Least-Mean-Square Algorithm

This is probably the most important area of study in the course!

Introduction and the Big Picture

We've seen from our work in Wiener filters that given **R** and **p** we can compute the optimal FIR filter (optimal in the mean-squared error sense)

$$\boxed{\mathbf{w}_o = \mathbf{P}^{-1}\boldsymbol{\pi}}.$$

If we wish to avoid the matrix inverse, we can also search the performance surface for the optimal filter coefficients using the steepest-descent algorithm

$$\boxed{\mathbf{w}(n+1) = \boldsymbol{\omega}(\nu) + \mu[\boldsymbol{\pi} - \mathbf{P}\boldsymbol{\omega}(\nu)]}$$

where the step-size is chosen to satisfy

$$0 < \mu < \frac{2}{\lambda_{\mu\alpha\xi}}$$

and **w**(0) arbitrarily chosen. This results in

$$\lim_{n\to\infty} \boldsymbol{\omega}(\nu) = \boldsymbol{\omega}_o$$

and consequently

$$\lim_{n\to\infty} J(n) = J_{\min}.$$

The shortcoming of either method is that if the statistics of the signal change, **R** and **p** are no longer valid and our filter **w**(*n*) may not be optimal anymore. A more serious shortcoming is that in all likelihood, **R** and **p** are *not* know.

We have a couple of ways around this.

1) Periodically estimate **R** and **p**, [$\hat{\mathbf{R}}(n)$ and $\hat{\mathbf{p}}(n)$] and compute the optimal (Wiener) filter, $\mathbf{w}_o(n) = \hat{\mathbf{P}}^{-1}(\nu)\hat{\boldsymbol{\pi}}(\nu)$ for every new estimate.
2) Periodically estimate **R** and **p**, [$\hat{\mathbf{R}}(n)$ and $\hat{\mathbf{p}}(n)$] and search for $\mathbf{w}_o$

$$\mathbf{w}(n+1) = \boldsymbol{\omega}(\nu) + \mu[\hat{\boldsymbol{\pi}}(\nu) - \hat{\mathbf{P}}(\nu)\boldsymbol{\omega}(\nu)].$$

We hope our search is completed before our new estimate comes in otherwise we won't be filtering optimally.

How "optimal" our filtering is depends on how good our estimates, $\hat{\mathbf{R}}$ and $\hat{\mathbf{p}}$ are.

For now we'll investigate a way to do 2) and thus avoid the matrix inverse. This will lead to the celebrated family of least-mean-square (LMS) algorithms.

Later we'll revisit 1) and attempt to recursively compute $\mathbf{R}^{-1}$ rather than recalculating $\mathbf{R}^{-1}$ every sample. This will lead to the family of recursive least-squares (RLS) algorithms.

Least-Mean-Square Algorithm

Recall our steepest-descent algorithm

$$\mathbf{w}(n+1) = \boldsymbol{\omega}(\nu) + \mu'[-\nabla\hat{\mathcal{J}}(\nu)]$$

where

$$\begin{aligned}\nabla J(n) &= -2E[\mathbf{u}(n)e^*(n)] \\ &= -2\mathbf{p} + 2\mathbf{R}\mathbf{w}(n)\end{aligned}.$$

This leads to the recursion

$$\begin{aligned}\mathbf{w}(n+1) &= \boldsymbol{\omega}(\nu) + \mu'[2\boldsymbol{\pi} - 2\mathbf{P}\boldsymbol{\omega}(\nu)] \\ &= \boldsymbol{\omega}(\nu) + \mu[\boldsymbol{\pi} - \mathbf{P}\boldsymbol{\omega}(\nu)]\end{aligned}.$$

We note that **R** is calculated by averaging an *infinite* number of outer products, $\mathbf{u}(n)\mathbf{u}^H(\nu)$. Now in the more realistic setting, **R** is unknown and we must estimate

$$\mathbf{R} = E[\mathbf{u}(\nu)\mathbf{u}^H(\nu)].$$

Therefore we use an *instantaneous* estimate based on a *single* outer product $\mathbf{u}(n)\mathbf{u}^H(\nu)$

$$\hat{\mathbf{R}}(n) = \mathbf{u}(\nu)\mathbf{u}^H(\nu).$$

Likewise we choose an *instantaneous* estimate of **p**

$$\hat{\mathbf{p}}(n) = \mathbf{u}(\nu)\delta^*(\nu).$$

Substitution into our recursion (SD) we have

$$\begin{aligned}\hat{\mathbf{w}}(n+1) &= \hat{\boldsymbol{\omega}}(\nu) + \mu[\hat{\boldsymbol{\pi}}(\nu) - \hat{\mathbf{P}}(\nu)\hat{\boldsymbol{\omega}}(\nu)] \\ &= \hat{\boldsymbol{\omega}}(\nu) + \mu[\mathbf{u}(\nu)\delta^*(\nu) - \mathbf{u}(\nu)\mathbf{u}^H(\nu)\hat{\boldsymbol{\omega}}(\nu)]. \\ &= \hat{\boldsymbol{\omega}}(\nu) + \mu\mathbf{u}(\nu)[\delta^*(\nu) - \mathbf{u}^H(\nu)\hat{\boldsymbol{\omega}}(\nu)]\end{aligned}$$

We note that our estimation error is given by

$$\begin{aligned}e^*(n) &= \delta^*(\nu) - \psi(\nu) \\ &= \delta^*(\nu) - \mathbf{u}^H(\nu)\hat{\boldsymbol{\omega}}(\nu)\end{aligned}$$

and hence we arrive at the final LMS update equation

$$\boxed{\hat{\mathbf{w}}(n+1) = \hat{\boldsymbol{\omega}}(\nu) + \mu\mathbf{u}(\nu)\varepsilon^*(\nu)}.$$

Figure 9.1a

LMS Algorithm

Initialize (optional)

$\mathbf{u}(0) = \hat{\boldsymbol{\omega}}(0) = [0 \Lambda\ 0]^T$

for $k \geq 0$

$e(n) = \delta(\nu) - \hat{\boldsymbol{\omega}}^H(\nu)\mathbf{u}(\nu)$

$\hat{\boldsymbol{\omega}}(\nu+1) = \hat{\boldsymbol{\omega}}(\nu) + \mu\mathbf{u}(\nu)\varepsilon^*(\nu)$

end

Comments

1) $e^*(n)$ is first calculated using the current coefficient vector, $\hat{\mathbf{w}}(n)$.

2) The correction to the coefficient vector, $\mu \mathbf{u}(n) e^*(n)$ is then added to the current coefficient vector, $\hat{\mathbf{w}}(n)$. Note we are required to buffer $u(n)$, ..., $u(n - M + 1)$ to do the update but we need to do this anyway to do the filtering, $y(n) = \hat{\mathbf{w}}^H(n)\mathbf{u}(n)$.

3) Filtering and coefficient update requires (calculate this)

$2M + 1$ complex multiplications
$2M$ complex additions

Note this is an $O(2M)$ algorithm and is extremely simple to implement. Simplicity ($2M + 2$ DSP instructions/cycles) opens up the possibility of real-time applications even for large M.

4) LMS algorithm is the benchmark by which all other adjustment algorithms are compared to.

5) The consequence of using $\hat{\mathbf{R}}$ and $\hat{\mathbf{p}}$ is that we use an estimate of the gradient, $\hat{\nabla}J(n)$. Recall that in SD we always made a change in the direction opposite $\nabla J(n)$. Now we'll be making a change in the direction opposite the $\hat{\nabla}J(n)$. If our estimate, $\hat{\nabla}J(n)$ has the wrong sign, we'll be moving in the wrong direction. However, since the LMS is recursive in nature the algorithm itself effectively averages these estimates during the course of adaptation and we head toward the optimal Wiener filter (still to be proved).

Nevertheless our trajectory will be *noisy*, i.e. we will shake, rattle, and roll on our way to the optimal filter. Algorithms which use a noisy gradient estimate are classified as *stochastic gradient algorithms*.

Examples

You should examine these examples for an appreciation of adaptive filtering techniques to a wide variety of signal processing problems. Note that all examples are classified into one of our four configurations discussed in Chapter 1.

Stability and Performance Analysis of the LMS Algorithm

Assuming that μ is selected properly, the SD algorithm achieves (note no expectations)

$$\lim_{n \to \infty} \mathbf{w}(n) = \mathbf{w}_o$$

or

$$\lim_{n \to \infty} \boldsymbol{\xi}(n) = \lim_{n \to \infty} [\mathbf{w}(n) - \mathbf{w}_o] = \mathbf{0}$$

and thus

$$\lim_{n \to \infty} J(n) = J_{\min} .$$

For the LMS, however, we use $\hat{\mathbf{R}}(n)$, $\hat{\mathbf{p}}(n)$ as instantaneous estimates for $\mathbf{R}$, $\mathbf{p}$ (resp.) or equivalently an instantaneous gradient estimate. Therefore we cannot assume

$$\lim_{n \to \infty} \hat{\mathbf{w}}(n) = \mathbf{w}_o$$

nor

$$\lim_{n\to\infty} \vartheta(n) = \vartheta_{\mu v} \ .$$

Since the LMS algorithm is based on estimates of **R** and **p**, our stability analysis will have to examine the above criteria in a mean-sense where we take into account an expectation of the above criteria based on our estimates. In this case we'll show convergence on two levels:

1) Convergence of the mean (weaker condition, easy to show), i.e. $\lim_{n\to\infty} E[\hat{\boldsymbol{\omega}}(n)] = \boldsymbol{\omega}_o$ or equivalently

$$\lim_{n\to\infty} E[\hat{\boldsymbol{\omega}}(n) - \boldsymbol{\omega}_o] = \lim_{n\to\infty} E[\varepsilon(n)] = 0$$

2) Convergence in the mean square (stronger condition, hard to show), i.e. $\lim_{n\to\infty} J(n) = c$ where c is a constant and $c \geq J_{\min}$. Note that condition 1) suggests on *average* $\hat{\mathbf{w}}(n) \to \mathbf{w}_o$ but this does not mean $\lim_{n\to\infty} J(n) \approx J_{\min}$ everytime.

Remember that we're ultimately after $\hat{\mathbf{w}}(n)$ that minimizes the MSE so convergence in the mean square will be the stronger, more important condition to prove.