9.1 Convergence rate of LMS

Reconsider the problem of predicting \(d(n)\) - the desired output at time \(n\), from the last \(k\) elements in the input sequence: \(u(n) = [u(n), u(n-1), \ldots, u(n-k+1)]\). The error at iteration \(n\) is

\[\epsilon(n) = d(n) - \hat{w}(n)^T u(n)\]

The LMS algorithm is derived by computing the gradient of \(\epsilon(n)^2\) with respect to \(w\), which yields the update rule

\[w(n+1) = w(n) + \mu \epsilon(n) u(n)\]

This is a simple and easy to compute update, however, it’s convergence can be slow.

The convergence rate of LMS is determined by the learning rate \(\mu\) and the eigen-values of the auto-correlation matrix \(R\) (see definition in a previous class), denoted \(\lambda_1 \geq \lambda_2 \geq \cdots\). To ensure convergence we have to set the learning rate sufficiently low:

\[\mu \leq \frac{2}{\lambda_1}\]

This causes a problem when \(\lambda_1\) is much larger than some of the other eigen-values. Setting \(\mu\) small enough to ensure convergence in the direction of the first eigen-vector results in very slow convergence in the directions of eigen-vectors with small eigen-value. This is called the “eigenvalue spread problem”.

The Recursive Least Squares method (RLS) overcomes the eigenvalue spread problem by, in effect, inverting the correlation matrix so that the update rule operates in a normalized space, where all of the eigen-vectors are equal. On the other hand, RLS is more computationally intensive than LMS.

9.2 Recursive Least Squares

(From Haykin’s book, page 436)

RLS is based on minimizing the exponentially discounted average square error. Denoting the error at time \(n\) by \(\epsilon(n)\), we define the exponentially discounted error to be

\[\mathcal{E}(n) = \sum_{i=1}^{n} \gamma^{n-i} \epsilon(i)^2 = \sum_{i=1}^{n} \gamma^{n-i} (d(i) - w(i)^T u(i))^2\]

The parameter \(\gamma\) is the “discount factor”, also called the “forgetting factor”. It is set to a value smaller but close to 1. When \(\gamma = 1\) we get the time-average of the squared error. If the stochastic process generating the sequence is stationary and ergodic then this time average converges to the Wiener optimal solution for large values of \(n\).

Minimizing the cost function with respect to \(w(n)\) for small values of \(n\) results in an ill-posed problem and an estimation method that is unstable. To cure this problem we add a so-called “regularization term”
to the cost function and redefine it as

\[ E(n) = \sum_{i=1}^{n} \gamma^{n-i} \epsilon(i)^2 + \delta \gamma^n \|w(i)\|^2 \]

Where \( \delta > 0 \) is a parameter that controls the level of regularization. We set \( \delta \) high if the signal-to-noise ratio is low. The effect of the regularization term vanishes as \( n \) increases.

Setting the gradient of \( E(n) \) with respect to \( w(n) \) to zero yields equations that are very similar to the Wiener equations for the expected squared error. Instead of the covariance matrix \( R \) we have the matrix \( \Phi \) defined as:

\[ \Phi(n) = \sum_{i=1}^{n} \gamma^i u(i)u(i)^T + \delta \gamma^n I \]

Where \( u(i)u(i)^T \) is the \( k \times k \) matrix defined by the outer product of \( u \) with itself. The sum \( \sum_{i=1}^{n} \gamma^i u(i)u(i)^T \) is the exponentially discounted covariance and \( \delta \gamma^n I \) is the regularization term (\( I \) is the unit matrix). We can define \( \Phi \) recursively as follows:

\[ \Phi(n) = \gamma \Phi(n-1) + u(n)u(n)^T \]

Similarly, the cross-correlation vector \( P \) in the Weiner equation is replaced by a

\[ z(n) = \sum_{i=1}^{n} \gamma^{n-i} u(i)d(i) \]

Which can be expressed recursively as

\[ z(n) = \gamma z(n-1) + u(n)d(n) \]

The formula for the optimal \( \hat{w}(n) \) is in this case.

\[ \Phi(n)\hat{w}(n) = z(n) \]

Note that unlike Weiner’s equation, which relates expected values of the signals, the values related here are random variables which are functions of the particular random sequence.

Our goal is to compute \( \hat{w}(n) \) at each iteration. Done naively, this requires inverting the matrix \( \Phi(n) \) at each iteration. However, there is an efficient recursion to for computing \( \Phi^{-1}(n), z(n) \) and \( \hat{w}(n) \) from \( \Phi^{-1}(n-1), z(n-1), \hat{w}(n-1) \) and \( u(n) \). This method requires \( O(k^2) \) computation time for each update step.

We define \( P(n) = \Phi^{-1}(n) \). Each iteration of RLS consists of the following steps:

1. \( \pi(n) = P(n-1)u(n) \)

2. Compute the gain vector:

\[ k(n) = \frac{\pi(n)}{\gamma + u(n)^T \pi(n)} \]

3. Compute the updated inverse matrix

\[ P(n) = \gamma^{-1}P(n-1) - \gamma^{-1}k(n)u(n)^TP(n-1) \]
4. Compute the prediction error
\[ \epsilon(n) = d(n) - \hat{w}(n-1)^T u(n) \]

5. Update the weight vector
\[ \hat{w}(n) = \hat{w}(n-1) + k(n)\epsilon(n) \]

If \( \gamma = 1 \) we get a time-based estimate of the Wiener optimal filter. When \( \gamma < 1 \) the more recent vectors are weighted more heavily and we get a solution.

### 9.3 The probabilistic interpretation of least squares

It is sometimes useful to give a probabilistic interpretation to the task of minimizing square error. Consider the finding \( w \) to minimize the cumulative squared error \( \sum_n \epsilon(n)^2 \) where
\[ \epsilon(n) = d(n) - w^T u(n) \]

We can think of \( d(n) \) as a sum of a predictable signal \( w^T u(n) \) and white noise \( \epsilon(n) \). As the noise is white the probability density of \( \epsilon(n) \) is a unit variance, zero mean distribution, independent for each \( n \):
\[ p(\epsilon(n)) = \frac{1}{\sqrt{2\pi}} e^{-\epsilon(n)^2/2} \]

Which means that the probability density of the sequence of errors \( \epsilon(1), \ldots, \epsilon(n) \) is
\[ p(\epsilon(1), \ldots, \epsilon(n)) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{-\epsilon(i)^2/2} = \frac{1}{(2\pi)^{n/2}} \exp \left( -\frac{1}{2} \sum_{i=1}^{n} \epsilon(i)^2 \right) \]

This probability can be thought of the likelihood of the weight vector \( w \) given the signals \( u(n) \) and \( d(n) \).
\[ L(w) = p(\epsilon(1), \ldots, \epsilon(n)) \]

To estimate \( w \) we often want to use the value of \( w \) which maximizes the likelihood, or, equivalently, maximizes the log likelihood:
\[ \log L(w) = \sum_{i=1}^{n} \log p(\epsilon(i)) = \frac{1}{(2\pi)^{n/2}} - \frac{1}{2} \sum_{i=1}^{n} \epsilon(i)^2 \]

The constant term and the factor of \( \frac{1}{2} \) can be ignored and we find that maximizing the likelihood is equivalent to minimizing the sum of the squares of the errors.

So far, this has been little more than symbolic manipulation. Things get more interesting when we want to model non-stationary stochastic systems.

### 9.4 Hidden Markov Models

Suppose that the best weight vector \( \hat{w}(n) \) changes as a function of \( n \). How can we estimate it?

If \( \hat{w}(n) \) can change arbitrarily at each iteration then it is impossible to estimate. The weight vector can always be set in many ways so that the error \( \epsilon(n) \) is zero at each iteration. We need therefore to restrict the transitions in some way.

One popular way to characterize nonstationary processes is by using a variant of the hidden markov model (HMM). In a hidden markov model there are two random sequences, the hidden state sequence \( s(1), s(2), \ldots \)
and the observed sequence \(x(1), x(2), \ldots\). The state sequence cannot be observed directly but only through its effect on the observed sequence. The hidden states vary over a finite set of size \(H\).

The state sequence is Markovian, that is to say, the state at time \(t\) depends on the previous states only through the state at time \(t - 1\)

\[
P(S(t) = s(t)|S(t-1) = s(t-1), S(t-2) = s(t-2), \ldots, S(1) = s(1)) = P(S(t) = s(t)|S(t-1) = s(t-1))
\]

Where \(s(t)\) denotes an actual value attained by the random variable \(S(t)\).

As the set of states is finite, we can fully describe the Markov process by an \(H \times H\) transition matrix:

\[
M(i, j) = P(S(t) = i|S(t-1) = j)
\]

Let’s see how we can use HMM’s to model a non-stationary stochastic signal. Suppose \(w_1, w_2\) are two weight vectors of length \(k\) that define two auto-regressive models.

\[
u(n + 1) = w_1^T u(n) + \nu(n)
\]

Where \(\nu(n)\) is white noise. The hidden state identifies the weight vector used in the auto-regressive model.

Assume that the transition matrix is

\[
M = \begin{bmatrix}
1 - \epsilon & \epsilon \\
\epsilon & 1 - \epsilon
\end{bmatrix}
\]

for some small \(\epsilon > 0\). This matrix corresponds to the assumption that with probability \(1 - \epsilon\) the system stays in the same state and with probability \(\epsilon\) it switches to the other state.

Using these assumptions we can compute the probability distribution of the hidden state \(S(i)\). To simplify notation we use \(X(i : j)\) for \(j < i\) to denote \(X(i), X(i-1), \ldots, X(j)\). We denote the joint probability of the hidden state at time \(i\) and the past observations by

\[
A_i(s) = P(S(i) = s, X(i : 1) = x(i : 1))
\]

We will compute the conditional probability of the hidden state given the observation using the formula

\[
B_i(v) = P(S(i) = s|X(i : 1) = x(i : 1)) = \frac{A_i(s)}{\sum_v A_i(v)}
\]

Therefor it is sufficient to find a formula for \(A_i(s)\) that is correct up to a multiplicative constant.

We can write the following recursion involving \(A_i(s)\):

\[
A_i(s) = \sum_v P(S(i) = s, S(i-1) = v, X(i : 1) = x(i : 1))
\]

\[
\propto P(X(i) = x(i)|S(i) = s, X(i-1 : i-1) = x(i-1 : i-1)) \\
\sum_v P(S(i) = s|S(i-1) = v)P(S(i-1) = v|X(i-1 : 1)) \\
\propto P(X(i) = x(i)|S(i) = s, X(i-1 : i-1) = x(i-1 : i-1)) \sum_v P(S(i) = s|S(i-1) = v)B_{i-1}(v)
\]

We now have a fully specified formula for \(A_i(s)\):

\[
P(X(i) = x(i)|S(i) = s, X(i-1 : i-1) = x(i-1 : i-1)) \propto \exp \left( -\frac{1}{2} (x(i) - w_x^T x(i-1 : i-1))^2 \right)
\]

and

\[
P(S(i) = s|S(i-1) = v) = \begin{cases}
1 - \epsilon & \text{if } s = v \\
\epsilon & \text{otherwise}
\end{cases}
\]