自适应滤波方法（二）

主讲人 宋 辉

清华大学电子工程系 博士
滴滴AI Labs 语音技术部

- 4.1 Recursive Least Square(RLS)算法
- 4.2 Affine Projection(AP)算法
- 4.3 实战

课程回顾

从一个N阶线性系统出发... ...

目标：设计一个N阶滤波器，它的参数为 $\mathbf{w}(n)$，则滤波器输出为：

$$y(n)=\sum_{i=0}^{N-1} w_i(n)x(n-i)=\mathbf{w}^T(n)\mathbf{x}(n)=\mathbf{x}^T(n)\mathbf{w}(n) \tag{3.1}$$

上式中：

$$\mathbf{x}(n)=[x(n),\quad x(n-1),\quad \ldots\quad x(n-N+1)]^T \tag{3.2}$$

$$\mathbf{w}(n)=[w_0(n),\quad w_1(n),\quad \ldots\quad w_{N-1}(n)]^T \tag{3.3}$$

期望输出为 $d(n)$，定义误差信号：

$$e(n)=d(n)-y(n)=d(n)-\mathbf{w}^T(n)\mathbf{x}(n) \tag{3.4}$$

课程回顾

根据最小均方误差(MMSE)准则，最小化目标函数： $J(\mathbf{w})$

$$J(\mathbf{w})=E\left\{|e(n)|^2\right\}=E\left\{\left|d(n)-\mathbf{w}^T(n)\mathbf{x}(n)\right|^2\right\} \tag{3.5}$$

为了最小化均方误差函数，需计算 $J(\mathbf{w})$对 $\mathbf{w}$ 的导数，令导数为零：

$$E\left\{\mathbf{x}(n)\mathbf{x}^T(n)\right\}\mathbf{w}(n)-E\left\{\mathbf{x}(n)d(n)\right\}=\mathbf{R}\mathbf{w}-\mathbf{r}=\mathbf{0} \tag{3.6}$$

可得到：

$$\boxed{\mathbf{w}_{opt}=\mathbf{R}^{-1}\mathbf{r}} \tag{3.7}$$

(3.6)定义的滤波器称为Wiener滤波器。Wiener滤波器是均方误差最小意义上的统计最优滤波器。

4.1 Recursive Least Square(RLS)算法

MMSE是一个均匀加权的最优化问题，也就是说，每一时刻的误差信号对目标函数的贡献权重是相同的。

采用指数加权的方式，重新定义目标函数：

$$J_n(\mathbf{w}) = \sum_{i=0}^{n} \lambda^{n-i} \left|e(i)\right|^2 = \sum_{i=0}^{n} \lambda^{n-i} \left|d(i) - \mathbf{w}^T(n)\mathbf{x}(i)\right|^2 \tag{4.1}$$

式中，$0 < \lambda \leqslant 1$ 称为遗忘因子。

遗忘因子的作用是对离 n 时刻越近的误差加比较大的权重，而对离 n 时刻越远的误差加比较小的权重，也就是说对各个时刻的误差具有一定的遗忘作用。

如果 $\lambda = 1$ ，表示无任何遗忘功能，或具有无穷记忆功能，此时RLS退化为LS方法。

如果 $\lambda \to 0$，表示只有当前时刻的误差起作用，而过去时刻的误差完全被遗忘。

4.1 RLS算法

$$J_n(\mathbf{w}) = \sum_{i=0}^{n} \lambda^{n-i} |e(i)|^2 = \sum_{i=0}^{n} \lambda^{n-i} \left|\boxed{d(i) - \mathbf{w}^T(n)\mathbf{x}(i)}\right|^2 \qquad (4.1)$$

特别注意，对于误差的定义： $e(i) = d(i) - \mathbf{w}^T(n)\mathbf{x}(i)$ 用的是 $\mathbf{w}^T(n)$，而不是 $\mathbf{w}^T(i)$。

原因：在自适应更新过程中，滤波器总是变得越来越好，这意味着对于任何的 $i < n$，$\left|d(i) - \mathbf{w}^T(n)\mathbf{x}(i)\right|$ 总是比 $\left|d(i) - \mathbf{w}^T(i)\mathbf{x}(i)\right|$ 小。因此，由前者构成的目标函数(4.1)总是比由后者构成的目标函数更小，是一种更为合理的选择。

由于计算 i 时刻的误差 $e(i)$，用到了 n 时刻的滤波器系数，因此(4.1)式也被称为后验估计误差。

4.1 RLS算法

最小化目标函数 $J_n(\mathbf{w})$

令 $J_n(\mathbf{w})$ 的梯度等于0，得出：

$$\mathbf{w}(n) = \mathbf{R}^{-1}(n)\mathbf{r}(n) \tag{4.2}$$

其中：

$$\mathbf{R}(n) = \sum_{i=0}^{n} \lambda^{n-i}\mathbf{x}(i)\mathbf{x}^T(i) \tag{4.3}$$

$$\mathbf{r}(n) = \sum_{i=0}^{n} \lambda^{n-i}\mathbf{x}(i)d(i) \tag{4.4}$$

可以看到，最小化指数加权的目标函数，它的解仍然是维纳解。

求解滤波器参数 $\boldsymbol{w}(n)$

$$\mathbf{R}(n)=\sum_{i=0}^{n}\lambda^{n-i}\mathbf{x}(i)\mathbf{x}^{T}(i) \tag{4.3}$$

$$\mathbf{r}(n)=\sum_{i=0}^{n}\lambda^{n-i}\mathbf{x}(i)d(i) \tag{4.4}$$

根据 $\mathbf{R}(n)$ 和 $\mathbf{r}(n)$ 的等式，得其时间递推公式：

$$\mathbf{R}(n)=\lambda\mathbf{R}(n-1)+\mathbf{x}(n)\mathbf{x}^{T}(n) \tag{4.5}$$

$$\mathbf{r}(n)=\lambda\mathbf{r}(n-1)+\mathbf{x}(n)d(n) \tag{4.6}$$

$$\begin{matrix}\mathbf{R}(n-1)\\ \mathbf{r}(n-1)\end{matrix} \xrightarrow[\text{递推公式(4.6)}]{\text{递推公式(4.5)}} \boxed{\begin{matrix}\mathbf{R}(n)\\ \mathbf{r}(n)\end{matrix}} \xrightarrow{\text{矩阵求逆}} \boxed{\begin{matrix}\mathbf{R}^{-1}(n)\\ \mathbf{r}(n)\end{matrix}} \xrightarrow{\mathbf{w}(n)=\mathbf{R}^{-1}(n)\mathbf{r}(n)} \boldsymbol{w}(n)$$

弊端：需要计算每个 $\mathbf{R}(n)$ 的逆矩阵，很不划算。更优的方法?

4.1 RLS算法

探索 $\mathbf{P}(n)=\mathbf{R}^{-1}(n)$ 的时间递推公式

$$\mathbf{P}(n)=\mathbf{R}^{-1}(n)=\left(\lambda\mathbf{R}(n-1)+\mathbf{x}(n)\mathbf{x}^{T}(n)\right)^{-1}$$

根据矩阵求逆引理：

$$\boxed{(A+BCD)^{-1}=A^{-1}-A^{-1}B(C^{-1}+DA^{-1}B)^{-1}DA^{-1}}$$

令： $A=\mathbf{R}(n-1)$ $B=\mathbf{x}(n)$ $C=\frac{1}{\lambda}$ $D=\mathbf{x}^{T}(n)$

可以得到：

$$\begin{aligned}\mathbf{P}(n)&=\frac{1}{\lambda}\left[\mathbf{P}(n-1)-\frac{\mathbf{P}(n-1)\mathbf{x}(n)\mathbf{x}^{T}(n)\mathbf{P}(n-1)}{\lambda+\mathbf{x}^{T}(n)\mathbf{P}(n-1)\mathbf{x}(n)}\right]\\&=\frac{1}{\lambda}\left[\mathbf{P}(n-1)-\mathbf{k}(n)\mathbf{x}^{T}(n)\mathbf{P}(n-1)\right]\end{aligned} \tag{4.7}$$

其中，增益向量：

$$\mathbf{k}(n)=\frac{\mathbf{P}(n-1)\mathbf{x}(n)}{\lambda+\mathbf{x}^{T}(n)\mathbf{P}(n-1)\mathbf{x}(n)} \tag{4.8}$$

4.1 RLS算法

探索 $w(n)$ 的时间递推公式

$$\begin{aligned}\mathbf{P}(n)\mathbf{x}(n) &= \frac{1}{\lambda}\left[\mathbf{P}(n-1)\mathbf{x}(n) - \mathbf{k}(n)\mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}(n)\right] \\ &= \frac{1}{\lambda}\left\{\left[\lambda + \mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}(n)\right]\mathbf{k}(n) - \mathbf{k}(n)\mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}(n)\right\} \\ &= \mathbf{k}(n)\end{aligned}$$

$$\begin{aligned}\mathbf{w}(n) &= \mathbf{R}^{-1}(n)\mathbf{r}(n) = \mathbf{P}(n)\mathbf{r}(n) \\ &= \frac{1}{\lambda}\left[\mathbf{P}(n-1) - \mathbf{k}(n)\mathbf{x}^T(n)\mathbf{P}(n-1)\right]\left[\lambda\mathbf{r}(n-1) + \mathbf{x}(n)d(n)\right] \\ &= \mathbf{P}(n-1)\mathbf{r}(n-1) - \mathbf{k}(n)\mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{r}(n-1) \\ &\quad + \frac{1}{\lambda}d(n)\left[\mathbf{P}(n-1)\mathbf{x}(n) - \mathbf{k}(n)\mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}(n)\right] \\ &= \mathbf{w}(n-1) + d(n)\mathbf{k}(n) - \mathbf{k}(n)\mathbf{x}^T(n)\mathbf{w}(n-1) \\ &= \mathbf{w}(n-1) + \mathbf{k}(n)\left[d(n) - \mathbf{x}^T(n)\mathbf{w}(n-1)\right] \\ &= \mathbf{w}(n-1) + \mathbf{k}(n)\varepsilon(n) \qquad (4.9)\end{aligned}$$

这就是标准RLS算法的更新公式，这里的 $\varepsilon(n)$ 为先验估计误差。

4.1 RLS算法

标准RLS算法的执行流程：

1) 初始化： $\mathbf{w}(0)=\mathbf{0}$，$\mathbf{P}(0)=\delta^{-1}\mathbf{I}$， δ 是一个很小的正数。

2) 对每一个时刻 *n*，重复如下计算：

先验误差： $\varepsilon(n)=d(n)-\mathbf{w}^T(n-1)\mathbf{x}(n)$

增益向量： $\mathbf{k}(n)=\dfrac{\mathbf{P}(n-1)\mathbf{x}(n)}{\lambda+\mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}(n)}$

逆矩阵更新： $\mathbf{P}(n)=\dfrac{1}{\lambda}\left[\mathbf{P}(n-1)-\mathbf{k}(n)\mathbf{x}^T(n)\mathbf{P}(n-1)\right]$

滤波器更新： $\mathbf{w}(n)=\mathbf{w}(n-1)+\mathbf{k}(n)\varepsilon(n)$

4.1 RLS算法

RLS v.s. LMS

一个广泛的共识是，RLS算法的收敛速度和跟踪性能都优于LMS算法，所付出的代价是需要更复杂的计算。

算法	误差计算	收敛速度	跟踪性能	计算代价
LMS	后验误差	慢	慢	小
RLS	先验误差	快	快	大

由于RLS使用了自相关矩阵的逆矩阵的递推，所以，一旦输入信号的自相关矩阵接近奇异时，RLS的收敛速度和跟踪性能会严重恶化。

4.2 AP算法

仿射投影算法（Affine projection algorithm）是运算量介于LMS和RLS之间的一种自适应算法。

仍然考虑一个N阶系统 $\mathbf{w}(n)$ ：

注意此处考虑更一般的模型，输入和权重可以是复数。

$$y(n) = \mathbf{w}^H(n)\mathbf{x}(n) = \mathbf{x}^T(n)\mathbf{w}^*(n) \tag{4.11}$$

这一次，我们把 L 个输出信号组成一个向量：

$$\mathbf{y}(n) = [y(n),\ \ y(n-1),\ \ \dots\ y(n-L+1))]^T \tag{4.12}$$

与之对应的，输入信号由向量变成矩阵：

$$\mathbf{X}(n) = [\mathbf{x}(n),\ \ \mathbf{x}(n-1),\ \ \dots, \mathbf{x}(n-L+1)] \tag{4.13}$$

4.2 AP算法

此时，滤波器的输出变成了如下形式：

$$\mathbf{y}(n) = \mathbf{X}^T(n)\mathbf{w}^*(n) \tag{4.14}$$

令滤波器的自适应更新公式为：

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mu\Delta\mathbf{w}_n \tag{4.15}$$

将(4.15)代入(4.14)，得：

$$\mathbf{y}(n) = \mathbf{X}^T(n)\mathbf{w}^*(n-1) + \mu\mathbf{X}^T(n)\Delta\mathbf{w}_n^* \tag{4.16}$$

令滤波器的实际输出作为期望输出 $d(n)$，则先验误差向量：

$$\mathbf{e}(n) = \mathbf{y}(n) - \mathbf{X}^T(n)\mathbf{w}^*(n-1) \tag{4.17}$$

4.2 AP算法

为了简化，我们令 $\mu = 1$，则：

$$\mathbf{X}^T(n)\Delta\mathbf{w}_n^* = \mathbf{e}(n) \tag{4.18}$$

$$\mathbf{X}^H(n)\Delta\mathbf{w}_n = \mathbf{e}^*(n) \tag{4.19}$$

发现了什么?

滤波器系数的更新量 $\Delta \mathrm{w}_n^*$（$\Delta \mathrm{w}_n$），是由 L 个线性方程组成的线性方程组决定的。

线性方程组相关内容补充

对于线性方程组： $\mathbf{A}_{m\times n}\mathbf{x}_{n\times 1}=\mathbf{b}_{m\times 1}$

考虑（行/列）满秩的情况，分下面三种情况：

（一）如果 $m = n$，则：

$$\mathbf{x}=\mathbf{A}^{-1}\mathbf{b}$$ 唯一解、精确解

（二）如果 $m < n$，即方程的个数小于未知数的个数，此时方程组有无穷多个解。为了得到唯一解，必须增加约束条件，要求 x 的范数最小，这样得到的解称为最小范数解。

$$\mathbf{x}=\mathbf{A}^{H}\left(\mathbf{A}\mathbf{A}^{H}\right)^{-1}\mathbf{b}$$ 最小范数解

（三）如果 $m > n$，即方程的个数大于未知数的个数，此时方程组不存在精确解，只存在近似解。我们自然希望找到一个使方程组两边的误差平方和为最小的解，即最小二乘解。

$$\mathbf{x}=\left(\mathbf{A}^{H}\mathbf{A}\right)^{-1}\mathbf{A}^{H}\mathbf{b}$$ 最小二乘解

4.2 AP算法

利用线性方程组的结论，重新观察(4.19)式：

$$\mathbf{X}^H(n)\Delta\mathbf{w}_n = \mathbf{e}^*(n) \tag{4.19}$$

分为两种情况讨论：

<u>**情况1**</u>：$1 \leqslant L < N$，方程组为欠定方程，具有唯一的最小范数解：

$$\Delta\mathbf{w}_n = \mathbf{X}(n)\left(\mathbf{X}^H(n)\mathbf{X}(n)\right)^{-1}\mathbf{e}^*(n) \tag{4.20}$$

此时，自适应算法(4.15)变为：

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mu\mathbf{X}(n)\left(\mathbf{X}^H(n)\mathbf{X}(n)\right)^{-1}\mathbf{e}^*(n) \tag{4.21}$$

如果 $L = 1$，则退化为NLMS算法：

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mu e^*(n)\frac{\mathbf{x}(n)}{\mathbf{x}^H(n)\mathbf{x}(n)} \tag{4.22}$$

实际中，L 取2、3就足够了。

4.2 AP算法

利用线性方程组的结论，重新观察(4.19)式：

$$\mathbf{X}^H(n)\Delta\mathbf{w}_n=\mathbf{e}^*(n) \tag{4.19}$$

<u>情况2</u>： $L \geqslant N$，方程组为超定方程，其唯一解为最小二乘解：

$$\Delta\mathbf{w}_n=\left(\mathbf{X}(n)\mathbf{X}^H(n)\right)^{-1}\mathbf{X}(n)\mathbf{e}^*(n) \tag{4.23}$$

此时，自适应算法(4.15)变为：

$$\mathbf{w}(n)=\mathbf{w}(n-1)+\mu\left(\mathbf{X}(n)\mathbf{X}^H(n)\right)^{-1}\mathbf{X}(n)\mathbf{e}^*(n) \tag{4.24}$$

如果我们令：$\mu=1$，$\mathbf{R}(n)=\frac{1}{L}\mathbf{X}(n)\mathbf{X}^H(n)$，$\mathbf{r}(n)=\frac{1}{L}\mathbf{X}(n)\mathbf{y}^*(n)$，并结合等式(4.17)，可得：

$$\begin{aligned}\mathbf{w}(n)&=\mathbf{w}(n-1)+\left(\mathbf{X}(n)\mathbf{X}^H(n)\right)^{-1}\mathbf{X}(n)\left(\mathbf{y}^*(n)-\mathbf{X}^H(n)\mathbf{w}(n-1)\right)\\&=\mathbf{w}(n-1)+\mathbf{R}^{-1}(n)\mathbf{r}(n)-\mathbf{R}^{-1}(n)\mathbf{R}(n)\mathbf{w}(n-1)\end{aligned} \tag{4.25}$$

等价于RLS算法。

4.2 AP算法

仿射投影算法提供了一套自适应滤波分析的统一框架，这套框架将LMS、AP、RLS三种算法串联在一起。

AP算法是计算复杂度和性能介于LMS和RLS之间的自适应算法，在AEC、multi-channel等场景中都有应用。

大家可以根据实际项目中对计算复杂度、收敛速度的要求，选择合适的自适应算法，解决自身场景的问题。

本节介绍的RLS、AP算法都有各自的快速算法，可以提高收敛速度或减少计算量，大家如果感兴趣可以自行查阅相关文献。

自适应滤波方法总结

维纳滤波

平稳环境、最小均方意义下的统计最优滤波器

最小均方族自适应滤波器

LMS滤波器

分块LMS滤波器

归一化LMS滤波器

频域自适应滤波器

子带自适应滤波器

仿射投影自适应滤波器

收敛速度和计算复杂度介于LMS和RLS之间。

递归最小二乘自适应滤波器

RLS自适应滤波器提供更快的收敛速度和跟踪性能。

本章回顾

- 4.1 Recursive Least Square(RLS)算法
- 4.2 Affine Projection(AP)算法

LMS和RLS算法的实现

请大家编程实现基本LMS算法和RLS算法：

```
int conv(short* inputdata, long inputdata_length, double* rir, long rir_length, short* outputdata);
```

输入：input：一个新的输入sample： $x(n)$
adapt_filter：自适应滤波器buffer
filter_length：自适应滤波器的阶数

输出：err：滤波后的误差信号 $e(n)$
方法：分别用基本LMS算法、RLS算法实现自适应滤波过程
语言：C/C++

函数的调用，外层的语音数据和冲激响应函数，我们都已经准备好，请大家独立完成adapt_filt.cpp中的adapt_filtering这个核心函数。

为了简化问题，我们采用白噪声作为输入信号（audio.raw），并且令期望输出信号d(n)等于输入信号。很显然，最优滤波器就是一个简单的delta函数，大家可以根据滤波器收敛后的系数判断算法实现的正确性。

感谢聆听!

Thanks for Listening