Adaptive Filtering

In this section, we will see how we can use least-squares to **learn** the impulse response of a linear time-invariant system by observing its input and output. We have seen already how we might set this up as a linear inverse problem (see the example at the beginning of Chapter II of these notes), and using the work we did on recursive least-squares earlier in this chapter, you might imagine how we could solve this problem in a streaming manner[1].

What we will do differently in this section is explicitly set the problem up as a basic statistical estimation problem, and then show how this leads to an even cheaper (computationally) algorithm than recursive least-squares. This algorithm (called the LMS[2] filter), introduced in 1960, is also the first example of a **stochastic gradient** algorithm, which is currently a hot topic in large-scale machine learning.

Here is a careful problem statement. We observe the convolution

$$y[n] = \sum_{k=0}^{N-1} h_*[k]u[n-k],$$

where

- $u[n]$ is the input signal (observed),
- $y[n]$ is the output signal (observed), and
- $h_*[n]$ is the impulse response of (finite) length N (unknown).

We want to estimate $\boldsymbol{h}_* \in \mathbb{R}^N$ after observing $u[n]$ and $y[n]$ over some amount of time. We also want our estimate to naturally adjust to the impulse response drifting over time.

[1] In fact, we will make this connection explicitly a little later in this section.
[2] Least Mean Squares.

Our model is that the input signal $u[n]$ is a **wide-sense stationary** random process. This means that its second-order statistics do not change over time. More precisely, we will assume that every sample of the input signal is zero-mean:

$$\mathrm{E}[u[n]] = 0, \quad \text{for all } n,$$

and that the correlation (covariance) between two samples depends only on the difference between the sample locations[3]:

$$\underbrace{R[n,\ell]}_{\text{function of 2 variables}} \quad = \quad \mathrm{E}[u[n]u[\ell]] \quad = \quad \underbrace{R_{uu}[n-\ell]}_{\text{function of 1 variable}} \quad .$$

Note that a consequence of this is that the variance $\sigma^2[n] = \mathrm{E}[u[n]^2] = R_{uu}[0]$ is constant for all n.

Associated with each set of candidate coefficients $h[n]$, $n = 0, \ldots, N-1$ is the error (or residual)

$$e[n] = y[n] - \sum_{k=0}^{N-1} h[k]u[n-k].$$

With the model in place, we want to find the filter coefficients that minimize the **mean square-error** of the error signal:

$$\min_{\boldsymbol{h} \in \mathbb{R}^N} \mathrm{E}[|e[n]|^2]. \tag{1}$$

A quick calculation shows that the wide-sense stationarity of $u[n]$

[3] This is like saying the covaraince "matrix" is Toeplitz, where we put the word matrix in quotes since the indices $n, \ell \in \mathbb{Z}$ extend infinitely in either direction.

means that this variance is independent of n:

$$\begin{aligned}
\mathrm{E}[|e[n]|^2] &= \mathrm{E}\left[\sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1}(h_*[k]-h[k])(h_*[\ell]-h[\ell])u[n-k]u[n-\ell]\right] \\
&= \sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1}(h_*[k]-h[k])(h_*[\ell]-h[\ell])\,\mathrm{E}[u[n-k]u[n-\ell]] \\
&= \sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1}(h_*[k]-h[k])(h_*[\ell]-h[\ell])R_{uu}[\ell-k].
\end{aligned}$$

(Notice that n does not appear anywhere in that last expression.)

The Wiener-Hopf equations

Minimizing (1) is a standard least-squares problem. We will derive the conditions for the minimizer by calculating the gradient:

$$\begin{aligned}
\frac{\partial\,\mathrm{E}\,|e[n]|^2}{\partial h[k]} &= \mathrm{E}\left[\frac{\partial |e[n]|^2}{\partial h[k]}\right] \\
&= 2\,\mathrm{E}\left[e[n]\frac{\partial e[n]}{\partial h[k]}\right] \quad \text{(chain rule)} \\
&= -2\,\mathrm{E}[e[n]u[n-k]] \\
&= -2\,\mathrm{E}[y[n]u[n-k]] + 2\sum_{i=0}^{N-1} h[i]\,\mathrm{E}[u[n-i]u[n-k]]
\end{aligned}$$

Define the **cross correlation** between the output $y[n]$ and input $u[n]$ as

$$R_{uy}[k] = \mathrm{E}[y[n]u[n-k]].$$

(For similar reasons as in the previous section, the cross-correlation will only depend on the difference between the indexes.) Then the

optimality condition[4]

$$\frac{\partial \, \mathrm{E}\,|e[n]|^2}{\partial h[k]} = 0, \quad \text{for } k = 0, 1, \ldots, N-1,$$

can be written as

$$\boldsymbol{R}\boldsymbol{h} = \boldsymbol{p}, \tag{2}$$

where

$$\boldsymbol{h} = \begin{bmatrix} h[0] \\ h[1] \\ \vdots \\ h[N-1] \end{bmatrix},$$

$$\boldsymbol{R} = \begin{bmatrix} R_{uu}[0] & R_{uu}[1] & \cdots & R_{uu}[N-1] \\ R_{uu}[1] & R_{uu}[0] & \cdots & R_{uu}[N-2] \\ \vdots & \vdots & \ddots & \vdots \\ R_{uu}[N-1] & R_{uu}[N-2] & \cdots & R_{uu}[0] \end{bmatrix},$$

$$\boldsymbol{p} = \begin{bmatrix} R_{uy}[0] \\ R_{uy}[1] \\ \vdots \\ R_{uy}[N-1] \end{bmatrix}.$$

These are the so-called *Wiener-Hopf* equations.

So given intimate knowledge of the statistics of input and output (i.e. the autocorrelation function $\boldsymbol{R}_{uu}$ and cross-correlation function $\boldsymbol{R}_{uy}$ at lags 0 to $N-1$), we can find the optimal set of filter weights by solving a symmetric positive-definite (and Toeplitz!) system of equations.

[4]It is easy to check that this program is convex.

The LMS algorithm

In the vast majority of applications, we have to build up estimates of $\boldsymbol{p}$ and $\boldsymbol{R}$ from the observed data. The popular LMS algorithm uses very crude estimates which have the advantage that they are simple, result in inexpensive computations, and allow the filter to adapt to changing statistics in an agile way.

For fixed $\boldsymbol{R}$ and $\boldsymbol{p}$, the system of equations in (2) can be solved iteratively using steepest descent. Given an estimate $\boldsymbol{h}_n$ at iteration n, we update our solution by moving in the direction of the gradient:

$$\boldsymbol{h}_{n+1} = \boldsymbol{h}_n + \mu_n \left(\boldsymbol{p} - \boldsymbol{R}\boldsymbol{h}_n\right), \tag{3}$$

where μ_n is some stepsize. We have seen that in this case, there is a clear best-choice for the μ_n:

$$\mu_n = \frac{\boldsymbol{r}_n^T \boldsymbol{r}_n}{\boldsymbol{r}_n^T \boldsymbol{R} \boldsymbol{r}_n}, \quad \text{where} \quad \boldsymbol{r}_n = \boldsymbol{p} - \boldsymbol{R}\boldsymbol{h}_n,$$

as this will minimize the functional along the ray starting at $\boldsymbol{h}_n$ and proceeding in the direction of the gradient $\boldsymbol{r}_n$.

The LMS algorithm works from iterations similar in form to (3). But now we assume that at each iteration, we receive a new input sample $u[n]$ and new output sample $y[n] = \sum_{k=0}^{N-1} h_*[k]u[n-k] + \text{noise}$. LMS uses this new information to form the simplest estimate of $\boldsymbol{R}$ and $\boldsymbol{p}$ possible. Let $\boldsymbol{u}_n$ be the vector consisting of the last N input samples:

$$\boldsymbol{u}_n = \begin{bmatrix} u[n] \\ u[n-1] \\ \vdots \\ u[n-N+1] \end{bmatrix}.$$

LMS sets

$$\tilde{\boldsymbol{R}}_n = \boldsymbol{u}_n \boldsymbol{u}_n^T, \quad \tilde{\boldsymbol{p}}_n = y[n]\boldsymbol{u}_n,$$

which essentially estimates the statistics from a single point — note that $\mathrm{E}[\tilde{\boldsymbol{R}}_n] = \boldsymbol{R}$ and $\mathrm{E}[\tilde{\boldsymbol{p}}_n] = \boldsymbol{p}$. The iteration (3) then becomes

$$\boldsymbol{h}_{n+1} = \boldsymbol{h}_n + \mu \left(y[n] - \boldsymbol{u}_n^T \boldsymbol{h}_n \right) \boldsymbol{u}_n.$$

Note that $\boldsymbol{u}_n^T \boldsymbol{h}_n = \tilde{y}[n]$ is the output of your candidate filter whose taps are the current weights $\boldsymbol{h}_n$. We have also fixed the stepsize.

LMS Adaptive Filter

Initialize: $\boldsymbol{h}_0 = \boldsymbol{0}$

for $n = 1, 2, 3, \ldots$ **do**

(observe input $u[n]$ and output $y[n]$)

$\boldsymbol{u}_n = \begin{bmatrix} u[n] & u[n-1] & \cdots & u[n-N+1] \end{bmatrix}^{\mathrm{T}}$

$\boldsymbol{h}_n = \boldsymbol{h}_{n-1} + \mu \left(y[n] - \boldsymbol{u}_n^T \boldsymbol{h}_{n-1} \right) \boldsymbol{u}_n$

end for

It can be shown (but we will not do so here) that the LMS algorithm converges when the stepsize is small enough, specifically

$$\mu \ \leq \ \frac{2}{\mathrm{trace}(\boldsymbol{R})} = \frac{2}{N R_{uu}[0]}.$$

There are also many results that give the speed of convergence under various assumptions on the true $\boldsymbol{h}_*$ and covariance matrix $\boldsymbol{R}$. In general, these results say that to get within ϵ relative error, we need either $\sim 1/\epsilon$ or $\sim 1/\epsilon^2$ iterations (depending on the assumptions). Notice that this is dramatically worse than the $\sim \log(1/\epsilon)$ required

for steepest descent[5]. But also notice that the iterations are much cheaper.

Notice that the direction we move depends only on the current output value and the last N input samples. This means LMS naturally tracks changing filter coefficients. The convergence results mentioned above can be very easily translated into characterizations of how closely we can track dynamic $\boldsymbol{h}_*$.

Adaptive filtering using RLS

Of course, we can use our recursive least-squares (RLS) algorithm in a similar manner to do adaptive filtering as well.

There are multiple ways we might derive this algorithm. The first is from a standard least-squares perspective. When we observe

$$y[n] = \sum_{k=0}^{N-1} h_*[k]u[n-k],$$

this is the same as

$$y[n] = \boldsymbol{A}_n \boldsymbol{h}_*,$$

where $\boldsymbol{A}_n$ is the $1 \times N$ matrix

$$\boldsymbol{A}_n = \begin{bmatrix} u[n] & u[n-1] & \dots & u[n-N+1] \end{bmatrix} = \boldsymbol{u}_n^{\mathrm{T}}.$$

With all of the measurement vectors upto time M stacked up:

$$\underline{\boldsymbol{A}}_M = \begin{bmatrix} \boldsymbol{A}_0 \\ \boldsymbol{A}_1 \\ \vdots \\ \boldsymbol{A}_M \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_0^{\mathrm{T}} \\ \boldsymbol{u}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{u}_M^{\mathrm{T}} \end{bmatrix}, \quad \underline{\boldsymbol{y}}_M = \begin{bmatrix} y[0] \\ y[1] \\ \vdots \\ y[M] \end{bmatrix},$$

[5]Suppose $\epsilon = 10^{-3}$. What are $\log(1/\epsilon)$, $1/\epsilon$, $1/\epsilon^2$?

we see that

$$\underline{\boldsymbol{A}}_M^{\mathrm{T}}\underline{\boldsymbol{A}}_M = \sum_{m=0}^{M} \boldsymbol{u}_m \boldsymbol{u}_m^{\mathrm{T}}.$$

Thus $\frac{1}{M+1}\underline{\boldsymbol{A}}_M^{\mathrm{T}}\underline{\boldsymbol{A}}_M$ is an estimate of the covariance matrix $\boldsymbol{R}$ in the previous section based on all of the samples up until time M. Likewise,

$$\underline{\boldsymbol{A}}_M^{\mathrm{T}}\underline{\boldsymbol{y}}_M = \sum_{m=0}^{M} y[m]\boldsymbol{u}_m$$

gives us a scaled estimate of $\boldsymbol{p}$. So the standard least-squares optimality condition

$$\underline{\boldsymbol{A}}_M^{\mathrm{T}}\underline{\boldsymbol{A}}_M\boldsymbol{h} = \underline{\boldsymbol{A}}_M^{\mathrm{T}}\underline{\boldsymbol{y}}_M$$

can be interpreted as the Wiener-Hopf equations with estimated covariance and cross-covariances:

$$\tilde{\boldsymbol{R}} = \frac{1}{M+1}\underline{\boldsymbol{A}}_M^{\mathrm{T}}\underline{\boldsymbol{A}}_M, \quad \tilde{\boldsymbol{p}} = \frac{1}{M+1}\underline{\boldsymbol{A}}_M^{\mathrm{T}}\underline{\boldsymbol{y}}_M.$$

Now we can apply the recursive least-squares algorithm from page 58 of these notes, giving us the following iteration (we have moved things around to make things as efficient as possible for this special case):

$$\begin{aligned}
\boldsymbol{v}_n &= \boldsymbol{P}_{n-1}\boldsymbol{u}_n \\
\boldsymbol{k}_n &= \frac{1}{1+\boldsymbol{u}_n^{\mathrm{T}}\boldsymbol{v}_n}\,\boldsymbol{v}_n \\
\boldsymbol{h}_n &= \boldsymbol{h}_{n-1} + (y[n] - \boldsymbol{u}_n^{\mathrm{T}}\boldsymbol{h}_{n-1})\boldsymbol{k}_n \\
\boldsymbol{P}_n &= \boldsymbol{P}_{n-1} - \boldsymbol{k}_n\boldsymbol{v}_n^{\mathrm{T}}\boldsymbol{P}_{n-1}.
\end{aligned}$$

To make the filter more agile to dynamic $\boldsymbol{h}_*$, it is typical in practice to slowly forget old inputs. This amounts to weighting the terms in

the covariance estimation with the most recent observations being weighted the most heavily:

$$\tilde{\boldsymbol{R}} = \sum_{m=0}^{M} \lambda^{M-m} \boldsymbol{u}_m \boldsymbol{u}_m^{\mathrm{T}}, \tag{4}$$

for some constant λ just a little bit smaller than 1. We can still apply the matrix inversion lemma in the same way, resulting in the algorithm below.

RLS Adaptive Filter

Initialize: $\boldsymbol{h}_0 = \boldsymbol{0}$, $\boldsymbol{P}_0 = \delta^{-1}\mathbf{I}$.

for $n = 1, 2, 3, \ldots$ **do**

(observe input $u[n]$ and output $y[n]$)

$\boldsymbol{u}_n = \begin{bmatrix} u[n] & u[n-1] & \cdots & u[n-N+1] \end{bmatrix}^{\mathrm{T}}$

$\boldsymbol{v}_n = \boldsymbol{P}_{n-1}\boldsymbol{u}_n$

$\boldsymbol{k}_n = \frac{1}{\lambda + \boldsymbol{u}_n^{\mathrm{T}}\boldsymbol{v}_n}\boldsymbol{v}_n$

$\boldsymbol{h}_n = \boldsymbol{h}_{n-1} + (y[n] - \boldsymbol{u}_n^{\mathrm{T}}\boldsymbol{h}_{n-1})\boldsymbol{k}_n$

$\boldsymbol{P}_n = \lambda^{-1}\boldsymbol{P}_{n-1} - \lambda^{-1}\boldsymbol{k}_n\boldsymbol{v}_n^{\mathrm{T}}\boldsymbol{P}_{n-1}$

end for

The RLS algorithm tends to converge much more quickly than LMS, but notice that each iteration is much more computationally expensive — we require a matrix-vector multiply instead of just a few inner products.

We can concretely connect the parameters δ, λ in the algorithm above to the problem formulation. Recall that we are building up an estimate of the covariance matrix as we proceed in time; this estimate

is $\boldsymbol{P}_n^{-1}$. Taking $\boldsymbol{P}_0 = \delta^{-1}\mathbf{I}$ corresponds to an initial covariance estimate of $\delta\mathbf{I}$. This seems like as good a starting point as any. This initialization eventually fades away, as

$$\boldsymbol{P}_n^{-1} = \sum_{m=0}^{n} \lambda^{n-m} \boldsymbol{u}_m \boldsymbol{u}_m^{\mathrm{T}} + \delta\lambda^n \mathbf{I},$$

which goes to (4) as n increases.

The algorithm above can also we interpreted as solving a regularized weighted least-squares problem. At time n, we are choosing $\boldsymbol{h}_n$ that solves

$$\underset{\boldsymbol{h}}{\text{minimize}} \ \sum_{m=0}^{n} \lambda^m \left| y[m] - \boldsymbol{u}_m^{\mathrm{T}} \boldsymbol{h} \right|^2 + \delta\lambda^n \|\boldsymbol{h}\|_2^2.$$

This is a Tikhonov-regularized least-squares problem, where the regularization parameter is getting smaller as n increases.