Recursive Least Squares

Ryan J. Kinnear

February 7, 2017

1 Introduction

A derivation of the classic recursive least squares algorithm. I have drawn in part from the book Statistical Digital Signal Processing and Modeling by Monson H. Hayes. The derivation is done for complex valued data, but it is trivial to modify for real valued data by simply replacing H by T, ignoring complex conjugates, and adding a $\frac{1}{2}$ factor to the error function $\mathcal{E}(t)$. Also note that the `RLS.py` implements the algorithm assuming real data.

2 Classic RLS

Suppose we are interested in making online estimates of some signal $d(n)$. We are given access to measurements of a related signal $x(n)$. Suppose further that after we make our estimate of $d(n)$ using our knowledge at time t (denoted $\hat{d}_t(n)$), nature either shows us the true value of $d(n)$, or tells us what error we made, so that we can determine how close our estimate was. We will make linear estimates of $d(n)$ given $x(n)$ using an adaptive filter of order $p+1$ as

$$\begin{aligned}\hat{d}_t(n) &= \sum_{\tau=0}^{p} w_t(\tau)x(n-\tau) \\ &= w_t^{\mathsf{T}} x_n\end{aligned} \tag{1}$$

Where we have $w_t = [w_t(0)\dots w_t(p)]^{\mathsf{T}} \in \mathbb{C}^{p+1}$ the weight vector of the filter at time t, and $x_n = [x(n)\dots x(n-p)]^{\mathsf{T}}$ the past $p+1$ measured samples from time n. Note that our convention is that every signal is 0 prior to time 1. Further, we must be careful with indices. Since we have an adaptive filter, we can view $\hat{d}_t(n)$ as a sequence both in t and n $(n \le t)$, where the t index specifies which weight vector was used for the estimate, and n specifies which $d(n)$ we were estimating.

Let $e_t(n) = d(n) - \hat{d}_t(n) = d(n) - w_t^{\mathsf{T}} x_n$ be the error made predicting $d(n)$ using the filter coefficients at time t. Note that it is here that we need nature to provide for us the true value of $d(n)$ (directly providing $e_t(t)$ is sufficient, as we will see). We will design this filter by minimizing an exponentially weighted squared error loss:

$$\begin{aligned}\mathcal{E}(t) &= \sum_{n=1}^{t} \lambda^{n-t} |e_t(n)|^2 \\ &= (d_t - \hat{d}_t)^{\mathsf{H}} \Lambda_t (d_t - \hat{d}_t) \\ &= (d_t - X_t^{\mathsf{T}} w_t)^{\mathsf{H}} \Lambda_t (d_t - X_t^{\mathsf{T}} w_t).\end{aligned} \tag{2}$$

Where $d_t = [d(t) \dots d(1)]^{\mathsf{T}} \in \mathbb{C}^t$, $\hat{d}_t = [\hat{d}_t(t) \dots \hat{d}_t(1)]^{\mathsf{T}} = X_t^{\mathsf{T}} w_t$, where $X_t = [x_t \dots x_1] \in \mathbb{C}^{(p+1)\times t}$, $\Lambda_t = \mathbf{Diag}(1, \lambda, \lambda^2, \dots, \lambda^t) \in \mathbb{R}^{t \times t}$. I stress again to think carefully about the indices, $\mathcal{E}(t)$ is the exponentially decaying error calculated over the entire past of given data had we used the filter at time t.

And, hence (suppressing the time indices):

$$\mathcal{E}(t) = d^{\mathsf{H}} \Lambda d - d^{\mathsf{H}} \Lambda X^{\mathsf{T}} w - w^{\mathsf{H}} X^* \Lambda d + w^{\mathsf{H}} X^* \Lambda X^{\mathsf{T}} w.$$

Taking the gradient of $\mathcal{E}(t)$ with respect to w^* we obtain

$$\nabla_{w^*} \mathcal{E}(t) = X^* \Lambda X^{\mathsf{T}} w - X^{\mathsf{H}} \Lambda d.$$

Setting this gradient to 0 nets us the update formula for w:

$$\begin{aligned} w_{t+1} &= (X_t^* \Lambda X_t^{\mathsf{T}})^{-1} (X_t^{\mathsf{H}} \Lambda d) \\ &= R(t)^{-1} P(t). \end{aligned} \tag{3}$$

Notice that $R(t)$ is the correlation matrix of the exponentially weighted x vector and must be positive semi-definite. $P(t)$ is the cross correlation between the exponentially weighted x vector and the desired signal d. The similarity to the LMMSE estimator is clear.

In order to derive an online (or recursive) update formula for w_t we need to obtain a recursive formula for $R(t)^{-1}$ in terms of $R(t-1)^{-1}$ and similarly for $P(t)$.

The following are clear:

$$R(t) = \lambda R(t-1) + x_t^* x_t^\mathsf{T}, \tag{4}$$
$$P(t) = \lambda P(t-1) + d(t) x_t^*. \tag{5}$$

We then apply the Sherman-Morrison rank 1 update to $R(t)^{-1}$ to obtain:

$$\begin{aligned} R(t)^{-1} &= [\lambda R(t-1) + x_t^* x_t^\mathsf{T}]^{-1} \\ &= \frac{1}{\lambda} R(t-1)^{-1} - \frac{\lambda^{-2} R(t-1)^{-1} x_t^* x_t^\mathsf{T} R(t-1)^{-1}}{1 + \lambda^{-1} x_t^\mathsf{T} R(t-1)^{-1} x_t^*} \\ &= \frac{1}{\lambda} [I - g(t) x_t^\mathsf{T}] R(t-1)^{-1}. \end{aligned}$$

Where

$$g(t) = \frac{\lambda^{-1} R(t-1)^{-1} x_t^* x_t}{1 + \lambda^{-1} x_t^\mathsf{T} R(t-1)^{-1} x_t^*} \tag{6}$$

is referred to as the "**gain vector**".

By multiplying through by the bottom of equation 6 and applying the first formula for $R(t)^{-1}$ above, we get the relation:

$$R(t) g(t) = x_t^*. \tag{7}$$

Either of 6 or 7 can be used to calculate $g(t)$, depending on whether or not you want to store both $R(t)$ and $R(t)^{-1}$. Solving the linear system 7 is more numerically stable than the direct calculation of 6. Further, since $R(t)$ is positive semi-definite, we could compute the rank 1 updates of it's Cholesky decomposition, and use this to efficiently solve 7.

In any case, we are now in a position to efficiently calculate w_{t+1}:

$$\begin{aligned} w_{t+1} &= R(t)^{-1} P(t) \\ &= R(t)^{-1} [\lambda P(t-1) + d(t) x_t^*] \\ &\overset{(a)}{=} [R(t-1)^{-1} - g(t) x_t^\mathsf{T} R(t-1)^{-1}] P(t-1) + d(t) g(t) \\ &= w_t - g(t) x_t^\mathsf{T} w_t + d(t) g(t) \\ &= w_t + e_t(t) g(t), \end{aligned}$$

where (a) uses equation 7 and recall that $e_t(t) = d(t) - w_t^\mathsf{T} x_t$ is the error on the previous estimate of $d(t)$. It remarkable that the only feedback necessary to calculate the least squares filter over the entire history of data is $e_t(t)$. Furthermore, mere estimates of $e_t(t)$ are usually sufficient.

It is typical to initialize $R(t)$ as δI for some small $\delta > 0$ in order to ensure the matrix is invertible at the beginning of the recursion. This initialization injects some bias into the algorithm, but it is quickly washed out by the exponential decay. If prior knowledge is available it can be used to initialize $R(t)$.

The summary of the algorithm is here:

```
**Data**: $\delta > 0$, $\lambda \in (0, 1]$, $p \in \mathbb{N}$
**Result**: $\hat{d}_t(t), t = 1, 2, \ldots$
**Initialize**: $R(0) = \delta I_{p+1}$, $w_1 = 0$
**for** $t = 1, 2, \ldots$ **do**
    **measure** : $x(t)$
    $\hat{d}_t(t) = w_t^{\mathsf{T}} x_t$
    **output** : $\hat{d}_t(t)$
    **measure** : $e_t(t)$ // An estimate will suffice
    $w_{t+1} = w_t + e_t(t) g(t)$
    $R(t) = \lambda R(t-1) + x_t^* x_t^{\mathsf{T}}$
    $\underset{g(t)}{\textbf{solve}}$ : $R(t) g(t) = x_t^*$
    $R(t)^{-1} = \lambda^{-1}[I - g(t) x_t^{\mathsf{T}}] R(t-1)^{-1}$
    $R(t)^{-1} = \frac{1}{2}[R(t)^{-1} + R(t)^{-\mathsf{T}}]$ // not strictly necessary
**end**
```

A Python implementation, as well as some examples should accompany this document. Otherwise, see `https://github.com/RJTK`