Adaptive Filtering: LMS Algorithm*

Douglas L. Jones
Swaroop Appadwedula
Matthew Berry
Mark Haun
Dima Moussa
Daniel Sachs

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License †

Abstract

This module introduces adaptive filters through the example of system identification using the LMS algorithm. The adaptive filter adjusts its coefficients to minimize the mean-square error between its output and that of an unknown system.

1 Introduction

Figure 1 is a block diagram of system identification using adaptive filtering. The objective is to change (adapt) the coefficients of an FIR filter, W, to match as closely as possible the response of an unknown system, H. The unknown system and the adapting filter process the same input signal $x[n]$ and have outputs $d[n]$(also referred to as the desired signal) and $y[n]$.

*Version 2.14: Jun 1, 2009 9:22 am -0500
†http://creativecommons.org/licenses/by/1.0

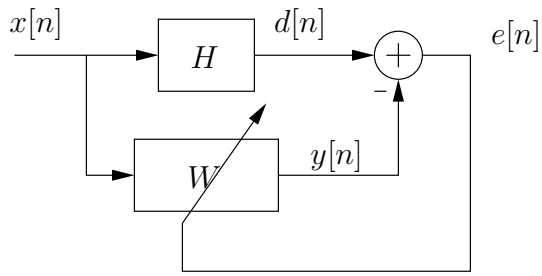

Figure 1: System identification block diagram.

1.1 Gradient-descent adaptation

The adaptive filter, W, is adapted using the least mean-square algorithm, which is the most widely used adaptive filtering algorithm. First the error signal, $e[n]$, is computed as $e[n] = d[n] - y[n]$, which measures the difference between the output of the adaptive filter and the output of the unknown system. On the basis of this measure, the adaptive filter will change its coefficients in an attempt to reduce the error. The coefficient update relation is a function of the error signal squared and is given by

$$h_{n+1}[i] = h_n[i] + \frac{\mu}{2}\left(-\frac{\partial(|e|)^2}{\partial h_n[i]}\right) \tag{1}$$

The term inside the parentheses represents the gradient of the squared-error with respect to the ith coefficient. The gradient is a vector pointing in the direction of the change in filter coefficients that will cause the greatest increase in the error signal. Because the goal is to minimize the error, however, (1) updates the filter coefficients in the direction opposite the gradient; that is why the gradient term is negated. The constant μ is a step-size, which controls the amount of gradient information used to update each coefficient. After repeatedly adjusting each coefficient in the direction opposite to the gradient of the error, the adaptive filter should converge; that is, the difference between the unknown and adaptive systems should get smaller and smaller.

To express the gradient decent coefficient update equation in a more usable manner, we can rewrite the derivative of the squared-error term as

$$\begin{aligned} \frac{\partial(|e|)^2}{\partial h[i]} &= 2\frac{\partial e}{\partial h[i]}e \\ &= 2\frac{\partial(d-y)}{\partial h[i]}e \\ &= \left(2\frac{\partial\left(d-\sum_{i=0}^{N-1}h[i]x[n-i]\right)}{\partial h[i]}\right)(e) \end{aligned} \tag{2}$$

$$\frac{\partial(|e|)^2}{\partial h[i]} = 2(-x[n-i])e \tag{3}$$

which in turn gives us the final LMS coefficient update,

$$h_{\text{n}+1}[i] = h_n[i] + \mu e x[n-i] \tag{4}$$

The step-size μ directly affects how quickly the adaptive filter will converge toward the unknown system. If μ is very small, then the coefficients change only a small amount at each update, and the filter converges slowly. With a larger step-size, more gradient information is included in each update, and the filter converges more quickly; however, when the step-size is too large, the coefficients may change too quickly and the filter will diverge. (It is possible in some cases to determine analytically the largest value of μ ensuring convergence.)

2 MATLAB Simulation

Simulate the system identification block diagram shown in Figure 1.

Previously in MATLAB, you used the `filter` command or the `conv` command to implement shift-invariant filters. Those commands will not work here because adaptive filters are shift-varying, since the coefficient update equation changes the filter's impulse response at every sample time. Therefore, implement the system identification block on a sample-by-sample basis with a `do` loop, similar to the way you might implement a time-domain FIR filter on a DSP. For the "unknown" system, use the fourth-order, low-pass, elliptical, IIR filter designed for the IIR Filtering: Filter-Design Exercise in MATLAB[1].

Use Gaussian random noise as your input, which can be generated in MATLAB using the command `randn`. Random white noise provides signal at all digital frequencies to train the adaptive filter. Simulate the system with an adaptive filter of length 32 and a step-size of 0.02. Initialize all of the adaptive filter coefficients to zero. From your simulation, plot the error (or squared-error) as it evolves over time and plot the frequency response of the adaptive filter coefficients at the end of the simulation. How well does your adaptive filter match the "unknown" filter? How long does it take to converge?

Once your simulation is working, experiment with different step-sizes and adaptive filter lengths.

3 Processor Implementation

Use the same "unknown" filter as you used in the MATLAB simulation.

Although the coefficient update equation is relatively straightforward, consider using the `lms` instruction available on the TI processor, which is designed for this application and yields a very efficient implementation of the coefficient update equation.

To generate noise on the DSP, you can use the PN generator from the Digital Transmitter: Introduction to Quadrature Phase-Shift Keying[2], but shift the PN register contents up to make the sign bit random. (If the sign bit is always zero, then the noise will not be zero-mean and this will affect convergence.) Send the desired signal, $d[n]$, the output of the adaptive filter, $y[n]$, and the error to the D/A for display on the oscilloscope.

When using the step-size suggested in the MATLAB simulation section, you should notice that the error converges very quickly. Try an extremely small μ so that you can actually watch the amplitude of the error signal decrease towards zero.

4 Extensions

If your project requires some modifications to the implementation here, refer to *Haykin* [1] and consider some of the following questions regarding such modifications:

- How would the system in Figure 1 change for different applications? (noise cancellation, equalization, *etc.*)
- What happens to the error when the step-size is too large or too small?
- How does the length of an adaptive FIR filters affect convergence?
- What types of coefficient update relations are possible besides the described LMS algorithm?

[1] "IIR Filtering: Filter-Design Exercise in MATLAB" <http://cnx.org/content/m10623/latest/>
[2] "Digital Transmitter: Introduction to Quadrature Phase-Shift Keying" <http://cnx.org/content/m10042/latest/>

References

[1] S. Haykin. *Adaptive Filter Theory*. Prentice Hall, 3rd edition edition, 1996.