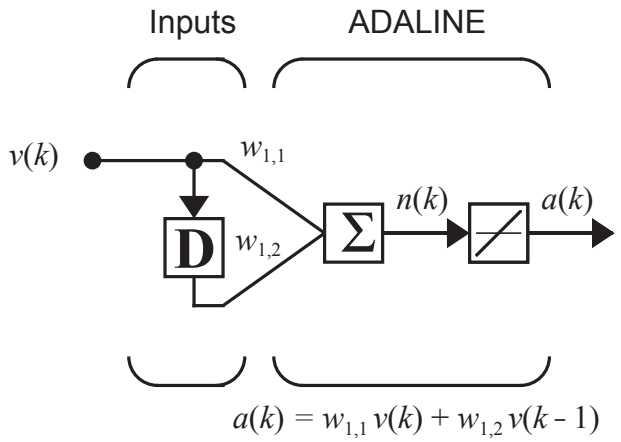



Figure 10.7 Adaptive Filter for Noise Cancellation

We can apply the mathematical relationships developed in the previous sections of this chapter to analyze this system. In order to do so, we will first need to find the input correlation matrix $\mathbf{R}$ and the input/target cross-correlation vector $\mathbf{h}$:

$$\mathbf{R} = [\mathbf{z}\mathbf{z}^T] \text{ and } \mathbf{h} = E[t\mathbf{z}]. \tag{10.63}$$

In our case the input vector is given by the current and previous values of the noise source:

$$\mathbf{z}(k) = \begin{bmatrix} v(k) \\ v(k-1) \end{bmatrix}, \tag{10.64}$$

while the target is the sum of the current signal and filtered noise:

$$t(k) = s(k) + m(k). \tag{10.65}$$

Now expand the expressions for $\mathbf{R}$ and $\mathbf{h}$ to give

$$\mathbf{R} = \begin{bmatrix} E[v^2(k)] & E[v(k)v(k-1)] \\ E[v(k-1)v(k)] & E[v^2(k-1)] \end{bmatrix}, \tag{10.66}$$

and

$$\mathbf{h} = \begin{bmatrix} E[(s(k)+m(k))v(k)] \\ E[(s(k)+m(k))v(k-1)] \end{bmatrix}. \tag{10.67}$$

To obtain specific values for these two quantities we must define the noise signal $v$, the EEG signal $s$ and the filtered noise $m$. For this exercise we will assume: the EEG signal is a white (uncorrelated from one time step to the next) random signal uniformly distributed between the values -0.2 and +0.2, the noise source (60-Hz sine wave sampled at 180 Hz) is given by

$$v(k) = 1.2 \sin\left(\frac{2\pi k}{3}\right), \tag{10.68}$$

and the filtered noise that contaminates the EEG is the noise source attenuated by a factor of 10 and shifted in phase by $\pi/2$:

$$m(k) = 0.12 \sin\left(\frac{2\pi k}{3} + \frac{\pi}{2}\right). \tag{10.69}$$

Now calculate the elements of the input correlation matrix $\mathbf{R}$:

$$E[v^2(k)] = (1.2)^2 \frac{1}{3} \sum_{k=1}^{3} \left(\sin\left(\frac{2\pi k}{3}\right)\right)^2 = (1.2)^2 0.5 = 0.72, \tag{10.70}$$

$$E[v^2(k-1)] = E[v^2(k)] = 0.72, \tag{10.71}$$

$$\begin{aligned} E[v(k)v(k-1)] &= \frac{1}{3} \sum_{k=1}^{3} \left(1.2 \sin\frac{2\pi k}{3}\right)\left(1.2 \sin\frac{2\pi(k-1)}{3}\right) \\ &= (1.2)^2 0.5 \cos\left(\frac{2\pi}{3}\right) = -0.36 \end{aligned} \tag{10.72}$$

(where we have used some trigonometric identities).

Thus $\mathbf{R}$ is

$$\mathbf{R} = \begin{bmatrix} 0.72 & -0.36 \\ -0.36 & 0.72 \end{bmatrix}. \tag{10.73}$$

The terms of $\mathbf{h}$ can be found in a similar manner. We will consider the top term in Eq. (10.67) first:

$$E[(s(k)+m(k))v(k)] = E[s(k)v(k)] + E[m(k)v(k)]. \tag{10.74}$$

Here the first term on the right is zero because $s(k)$ and $v(k)$ are independent and zero mean. The second term is also zero:

$$E[m(k)v(k)] = \frac{1}{3}\sum_{k=1}^{3}\left(0.12\ \sin\left(\frac{2\pi k}{3}+\frac{\pi}{2}\right)\right)\left(1.2\sin\frac{2\pi k}{3}\right) = 0 \qquad (10.75)$$

Thus, the first element of $\mathbf{h}$ is zero.

Next consider the second element of $\mathbf{h}$:

$$E[(s(k)+m(k))v(k-1)] = E[s(k)v(k-1)] + E[m(k)v(k-1)]\ . \qquad (10.76)$$

As with the first element of $\mathbf{h}$, the first term on the right is zero because $s(k)$ and $v(k-1)$ are independent and zero mean. The second term is evaluated as follows:

$$E[m(k)v(k-1)] = \frac{1}{3}\sum_{k=1}^{3}\left(0.12\ \sin\left(\frac{2\pi k}{3}+\frac{\pi}{2}\right)\right)\left(1.2\ \sin\frac{2\pi(k-1)}{3}\right)$$

$$= -0.0624\ . \qquad (10.77)$$

Thus, $\mathbf{h}$ is

$$\mathbf{h} = \begin{bmatrix} 0 \\ -0.0624 \end{bmatrix}. \qquad (10.78)$$

The minimum mean square error solution for the weights is given by Eq. (10.18):

$$\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h} = \begin{bmatrix} 0.72 & -0.36 \\ -0.36 & 0.72 \end{bmatrix}^{-1}\begin{bmatrix} 0 \\ -0.0624 \end{bmatrix} = \begin{bmatrix} -0.0578 \\ -0.1156 \end{bmatrix}. \qquad (10.79)$$

Now, what kind of error will we have at the minimum solution? To find this error recall Eq. (10.12):

$$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\mathbf{x}\ . \qquad (10.80)$$

We have just found $\mathbf{x}^*$, $\mathbf{h}$ and $\mathbf{R}$, so we only need to find $c$:

$$c = E[t^2(k)] = E[(s(k)+m(k))^2]$$

$$= E[s^2(k)] + 2E[s(k)m(k)] + E[m^2(k)]\ . \qquad (10.81)$$

The middle term is zero because $s(k)$ and $m(k)$ are independent and zero mean. The first term, the mean squared value of the random signal, can be calculated as follows:

$$E[s^2(k)] = \frac{1}{0.4}\int_{-0.2}^{0.2} s^2 ds = \frac{1}{3(0.4)} s^3 \Big|_{-0.2}^{0.2} = 0.0133 . \tag{10.82}$$

The mean square value of the filtered noise is

$$E[m^2(k)] = \frac{1}{3}\sum_{k=1}^{3} \left\{ 0.12 \sin\left(\frac{2\pi}{3} + \frac{\pi}{2}\right) \right\}^2 = 0.0072 , \tag{10.83}$$

so that

$$c = 0.0133 + 0.0072 = 0.0205 . \tag{10.84}$$

Substituting $\mathbf{x}^*$, $\mathbf{h}$ and $\mathbf{R}$ into Eq. (10.80), we find that the minimum mean square error is

$$F(\mathbf{x}^*) = 0.0205 - 2(0.0072) + 0.0072 = 0.0133 . \tag{10.85}$$

The minimum mean square error is the same as the mean square value of the EEG signal. This is what we expected, since the "error" of this adaptive noise canceller is in fact the reconstructed EEG signal.

Figure 10.8 illustrates the trajectory of the LMS algorithm in the weight space with learning rate $\alpha = 0.1$. The system weights $w_{1,1}$ and $w_{1,2}$ in this simulation were initialized arbitrarily to 0 and -2, respectively. You can see from this figure that the LMS trajectory looks like a noisy version of steepest descent.

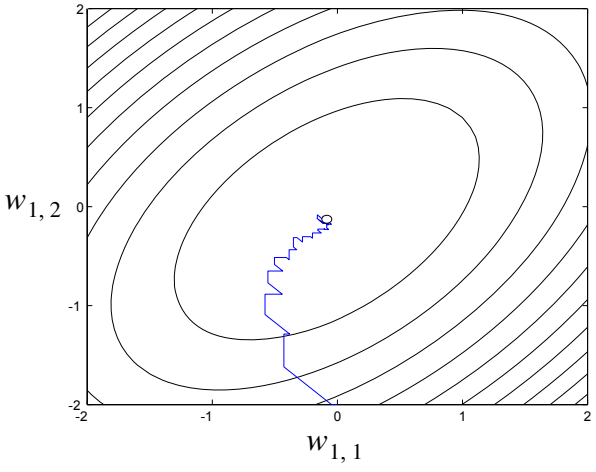



Figure 10.8 LMS Trajectory for $\alpha = 0.1$

Note that the contours in this figure reflect the fact that the eigenvalues and eigenvectors of the Hessian matrix ($\mathbf{A} = 2\mathbf{R}$) are

$$\lambda_1 = 2.16,\ \mathbf{z}_1 = \begin{bmatrix} -0.7071 \\ 0.7071 \end{bmatrix},\ \lambda_2 = 0.72,\ \mathbf{z}_2 = \begin{bmatrix} -0.7071 \\ -0.7071 \end{bmatrix}. \tag{10.86}$$

(Refer back to our discussion in Chapter 8 on the eigensystem of the Hessian matrix.)

If the learning rate is decreased, the LMS trajectory is smoother than that shown in Figure 10.8, but the learning proceeds more slowly. If the learning rate is increased, the trajectory is more jagged and oscillatory. In fact, as noted earlier in this chapter, if the learning rate is increased too much the system does not converge at all. The maximum stable learning rate is $\alpha < 2/2.16 = 0.926$.

In order to judge the performance of our noise canceller, consider Figure 10.9. This figure illustrates how the filter adapts to cancel the noise. The top graph shows the restored and original EEG signals. At first the restored signal is a poor approximation of the original EEG signal. It takes about 0.2 second (with $\alpha = 0.1$) for the filter to adjust to give a reasonable restored signal. The mean square difference between the original and restored signal over the last half of the experiment was 0.002. This compares favorably with the signal mean square value of 0.0133. The difference between the original and restored signal is shown in the lower graph.

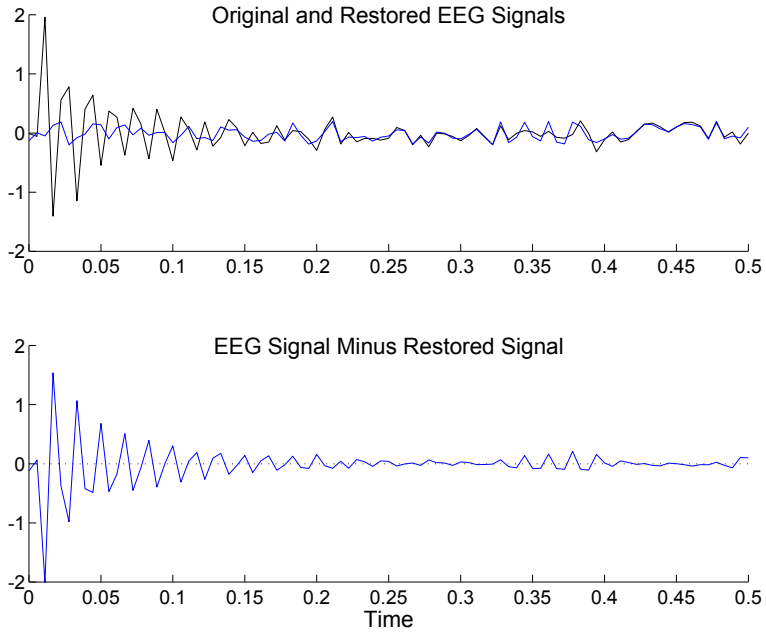
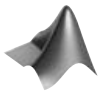



Figure 10.9 Adaptive Filter Cancellation of Contaminating Noise

You might wonder why the error does not go to zero. This is because the LMS algorithm is approximate steepest descent; it uses an estimate of the gradient, not the true gradient, to update the weights. The estimate of the gradient is a noisy version of the true gradient. This will cause the weights to continue to change slightly, even after the mean square error is at the minimum point. You can see this effect in Figure 10.8.



*To experiment with the use of this adaptive noise cancellation filter, use the MATLAB® Neural Network Design Demonstration Adaptive Noise Cancellation* (**nnd10nc**). *A more complex noise source and actual EEG data are used in the Demonstration Electroencephalogram Noise Cancellation* (**nnd10eeg**).

## Echo Cancellation

Another very important practical application of adaptive noise cancellation is echo cancellation. Echoes are common in long distance telephone lines because of impedance mismatch at the "hybrid" device that forms the junction between the long distance line and the customer's local line. You may have experienced this effect on international telephone calls.

Figure 10.10 illustrates how an adaptive noise cancellation filter can be used to reduce these echoes [WiWi85]. At the end of the long distance line the incoming signal is sent to an adaptive filter, as well as to the hybrid device. The target output of the filter is the output of the hybrid. The filter thus tries to cancel the part of the hybrid output that is correlated with the input signal — the echo.

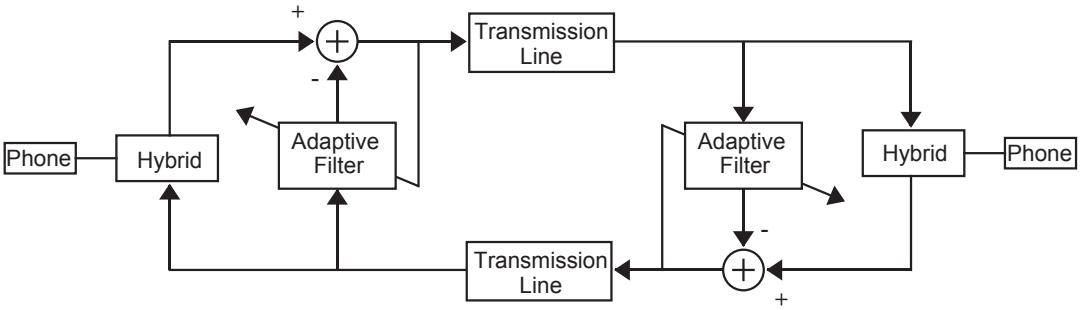
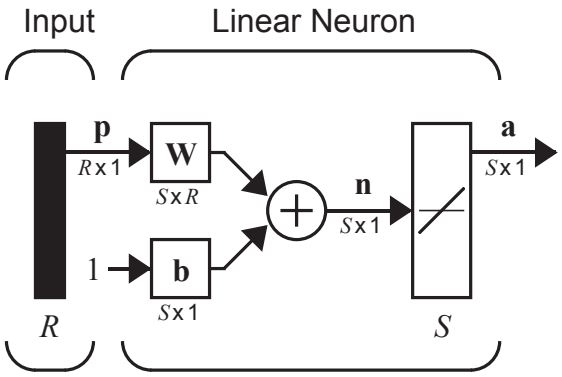



Figure 10.10 Echo Cancellation System

# Summary of Results

## ADALINE




## Mean Square Error

$$F(\mathbf{x}) = E[e^2] = E[(t-a)^2] = E[(t-\mathbf{x}^T\mathbf{z})^2]$$

$$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\mathbf{x},$$

$$c = E[t^2],\ \mathbf{h} = E[t\mathbf{z}] \text{ and } \mathbf{R} = E[\mathbf{z}\mathbf{z}^T]$$

Unique minimum, if it exists, is $\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}$.

Where $\mathbf{x} = \begin{bmatrix} {}_1\mathbf{w} \\ b \end{bmatrix}$ and $\mathbf{z} = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}$.

## LMS Algorithm

$$\mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha\mathbf{e}(k)\mathbf{p}^T(k)$$

$$\mathbf{b}(k+1) = \mathbf{b}(k) + 2\alpha\mathbf{e}(k)$$

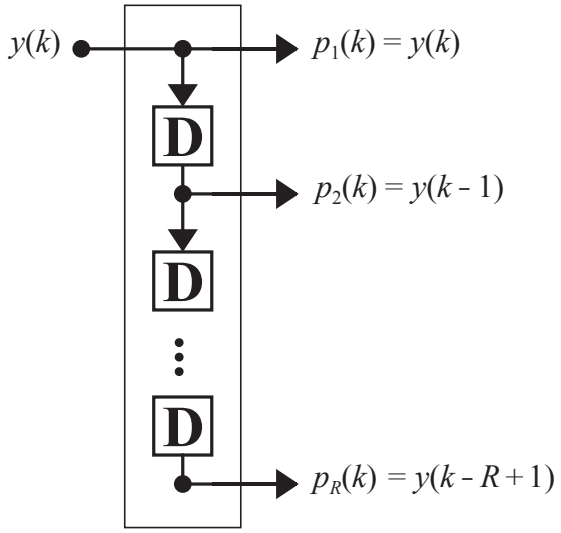
### Convergence Point

$$\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}$$

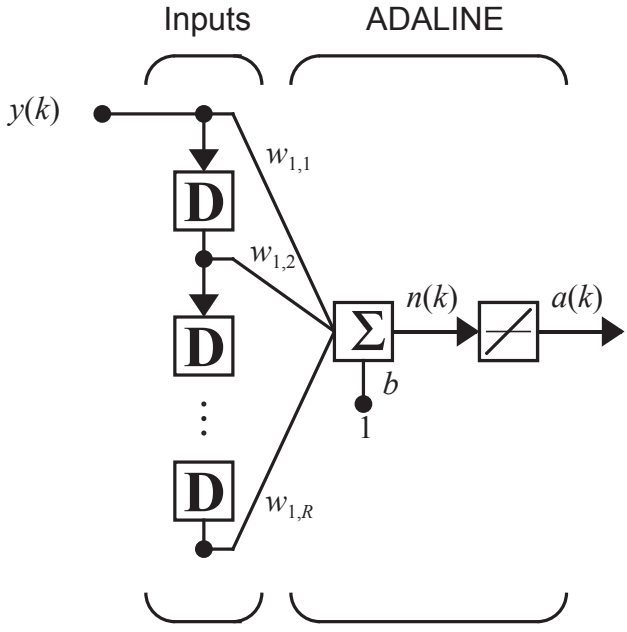
### Stable Learning Rate

$0 < \alpha < 1/\lambda_{max}$ where $\lambda_{max}$ is the maximum eigenvalue of $\mathbf{R}$

## Tapped Delay Line




## Adaptive Filter ADALINE




$$a(k) = purelin(\mathbf{W}\mathbf{p}(k) + b)$$

$$a(k) = purelin(\mathbf{W}\mathbf{p} + b) = \sum_{i=1}^{R} w_{1,i}\, y(k - i + 1) + b$$

# Solved Problems

**P10.1 Consider the ADALINE filter in Figure P10.1.**

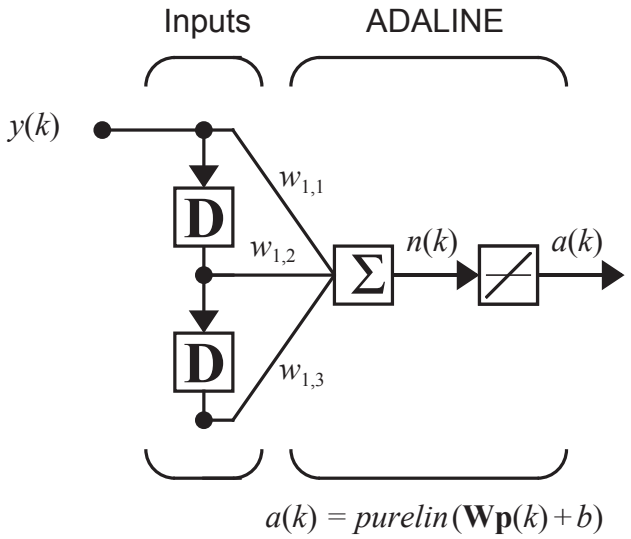



Figure P10.1 ADALINE Filter

**Suppose that**

$$w_{1,1} = 2, \quad w_{1,2} = -1, \quad w_{1,3} = 3,$$

**and the input sequence is**

$$\{y(k)\} = \{\ldots, 0, 0, 0, 5, -4, 0, 0, 0, \ldots\}$$

**where** $y(0) = 5$**,** $y(1) = -4$**, etc.**

i. **What is the filter output just prior to** $k = 0$**?**

ii. **What is the filter output from** $k = 0$ **to** $k = 5$**?**

iii. **How long does** $y(0)$ **contribute to the output?**

**i**. Just prior to $k = 0$ three zeros have entered the filter, and the output is zero.

**ii**. At $k = 0$ the digit "5" has entered the filter, and it will be multiplied by $w_{1,1}$, which has the value 2, so that $a(0) = 10$. This can be viewed as the matrix operation:

$$a(0) = \mathbf{W}\mathbf{p}(0) = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \end{bmatrix} \begin{bmatrix} y(0) \\ y(-1) \\ y(-2) \end{bmatrix} = \begin{bmatrix} 2 & -1 & 3 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix} = 10 .$$

Similarly, one can calculate the next outputs as

$$a(1) = \mathbf{W}\mathbf{p}(1) = \begin{bmatrix} 2 & -1 & 3 \end{bmatrix} \begin{bmatrix} -4 \\ 5 \\ 0 \end{bmatrix} = -13$$

$$a(2) = \mathbf{W}\mathbf{p}(2) = \begin{bmatrix} 2 & -1 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ -4 \\ 5 \end{bmatrix} = 19$$

$$a(3) = \mathbf{W}\mathbf{p}(3) = \begin{bmatrix} 2 & -1 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -4 \end{bmatrix} = -12 , \; a(4) = \mathbf{W}\mathbf{p}(4) = \begin{bmatrix} 2 & -1 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = 0 .$$

All remaining outputs will be zero.

**iii**. The effects of $y(0)$ last from $k = 0$ through $k = 2$, so it will have an influence for three time intervals. This corresponds to the length of the impulse response of this filter.

**P10.2 Suppose that we want to design an ADALINE network to distinguish between various categories of input vectors. Let us first try the categories listed below:**

$$\textbf{Category I:} \quad \mathbf{p}_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}^T \textbf{ and } \mathbf{p}_2 = \begin{bmatrix} -1 & -1 \end{bmatrix}^T$$

$$\textbf{Category II:} \quad \mathbf{p}_3 = \begin{bmatrix} 2 & 2 \end{bmatrix}^T .$$

**i. Can an ADALINE network be designed to make such a distinction?**

**ii. If the answer to part (i) is yes, what set of weights and bias might be used?**

**Next consider a different set of categories.**

$$\textbf{Category III:} \quad \mathbf{p}_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}^T \textbf{ and } \mathbf{p}_2 = \begin{bmatrix} 1 & -1 \end{bmatrix}^T$$

**Category IV:** $\mathbf{p}_3 = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$.

**iii. Can an ADALINE network be designed to make such a distinction?**

**iv. If the answer to part (iii) is yes, what set of weights and bias might be used?**

**i**. The input vectors are plotted in Figure P10.2.

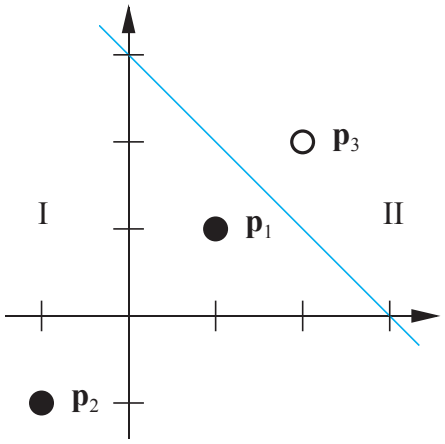



Figure P10.2 Input Vectors for Problem P10.1 (i)

The blue line in this figure is a decision boundary that separates the two categories successfully. Since they are linearly separable, an ADALINE network will do the job.

**ii**. The decision boundary passes through the points $(3, 0)$ and $(0, 3)$. We know these points to be the intercepts $-b/w_{1,1}$ and $-b/w_{1,2}$. Thus, a solution

$$b = 3,\ w_{1,1} = -1,\ w_{1,2} = -1,$$

is satisfactory. Note that if the output of the ADALINE is positive or zero the input vector is classified as Category I, and if the output is negative the input vector is classified as Category II. This solution also provides for error, since the decision boundary bisects the line between $\mathbf{p}_1$ and $\mathbf{p}_3$.

**iii**. The input vectors to be distinguished are shown in Figure P10.3. The vectors in the figure are not linearly separable, so an ADALINE network cannot distinguish between them.

**iv**. As noted in part (iii), an ADALINE cannot do the job, so there are no values for the weights and bias that are satisfactory.

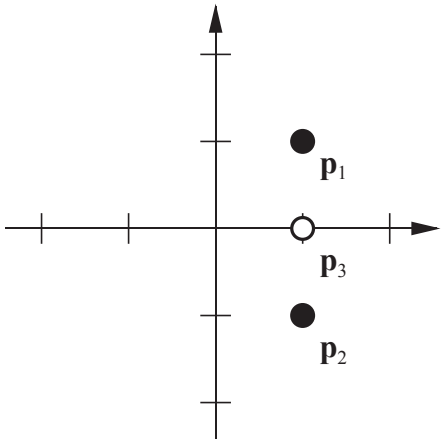



Figure P10.3 Input Vectors for Problem P10.1 (iii)

**P10.3 Suppose that we have the following input/target pairs:**

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\1\end{bmatrix}, t_1 = 1\right\}, \left\{\mathbf{p}_2 = \begin{bmatrix}1\\-1\end{bmatrix}, t_2 = -1\right\}.$$

**These patterns occur with equal probability, and they are used to train an ADALINE network with no bias. What does the mean square error performance surface look like?**

First we need to calculate the various terms of the quadratic function. Recall from Eq. (10.11) that the performance index can be written as

$$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\mathbf{x}.$$

Therefore we need to calculate $c$, $\mathbf{h}$ and $\mathbf{R}$.

The probability of each input occurring is 0.5, so the probability of each target is also 0.5. Thus, the expected value of the square of the targets is

$$c = E[t^2] = (1)^2(0.5) + (-1)^2(0.5) = 1.$$

In a similar way, the cross-correlation between the input and the target can be calculated:

$$\mathbf{h} = E[t\mathbf{z}] = (0.5)(1)\begin{bmatrix}1\\1\end{bmatrix} + (0.5)(-1)\begin{bmatrix}1\\-1\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix}.$$

Finally, the input correlation matrix $\mathbf{R}$ is

$$\mathbf{R} = E[\mathbf{z}\mathbf{z}^T] = \mathbf{p}_1\mathbf{p}_1^T(0.5) + \mathbf{p}_2\mathbf{p}_2^T(0.5)$$

$$= (0.5)\left[\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}1 & 1\end{bmatrix} + \begin{bmatrix}1\\-1\end{bmatrix}\begin{bmatrix}1 & -1\end{bmatrix}\right] = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$$

Therefore the mean square error performance index is

$$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\mathbf{x}$$

$$= 1 - 2\begin{bmatrix}w_{1,1} & w_{1,2}\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} + \begin{bmatrix}w_{1,1} & w_{1,2}\end{bmatrix}\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\begin{bmatrix}w_{1,1}\\w_{1,2}\end{bmatrix}$$

$$= 1 - 2w_{1,2} + w_{1,1}^2 + w_{1,2}^2$$

The Hessian matrix of $F(\mathbf{x})$, which is equal to $2\mathbf{R}$, has both eigenvalues at 2. Therefore the contours of the performance surface will be circular. To find the center of the contours (the minimum point), we need to solve Eq. (10.18):

$$\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h} = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}^{-1}\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix}.$$

Thus we have a minimum at $w_{1,1} = 0$, $w_{1,2} = 1$. The resulting mean square error performance surface is shown in Figure P10.4.

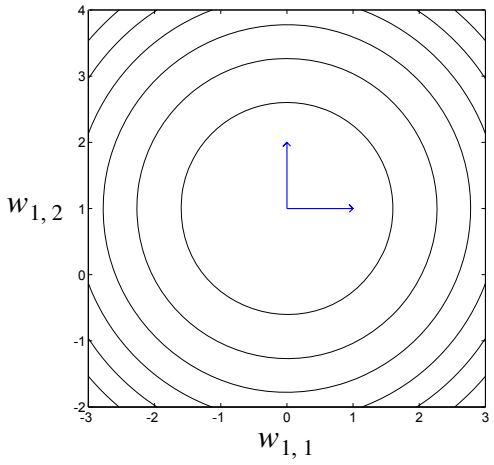


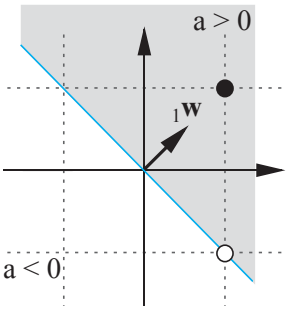

Figure P10.4 Contour Plot of $F(\mathbf{x})$ for Problem P10.3

**P10.4 Consider the system of Problem P10.3 again. Train the network using the LMS algorithm, with the initial guess set to zero and a learning rate $\alpha = 0.25$. Apply each reference pattern only once during training. Draw the decision boundary at each stage.**

Assume the input vector $\mathbf{p}_1$ is presented first. The output, error and new weights are calculated as follows:

$$a(0) = purelin\left[\begin{bmatrix}0 & 0\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\right] = 0,$$

$$e(0) = t(0) - a(0) = 1 - 0 = 1,$$



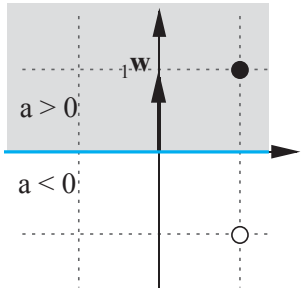

$$\mathbf{W}(1) = \mathbf{W}(0) + 2\alpha e(0)\mathbf{p}(0)^T = \begin{bmatrix}0 & 0\end{bmatrix} + 2\left(\frac{1}{4}\right)(1)\begin{bmatrix}1 & 1\end{bmatrix} = \begin{bmatrix}\frac{1}{2} & \frac{1}{2}\end{bmatrix}.$$

The decision boundary associated with these weights is shown to the left.

Now apply the second input vector:

$$a(1) = purelin\left\{\begin{bmatrix}\frac{1}{2} & \frac{1}{2}\end{bmatrix}\begin{bmatrix}1\\-1\end{bmatrix}\right\} = 0,$$

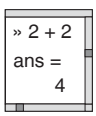
$$e(1) = t(1) - a(1) = -1 - 0 = -1,$$




$$\mathbf{W}(2) = \mathbf{W}(1) + 2\alpha e(1)\mathbf{p}(1)^T = \begin{bmatrix}\frac{1}{2} & \frac{1}{2}\end{bmatrix} + 2\left(\frac{1}{4}\right)(-1)\begin{bmatrix}1 & -1\end{bmatrix} = \begin{bmatrix}0 & 1\end{bmatrix}.$$

The decision boundary associated with these weights is shown to the left. This boundary shows real promise. It is exactly halfway between the input vectors. You might verify for yourself that each input vector, when applied, yields its correct associated target. (What set of weights would be optimal if the targets associated with the two input vectors were exchanged?)

**P10.5 Now consider the convergence of the system of Problems P10.3 and P10.4. What is the maximum stable learning rate for the LMS algorithm?**




The LMS convergence is determined by the learning rate $\alpha$, which should not exceed the reciprocal of the largest eigenvalue of $\mathbf{R}$. We can determine this limit by finding these eigenvalues using MATLAB.

```
[V,D] = eig (R)
V =
        1      0
        0      1

D=
        1       0
        0       1
```

The diagonal terms of matrix `D` give the eigenvalues, 1 and 1, while the columns of `V` show the eigenvectors. Note, incidentally, that the eigenvectors have the same direction as those shown in Figure P10.4.

The largest eigenvalue, $\lambda_{max} = 1$, sets the upper limit on the learning rate at

$$\alpha < 1/\lambda_{max} = 1/1 = 1 .$$

The suggested learning rate in the previous problem was 0.25, and you found (perhaps) that the LMS algorithm converged quickly. What do you suppose happens when the learning rate is 1.0 or larger?

**P10.6 Consider the adaptive filter ADALINE shown in Figure P10.5. The purpose of this filter is to predict the next value of the input signal from the two previous values. Suppose that the input signal is a stationary random process, with autocorrelation function given by**

$$C_y(n) = E[y(k)y(k+n)]$$

$$C_y(0) = 3 , \; C_y(1) = -1 , \; C_y(2) = -1 .$$

**i. Sketch the contour plot of the performance index (mean square error).**

**ii. What is the maximum stable value of the learning rate ($\alpha$) for the LMS algorithm?**

**iii. Assume that a very small value is used for $\alpha$. Sketch the path of the weights for the LMS algorithm, starting with initial guess $\mathbf{W}(0) = \begin{bmatrix} 0.75 & 0 \end{bmatrix}^T$. Explain your procedure for sketching the path.**

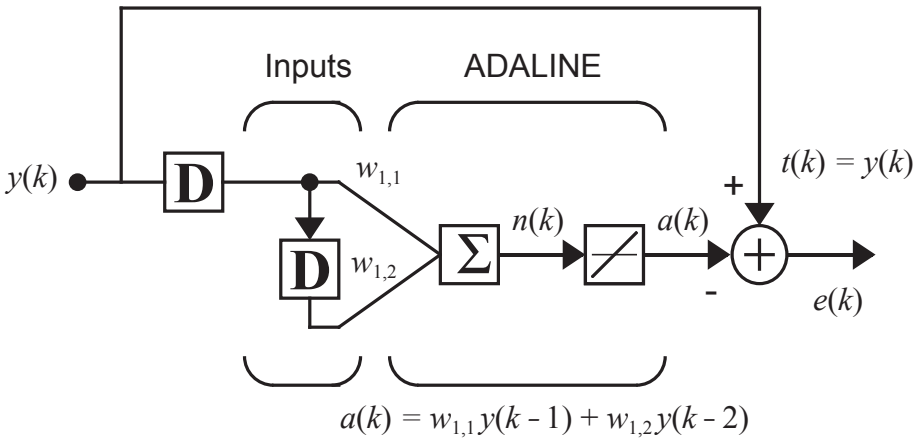



Figure P10.5 Adaptive Predictor

**i**. To sketch the contour plot we first need to find the performance index and the eigenvalues and eigenvectors of the Hessian matrix. First note that the input vector is given by

$$\mathbf{z}(k) = \mathbf{p}(k) = \begin{bmatrix} y(k-1) \\ y(k-2) \end{bmatrix}.$$

Now consider the performance index. Recall from Eq. (10.12) that

$$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\mathbf{x}.$$

We can calculate the constants in the performance index as shown below:

$$c = E[t^2(k)] = E[y^2(k)] = C_y(0) = 3,$$

$$\mathbf{R} = E[\mathbf{z}\mathbf{z}^T] = E\begin{bmatrix} y^2(k-1) & y(k-1)y(k-2) \\ y(k-1)y(k-2) & y^2(k-2) \end{bmatrix}$$

$$= \begin{bmatrix} C_y(0) & C_y(1) \\ C_y(1) & C_y(0) \end{bmatrix} = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}$$

$$\mathbf{h} = E[t\,\mathbf{z}] = E\begin{bmatrix} y(k)y(k-1) \\ y(k)y(k-2) \end{bmatrix} = \begin{bmatrix} C_y(1) \\ C_y(2) \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}.$$

The optimal weights are

$$\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h} = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}^{-1} \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 3/8 & 1/8 \\ 4/8 & 3/8 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1/2 \\ -1/2 \end{bmatrix}.$$

The Hessian matrix is

$$\nabla^2 F(\mathbf{x}) = \mathbf{A} = 2\mathbf{R} = \begin{bmatrix} 6 & -2 \\ -2 & 6 \end{bmatrix}.$$

Now we can get the eigenvalues:

$$\left| \mathbf{A} - \lambda \mathbf{I} \right| = \begin{vmatrix} 6-\lambda & -2 \\ -2 & 6-\lambda \end{vmatrix} = \lambda^2 - 12\lambda + 32 = (\lambda - 8)(\lambda - 4).$$

Thus,

$$\lambda_1 = 4, \qquad \lambda_2 = 8.$$

To find the eigenvectors we use

$$[\mathbf{A} - \lambda \mathbf{I}]\mathbf{v} = 0.$$

For $\lambda_1 = 4$,

$$\begin{bmatrix} 2 & -2 \\ -2 & 2 \end{bmatrix} \mathbf{v}_1 = 0 \qquad \mathbf{v}_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix},$$

and for $\lambda_2 = 8$,

$$\begin{bmatrix} -2 & -2 \\ -2 & -2 \end{bmatrix} \mathbf{v}_2 = 0 \qquad \mathbf{v}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

Therefore the contours of $F(\mathbf{x})$ will be elliptical, with the long axis of each ellipse along the first eigenvector, since the first eigenvalue has the smallest magnitude. The ellipses will be centered at $\mathbf{x}^*$. The contour plot is shown in Figure P10.6.

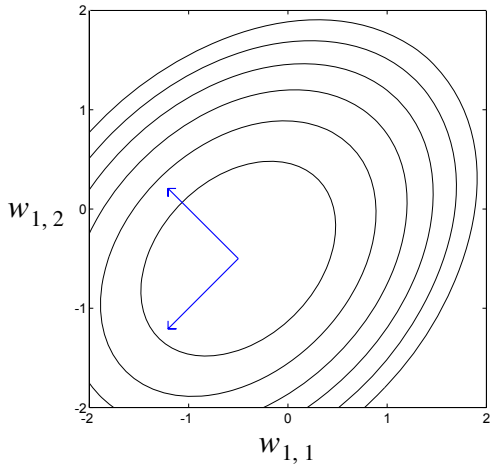



Figure P10.6 Error Contour for Problem P10.6

You might check your sketch by writing a MATLAB M-file to plot the contours.

**ii**. The maximum stable learning rate is the reciprocal of the maximum eigenvalue of $\mathbf{R}$, which is the same as twice the reciprocal of the largest eigenvalue of the Hessian matrix $\nabla^2 F(\mathbf{x}) = \mathbf{A}$:

$$\alpha < 2/\lambda_{max} = 2/8 = 0.25 .$$

**iii**. The LMS algorithm is approximate steepest descent, so the trajectory for small learning rates will move perpendicular to the contour lines, as shown in Figure P10.7.

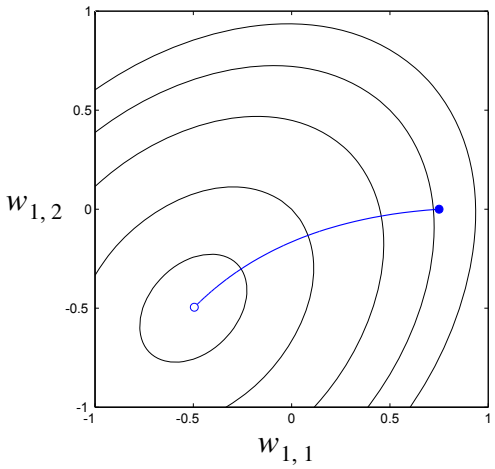



Figure P10.7 LMS Weight Trajectory

**P10.7 The pilot of an airplane is talking into a microphone in his cockpit. The sound received by the air traffic controller in the tower is garbled because the pilot's voice signal has been contaminated by engine noise that reaches his microphone. Can you suggest an adaptive ADALINE filter that might help reduce the noise in the signal received by the control tower? Explain your system.**

The engine noise that has been inadvertently added to the microphone input can be minimized by using the adaptive filtering system shown in Figure P10.8. A sample of the engine noise is supplied to an adaptive filter through a microphone in the cockpit. The desired output of the filter is the contaminated signal coming from the pilot's microphone. The filter attempts to reduce the "error" signal to a minimum. It can do this only by subtracting the component of the contaminated signal that is linearly correlated with the engine noise (and presumably uncorrelated with the pilot's voice). The result is that a clear voice signal is sent to the control tower, in spite of the fact that the engine noise got into the pilot's microphone along with his voice signal. (See [WiSt85] for discussion of similar noise cancellation systems.)

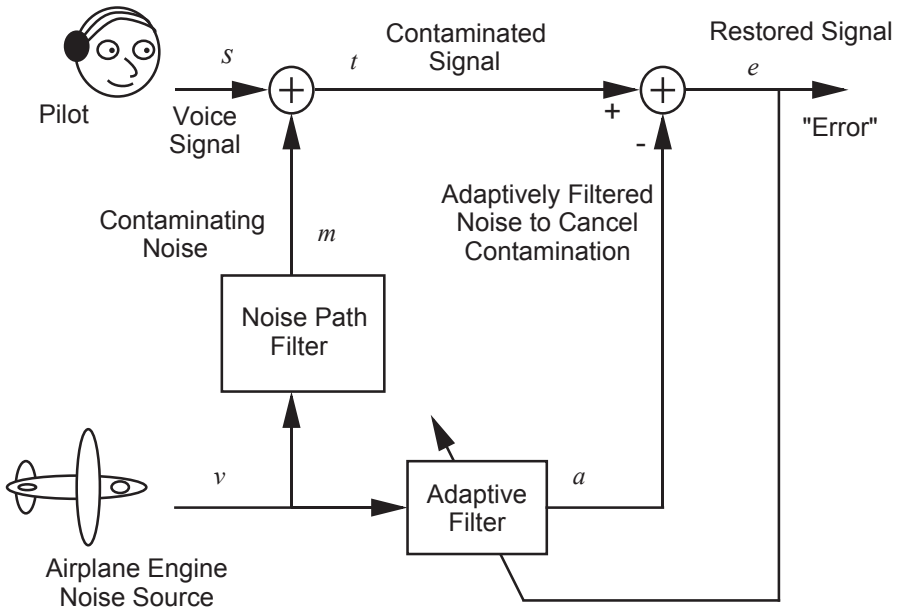



Figure P10.8 Filtering Engine Noise from Pilot's Voice Signal

**P10.8 This is a classification problem like that described in Problems P4.3 and P4.5, except that here we will use an ADALINE network and the LMS learning rule rather than the perceptron learning rule. First we will describe the problem.**

**We have a classification problem with four classes of input vector. The four classes are**

$$\text{class 1:}\left\{\mathbf{p}_1 = \begin{bmatrix}1\\1\end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix}1\\2\end{bmatrix}\right\}, \text{class 2: } \left\{\mathbf{p}_3 = \begin{bmatrix}2\\-1\end{bmatrix}, \mathbf{p}_4 = \begin{bmatrix}2\\0\end{bmatrix}\right\},$$

$$\text{class 3:}\left\{\mathbf{p}_5 = \begin{bmatrix}-1\\2\end{bmatrix}, \mathbf{p}_6 = \begin{bmatrix}-2\\1\end{bmatrix}\right\}, \text{class 4: } \left\{\mathbf{p}_7 = \begin{bmatrix}-1\\-1\end{bmatrix}, \mathbf{p}_8 = \begin{bmatrix}-2\\-2\end{bmatrix}\right\}.$$

**Train an ADALINE network to solve this problem using the LMS learning rule. Assume that each pattern occurs with probability** $1/8$**.**

Let's begin by displaying the input vectors, as in Figure P10.9. The light circles ○ indicate class 1 vectors, the light squares □ indicate class 2 vectors, the dark circles ● indicate class 3 vectors, and the dark squares ■ indicate class 4 vectors. These input vectors can be plotted as shown in Figure P10.9.

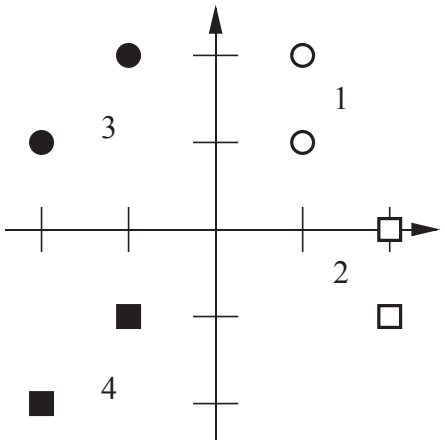



Figure P10.9 Input Vectors for Problem P10.8

We will use target vectors similar to the ones we introduced in Problem P4.3, except that we will replace any targets of 0 by targets of -1. (The perceptron could only output 0 or 1.) Thus, the training set will be:

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\1\end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix}-1\\-1\end{bmatrix}\right\}\left\{\mathbf{p}_2 = \begin{bmatrix}1\\2\end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix}-1\\-1\end{bmatrix}\right\}\left\{\mathbf{p}_3 = \begin{bmatrix}2\\-1\end{bmatrix}, \mathbf{t}_3 = \begin{bmatrix}-1\\1\end{bmatrix}\right\}$$

$$\left\{\mathbf{p}_4 = \begin{bmatrix}2\\0\end{bmatrix}, \mathbf{t}_4 = \begin{bmatrix}-1\\1\end{bmatrix}\right\}\left\{\mathbf{p}_5 = \begin{bmatrix}-1\\2\end{bmatrix}, \mathbf{t}_5 = \begin{bmatrix}1\\-1\end{bmatrix}\right\}\left\{\mathbf{p}_6 = \begin{bmatrix}-2\\1\end{bmatrix}, \mathbf{t}_6 = \begin{bmatrix}1\\-1\end{bmatrix}\right\}$$

$$\left\{\mathbf{p}_7 = \begin{bmatrix}-1\\-1\end{bmatrix}, \mathbf{t}_7 = \begin{bmatrix}1\\1\end{bmatrix}\right\}\left\{\mathbf{p}_8 = \begin{bmatrix}-2\\-2\end{bmatrix}, \mathbf{t}_8 = \begin{bmatrix}1\\1\end{bmatrix}\right\}$$

Also, we will begin as in Problem P4.5 with the following initial weights and biases:

$$\mathbf{W}(0) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{b}(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Now we are almost ready to train an ADALINE network using the LMS rule. We will use a learning rate of $\alpha = 0.04$, and we will present the input vectors in order according to their subscripts. The first iteration is

$$\mathbf{a}(0) = purelin\,(\mathbf{W}(0)\mathbf{p}(0) + \mathbf{b}(0)) = purelin\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$\mathbf{e}(0) = \mathbf{t}(0) - \mathbf{a}(0) = \begin{bmatrix} -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} -3 \\ -3 \end{bmatrix}$$

$$\mathbf{W}(1) = \mathbf{W}(0) + 2\alpha\mathbf{e}(0)\mathbf{p}^T(0)$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + 2(0.04)\begin{bmatrix} -3 \\ -3 \end{bmatrix}\begin{bmatrix} 1 & 1 \end{bmatrix} = \begin{bmatrix} 0.76 & -0.24 \\ -0.24 & 0.76 \end{bmatrix}$$

$$\mathbf{b}(1) = \mathbf{b}(0) + 2\alpha\mathbf{e}(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 2(0.04)\begin{bmatrix} -3 \\ -3 \end{bmatrix} = \begin{bmatrix} 0.76 \\ 0.76 \end{bmatrix}.$$

The second iteration is

$$\mathbf{a}(1) = purelin\,(\mathbf{W}(1)\mathbf{p}(1) + \mathbf{b}(1))$$

$$= purelin\left(\begin{bmatrix} 0.76 & -0.24 \\ -0.24 & 0.76 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.76 \\ 0.76 \end{bmatrix}\right) = \begin{bmatrix} 1.04 \\ 2.04 \end{bmatrix}$$

$$\mathbf{e}(1) = \mathbf{t}(1) - \mathbf{a}(1) = \begin{bmatrix} -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 1.04 \\ 2.04 \end{bmatrix} = \begin{bmatrix} -2.04 \\ -3.04 \end{bmatrix}$$

$$\mathbf{W}(2) = \mathbf{W}(1) + 2\alpha\mathbf{e}(1)\mathbf{p}^T(1)$$

$$= \begin{bmatrix} 0.76 & -0.24 \\ -0.24 & 0.76 \end{bmatrix} + 2(0.04)\begin{bmatrix} -2.04 \\ -3.04 \end{bmatrix}\begin{bmatrix} 1 & 2 \end{bmatrix} = \begin{bmatrix} 0.5968 & -0.5664 \\ -0.4832 & 0.2736 \end{bmatrix}$$

$$\mathbf{b}(2) = \mathbf{b}(1) + 2\alpha\mathbf{e}(1) = \begin{bmatrix}0.76\\0.76\end{bmatrix} + 2(0.04)\begin{bmatrix}-2.04\\-3.04\end{bmatrix} = \begin{bmatrix}0.5968\\0.5168\end{bmatrix}.$$

If we continue until the weights converge we find

$$\mathbf{W}(\infty) = \begin{bmatrix}-0.5948 & -0.0523\\0.1667 & -0.6667\end{bmatrix}, \quad \mathbf{b}(\infty) = \begin{bmatrix}0.0131\\0.1667\end{bmatrix}.$$

The resulting decision boundaries are shown in Figure P10.10. Compare this result with the final decision boundaries created by the perceptron learning rule in Problem P4.5 (Figure P4.7). The perceptron rule stops training when all the patterns are classified correctly. The LMS algorithm moves the boundaries as far from the patterns as possible.

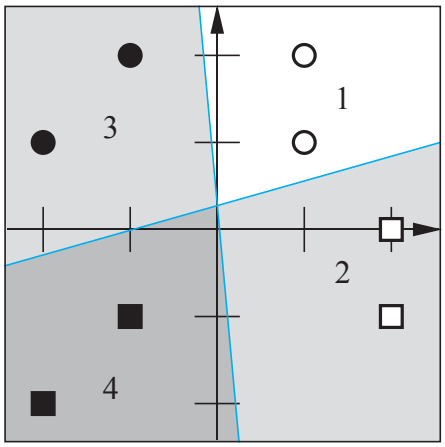



Figure P10.10 Final Decision Boundaries for Problem P10.8

**P10.9 Repeat the work of Widrow and Hoff on a pattern recognition problem from their classic 1960 paper [WiHo60]. They wanted to design a recognition system that would classify the six patterns shown in Figure P10.11.**

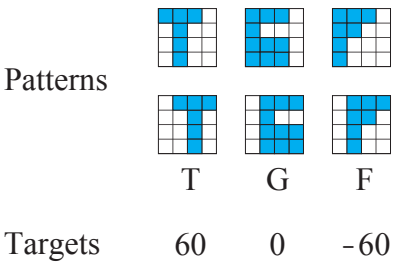



Figure P10.11 Patterns and Their Classification Targets

**These patterns represent the letters T, G and F, in an original form on the top and in a shifted form on the bottom. The targets for these letters (in their original and shifted forms) are +60, 0 and -60, respectively. (The values of 60, 0 and -60 were nice for use on the face of a meter that Widrow and Hoff used to display their network output.) The objective is to train a network so that it will classify the six patterns into the appropriate T, G or F groups.**

The blue squares in the letters will be assigned the value +1, and the white squares will be assigned the value -1. First we convert each of the letters into a single 16-element vector. We choose to do this by starting at the upper left corner, going down the left column, then going down the second column, etc. For example, the vector corresponding to the unshifted letter T is

$$\mathbf{p}_1 = \begin{bmatrix} 1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}^T$$

We have such an input vector for each of the six letters.

The ADALINE network that we will use is shown in Figure P10.12.

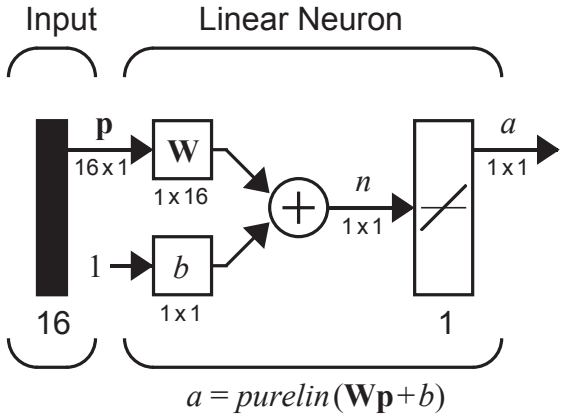



Figure P10.12 Adaptive Pattern Classifier

(Widrow and Hoff built their own machine to realize this ADALINE. According to them, it was “about the size of a lunch pail.”)

Now we will present the six vectors to the network in a random sequence and adjust the weights of the network after each presentation using the LMS algorithm with a learning rate of $\alpha = 0.03$. After each adjustment of weights, all six vectors will be presented to the network to generate their outputs and corresponding errors. The sum of the squares of the errors will be examined as a measure of the quality of the network.

Figure P10.13 illustrates the convergence of the network. The network is trained to recognize these six characters in about 60 presentations, or roughly 10 for each of the possible input vectors.

The results shown in Figure P10.13 are quite like those obtained and published by Widrow and Hoff some 35 years ago. Widrow and Hoff did good science. One can indeed duplicate their work, even decades later (without a lunch pail).

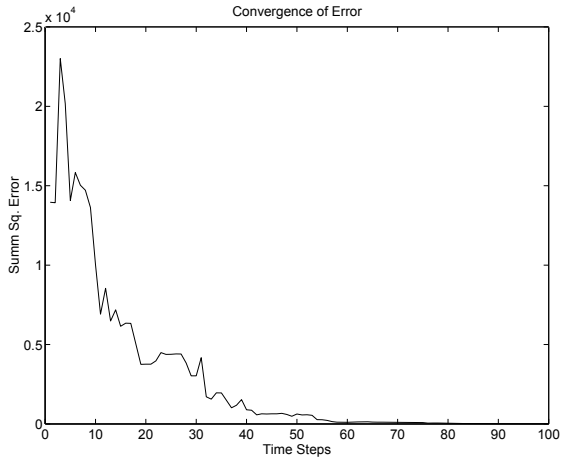
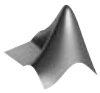



Figure P10.13 Error Convergence with Learning Rate of 0.03



*To experiment with this character recognition problem, use the MATLAB® Neural Network Design Demonstration Linear Pattern Classification* (`nnd10lc`). *Notice the sensitivity of the network to noise in the input pattern.*

# Epilogue

In this chapter we have presented the ADALINE neural network and the LMS learning rule. The ADALINE network is very similar to the perceptron network of Chapter 4, and it has the same fundamental limitation: it can only classify linearly separable patterns. In spite of this limitation on the network, the LMS algorithm is in fact more powerful than the perceptron learning rule. Because it minimizes mean square error, the algorithm is able to create decision boundaries that are more robust to noise than those of the perceptron learning rule.

The ADALINE network and the LMS algorithm have found many practical applications. Even though they were first presented in the late 1950s, they are still very much in use in adaptive filtering applications. For example, echo cancellers using the LMS algorithm are currently employed on many long distance telephone lines. (Chapter 14 provides more extensive coverage of dynamic networks, which are widely used for filtering, prediction and control.)

In addition to its importance as a practical solution to many adaptive filtering problems, the LMS algorithm is also important because it is the forerunner of the backpropagation algorithm, which we will discuss in Chapters 11 through 14. Like the LMS algorithm, backpropagation is an approximate steepest descent algorithm that minimizes mean square error. The only difference between the two algorithms is in the manner in which the derivatives are calculated. Backpropagation is a generalization of the LMS algorithm that can be used for multilayer networks. These more complex networks are not limited to linearly separable problems. They can solve arbitrary classification problems.

# Further Reading

[AnRo89] J. A. Anderson, E. Rosenfeld, *Neurocomputing: Foundations of Research*, Cambridge, MA: MIT Press, 1989.

*Neurocomputing* is a fundamental reference book. It contains over forty of the most important neurocomputing writings. Each paper is accompanied by an introduction that summarizes its results and gives a perspective on the position of the paper in the history of the field.

[StDo84] W. D. Stanley, G. R. Dougherty, R. Dougherty, *Digital Signal Processing*, Reston VA: Reston, 1984

[WiHo60] B. Widrow, M. E. Hoff, "Adaptive switching circuits," *1960 IRE WESCON Convention Record*, New York: IRE Part 4, pp. 96–104.

This seminal paper describes an adaptive perceptron-like network that can learn quickly and accurately. The authors assumed that the system had inputs, a desired output classification for each input, and that the system could calculate the error between the actual and desired output. The weights are adjusted, using a gradient descent method, so as to minimize the mean square error. (Least mean square error or LMS algorithm.)

This paper is reprinted in [AnRo88].

[WiSt 85] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1985.

This informative book describes the theory and application of adaptive signal processing. The authors include a review of the mathematical background that is needed, give details on their adaptive algorithms, and then discuss practical information about many applications.

[WiWi 88] B. Widrow and R. Winter, "Neural nets for adaptive filtering and adaptive pattern recognition," *IEEE Computer Magazine*, March 1988, pp. 25–39.

This is a particularly readable paper that summarizes applications of adaptive multilayer neural networks. The networks are applied to system modeling, statistical prediction, echo cancellation, inverse modeling and pattern recognition.

# Exercises

**E10.1** An adaptive filter ADALINE is shown in Figure E10.1. Suppose that the weights of the network are given by

$$w_{1,1} = 1,\ w_{1,2} = -4,\ w_{1,3} = 2,$$

and the input to the filter is

$$\{y(k)\} = \{\ldots, 0, 0, 0, 1, 1, 2, 0, 0, \ldots\}.$$

Find the response $\{a(k)\}$ of the filter.

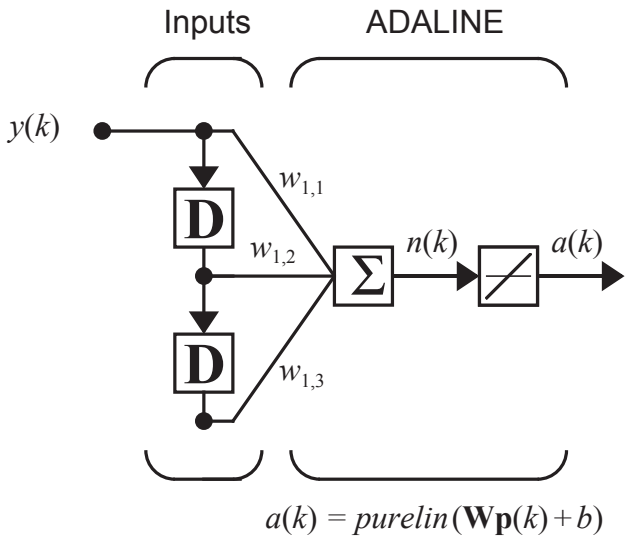



Figure E10.1 Adaptive Filter ADALINE for Exercise E10.1

**E10.2** In Figure E10.2 two classes of patterns are given.

**i.** Use the LMS algorithm to train an ADALINE network to distinguish between class I and class II patterns (we want the network to identify horizontal and vertical lines).

**ii.** Can you explain why the ADALINE network might have difficulty with this problem?

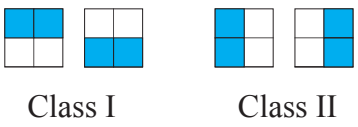



Figure E10.2 Pattern Classification Problem for Exercise E10.2

**E10.3** Suppose that we have the following two reference patterns and their targets:

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\1\end{bmatrix}, t_1 = 1\right\}, \left\{\mathbf{p}_2 = \begin{bmatrix}1\\-1\end{bmatrix}, t_2 = -1\right\}.$$

In Problem P10.3 these input vectors to an ADALINE were assumed to occur with equal probability. Now suppose that the probability of vector $\mathbf{p}_1$ is 0.75 and that the probability of vector $\mathbf{p}_2$ is 0.25. Does this change of probabilities change the mean square error surface? If yes, what does the surface look like now? What is the maximum stable learning rate?

**E10.4** In this exercise we will modify the reference pattern $\mathbf{p}_2$ from Problem P10.3:

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\1\end{bmatrix}, t_1 = 1\right\}, \left\{\mathbf{p}_2 = \begin{bmatrix}-1\\-1\end{bmatrix}, t_2 = -1\right\}.$$

**i.** Assume that the patterns occur with equal probability. Find the mean square error and sketch the contour plot.

**ii.** Find the maximum stable learning rate.

```
» 2 + 2
ans =
     4
```

**iii.** Write a MATLAB M-file to implement the LMS algorithm for this problem. Take 40 steps of the algorithm for a stable learning rate. Use the zero vector as the initial guess. Sketch the trajectory on the contour plot.

**iv.** Take 40 steps of the algorithm after setting the initial values of both parameters to 1. Sketch the final decision boundary.

**v.** Compare the final parameters from parts (iii) and (iv). Explain your results.

**E10.5** We again use the reference patterns and targets from Problem P10.3, and assume that they occur with equal probability. This time we want to train an ADALINE network with a bias. We now have three parameters to find: $w_{1,1}$, $w_{1,2}$ and $b$.

**i.** Find the mean square error and the maximum stable learning rate.

```
» 2 + 2
ans =
     4
```

**ii.** Write a MATLAB M-file to implement the LMS algorithm for this problem. Take 40 steps of the algorithm for a stable learning rate. Use the zero vector as the initial guess. Sketch the final decision boundary.

**iii.** Take 40 steps of the algorithm after setting the initial values of all parameters to 1. Sketch the final decision boundary.

**iv.** Compare the final parameters and the decision boundaries from parts (iii) and (iv). Explain your results.

**E10.6** We have two categories of vectors. Category I consists of

$$\left\{\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}-1\\2\end{bmatrix}\right\}.$$

Category II consists of

$$\left\{\begin{bmatrix}0\\-1\end{bmatrix}, \begin{bmatrix}-4\\1\end{bmatrix}\right\}.$$

We want to train a single-neuron ADALINE network without a bias to recognize these categories ($t$ = 1 for Category I and $t$ = -1 for Category II). Assume that each pattern occurs with equal probability.

**i.** Draw the network diagram.

**ii.** Take four steps of the LMS algorithm, using the zero vector as the initial guess. (one pass through the four vectors above - present each vector once). Use a learning rate of 0.1.

**iii.** What are the optimal weights?

**iv.** Sketch the optimal decision boundary.

**v.** How do you think the boundary would change if the network were allowed to have a bias? If the boundary would change, indicate the approximate new position on your sketch of part iv. You do not need to perform any calculations here - just explain your reasoning.

**E10.7** Suppose that we have the following three reference patterns and their targets:

$$\left\{\mathbf{p}_1 = \begin{bmatrix}3\\6\end{bmatrix}, t_1 = \begin{bmatrix}75\end{bmatrix}\right\}, \left\{\mathbf{p}_2 = \begin{bmatrix}6\\3\end{bmatrix}, t_2 = \begin{bmatrix}75\end{bmatrix}\right\}, \left\{\mathbf{p}_3 = \begin{bmatrix}-6\\3\end{bmatrix}, t_3 = \begin{bmatrix}-75\end{bmatrix}\right\}.$$

Each pattern is equally likely.

**i.** Draw the network diagram for an ADALINE network with no bias that could be trained on these patterns.

**ii.** We want to train the ADALINE network with no bias using these patterns. Sketch the contour plot of the mean square error performance index.

**iii.** Find the maximum stable learning rate for the LMS algorithm.

**iv.** Sketch the trajectory of the LMS algorithm on your contour plot. Assume a very small learning rate, and start with all weights equal to zero. This does not require any calculations.

**E10.8** Suppose that we have the following two reference patterns and their targets:

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\2\end{bmatrix}, t_1 = \begin{bmatrix}-1\end{bmatrix}\right\}, \left\{\mathbf{p}_2 = \begin{bmatrix}-2\\1\end{bmatrix}, t_2 = \begin{bmatrix}1\end{bmatrix}\right\}.$$

The probability of vector $\mathbf{p}_1$ is 0.5 and the probability of vector $\mathbf{p}_2$ is 0.5.We want to train an ADALINE network without a bias on this data set.

**i.** Sketch the contour plot of the mean square error performance index.

**ii.** Sketch the optimal decision boundary.

**iii.** Find the maximum stable learning rate.

**iv.** Sketch the trajectory of the LMS algorithm on your contour plot. Assume a very small learning rate, and start with initial weights $\mathbf{W}(0) = \begin{bmatrix}0 & 1\end{bmatrix}$.

**E10.9** We have the following input/target pairs:

$$\left\{\mathbf{p}_1 = \begin{bmatrix}4\\2\end{bmatrix}, t_1 = 5\right\}, \left\{\mathbf{p}_2 = \begin{bmatrix}2\\-4\end{bmatrix}, t_2 = -2\right\}, \left\{\mathbf{p}_3 = \begin{bmatrix}-4\\4\end{bmatrix}, t_3 = 9\right\}.$$

The first two pair each occurs with probability of 0.25, and the third pair occurs with probability 0.5. We want to train a single-neuron ADALINE network without a bias to perform the desired mapping.

**i.** Draw the network diagram.

**ii.** What is the maximum stable learning rate?

**iii.** Perform one iteration of the LMS algorithm. Apply the input $\mathbf{p}_1$ and use a learning rate of $\alpha = 0.1$. Start from the initial weights $\mathbf{x}_0 = \begin{bmatrix}0 & 0\end{bmatrix}^T$.

**E10.10** Repeat E10.9 for the following input/target pairs:

$$\left\{\mathbf{p}_1 = \begin{bmatrix}2\\-4\end{bmatrix}, t_1 = 1\right\}, \left\{\mathbf{p}_2 = \begin{bmatrix}-4\\4\end{bmatrix}, t_2 = -1\right\}, \left\{\mathbf{p}_3 = \begin{bmatrix}4\\2\end{bmatrix}, t_3 = 1\right\}.$$

The first two pair each occurs with probability of 0.25, and the third pair occurs with probability 0.5. We want to train a single-neuron ADALINE network without a bias to perform the desired mapping.

**E10.11** We want to train a single-neuron ADALINE network without a bias, using the following training set, which categorizes vectors into two classes. Each pattern occurs with equal probability.

$$\left\{\mathbf{p}_1 = \begin{bmatrix}-1\\2\end{bmatrix}, t_1 = -1\right\}\left\{\mathbf{p}_2 = \begin{bmatrix}2\\-1\end{bmatrix}, t_2 = -1\right\}\left\{\mathbf{p}_3 = \begin{bmatrix}0\\-1\end{bmatrix}, t_3 = 1\right\}\left\{\mathbf{p}_4 = \begin{bmatrix}-1\\0\end{bmatrix}, t_4 = 1\right\}$$

**i.** Draw the network diagram.

**ii.** Take one step of the LMS algorithm (present $\mathbf{p}_1$ only) starting from the initial weight $\mathbf{W}(0) = \begin{bmatrix}0 & 0\end{bmatrix}$. Use a learning rate of 0.1.

**iii.** What are the optimal weights? Show all calculations.

**iv.** Sketch the optimal decision boundary.

**v.** How do you think the boundary would change if the network were allowed to have a bias? Indicate the approximate new position on your sketch of part iv.

**vi.** What is the maximum stable learning rate for the LMS algorithm?

**vii.** Sketch the contour plot of the mean square error performance surface.

**viii.** On your contour plot of part vii, sketch the path of the LMS algorithm for a very small learning rate (e.g., 0.001) starting from the initial condition $\mathbf{W}(0) = \begin{bmatrix}2 & 0\end{bmatrix}$. This does not require any calculations, but explain how you obtained your answer.

**E10.12** Suppose that we have the following three reference patterns and their targets:

$$\left\{\mathbf{p}_1 = \begin{bmatrix}2\\4\end{bmatrix}, t_1 = \begin{bmatrix}26\end{bmatrix}\right\}, \left\{\mathbf{p}_2 = \begin{bmatrix}4\\2\end{bmatrix}, t_2 = \begin{bmatrix}26\end{bmatrix}\right\}, \left\{\mathbf{p}_3 = \begin{bmatrix}-2\\-2\end{bmatrix}, t_3 = \begin{bmatrix}-26\end{bmatrix}\right\}.$$

The probability of vector $\mathbf{p}_1$ is 0.25, the probability of vector $\mathbf{p}_2$ is 0.25 and the probability of vector $\mathbf{p}_3$ is 0.5.

**i.** Draw the network diagram for an ADALINE network with no bias that could be trained on these patterns.

**ii.** Sketch the contour plot of the mean square error performance index.

**iii.** Show the optimal decision boundary (for the weights that minimize mean square error) and verify that it separates the patterns into the appropriate categories.

**iv.** Find the maximum stable learning rate for the LMS algorithm. If the target values are changed from 26 and -26 to 2 and -2, how would this change the maximum stable learning rate?

**v.** Perform one iteration of the LMS algorithm, starting with all weights equal to zero, and presenting input vector $\mathbf{p}_1$. Use a learning rate of $\alpha = 0.5$.

**vi.** Sketch the trajectory of the LMS algorithm on your contour plot. Assume a very small learning rate, and start with all weights equal to zero.

**E10.13** Consider the adaptive predictor in Figure E10.3.

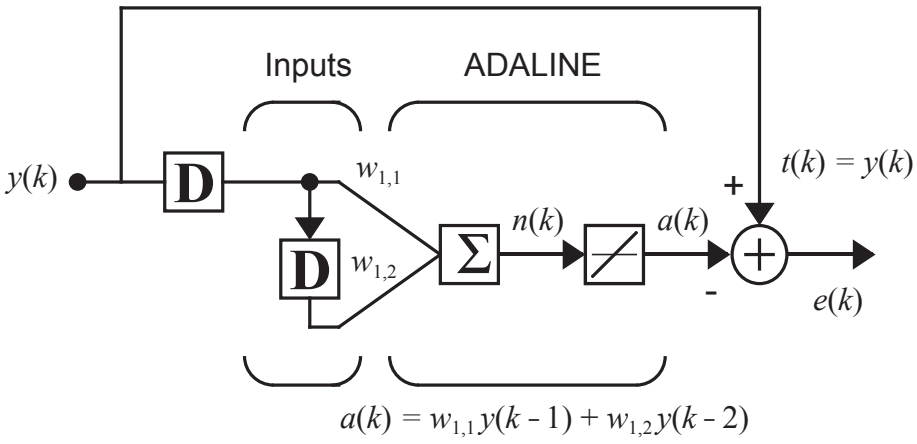



Figure E10.3 Adaptive Predictor for Exercise E10.13

Assume that $y(k)$ is a stationary process with autocorrelation function

$$C_y(n) = E[y(k)(y(k+n))] .$$

**i.** Write an expression for the mean square error in terms of $C_y(n)$.

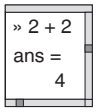
**ii.** Give a specific expression for the mean square error when

$$y(k) = \sin\left(\frac{k\pi}{5}\right).$$

**iii.** Find the eigenvalues and eigenvectors of the Hessian matrix for the mean square error. Locate the minimum point and sketch a rough contour plot.

**iv.** Find the maximum stable learning rate for the LMS algorithm.

**v.** Take three steps of the LMS algorithm by hand, using a stable learning rate. Use the zero vector as the initial guess.




**vi.** Write a MATLAB M-file to implement the LMS algorithm for this problem. Take 40 steps of the algorithm for a stable learning rate and sketch the trajectory on the contour plot. Use the zero vector as the initial guess. Verify that the algorithm is converging to the optimal point.

**vii.** Verify experimentally that the algorithm is unstable for learning rates greater than that found in part (iv).

**E10.14** Repeat Problem P10.9, but use the numerals “1”, “2” and “4”, instead of the letters “T”, “G” and “F”. Test the trained network on each reference pattern and on noisy patterns. Discuss the sensitivity of the network. (*Use the Neural Network Design Demonstration Linear Pattern Classification* (`nnd10lc`).)