Lecture 27: Recursive Least Squares

Mark Hasegawa-Johnson

ECE 401: Signal Processing

1. Review: LPC
2. Recursive Least Squares
3. Matrix recursion and vector prediction
4. How to make RLS efficient and stable
5. Summary

Outline

1. Review: LPC
2. Recursive Least Squares
3. Matrix recursion and vector prediction
4. How to make RLS efficient and stable
5. Summary

Linear predictive coding (LPC): Synthesis model

A linearly predictable signal is one whose samples, $s[n]$, can be estimated as the linear prediction of p previous samples plus some small error term $e[n]$:

$$s[n] = e[n] + \sum_{k=1}^{p} a_k s[n-k]$$

LPC: The Solution!

We will choose a_k to minimize the mean squared error (MSE):

$$\mathcal{E} = \sum_m e^2[m]$$

$$= \sum_m \left(s[m] - \sum_{k=1}^{p} a_k s[m-k] \right)^2$$

If you differentiate w.r.t. a_i, you get p linear equations in p unknowns:

$$r_s[i] = \sum_{k=1}^{p} a_k r_s[k-i]$$

. . . where. . .

$$r_s[i] \equiv \sum_m s[m-i]s[m]$$

The LPC solution as a matrix equation

$$\gamma = \mathbf{R}\mathbf{a}$$

$$\gamma = \begin{bmatrix} r_s[1] \\ \vdots \\ r_s[p] \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} r_s[0] & \cdots & r_s[p-1] \\ \vdots & \ddots & \vdots \\ r_s[p-1] & \cdots & r_s[0] \end{bmatrix}, \quad \mathbf{a} = \begin{bmatrix} a_1 \\ \vdots \\ a_p \end{bmatrix}$$

Outline

1 Review: LPC

2 Recursive Least Squares

3 Matrix recursion and vector prediction

4 How to make RLS efficient and stable

5 Summary

What if the autocorrelation changes rapidly?

- LPC assumes that the autocorrelation and LPC coefficients are stationary for at least one frame.

$$s[n] = e[n] + \sum_{k=1}^{p} a_k s[n-k]$$

- What if the autocorrelation is changing quickly, so that we need to update the coefficients every sample?

$$s[n] = e[n] + \sum_{k=1}^{p} a_k[n] s[n-k]$$

Recursive least squares (RLS)

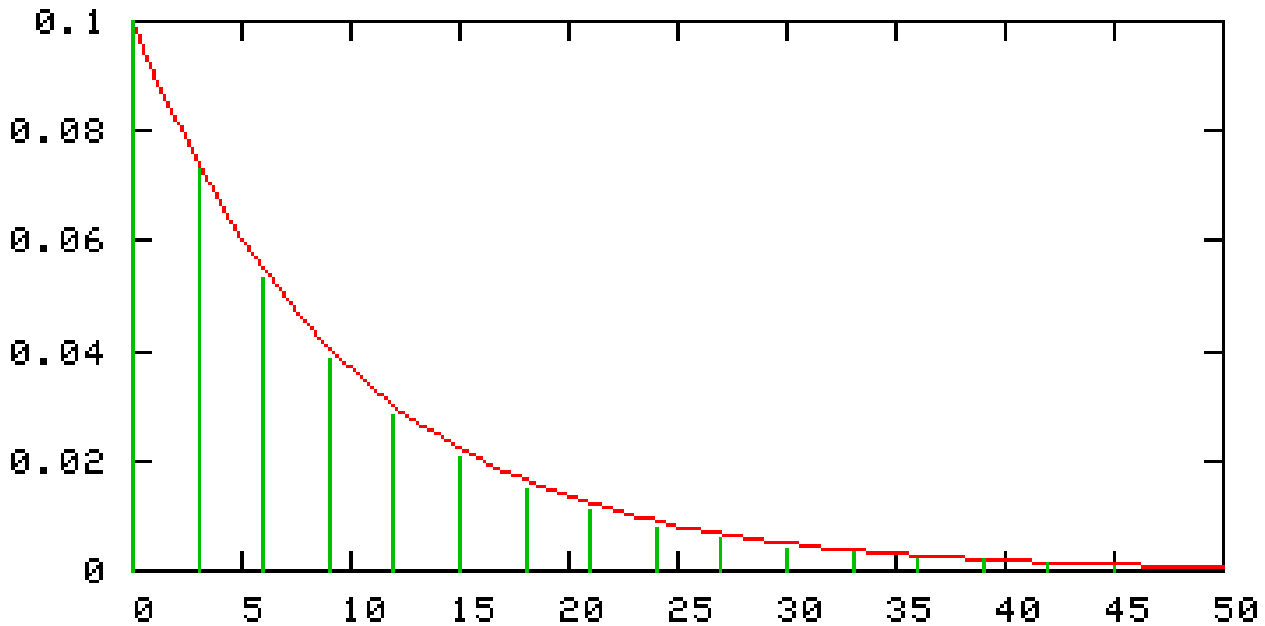
- RLS, independently invented by Gauss (1850s) and Lanczos (1952), recomputes the autocorrelation $r_s[i, k; n]$ and predictor coefficients $a_k[n]$ for every sample!
- The word "recursive" describes our estimate of the autocorrelation:

$$r_s[i, k; n] = s[n - i]s[n - k] + \lambda r_s[i, k; n]$$

...where $\lambda < 1$ is the forgetting factor.

The forgetting factor, λ

The forgetting factor, λ, gives a weight of $\lambda^0 = 1.0$ to the current sample, and progressively decreasing weights to older samples. For example, here's the curve for $\lambda = 0.9$:



https://commons.wikimedia.org/wiki/File:Geometrique_exponentielle.png

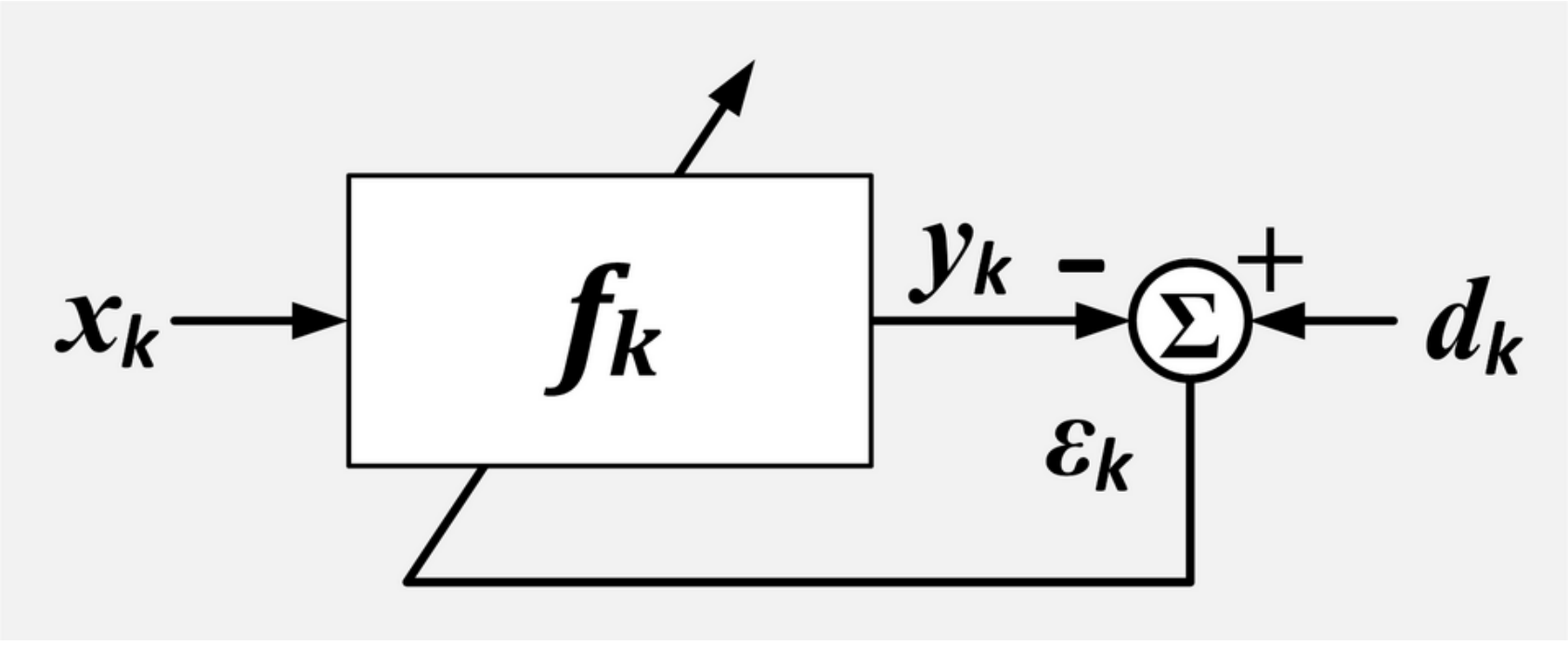
The RLS error term

- The error we want to minimize is $e^2[n] = (s[n] - \sum_k a_k[n]s[n-k])^2$.
- We could set that to zero with $a_1[n] = s[n]/s[n-1]$! It's too easy: it has only learned the current sample, it hasn't learned anything about the usystem that produced the samples
- A more useful definition: pretend that $a_k[n]$ has been applied for all recent samples, and minimize a weighted error term:

$$\begin{aligned}\mathcal{E} &= \sum_{m=0}^{\infty} \lambda^m e^2[n-m] \\ &= \sum_{m=0}^{\infty} \lambda^m \left(s[n-m] - \sum_k a_k[n]s[n-m-k] \right)^2\end{aligned}$$

Adaptive filtering

RLS is a type of adaptive filter, in which the desired signal, $d_k = s[n]$, is predicted from the past samples $\mathbf{x}_k = [s[n-1], \ldots, s[n-p]]^T$:



https://commons.wikimedia.org/wiki/File:Adaptive_Filter_Compact.png

Differentiating the RLS error term

Differentiating $\mathcal{E}$ w.r.t. a_i gives us

$$\frac{\partial \mathcal{E}}{\partial a_i} = r_s[0, i; n] - \sum_k a_k[n] r_s[i, k; n]$$

. . . and setting it to zero gives us. . .

$$r_s[0, i; n] = \sum_k a_k[n] r_s[i, k; n]$$

. . . where . . .

$$\begin{aligned} r_s[i, k; n] &\equiv s[n-i]s[n-k] + \lambda r_s[i, k; n-1] \\ &= \sum_{m=0}^{\infty} \lambda^m s[n-m-i]s[n-m-k] \end{aligned}$$

RLS: The solution!

The solution that minimizes the RLS error is $r_s[0, i; n] = \sum_k a_k[n] r_s[i, k; n]$, i.e.,

$$\boldsymbol{\gamma}[n] = \mathbf{R}[n]\mathbf{a}[n]$$

. . . where . . .

$$\boldsymbol{\gamma}[n] = \begin{bmatrix} r_s[0, 1; n] \\ \vdots \\ r_s[0, p; n] \end{bmatrix}, \quad \mathbf{R}[n] = \begin{bmatrix} r_s[1, 1; n] & \cdots & r_s[1, p-1; n] \\ \vdots & \ddots & \vdots \\ r_s[p-1, 1; n] & \cdots & r_s[p, p; n] \end{bmatrix},$$

Outline

1 Review: LPC

2 Recursive Least Squares

3 Matrix recursion and vector prediction

4 How to make RLS efficient and stable

5 Summary

Matrix recursion

Each autocorrelation is recursive:

$$r_s[i,k;n] = s[n-i]s[n-k] + \lambda r_s[i,k;n-1]$$

. . . and therefore $\mathbf{R}[n]$ and $\boldsymbol{\gamma}[n]$ are also recursive:

$$\mathbf{R}[n] = \mathbf{s}[n]\mathbf{s}^T[n] + \lambda \mathbf{R}[n-1]$$
$$\boldsymbol{\gamma}[n] = \mathbf{s}[n]s[n] + \lambda \boldsymbol{\gamma}[n-1]$$

. . . where $\mathbf{s}[n]$ is the delayed-signal vector

$$\mathbf{s}[n] = \begin{bmatrix} s[n-1] \\ \vdots \\ s[n-p] \end{bmatrix}$$

Vector prediction

Notice that, with these definitions, the predicted value of $s[n]$ is

$$\begin{aligned}\tilde{s}[n] &= \sum_k a_k[n]s[n-k] \\ &= \mathbf{s}^T[n]\mathbf{a}[n]\end{aligned}$$

The prediction error is

$$e[n] = s[n] - \mathbf{s}^T[n]\mathbf{a}[n]$$

RLS problem summary

To minimize the forgetting-weighted memory

$$\mathcal{E} = \sum_m \lambda^m \left(s[n-m] - \mathbf{s}^T[n-m]\mathbf{a}[n] \right)^2$$

. . . we find coefficients $\mathbf{a}[n]$ according to

$$\mathbf{a}[n] = \mathbf{R}^{-1}[n]\boldsymbol{\gamma}[n]$$

. . . where $\mathbf{R}[n]$ and $\boldsymbol{\gamma}[n]$ are defined recursively from $s[n]$ and $\mathbf{s}[n] = [s[n-1], \ldots, s[n-p]]^T$

$$\mathbf{R}[n] = \mathbf{s}[n]\mathbf{s}^T[n] + \lambda\mathbf{R}[n-1]$$
$$\boldsymbol{\gamma}[n] = \mathbf{s}[n]s[n] + \lambda\boldsymbol{\gamma}[n-1]$$

Try the quiz!

Go to the course web page and try the quiz!

Outline

1 Review: LPC

2 Recursive Least Squares

3 Matrix recursion and vector prediction

4 How to make RLS efficient and stable

5 Summary

Is RLS efficient and stable?

RLS has two big problems:

- **Inefficiency:** We need to invert $\mathbf{R}[n]$ at every sample time
- **Instability:** Sometimes $\mathbf{R}[n]$ might be nearly singular, in which case $\mathbf{R}^{-1}[n]$ and $\mathbf{a}[n]$ might become nearly infinite

Both problems are solved in the same way, using the **stable RLS algorithm**.

The stable RLS algorithm

At every sample n:

1. Compute the *a priori* error $\epsilon[n]$:

$$\epsilon[n] = s[n] - \mathbf{s}^T[n]\mathbf{a}[n-1]$$

2. Compute the update gain vector $\mathbf{g}[n]$
3. Update the coefficients

$$\mathbf{a}[n] = \mathbf{a}[n-1] + \mathbf{g}[n]\epsilon[n]$$

The gain vector and the precision matrix

- The gain vector is defined as $\mathbf{g}[n] = \mathbf{R}^{-1}[n]\mathbf{s}[n]$, which is not very useful because it still depends on $\mathbf{R}^{-1}[n]$. With the help of some linear algebra, though, we can rewrite it as

$$\mathbf{g}[n] = \frac{\mathbf{P}[n-1]\mathbf{s}[n]}{\lambda + \mathbf{s}^T[n]\mathbf{P}[n-1]\mathbf{s}[n]}$$

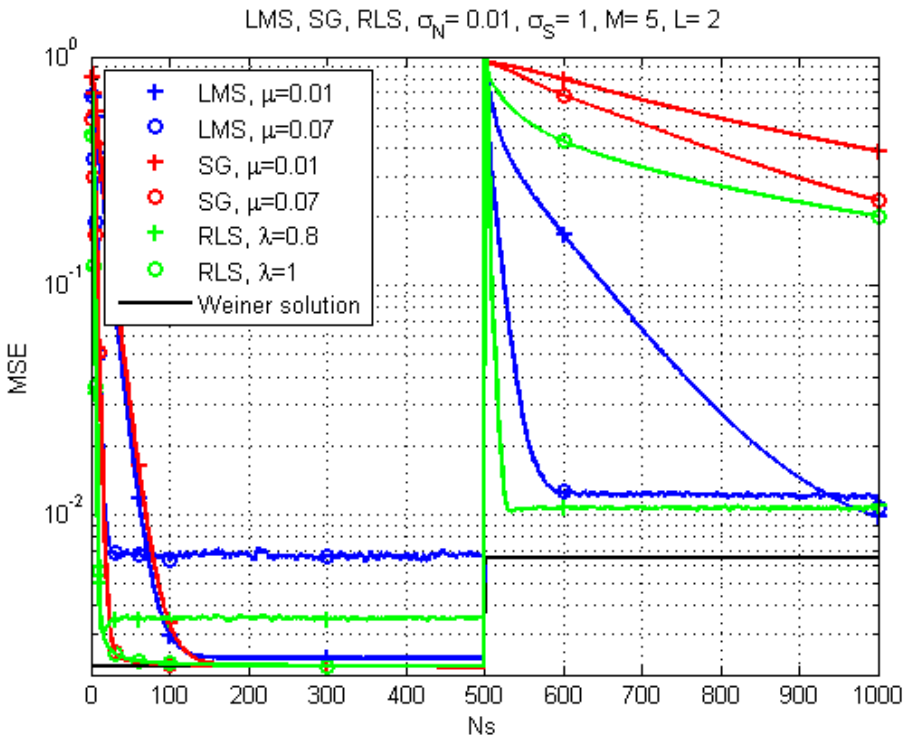
- To make the recursion work, we then need to update the precision matrix. The precision matrix is defined as $\mathbf{P}[n] = \mathbf{R}^{-1}[n]$, which, again, is not very useful, but using some linear algebra we can rewrite it as

$$\mathbf{P}[n] = \frac{1}{\lambda}\left(\mathbf{P}[n-1] - \mathbf{g}[n]\mathbf{s}^T[n]\mathbf{P}[n-1]\right)$$

- Notice that both of these formulas are computationally efficient (the worst computation, the matrix-vector multiplication, is $\mathcal{O}\left\{p^2\right\}$) and, as long as $\lambda > 0$, both are stable (finite inputs always give finite outputs).

Simulation example

- Stationary for 500 samples, then sudden change
- Wiener filter (black) knows the true $r_s[n]$
- LMS (least-mean-squares), SG (stochastic gradient), and RLS estimate $r_s[n]$ from the sample values
- $\lambda = 1.0$ converges to $\mathcal{E} = 0$ for the first system, but $\lambda = 0.8$ adapts better when the system changes.



https://commons.wikimedia.org/wiki/File:SG_RLS_LMS_chan_var.png

Outline

1 Review: LPC

2 Recursive Least Squares

3 Matrix recursion and vector prediction

4 How to make RLS efficient and stable

5 Summary

RLS problem summary

To minimize the forgetting-weighted memory

$$\mathcal{E} = \sum_m \lambda^m \left(s[n-m] - \mathbf{s}^T[n-m]\mathbf{a}[n] \right)^2$$

. . . we find coefficients $\mathbf{a}[n]$ according to

$$\mathbf{a}[n] = \mathbf{R}^{-1}[n]\boldsymbol{\gamma}[n]$$

. . . where $\mathbf{R}[n]$ and $\boldsymbol{\gamma}[n]$ are defined recursively from $s[n]$ and $\mathbf{s}[n] = [s[n-1], \ldots, s[n-p]]^T$

$$\mathbf{R}[n] = \mathbf{s}[n]\mathbf{s}^T[n] + \lambda\mathbf{R}[n-1]$$
$$\boldsymbol{\gamma}[n] = \mathbf{s}[n]s[n] + \lambda\boldsymbol{\gamma}[n-1]$$

The stable RLS algorithm

At every sample n:

1. Compute the *a priori* error $\epsilon[n]$:

$$\epsilon[n] = s[n] - \mathbf{s}^T[n]\mathbf{a}[n-1]$$

2. Compute the gain and precision as

$$\mathbf{g}[n] = \frac{\mathbf{P}[n-1]\mathbf{s}[n]}{\lambda + \mathbf{s}^T[n]\mathbf{P}[n-1]\mathbf{s}[n]}$$

$$\mathbf{P}[n] = \frac{1}{\lambda}\left(\mathbf{P}[n-1] - \mathbf{g}[n]\mathbf{s}^T[n]\mathbf{P}[n-1]\right)$$

3. Update the coefficients

$$\mathbf{a}[n] = \mathbf{a}[n-1] + \mathbf{g}[n]\epsilon[n]$$