

ECE 3640

Adaptive Filter Programming Assignment

Spring 2006

Introduction

An adaptive filter is a filter in which the coefficients are adjusted to force the output of the filter $y[t]$ to match some desired input signal $d[t]$. Usually adaptive filters are FIR. Schematically, the filter is shown in figure 1; the line passing through the box is suggestive of the notation for a variable resistor. In this figure, the error signal

$$e[t] = d[t] - y[t]$$

is fed back around to adjust the coefficients. The key idea of an adaptive filter is this: The coefficients of the filter are adjusted to make the squared error signal $|e[t]|^2$ be as small as possible *on average*, which results in $d[t]$ matching $y[t]$ as closely as possible *on average*. There are a variety of adaptation rules which are used to adjust the coefficients of the filter. In this lab you will be introduced to the least mean squares (LMS) algorithm.

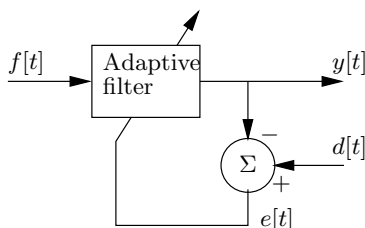


Figure 1: Representation of an adaptive filter

System identification

The basic adaptive filter idea is used in a variety of configurations for various applications, which we outline below.

An adaptive filter can estimate the the transfer function of an unknown plant (or system), using the configuration shown in figure 2. The adaptive filter and the plant are both driven by the same input signal, and the desired signal $d[t]$ is the plant output. The adaptive filter will converge to a “best” representation of the unknown system. In that case, the adaptive filter will be the best possible representation of the unknown plant. (If the system is an IIR system and the adaptive filter is an FIR system, or if the order of the adaptive filter is less than the order of the system, then the adaptive filter can be at best an approximation of the true system response.)

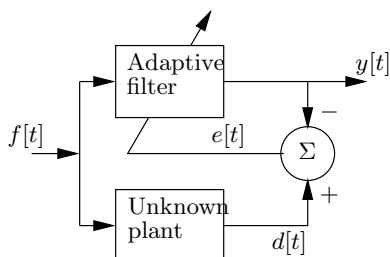


Figure 2: Identification of an unknown plant

Inverse system identification

When the adaptive filter is configured as shown in figure 3, then it will converge when the output of the adaptive filter matches the delayed input of the system $f[t]$ as closely as possible. Ideally, the adaptive filter will converge to the inverse of the plant, so that the cascade of the plant and the adaptive filter is simply a delay. This configuration is employed in some modems to reduce the effect of the channel on the transmitted signal. The signal representing a sequence of input bits ($f[t]$) passes through a channel with an unknown transfer function $H(z)$. At the receiver, the signal is processed by an adapted inverse system before detecting the bits.

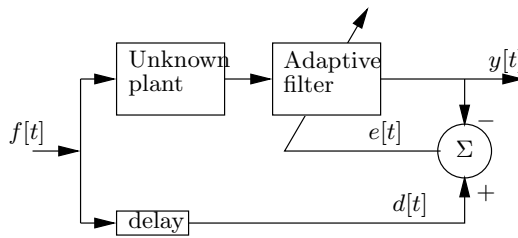


Figure 3: Adapting to the inverse of an unknown plant

Adaptive predictors

In the configuration shown in figure 4, the input to the adaptive filter is a delayed version of the desired signal. In this case, the adaptive filter converges in such a way as to provide a predictor of the input signal (if prediction is possible). For example, suppose there is a delay of 1. Then the input to the filter is $f[t - 1]$. However, the output of the adaptive filter is suppose to match $d[t] = f[t]$. That is, given the input $f[t - 1]$, the filter output is approximately $y[t] \approx f[t]$: the filter figures out out to predict $f[t]$. (Don't run out and try to take on the stock market with this, though!) Such predictors are used in a variety of applications, including data compression, pattern recognition, or spectrum estimation.

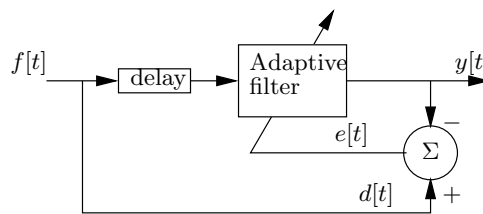


Figure 4: An adaptive predictor

Interference cancellation

In the context of interference cancellation, the signal $d[t]$ is commonly referred to as the “primary signal,” while the filter input is referred to as the “secondary signal.” The primary $d[t]$ is modeled as the sum of a signal of interest, $x[t]$, plus noise:

$$d[t] = x[t] + w[t].$$

The secondary input consists of a noise signal,

$$f[t] = n[t].$$

(see figure 5). As an example, suppose that a background acoustic noise source (say the hum of a fan), $w[t]$,

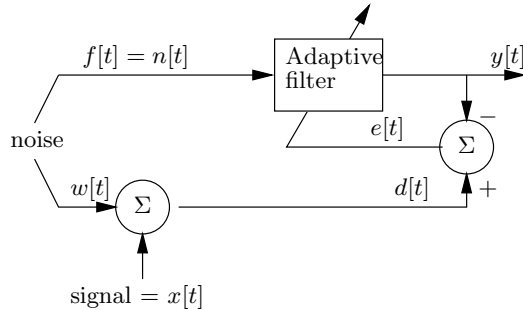


Figure 5: Configuration for interference cancellation

superimposed on a desired audio signal, $x[t]$, which is recorded using a microphone to form the primary input. A second microphone placed far from the desired signal records the noise, $n[t]$, but not the desired signal. There is a different acoustic transfer function between the source and each of the two microphones, hence $n[t]$ is not the same as $w[t]$. The adaptive filter is driven to minimize the error, which adapts to accommodate this difference in transfer function from the noise source. Thus, the resulting difference signal, $e[t]$, will have (insofar as possible) the noise from the reference signal subtracted from the noise from the primary signal.

The interference cancellation configuration has been used in several applications, such as noise cancellation, echo cancellation, and adaptive beamforming in array processing.

Description of the adaptive filter

Let $\mathbf{h} = [h_0, h_1, \dots, h_m]^T$ denote the coefficients of an FIR filter, and let $f[t]$ denote the input to the filter. Then the FIR filter output can be written

$$y[t] = \sum_{i=0}^m h_i f[t-i]. \quad (1)$$

It will be convenient to express this and other expressions using the language of vectors and matrices. Let $\mathbf{f}[t]$ be the vector formed by the current and previous input values,

$$\mathbf{f}[t] = [f[t], f[t-1], f[t-2], \dots, f[t-m]]^T.$$

Then the filter output of (1) can equivalently be given by

$$y[t] = \mathbf{f}[t]^T \mathbf{h}.$$

Suppose that the filter coefficients can change as a function of time (since this is an adaptive filter). Then we can write

$$\mathbf{h}[t] = [h_0[t], h_1[t], \dots, h_m[t]]^T.$$

Now the output of the filter due to the input at time t is

$$y[t] = \mathbf{f}[t]^T \mathbf{h}[t].$$

Suppose that, as in figure 1, we have a desired signal $d[t]$. Then the error between the filter output $y[t]$ and the desired signal $d[t]$ is

$$e[t] = d[t] - y[t].$$

The adaptive filter update rule is now quite simple: Given $e[t]$, the filter coefficients are updated according to

$$\boxed{\mathbf{h}[t+1] = \mathbf{h}[t] + \mu \mathbf{f}[t] e[t]}$$

where μ is some small number which controls how fast the filter can adapt. One interesting thing to note about this update rule is that if $e[t] = 0$ (that is, there is no error) then the adaptive filter weights do not change. This is as it should be: if the filter is doing what it is supposed to, why change?

The entire LMS adaptive filter algorithm can be described as follows:

Input: An input sample $f[t]$ and a desired filter output $d[t]$ and the current filter $\mathbf{h}[t]$, and an adjustment step size μ , and a filter length m .

Internal Storage: The vector \mathbf{f} .

Step 1: Form the vector $\mathbf{f}[t]$ using the current input sample $f[t]$ and previous input samples.

Step 2: Compute the filter output:

$$y[t] = \mathbf{f}[t]^T \mathbf{h}[t]$$

Step 3: Compute the error signal:

$$e[t] = d[t] - y[t]$$

Step 4: Update the filter coefficients

$$\mathbf{h}[t + 1] = \mathbf{h}[t] + \mu \mathbf{f}[t] e[t]$$

Output: Return $y[t]$, $d[t]$, $e[t]$, and $\mathbf{h}[t + 1]$.

How It Works

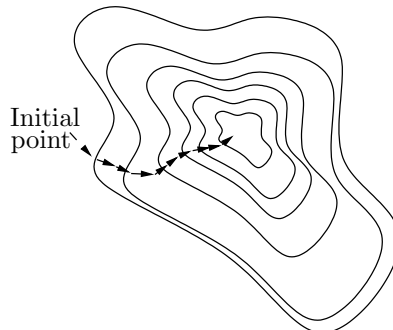
We form a way of “measuring” the error term. Let $J(\mathbf{h})$ be a be the mean squared error. That is, we take the error $e[t]$, square it, then compute the mean (average) value using the expectation operator. (This is the same expectation E as you learn about in your probability class.) We write

$$J(\mathbf{h}) = E[(d[t] - y[t])^2] = E[(d[t] - \mathbf{f}[t]^T \mathbf{h})^2]. \quad (2)$$

When \mathbf{h} has the correct value, then $J(\mathbf{h})$ is going to be small. The overall goal is to minimize $J(\mathbf{h})$ with respect to the coefficient vector \mathbf{h} .

It turns out that this can be done in a closed form way. In fact, the correct value is $\mathbf{h} = E[\mathbf{f}[t]\mathbf{f}[t]^T]^{-1}E[d[t]\mathbf{f}[t]]$. However, this solution does not allow the filter to *adapt* to changing circumstances.

Instead, we will find the solution by the *steepest descent* method. Rather than minimizing all in one step, as before, we compute the gradient of $J(\mathbf{h})$ with respect to \mathbf{h} , and update the current \mathbf{h} by moving in the direction of the negative gradient. That is, we “slide downhill” on the surface of $J(\mathbf{h})$. The idea of sliding downhill is conveyed in the following figure:



This figure shows the contours of a function (of two variables). At each point in the plane, the contours are orthogonal to the direction of the gradient, with the gradient pointing in the direction of greatest increase. Thus, starting at an initial point and moving some small distance from the point in the direction of the negative gradient at that point decreases the function value. Then starting at the new point and moving in the direction of the negative gradient at that point again decreases the function value. A series of such steps will eventually reach a point of local minimum. An update rule such as this is referred to as *steepest descent* or *gradient descent*.

We denote the gradient of $J(\mathbf{h})$ with respect to \mathbf{h} as $\frac{\partial}{\partial \mathbf{h}} J(\mathbf{h})$. Based on this, an update rule can be written as

$$\mathbf{h}[t+1] = \mathbf{h}[t] - \frac{\mu}{2} \frac{\partial}{\partial \mathbf{h}} J(\mathbf{w}) \Big|_{\mathbf{h}[t]} .$$

That is, the filter weights at the next time around, $\mathbf{h}[t+1]$, are obtained by moving from the current weights, $\mathbf{w}[t]$, in the direction of the negative gradient, evaluated at the current weights, $-\frac{\partial}{\partial J(\mathbf{h})} \Big|_{\mathbf{h}[t]}$. The quantity $\mu/2$ is a “step size,” indicating how far the move should be.

It can be shown that (see the exercises)

$$\frac{\partial}{\partial \mathbf{h}} J(\mathbf{h})$$

can be written as

$$\frac{\partial}{\partial \mathbf{h}} J(\mathbf{h}) = 2E[(y[t] - d[t])\mathbf{f}[t]] = -2E[e[t]\mathbf{f}[t]] \quad (3)$$

Hence the weight update rule is

$$\mathbf{h}[t+1] = \mathbf{h}[t] + \mu E[e(t)\mathbf{f}(t)].$$

Now, the expectation operator in this case is problematic to compute. We therefore make the *stochastic gradient assumption*: Assume that the mean value of a random number is close (enough) to the mean value. That is, assume that

$$E[e[t]\mathbf{f}[t]] \approx e[t]\mathbf{f}[t].$$

Then we throw away the expectation operator and obtain the weight update rule

$$\boxed{\mathbf{h}[t+1] = \mathbf{h}[t] + \mu e(t)\mathbf{f}(t).}$$

Assignment

Derivation of the algorithm Show that with $J(\mathbf{h})$ as defined in (2), that both of the equalities in (3) are true. Hint: You need some basic results from vector calculus.

Let \mathbf{h} be a $(m+1) \times 1$ column vector,

$$\mathbf{h} = \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_m \end{bmatrix}$$

and let \mathbf{a} be a $(m+1) \times 1$ column vector, and let R be a symmetric $(m+1) \times (m+1)$ matrix. The gradient of a scalar function $f(\mathbf{h})$ with respect to the vector \mathbf{h} is defined as the vector

$$\frac{\partial}{\partial \mathbf{h}} f = \begin{bmatrix} \frac{\partial f}{\partial h_0} \\ \frac{\partial f}{\partial h_1} \\ \vdots \\ \frac{\partial f}{\partial h_m} \end{bmatrix} .$$

Based on this definition, here is a short table of vector derivatives:

$$\boxed{\frac{\partial}{\partial \mathbf{a}} \mathbf{a}^T \mathbf{h} = \mathbf{a}}$$

$$\boxed{\frac{\partial}{\partial \mathbf{h}} \mathbf{h}^T R \mathbf{h} = 2R\mathbf{h}}$$

Programming the adaptive filter In C++, write a function (perhaps as a member of a class)

```
adfilt(double f, double d, double *h, int m, double mu,
       double &y, double &e)
```

which implements the LMS adaptive filter as described above.

Testing: System Identification Let $P(z) = 0.5 + 0.25z^{-1} + 0.4z^{-2} + 0.5z^{-3} + 0.6z^{-4} + 0.8z^{-5}$ denote a plant which is to be identified. (That is, you know what it is for purposes of simulation, but it models a plant that in “real life” would be unknown.)

1. Write a program which simulates the system shown in figure 2 to do system identification. Let the input $f[t]$ to the plant and adaptive filter be a *random* sequence of ± 1 .
2. Verify that the adaptive filter works by running the input sequence into the system. Plot the squared error signal $|e[t]|^2$ as a function of t and the *average* of $|e[t]|^2$, averaged over 100 independent runs of data. Verify that the average of $|e[t]|^2$ decreases as a function of time.

Do this for various values of μ , such as $\mu = 0.0001$, $\mu = 0.001$, $\mu = 0.01$, $\mu = 0.1$, $\mu = 0.5$, $\mu = 2$, plotting $|e[t]|^2$ and its average as a function of time in each case. Do this all for the cases when the number of coefficients in the filter is more than the number of coefficients in the “plant.” Plot the impulse response of the adaptive filter after it has converged, and comment on whether it looks like the impulse response of the plant.

3. Now repeat the above plots (the error and the average error) in the case when the number of coefficients in the adaptive filter is less than the number of coefficients in the plant, using a value of μ that you have found to be effective. What impulse response do you get in this case?

Experimentation Determine the largest value of μ that you can use. Do this for various filter lengths. Describe what happens when μ is too large, and explain why this occurs.

Testing: Inverse system identification 1. Write a program which simulates the system shown in figure 3. As before, let the input $f[t]$ be a random sequence of ± 1 . Try this first for an FIR plant, $P(z)$ as before, and use a long adaptive filter. After the filter has converged (as well as it can), see if it acts as an inverse filter by convolving $p[t] * h[t]$ — this should look something like a delta function. (Plot the convolved sequence $p[t] * h[t]$.) Experiment with various delays, filter lengths, and μ . Then try an IIR filter, $P(z) = \frac{1}{1-0.9z^{-1}}$. In this case, the inverse filter should be $1 - 0.9z^{-1}$ — see if the adaptive filter converges to something like this. Experiment with various values of μ and adaptive filter length. In all experiments, make plots of the error $|e[t]|^2$ and the average error.

Turn in:

1. Your derivations and other computations.
2. Listings of your programs.
3. The plots you have made, with descriptions of what they represent and what you learn from them.
4. The results of your experiments.
5. A discussion of what you learned with regard to convergence as a function of μ , the effect of the filter length on the performance, and the effect of the filter length on the speed of convergence. Discuss what you learned regarding the system identification and inverse system identification problems. Provide appropriate plots (e.g., squared error as a function of time, filter impulse response, end-to-end convolutions, etc.) to support your discussion. Discuss how such systems might be used in practice.

A note on writing: As you will learn in industry, the documentation part of an engineering project is extremely important. There is usually very little sense in creating something that is not documented, so that it may be fixed, improved, marketed, patented, transferred, or debugged. An important part of your education is learning to document correctly.

So, how much should you write for this project? I would recommend that you take the following guideline for this course: Write to yourself. Assume that you are explaining to a student who has not had this class, but who is otherwise a bright and capable person, what you have done, and why you have done it. Write in such a way that if your boss were to come to you twenty years from now and ask you to do something similar to this project, you could pick up what you have written for this project and use it as the starting point, with every step clearly explained and every result justified.

The grader will be looking for clear writing!