Linear Filtering

Adaptive Filtering with LMS

Adaptive Filtering I

- If the signal statistics change over time, we need to update the filter weights
- Adaptive algorithms can be used to estimate second-order statistics from data (Online/sequential learning)

The Least-Mean-Squares (LMS) Algorithm I

- Iterative scheme
  - $\theta^{(i)} = \theta^{(i-1)} + \mu_i \nabla \theta^{(i-1)}$
  - $J(\theta^{(i)}) < J(\theta^{(i-1)})$
- Normal equations: $\nabla J(\theta) = \Sigma_x \boldsymbol{\theta} - \boldsymbol{p} = 0$
  - Access to second-order statistics is needed

Deriving LMS I

Robbins-Monro algorithm:

$$J(\theta) = \mathbb{E}[\mathcal{L}(\theta, y, x)], \quad \nabla J(\theta) = \mathbb{E}[\nabla \mathcal{L}(\theta, y, x)]$$

$$\theta_n = \theta_{n-1} - \mu_n \nabla \mathcal{L}(\theta_{n-1}, y_n, x_n)$$

$$\sum_n \mu_n^2 < \infty, \quad \sum_n \mu_n \to \infty : \quad \text{Convergence conditions.}$$

- Derivation of the Update Rule:

$$J(\theta) = E[e^2]$$

$$\nabla J(\theta) = E[\nabla e^2] = E[2e\nabla e] = -2E[(y - \theta^T x)x]$$

Deriving LMS II

- Robbins-Monro Update Rule: $\theta_n = \theta_{n-1} + \mu_n(y_n - \theta_{n-1}^T x_n)x_n$

Algorithm LMS Algorithm

1: Initialize $\boldsymbol{\theta}^{-1} = \mathbf{0} \in \mathbb{R}^l$
2: Select the value of μ
3: **for** $n = 0, 1, \ldots$ **do**
4: $\quad e_n = y_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n$
5: $\quad \boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu e_n \mathbf{x}_n$
6: **end for**

- To ensure agility step size is fixed $\mu_n = \mu$
  - Robbins-Monro convergence assumptions are not valid
  - Step size must be chosen carefully

Convergence of LMS I

Coefficient error vector:

$$\boldsymbol{c}_n := \boldsymbol{\theta}_n - \boldsymbol{\theta}^*$$

Convergence in the mean,

$$\mathbb{E}[\boldsymbol{c}_n] \quad \text{as} \quad n \to \infty$$

is subject to the condition

$$0 < \mu < \frac{2}{\lambda_{\text{max}}}$$

Analyzing the covariance matrix, $\Sigma_{c,n} := \mathbb{E}[\boldsymbol{c}_n \boldsymbol{c}_n^T]$, as $n \to \infty$, we get:

$$0 < \mu < \frac{2}{\text{trace}\{\Sigma_x\}}$$

Convergence of LMS II

We want to find a value of μ, such that w get a small misalignment:

$$\mathcal{M} := \frac{J_{\text{exc}}}{J_{min}}$$

Excess MSE at time instant n:

$$J_{\text{exc},n} = \text{trace}\{\Sigma_x \Sigma_{c,n-1}\} : \text{excess MSE at time instant } n$$

Expected misalignment (misadjustment):

$$\mathcal{M} \approx \frac{1}{2}\mu \, \text{trace}\{\Sigma_x\}$$

Trade-off between small $\mathcal{M}$ and convergence speed

The Affine Projection Algorithm (APA) I

- **Motivation:** LMS is slow in convergence
- **Concept:** Re-using data (requires more memory)
- **Trade-off when choosing q:** Faster convergence at the expense of increased misalignment.
- **Optimization problem:** Combination of gradient descent and recursive least squares (RLS)

$$\boldsymbol{\theta}_n = \arg\min_{\boldsymbol{\theta}} ||\boldsymbol{\theta} - \boldsymbol{\theta}_{n-1}||^2 \quad \text{s.t.} \quad \mathbf{x}_{n-i}^T \boldsymbol{\theta} = y_{n-i}, \quad i = 0, 1, \ldots, q-1$$

- Lagrange multpliers
- Update rule:

$$e_n = y_n - \mathbf{X}_n \boldsymbol{\theta}_{n-1} \quad \boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu \mathbf{X}_n^T (\delta \mathbf{I} + \mathbf{X}_n \mathbf{X}_n^T)^{-1} e_n$$

Normalized Least Mean Squares (NLMS) I

- Special case of APA with $q = 1$

Algorithm Normalized Least Mean Squares (NLMS) Algorithm

1: Initialize $\boldsymbol{\theta}^{-1} = \mathbf{0} \in \mathbb{R}^l$
2: Select the value of $0 < \mu < 2$, and a small δ value (Stability)
3: **for** $n = 0, 1, \ldots$ **do**
4: $\quad e_n = y_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n$
5: $\quad \boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \frac{\mu}{\mathbf{x}_n^T \mathbf{x}_n + \delta} \mathbf{x}_n e_n$
6: **end for**

Exercise 4.3.1 I

- Noise-canceling using LMS and NMLS
- Two audio files:
  - y: Clean speech
  - x: Noisy speech
- Parameters:
  - Filter length: I
  - Step size: μ

Exercise 4.3.1 II

```
def lms(x, y, L, mu):
    N = np.size(x, 0)
    w = np.zeros(L,)
    yhat = np.zeros(N,)
    x = np.concatenate((np.zeros(L-1,), x), axis=0)
    for n in range(0, N):
        x_n = x[n:n+L]
        yhat[n] = w.T @ x_n
        e = y[n] - yhat[n]
        w = w + mu*e*x_n
    return yhat

L = 128
mu_lms = mu_nlms = 0.8
yhat_lms = lms(x, y, L, mu_lms)
```

Exercise 4.3.1 III

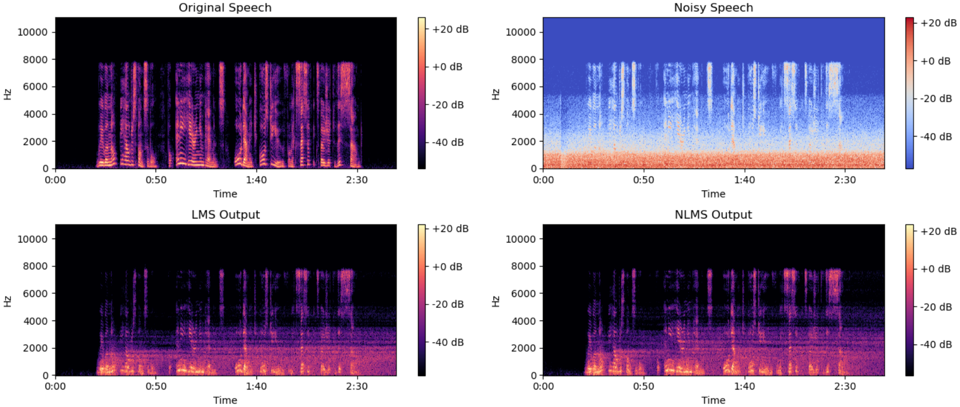

Figure: Using filter with l=128 and $\mu = 0.8$

Exercise 4.3.1 IV

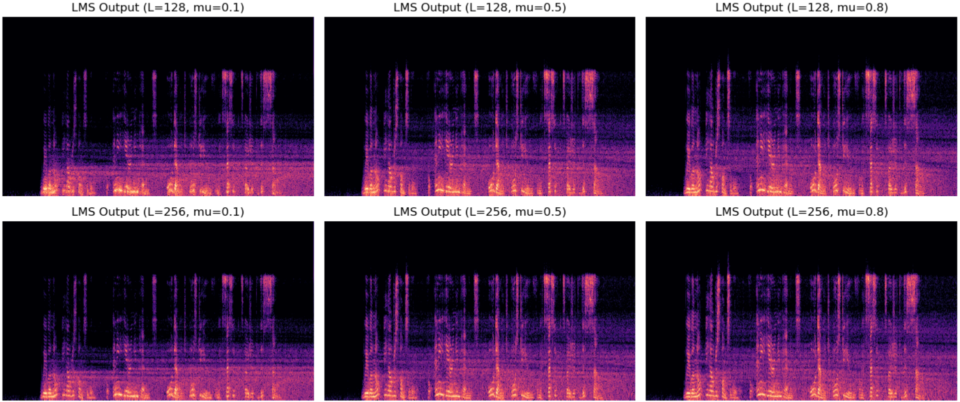

Figure: LMS with different filter lengths l and step sizes μ