Chapter 4: The Least-Mean-Square (LMS) Algorithm

Linear Adaptive Filtering

Kiran Bagale

July 2025

4.1 Introduction

Historical Context:

- Rosenblatt's perceptron: first learning algorithm for linearly separable pattern-classification problems
- LMS algorithm developed by Widrow and Hoff (1960): first linear adaptive-filtering algorithm for prediction and communication-channel equalization
- Development of LMS algorithm inspired by the perceptron

Common Feature: Both algorithms involve the use of a *linear combiner*, hence the designation "linear."

Why the LMS Algorithm is Remarkable

The LMS algorithm has established itself as:

- The **workhorse** for adaptive-filtering applications
- The **benchmark** against which other adaptive-filtering algorithms are evaluated

Reasons for its success:

1. **Computational complexity:** Linear with respect to adjustable parameters $\rightarrow$ computationally efficient yet effective
2. **Simplicity:** Simple to code and easy to build
3. **Robustness:** Robust with respect to external disturbances

From an engineering perspective, these qualities are highly desirable. The LMS algorithm has withstood the test of time.

4.2 FILTERING STRUCTURE OF THE LMS ALGORITHM

Problem Formulation:
Consider an unknown dynamic system stimulated by an input vector:

- Input elements: $x_1(i), x_2(i), \ldots, x_M(i)$
- Time index: $i = 1, 2, \ldots, n$
- System output: $d(i)$

External behavior described by the data set:

$$\mathcal{T} : \{x(i), d(i); i = 1, 2, \ldots, n, \ldots\} \tag{1}$$

where

$$x(i) = [x_1(i), x_2(i), \ldots, x_M(i)]^T \tag{2}$$

The sample pairs composing $\mathcal{T}$ are identically distributed according to an unknown probability law.

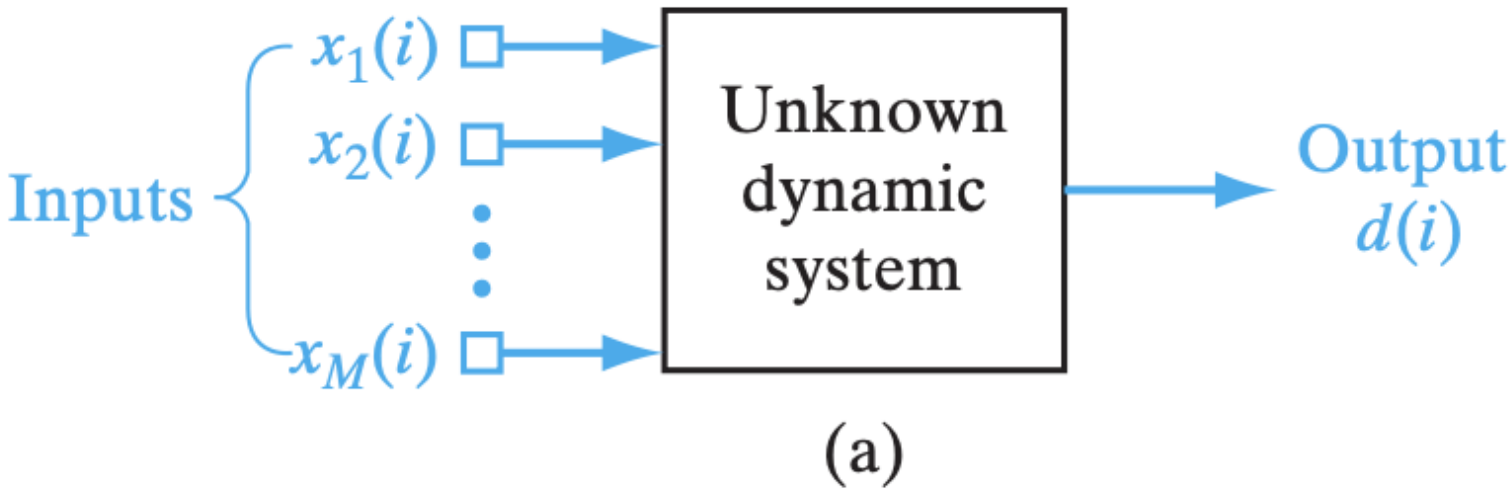
Input Vector Characteristics

Dimension M: Input dimensionality (dimensionality of the input space)

Origin of stimulus vector $x(i)$: Two fundamentally different ways:

1. **Spatial:** The M elements of $x(i)$ originate at different points in space
   - We speak of $x(i)$ as a *snapshot of data*
2. **Temporal:** The M elements of $x(i)$ represent the set of present and $(M-1)$ past values of some excitation
   - Values uniformly spaced in time

Figure 4.2a: Unknown Dynamic System



[Figure 4.2a] Unknown Dynamic System Block Diagram

Input: $x_1(i), x_2(i), \ldots, x_M(i)$

Output: $d(i)$

Design Objective

Problem: Design a multiple input-single output model of the unknown dynamic system by building it around a single linear neuron.

Algorithm Requirements:

- Start from an arbitrary setting of the neuron's synaptic weights
- Adjustments to synaptic weights in response to statistical variations are made on a continuous basis (time incorporated into the algorithm)
- Computations of weight adjustments completed within one sampling period

The neural model is referred to as an **adaptive filter**.

Adaptive Filter Operation

The adaptive filter operation consists of **two continuous processes**:

1. Filtering Process: Computation of two signals:

- **Output** $y(i)$: produced in response to the M elements of stimulus vector $x(i)$
- **Error signal** $e(i)$: obtained by comparing output $y(i)$ with corresponding output $d(i)$ from the unknown system

2. Adaptive Process: Automatic adjustment of synaptic weights according to the error signal $e(i)$

The combination of these processes constitutes a **feedback loop** around the neuron.

Figure 4.2b: Adaptive Filter Signal-Flow Graph

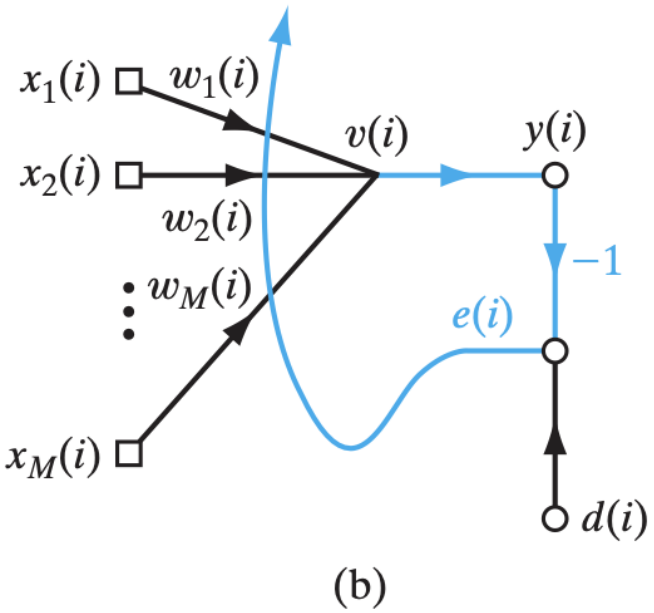


Figure 4.2b
Adaptive Filter Signal-Flow Graph

Shows the feedback loop structure with:
- Input vector $x(i)$
- Linear neuron with weights $w(i)$
- Output $y(i)$
- Error signal $e(i) = d(i) - y(i)$
- Weight adaptation mechanism

Mathematical Formulation: Linear Neuron

Since the neuron is linear, the output $y(i)$ is exactly the same as the induced local field $v(i)$:

$$y(i) = v(i) = \sum_{k=1}^{M} w_k(i)x_k(i) \tag{4.3}$$

where $w_1(i), w_2(i), \ldots, w_M(i)$ are the M synaptic weights of the neuron at time i.

Matrix form: Express $y(i)$ as an inner product:

$$y(i) = x^T(i)w(i) \tag{4.4}$$

where $w(i) = [w_1(i), w_2(i), \ldots, w_M(i)]^T$

Error Signal and Weight Adjustment

Error Signal: The neuron's output $y(i)$ is compared with the corresponding output $d(i)$ from the unknown system:

$$e(i) = d(i) - y(i) \tag{4.5}$$

Typically, $y(i) \neq d(i)$; hence their comparison results in the error signal.

Weight Adjustment Control: The manner in which the error signal $e(i)$ is used to control adjustments to the neuron's synaptic weights is determined by the **cost function** used to derive the adaptive-filtering algorithm.

This issue is closely related to **optimization**.

Unconstrained Optimization: A Review

Methods for Adaptive Filtering

Problem Statement

- Consider a cost function $e(\mathbf{w})$ that is continuously differentiable
- $\mathbf{w}$ is the unknown weight (parameter) vector
- Goal: Find optimal solution $\mathbf{w}^*$ that minimizes $e(\mathbf{w})$

Optimization Problem

Minimize the cost function $e(\mathbf{w})$ with respect to $\mathbf{w}$ such that:

$$e(\mathbf{w}^*) \leq e(\mathbf{w})$$

Necessary Condition for Optimality

Gradient Condition

$$\nabla e(\mathbf{w}^*) = \mathbf{0}$$

Where the gradient operator is:

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots, \frac{\partial}{\partial w_M}\right]^T$$

And the gradient vector is:

$$\nabla e(\mathbf{w}) = \left[\frac{\partial e}{\partial w_1}, \frac{\partial e}{\partial w_2}, \ldots, \frac{\partial e}{\partial w_M}\right]^T$$

Local Iterative Descent

- Start with initial guess $\mathbf{w}(0)$
- Generate sequence $\mathbf{w}(1), \mathbf{w}(2), \ldots$
- Cost function reduced at each iteration:

Descent Condition

$$e(\mathbf{w}(n+1)) \leq e(\mathbf{w}(n))$$

Important Note

Algorithm may diverge unless special precautions are taken!

Steepest Descent Algorithm

- Direction: Opposite to gradient vector
- Let $\mathbf{g} = \nabla e(\mathbf{w})$

Algorithm

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu \mathbf{g}(n)$$

Where:

- μ is the positive stepsize (learning-rate parameter)
- $\mathbf{g}(n)$ is the gradient vector at $\mathbf{w}(n)$

Correction Term

$$\Delta \mathbf{w}(n) = \mathbf{w}(n+1) - \mathbf{w}(n) = -\mu \mathbf{g}(n)$$

Convergence Analysis

Using first-order Taylor expansion:

$$e(\mathbf{w}(n+1)) \approx e(\mathbf{w}(n)) + \mathbf{g}^T(n)\Delta\mathbf{w}(n)$$

Substituting $\Delta\mathbf{w}(n) = -\mu\mathbf{g}(n)$:

$$e(\mathbf{w}(n+1)) \approx e(\mathbf{w}(n)) - \mu\|\mathbf{g}(n)\|^2$$

Result

For positive μ, the cost function decreases at each iteration (for small enough learning rates).

Learning Rate Effects

Small μ:

- Overdamped response
- Smooth trajectory
- Slow convergence

Large μ:

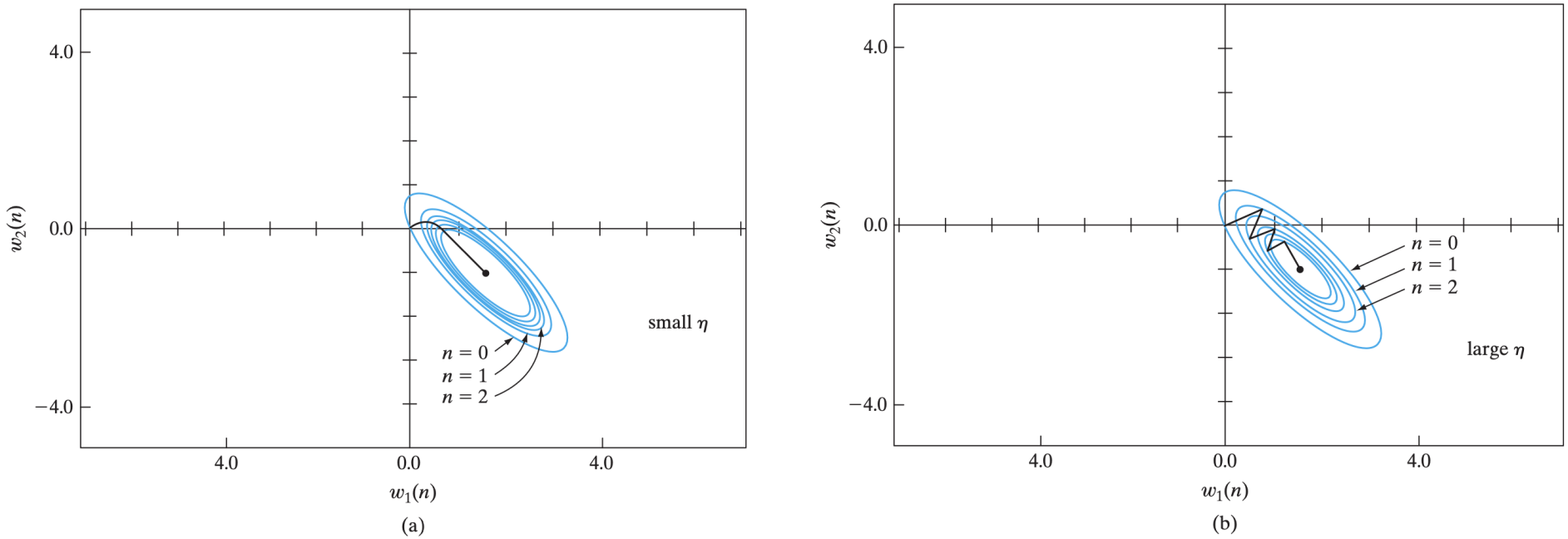
- Underdamped response
- Zigzagging path
- Faster but oscillatory

Too Large μ:

- Algorithm becomes unstable
- Divergence occurs

Key Point

Learning rate has profound influence on convergence behavior



[Figure 4.2] Trajectory of the method of steepest descent in a two-dimensional space for two different values of learning-rate parameter: (a) small μ (b) large μ. The coordinates w1 and w2 are elements of the weight vector w; they both lie in the w-plane.

Newton's Method

- Minimize quadratic approximation at each iteration
- Uses second-order Taylor expansion

Second-Order Approximation

$$\Delta e(\mathbf{w}(n)) \approx \mathbf{g}^T(n)\Delta\mathbf{w}(n) + \frac{1}{2}\Delta\mathbf{w}^T(n)\mathbf{H}(n)\Delta\mathbf{w}(n)$$

Where $\mathbf{H}(n)$ is the Hessian matrix:

$$\mathbf{H} = \nabla^2 e(\mathbf{w}) = \begin{bmatrix} \frac{\partial^2 e}{\partial w_1^2} & \frac{\partial^2 e}{\partial w_1 \partial w_2} & \cdots \\ \frac{\partial^2 e}{\partial w_2 \partial w_1} & \frac{\partial^2 e}{\partial w_2^2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

Note:The Hessian matrix, Hessian or (less commonly) Hesse matrix is a square matrix of second-order partial derivatives of a scalar valued function, or scalar field. Hessian is sometimes denoted by H or $\nabla\nabla$ or ∇^2 or $\nabla \otimes \nabla$ or D^2.

Newton's Method Algorithm

Minimizing the quadratic approximation:

$$\mathbf{g}(n) + \mathbf{H}(n)\Delta\mathbf{w}(n) = \mathbf{0}$$

Newton's Update Rule

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mathbf{H}^{-1}(n)\mathbf{g}(n)$$

Advantages

- Fast asymptotic convergence
- No zigzagging behavior

Limitations

- Requires $\mathbf{H}(n)$ to be positive definite
- High computational complexity

Gauss-Newton Method

- Addresses computational complexity of Newton's method
- Uses sum of squared errors cost function

Cost Function

$$e(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$$

Linearization

$$e'(i, \mathbf{w}) = e(i) + \left[\frac{\partial e(i)}{\partial \mathbf{w}}\right]^T (\mathbf{w} - \mathbf{w}(n))$$

In matrix form:

$$\mathbf{e}'(n, \mathbf{w}) = \mathbf{e}(n) + \mathbf{J}(n)(\mathbf{w} - \mathbf{w}(n))$$

Jacobian Matrix

The Jacobian $\mathbf{J}(n)$ is an $n \times M$ matrix:

$$\mathbf{J}(n) = \begin{bmatrix} \frac{\partial e(1)}{\partial w_1} & \frac{\partial e(1)}{\partial w_2} & \cdots & \frac{\partial e(1)}{\partial w_M} \\ \frac{\partial e(2)}{\partial w_1} & \frac{\partial e(2)}{\partial w_2} & \cdots & \frac{\partial e(2)}{\partial w_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e(n)}{\partial w_1} & \frac{\partial e(n)}{\partial w_2} & \cdots & \frac{\partial e(n)}{\partial w_M} \end{bmatrix}_{\mathbf{w}=\mathbf{w}(n)}$$

Relationship

$\mathbf{J}(n)$ is the transpose of the gradient matrix $\nabla \mathbf{e}(n)$

Note:The gradient and Jacobian are both related to the concept of derivatives in multivariable calculus, but they apply to different types of functions. The gradient is used for scalar-valued functions (functions that output a single number), while the Jacobian is used for vector-valued functions (functions that output a vector).

Gauss-Newton Update Rule

Minimizing $\frac{1}{2}\|\mathbf{e}'(n,\mathbf{w})\|^2$:

$$\mathbf{J}^T(n)\mathbf{e}(n) + \mathbf{J}^T(n)\mathbf{J}(n)(\mathbf{w} - \mathbf{w}(n)) = \mathbf{0}$$

Pure Gauss-Newton

$$\mathbf{w}(n+1) = \mathbf{w}(n) - (\mathbf{J}^T(n)\mathbf{J}(n))^{-1}\mathbf{J}^T(n)\mathbf{e}(n)$$

Requirements

- Only requires Jacobian (not Hessian)
- $\mathbf{J}^T(n)\mathbf{J}(n)$ must be nonsingular
- $\mathbf{J}(n)$ must have row rank n

Modified Gauss-Newton (Diagonal Loading)

To ensure nonsingularity, add diagonal matrix $\delta\mathbf{I}$:

Modified Update Rule

$$\mathbf{w}(n+1) = \mathbf{w}(n) - (\mathbf{J}^T(n)\mathbf{J}(n) + \delta\mathbf{I})^{-1}\mathbf{J}^T(n)\mathbf{e}(n)$$

This corresponds to the modified cost function:

$$e(\mathbf{w}) = \frac{1}{2}\left[\sum_{i=1}^{n} e^2(i) + \delta\|\mathbf{w} - \mathbf{w}(n)\|^2\right]$$

- δ is the regularization parameter
- Second term acts as a stabilizer
- Known as structural regularization

Method Comparison

Method	Convergence	Complexity	Stability
Steepest Descent	Slow	Low	Depends on μ
Newton's Method	Fast	High	Requires pos. def.
Gauss-Newton	Moderate	Moderate	With diagonal loading

Key Takeaways

- Trade-off between convergence speed and computational complexity
- Regularization techniques improve stability
- Choice depends on application requirements

4.3 THE WIENER FILTER

- **Wiener filtering** is one of the earliest techniques developed to reduce additive random noise in images.
- Assumes additive noise is a **stationary random process**, independent of pixel location.
- Minimizes the **mean square error** between the original and reconstructed image.
- Functions as a **space-varying low-pass filter**:
  - Uses a **low cutoff frequency** in low-detail (smooth) regions.
  - Uses a **high cutoff frequency** in high-detail regions (edges, textures).
- The **window size** controls the frequency cutoff:
  - **Larger windows** $\rightarrow$ lower cutoff frequency $\rightarrow$ more blurring and noise reduction.
  - **Smaller windows** $\rightarrow$ higher cutoff frequency $\rightarrow$ preserves more detail.

Least-Squares Filter using Gauss–Newton Method

- Least-squares filter minimizes the error:

$$\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{X}(n)\mathbf{w}(n)$$

- Gradient of error:

$$\nabla e(n) = -\mathbf{X}^T(n)$$

- Jacobian: $\mathbf{J}(n) = -\mathbf{X}(n)$
- Gauss–Newton update rule converges in one iteration:

$$\mathbf{w}(n+1) = (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)\mathbf{d}(n)$$

- Pseudoinverse form:

$$\mathbf{X}^+(n) = (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)$$

$$\Rightarrow \mathbf{w}(n+1) = \mathbf{X}^+(n)\mathbf{d}(n)$$

The weight vector $w(n+1)$ solves the linear least-squares problem, defined over an observation interval of duration n, as the product of two terms: the pseudo-inverse $\mathbf{X}^+(n)$ and the desired response $\mathbf{d}(n)$.

Limiting Form and Wiener Solution

- As $n \to \infty$, the least-squares solution becomes:

$$\mathbf{w}_o = \lim_{n\to\infty} (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)\mathbf{d}(n)$$

- For ergodic, stationary environments:

$$\mathbf{R}_{xx} = \mathbb{E}[\mathbf{x}(i)\mathbf{x}^T(i)] \quad \text{(autocorrelation matrix)}$$

$$\mathbf{r}_{dx} = \mathbb{E}[\mathbf{x}(i)d(i)] \quad \text{(cross-correlation vector)}$$

- Wiener solution:

$$\mathbf{w}_o = \mathbf{R}_{xx}^{-1}\mathbf{r}_{dx}$$

- Thus, the least-squares filter **asymptotically** becomes the Wiener filter.

Wiener Filter in Unknown Environments

- Wiener filter design requires knowledge of:
  - $\mathbf{R}_{xx}$: input correlation matrix
  - $\mathbf{r}_{dx}$: input-desired response cross-correlation
- In unknown or dynamic environments:
  - These statistics are not available.
  - Use **adaptive filtering** instead.
- One popular adaptive algorithm: **Least-Mean-Square (LMS)** algorithm.

Applications

- Adaptive filtering algorithms
- Neural network training
- Parameter estimation
- Signal processing optimization

Future Considerations

- Regularization techniques
- Convergence analysis
- Practical implementation issues

4.4 The Least-Mean-Square (LMS) Algorithm

What is the LMS Algorithm?

- LMS stands for **Least-Mean-Square**.
- It is an adaptive filtering algorithm used to adjust weights (parameters) in a model.
- Objective: Minimize the difference between the desired output and the actual output.
- It is widely used in applications like noise cancellation, system identification, and signal prediction.

Cost Function in LMS

- The LMS algorithm tries to minimize the cost function:

$$\varepsilon(\hat{w}) = \frac{1}{2}e^2(n) \tag{1}$$

- Here, $e(n)$ is the error between the desired and actual output:

$$e(n) = d(n) - \mathbf{x}^T(n)\hat{w}(n) \tag{2}$$

- $\hat{w}(n)$ is the weight vector (or parameters) we are adjusting.
- $\mathbf{x}(n)$ is the input vector at time n, and $d(n)$ is the desired output.

Gradient of the Cost Function

- To minimize the cost, we compute the gradient (slope) of the cost function:

$$\frac{\partial \varepsilon(\hat{w})}{\partial \hat{w}(n)} = -\mathbf{x}(n)e(n) \tag{3}$$

- This gives us the direction in which we should change the weights to reduce error.
- This method is called **gradient descent**.

LMS Weight Update Rule

- Using the gradient, the LMS algorithm updates weights as:

$$\hat{w}(n+1) = \hat{w}(n) + \mu \mathbf{x}(n) e(n) \tag{4}$$

- Here, μ is the **learning rate**, a small positive constant that controls how fast the weights are updated.
- This rule helps the algorithm learn from errors and improve over time.

Role of Learning Rate μ

- **Small** μ: Slower learning, more accurate and stable.
- **Large** μ: Faster learning, but may become unstable or inaccurate.
- Think of μ as a knob that adjusts how sensitive the algorithm is to new data.

Tip:

Choosing the right μ is important for good performance.

Stochastic Nature of LMS

- LMS uses only the current input-output pair to update weights (no need for full statistics).
- Because of this, the weight vector $\hat{w}(n)$ takes a **random path** (not smooth).
- Hence, LMS is called a **stochastic gradient algorithm**.
- Over time, the weights hover around the optimal solution.

Advantages of LMS Algorithm

- Simple and easy to implement.
- Requires only basic operations (addition and multiplication).
- Does not need knowledge of signal statistics.
- Suitable for real-time learning and adaptation.

LMS Algorithm: Step-by-Step Summary

Given:

- Input vector $\mathbf{x}(n)$
- Desired response $d(n)$
- Learning rate μ

Initialize: $\hat{w}(0) = 0$

For each time step n**:**

1. Compute error: $e(n) = d(n) - \hat{w}^T(n)\mathbf{x}(n)$
2. Update weights: $\hat{w}(n+1) = \hat{w}(n) + \mu\mathbf{x}(n)e(n)$

Matrix Form of LMS Update

- The update can also be written as:

$$\hat{w}(n+1) = [I - \mu \mathbf{x}(n)\mathbf{x}^T(n)]\hat{w}(n) + \mu \mathbf{x}(n)d(n) \tag{5}$$

- I is the identity matrix.
- This form shows how the input vector directly influences the weight update.
- Using the LMS algorithm, we recognize that

$$\hat{w}(n) = z^{-1}[\hat{w}(n+1)] \tag{6}$$

where z^{-1} is the unit-time delay operator, implying storage.

LMS as a Feedback System

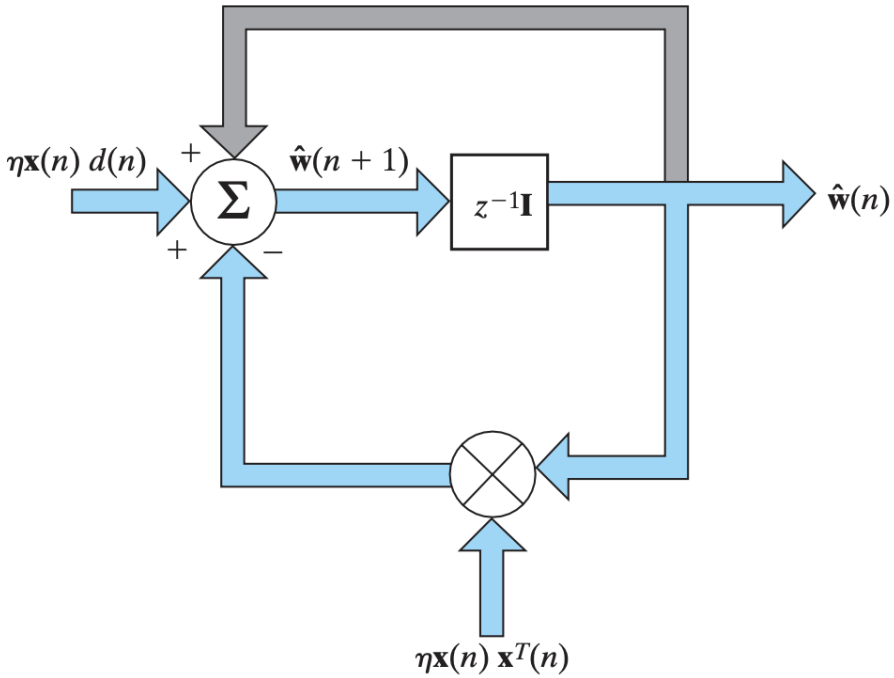


Fig 4.3

Signal-flow graph representation of the LMS algorithm. The graph embodies feedback depicted in color.

- The LMS algorithm includes a **feedback loop**, since the output depends on current weights, and weights are updated based on the output.
- The feedback affects the convergence behavior of the algorithm.
- Signal flow diagram (to be added by you) helps visualize this feedback process.

4.5 Markov Model: LMS Deviation from Wiener Filter

Why Use a Markov Model?

- We want to analyze how the LMS algorithm behaves over time compared to the optimal Wiener filter.
- For this, we define a new variable:

$$\boldsymbol{\xi}(n) = \mathbf{w}_o - \hat{\mathbf{w}}(n) \tag{1}$$

- $\boldsymbol{\xi}(n)$ is called the **weight-error vector**.
- It tells us how far the LMS estimate $\hat{\mathbf{w}}(n)$ is from the optimal Wiener solution $\mathbf{w}_o$.

LMS Evolution in Terms of Error Vector

- Recall LMS update equation:

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \mu \mathbf{x}(n) \left[d(n) - \mathbf{x}^T(n)\hat{\mathbf{w}}(n) \right] \tag{2}$$

- Using the definition , we get:

$$\boldsymbol{\xi}(n+1) = \mathbf{A}(n)\boldsymbol{\xi}(n) + \mathbf{f}(n) \tag{3}$$

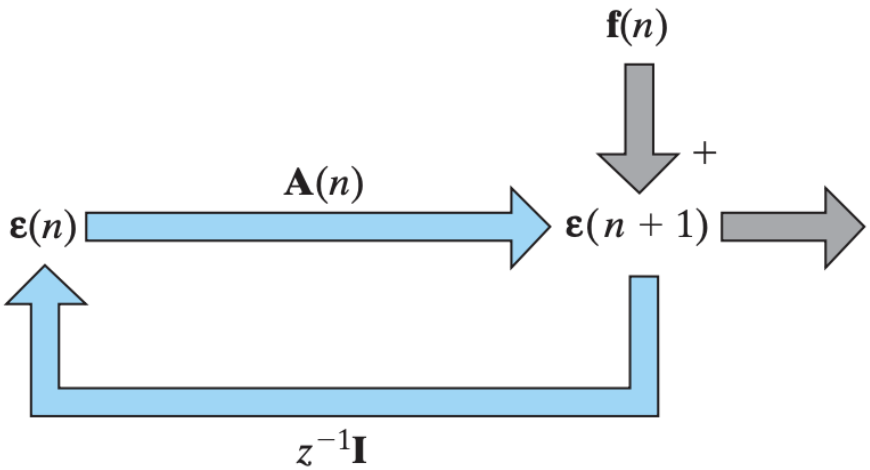


Fig 4.4

Signal-flow graph representation of the Markov model described; the graph embodies feedback depicted in color.

Transition Matrix and Noise Term

- The transition matrix $\mathbf{A}(n)$ is defined as:

$$\mathbf{A}(n) = \mathbf{I} - \mu \mathbf{x}(n)\mathbf{x}^T(n) \tag{4}$$

- The additive noise term is:

$$\mathbf{f}(n) = -\mu \mathbf{x}(n)e_o(n) \tag{5}$$

- Where the **optimal error signal** (Wiener filter error) is:

$$e_o(n) = d(n) - \mathbf{w}_o^T \mathbf{x}(n) \tag{6}$$

Understanding the Markov Model

- The LMS weight-error vector evolves based on:
  - Previous state $\xi(n)$
  - Transition dynamics via $\mathbf{A}(n)$
  - Noise force $\mathbf{f}(n)$
- This forms a **Markov process**:
  - Next state depends only on the current state (not full history).
  - The system is driven by noise, so it's stochastic.

Delay and Memory in LMS Model

- LMS has memory represented by delay operator:

$$\xi(n) = z^{-1}[\xi(n+1)]$$

- This shows feedback is present in the system.
- Feedback and memory play a key role in LMS convergence.
  Summary of the Markov Model
- Equation (3) describes how the weight-error vector changes over time.
- $\mathbf{A}(n)$ controls how much of the past error is retained.
- $\mathbf{f}(n)$ introduces randomness due to input signal and desired output.
- Provides a compact view of LMS dynamics compared to earlier signal-flow models.

4.6 The Langevin Equation and Brownian Motion

Motivation: Why Langevin Equation?

- LMS algorithm does not fully converge to a stable point.
- Instead, for small learning rate μ, it enters a **pseudo-equilibrium** state.
- The weight vector $\hat{w}(n)$ performs **random motion** around the Wiener solution w_o.
- This behavior is similar to **Brownian motion** - the random movement of particles suspended in a fluid (liquid or gas), described by the Langevin equation.

Physical Analogy: Particle in a Fluid

- Imagine a particle of mass m moving in a viscous fluid.
- The particle is affected by:
  1. **Frictional force:** $-\alpha v(t)$
  2. **Random fluctuations:** $F_f(t)$ (from surrounding fluid molecules)
- Velocity $v(t)$ is influenced by these forces.

Equipartition Law of Thermodynamics

- Average kinetic energy of the particle is given by:

$$\frac{1}{2}\mathbb{E}[v^2(t)] = \frac{1}{2}k_B T \tag{1}$$

- k_B: Boltzmann's constant
- T: Absolute temperature
- This relates thermal fluctuations to the average velocity of the particle.

The Langevin Equation

Equation of Motion

- Total force on the particle:

$$m\frac{dv}{dt} = -\alpha v(t) + F_f(t) \tag{2}$$

- $-\alpha v(t)$ is the damping (frictional) force (Stoke's law).
- $F_f(t)$ is the random fluctuating force from the fluid.
- Divide both sides of the motion equation by m:

$$\frac{dv}{dt} = -\gamma v(t) + \Gamma(t) \tag{3}$$

- Where:

$$\gamma = \frac{\alpha}{m}, \quad \Gamma(t) = \frac{F_f(t)}{m} \tag{4}$$

- $\Gamma(t)$ is called the **Langevin force**, a stochastic term due to thermal noise.
- Equation (3) is called the **Langevin Equation**.

Key Takeaways

- Langevin equation describes how a particle moves under:
  - Deterministic damping
  - Random fluctuating forces
- It models a system that never reaches perfect equilibrium, only fluctuates around it.
- This is exactly how the LMS algorithm behaves for small μ.

Connection to LMS:

The weight vector in LMS behaves like a particle undergoing Brownian motion near the optimal Wiener solution.

4.7 Kushner's Direct-Averaging Method

Why Use Direct-Averaging?

- The LMS algorithm's Markov model:

$$\boldsymbol{\xi}(n+1) = \mathbf{A}(n)\boldsymbol{\xi}(n) + \mathbf{f}(n)$$

- Is both:
  - **Nonlinear:** $\mathbf{A}(n)$ depends on $\mathbf{x}(n)\mathbf{x}^T(n)$.
  - **Stochastic:** $\{\mathbf{x}(n), d(n)\}$ are random samples.
- Makes rigorous analysis very difficult.

Kushner's Idea: Simplify the Model

- Under certain conditions, we can replace the complex model with a simplified one.
- This is called **Kushner's direct-averaging method**.
- Modified model:

$$\boldsymbol{\xi}_0(n+1) = \mathbf{A}(n)\boldsymbol{\xi}_0(n) + \mathbf{f}_0(n) \tag{1}$$

- New transition matrix:

$$\mathbf{A}(n) = \mathbf{I} - \mu\mathbb{E}[\mathbf{x}(n)\mathbf{x}^T(n)] \tag{2}$$

Assumptions for Kushner's Method

Kushner's method is valid when:

1. The learning rate μ is **sufficiently small**.
2. The noise term $\mathbf{f}(n)$ is **approximately independent** of the state $\xi(n)$.

These assumptions allow simplification from a stochastic model to a quasi-deterministic one.

- The modified model tracks the original model closely when $\mu \to 0$.
  Interpretation of the Modified Model
- $\xi_0(n)$ is the weight-error vector of the simplified (averaged) model.
- This model evolves deterministically with averaged data statistics.
- The randomness is averaged out using:

$$\mathbb{E}[\mathbf{x}(n)\mathbf{x}^T(n)] \text{ instead of } \mathbf{x}(n)\mathbf{x}^T(n)$$

- Easier to analyze stability and convergence.

Why It Works: Three Key Points

1. **Long Memory:** For small μ, the algorithm remembers past updates, making it smoother.
2. **Taylor Approximation:** Higher-order terms in μ are negligible, so linear terms dominate.
3. **Ergodicity:** Time averages can substitute ensemble averages.

Result:

The modified Markov model behaves nearly identically to the original LMS model for small μ.

Summary of Kushner's Method

- A powerful mathematical trick to simplify stochastic models.
- Applies to LMS convergence analysis.
- Replaces random components with expected values.
- Allows analytical convergence proofs and performance prediction.

4.8 Statistical LMS Learning Theory (Small μ)

- Now that we have:
  - Kushner's averaging method
  - A simplified model
- We can perform a principled statistical analysis of LMS.
- Requires a few reasonable assumptions.

Assumption I: Small Learning Rate μ

- A small μ justifies using the averaged Markov model:

$$\xi_0(n+1) = \left(I - \mu\mathbb{E}[\mathbf{x}(n)\mathbf{x}^T(n)]\right)\xi_0(n) + \mathbf{f}_0(n)$$

- This makes the LMS algorithm more robust and less sensitive to noise.
- Practical algorithms often choose small μ for stability.

Assumption II: White Estimation Error

- The error $e_o(n)$ from the Wiener filter is assumed to be white (uncorrelated over time).
- Based on the linear regression model:

$$d(n) = \mathbf{w}_o^T \mathbf{x}(n) + e_o(n) \tag{1}$$

- This means that the Wiener filter perfectly matches the environment model.

Assumption III: Joint Gaussianity

- The input vector $\mathbf{x}(n)$ and desired output $d(n)$ are assumed to be jointly Gaussian.
- This is a common and reasonable assumption in many physical systems.
- It simplifies the statistical analysis and derivations.

Natural Modes of the LMS Algorithm

- Let the input correlation matrix be:

$$\mathbf{R}_{xx} = \mathbb{E}[\mathbf{x}(n)\mathbf{x}^T(n)] \tag{2}$$

- Then the averaged transition matrix becomes:

$$\mathbf{A} = \mathbb{E}[I - \mu\mathbf{x}(n)\mathbf{x}^T(n)] = I - \mu\mathbf{R}_{xx} \tag{3}$$

Final Model for Statistical Analysis

- Substituting into the averaged Markov model:

$$\xi_0(n+1) = (I - \mu\mathbf{R}_{xx})\xi_0(n) + \mathbf{f}_0(n) \tag{4}$$

- This equation is the foundation for analyzing the behavior of LMS statistically.
- It shows how the error vector evolves under the influence of input correlation and noise.

4.9 Virtues and Limitations of the LMS Algorithm

Virtue 1: Simplicity and Efficiency

- LMS is easy to implement:

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \mu \mathbf{x}(n) e(n)$$

- Few lines of code are sufficient.
- Computational complexity is **linear** in the number of parameters.
- Ideal for real-time embedded systems and low-power devices.

Virtue 2: Robustness to Disturbances

- LMS is **model-independent**, works even without full knowledge of the system.
- It handles two main types of disturbances:
  - Initial weight-error: $\Delta\mathbf{w}(0) = \mathbf{w} - \hat{\mathbf{w}}(0)$
  - Explanational error: ε in the regression model:

$$d = \mathbf{w}^T\mathbf{x} + \varepsilon$$

- The algorithm can function even with poor initial guesses and modeling inaccuracies.

H_q Optimality: Worst-Case Design

- Transfer operator T maps disturbances to estimation error.
- H_q optimal design: Minimizes error energy under worst-case disturbance energy.
- A minimax game:
  - **Nature** (opponent) maximizes energy gain.
  - **Designer** chooses estimator to minimize error.
- LMS is optimal in the H_q (minimax) sense:

Plan for the worst scenario and optimize

Adaptability to Nonstationary Environments

- LMS performs well in both:
  - Stationary: Fixed statistics.
  - Nonstationary: Changing statistics.
- Can track time-varying Wiener solution $\mathbf{w}_o(n)$.
- Makes LMS a good choice in real-world dynamic systems.

Limitation 1: Slow Convergence

- LMS often needs:

$$\approx 10 \times \text{dimension of input vector}$$

  iterations to reach steady-state.
- Convergence becomes slower as input dimension increases.
- Not suitable for applications requiring fast adaptation.

Limitation 2: Sensitivity to Input Eigenstructure

- LMS performance is sensitive to the condition number of the input correlation matrix:

$$\mathbf{R}_{xx} = \mathbb{E}[\mathbf{x}(n)\mathbf{x}^T(n)]$$

- Condition number:

$$\kappa(\mathbf{R}) = \frac{\lambda_{\max}}{\lambda_{\min}} \tag{3.67}$$

- Large $\kappa(\mathbf{R}) \Rightarrow$ input is ill-conditioned $\Rightarrow$ LMS converges slowly or unstably.

Summary: LMS Algorithm

Key Strengths

- Simple and efficient to implement
- Robust to disturbances and model inaccuracies
- Adaptable to changing environments

Key Limitations

- Slow convergence, especially in high dimensions
- Sensitive to poorly conditioned inputs

4.9 Learning-Rate Annealing Schedules

Why Annealing the Learning Rate?

- LMS convergence is often slow because learning-rate μ is constant:

$$\mu(n) = \mu_0 \quad \text{for all } n \tag{1}$$

- A constant μ does not adapt to the algorithm's progress.
- We explore varying μ over time to improve both convergence speed and stability.

Stochastic Approximation Schedule

- In stochastic approximation (Robbins-Monro, 1951), the learning rate decays over time:

$$\mu(n) = \frac{c}{n} \tag{2}$$

- c is a constant.
- Guarantees convergence under certain conditions.
- However, for large c, early updates may be unstable (parameter blowup)

Search-Then-Converge Schedule

- Proposed by Darken and Moody (1992) to balance speed and stability:

$$\mu(n) = \frac{\mu_0}{1 + \frac{n}{\tau}} \tag{3.70}$$

- μ_0: initial learning rate, τ: search-time constant.
- Two distinct phases:
  1. **Search Phase** ($n \ll \tau$): $\mu(n) \approx \mu_0$ (fast exploration)
  2. **Converge Phase** ($n \gg \tau$): $\mu(n) \approx \frac{c}{n}$ (fine-tuning)

Learning-rate annealing schedules

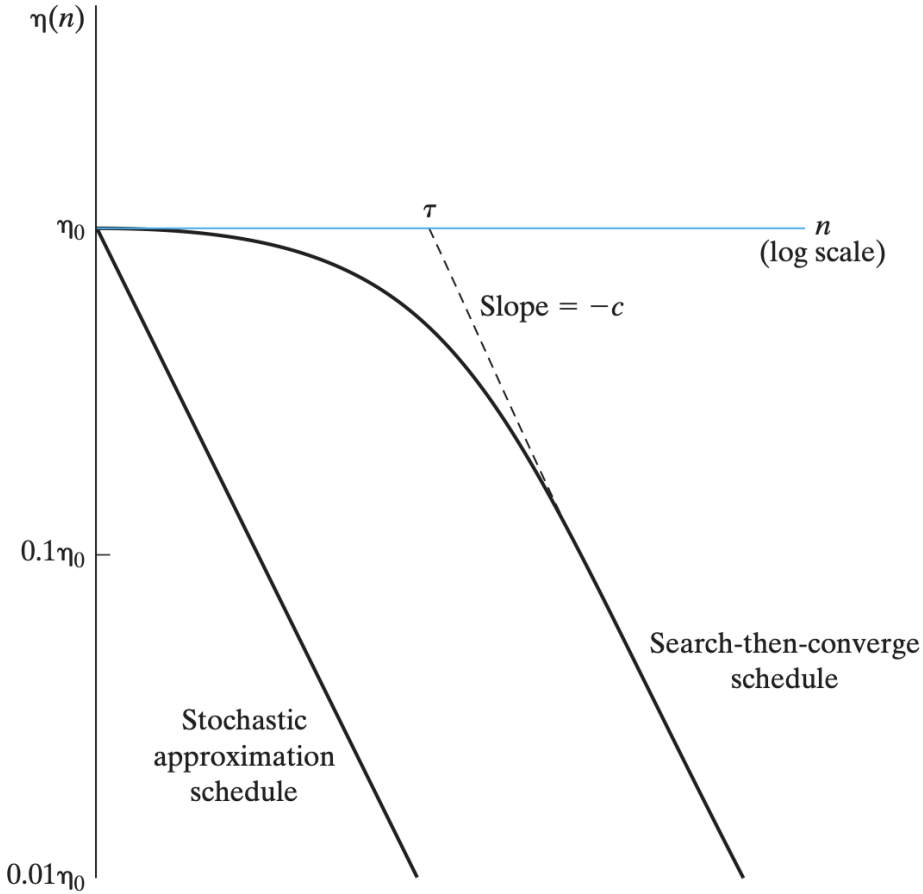


Fig: Annealing: The horizontal axis, printed in color, pertains to the standard LMS algorithm.

Intuition Behind the Schedule

- Early iterations: large μ helps find a ”good” region quickly.
- Later iterations: smaller μ helps settle near the optimal weights.
- Combines:
  - Fast convergence of standard LMS
  - Theoretical guarantees of stochastic approximation

Key Idea:

Start fast, then slow down to ensure precision and stability.

Summary: Annealing Approaches

- **Constant** μ: simple but slow convergence.
- $1/n$ **decay**: theoretically sound but unstable for small n.
- **Search-then-converge**: combines best of both.

Choose based on:

- Speed vs. stability tradeoff
- Stationarity of the environment
- Desired precision in final weights

Thank you!!!