SysEng 5212 /EE 5370
Introduction to Neural Networks and Applications

Week 4 : The Least-Mean-Square Algorithm

Cihan H Dagli, PhD

Professor of Engineering Management and Systems Engineering
Professor of Electrical and Computer Engineering
Founder and Director of Systems Engineering Graduate Program

dagli@mst..edu

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY
Rolla, Missouri, U.S.A.

Volunteers Tally

- Ameer Alam
- Murat Aslan
- Tatiana Cardona Sepulveda
- Prince Codjoe
- Xiongming Dai
- Jeffrey Dierker

Volunteers Tally

- Venkata Sai Abhishek Dwivadula
- Viraj Kishorkumar Gajjar
- Brian Guenther
- Anthony Guertin
- Timothy Guertin
- Joseph Hall

Volunteers Tally

- Seth Kitchen
- Gregory Leach
- Yu Li
- John Nganga
- Igor Povarich
- Jack Savage

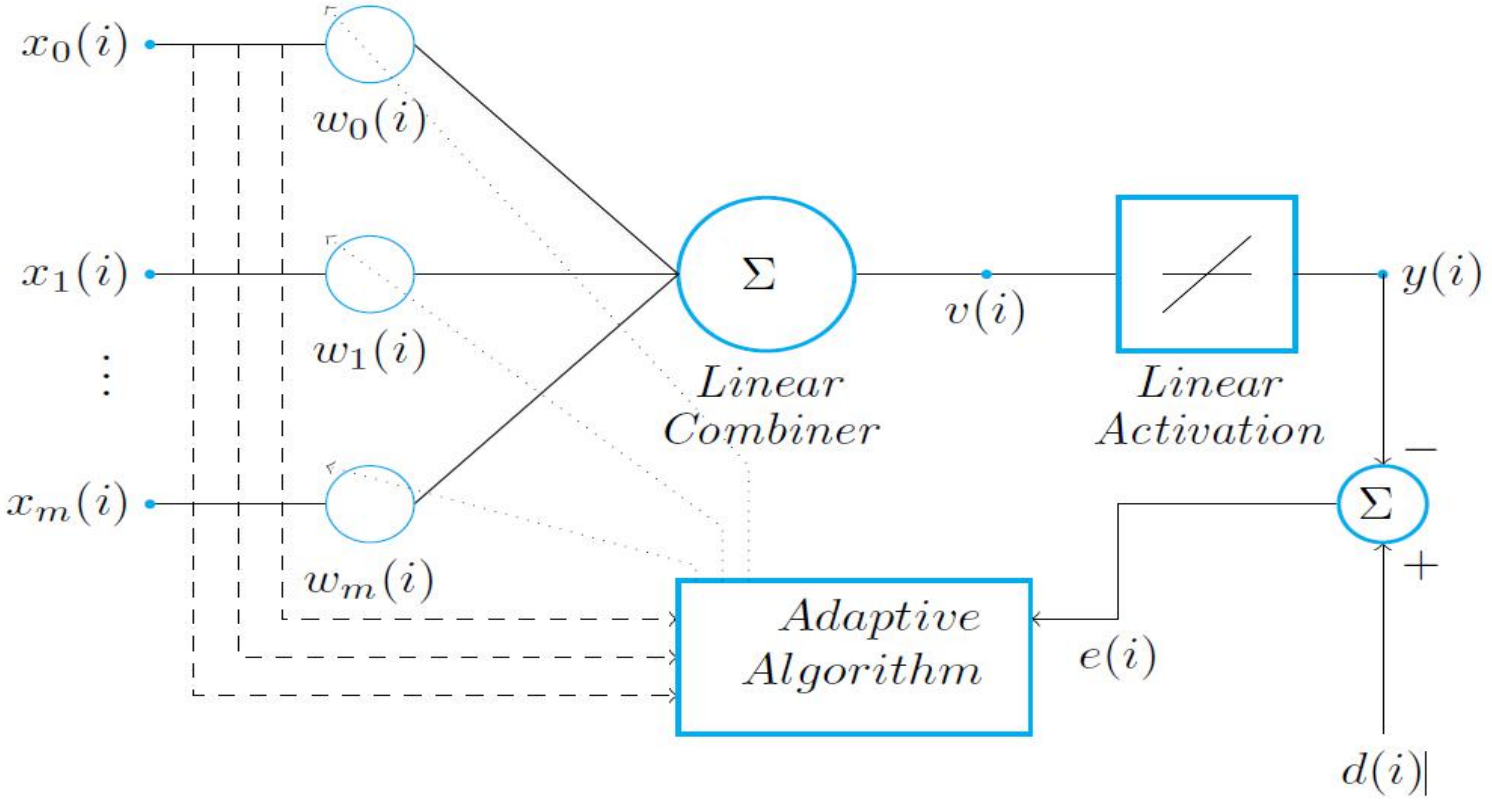
Volunteers Tally

- William Symolon
- Wayne Viers III
- Tao Wang
- Kari Ward
- Julia White
- Jun Xu

Lecture outline

- The linear neuron
- Unconstrained optimization
- Steepest descent algorithm
- Weiner-Hopf solution
- Least-Mean-Squares algorithm

Linear Model of a Neuron

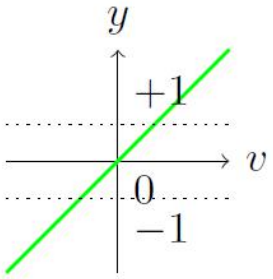


Linear Neuron Equations

Consider the input-output dataset where m is the dimensionality of the input space and i is the time step or iteration.

$$x(i) = [x_0(i),\ x_1(i), \ldots\ldots,\ x_m(i)]^T$$
$$\mathrm{d}(i) = [d_0(i),\ d_1(i), \ldots\ldots,\ d_m(i)]^T$$

Since the neuron has a linear transfer function, the output of the neuron is the same as its input,



$$y = v = \boldsymbol{w}x + b$$

MATLAB: y = purelin(v)

The output is given by,

$$y(i) = v(i) = \sum_{k=0}^{m} \boldsymbol{w}_k(i) x_k(i)$$

Without loss of generality we can assume that b = 0,

$$v(i) = \boldsymbol{w}^T(i) x(i)$$

To compute the error signal, we compare the neuron's output with the desired output: $e(i) = d(i) - \mathrm{y(i)}$

The linear combiner uses the error signal to adjust the network's weight. A basic error correction mechanism:

$$\Delta \boldsymbol{w}(i) = f[e(i)]$$

The Linear Neuron as an Adaptive Filter

Two continuous processes take place in the linear neuron model,

1. Filtering process: Two signals are being computed, an output *y(i)* and an error signal *e(i)*. This behavior of the linear combiner is similar to that of a linear transversal filter.
2. Adaptive process: Synaptic weights are automatically adjusted in accordance with the error signal, *e(i)*.

- Linear Neuron ⟺ Adaptive Filter

The manner in which the error signal is used to update weights depends upon the cost function of an algorithm.

The problem of finding the optimal weight vector w* is an optimization problem.

Weight Search as an Unconstrained Optimization Problem

Goal: To find w* that minimizes some cost function ε(w). The most common choice for E is the error signal

- The necessary condition for optimality is,

$$\nabla\epsilon(w^*) = 0$$

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

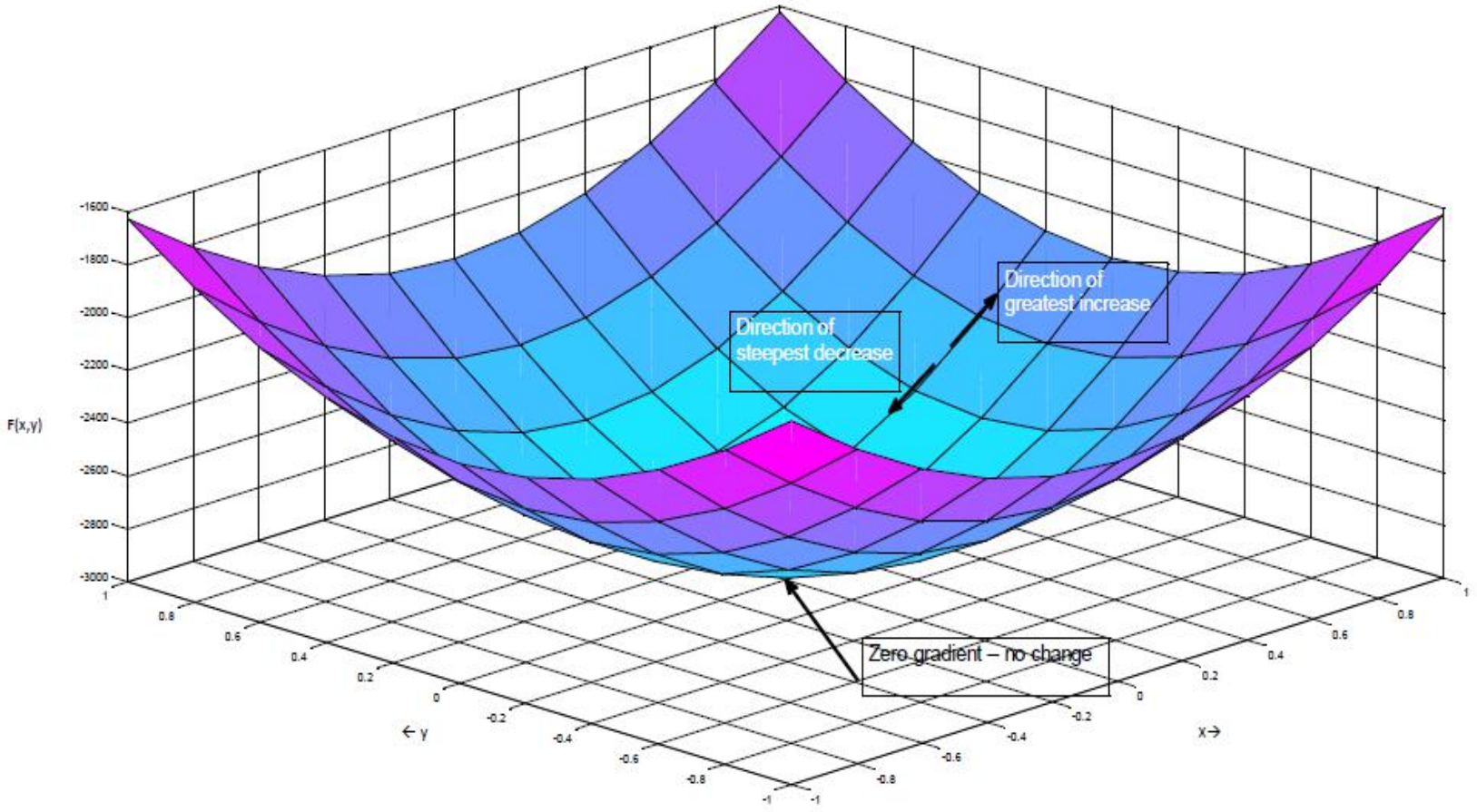
Thus the gradient of the cost function is,

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

What is the Gradient?

Partial derivative of a function of multiple variables or a vector

- F(x; y)
- The gradient is the vector of derivatives.


Direction of steepest decrease
Direction of greatest increase
Zero gradient – no change
F(x,y)
-1600
-1800
-2000
-2200
-2400
-2600
-2800
-3000
← y
x→
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1

Local Iterative Descent

One class of unconstrained optimization algorithms suited to our problem is based on the idea of *iterative descent*.

Initialize: Start with an initial guess, w(0)

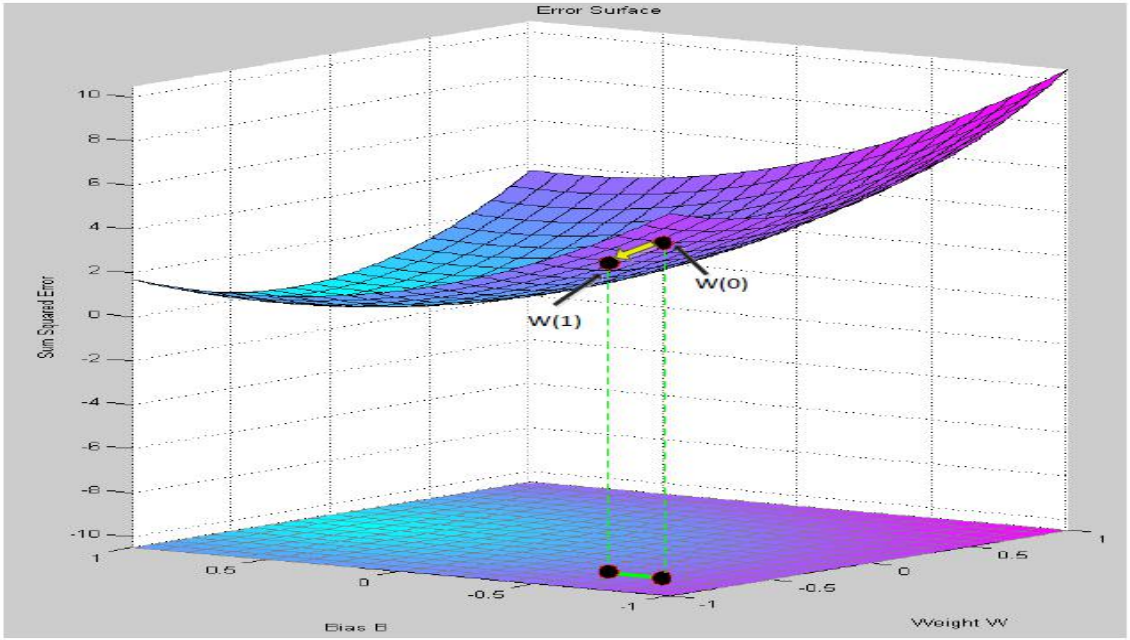
Iterate: Generate weight vectors w(1);w(2); : : : ;w(n) such that at each time step the cost function is less than the previous step.

$$\varepsilon(w(n+1)) < \varepsilon(w(n))$$

w(n + 1) is the new weight vector and w(n) is the old weight vector.

The goal is to eventually converge at the optimal weight vector w*.

Illustration of Iterative Descent



The new weight vector for the next iteration is chosen such that the overall cost function is minimized.

The Method of Steepest Descent (or Gradient Descent)

Well-known optimization algorithm in which the steepest descent is in a direction opposite to the gradient vector g,

$$g = \nabla \epsilon(w)$$

The weight update equation of the steepest descent algorithm is,

$$w(n + 1) = w(n) - \eta g(n)$$

where η is the learning rate parameter, and $g(n)$ is the gradient of the cost function when the weight is $w(n)$.

Correction Applied to Weights

What is the correction applied to the weights?

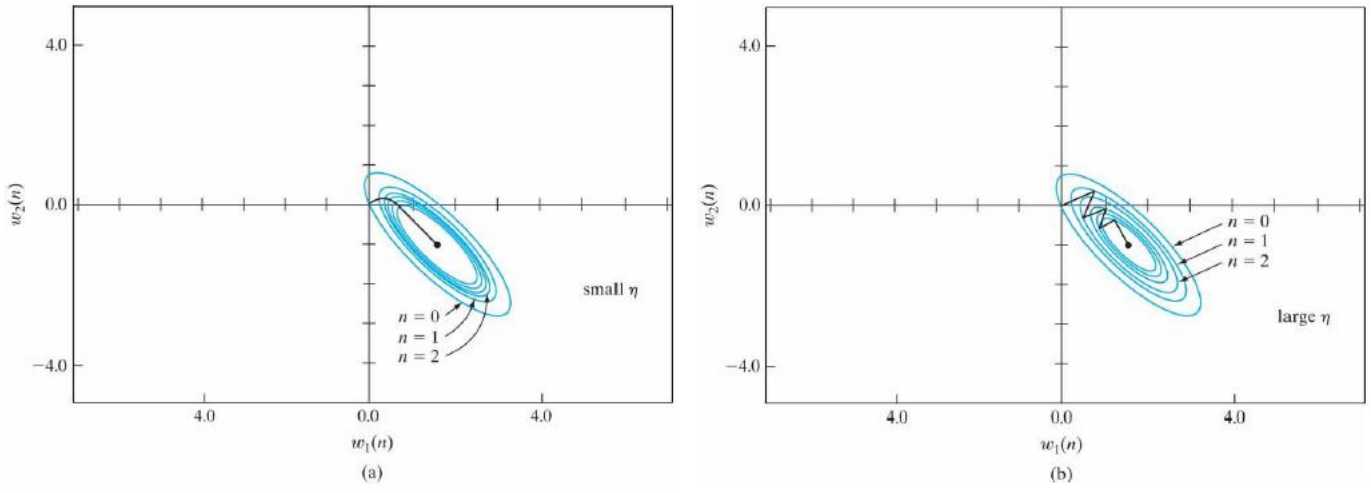
$$\Delta w(n) = w(n+1) - w(n)$$
$$= -\eta g(n)$$

As long as η is +ve and small, the condition for iterative descent

$$\varepsilon(w(n+1)) < \varepsilon(w(n))$$

is satisfied.

Trajectory of the Method of Steepest Descent



Trajectory of the method of steepest descent in a two-dimensional space for two different values of learning-rate parameter: (a) *small η* (b) *Large η* .The coordinates **w1** and **w2** are elements of the weight vector *w*; they both lie in the **W**-plane [1].

Comments on Steepest Descent

- Slow to converge to the optimal solution w*.

- The learning rate has a profound impact on the convergence behavior.
  - Smooth convergence trajectory with small .
  - Trajectory follows a zigzagging path with large values of .
  - If η exceeds a certain critical value, the algorithm diverges and becomes unstable.

Weiner-Hopf Solution

Classical approach to determining the optimal weight vector of an adaptive filter $\boldsymbol{W}^*$

Let $x(n)$ and $d(n)$ be the training input and desired response presented to the network at the nth iteration; w(n) is the current weight vector.

Network response and error signal,

$$v(n) = \boldsymbol{w}^T(n)x(n)$$

$$e(n) = d(n) - \mathrm{y(n)} = d(n) - \boldsymbol{w}^T(n)x(n)$$

Assuming x(n) and d(n) are wide-sense stationary processes, the cost function can be defined as,

$$\mathcal{E}(w) = \frac{1}{2}E\{e^2(n)\} = \frac{1}{2}E\{[d(n) - w^T(n)x(n)]^2\}$$

其实就是因为加了这个二分之一的系数，就导致了偏导上了一个2，而变成了为1的系数

Expanding the error function

$$\begin{aligned}\mathcal{E}(w) &= \frac{1}{2}E\left\{[d^2(n)\right\} - E\left\{d(n)x^T(n)\right\}w(n) \\ &+ \frac{1}{2}w^T(n)E\left\{x(n)x^T(n)\right\}w(n) \\ &= \frac{1}{2}E\left\{[d^2(n)\right\} - p^T w(n) + \frac{1}{2}w^T(n)C_x w(n)\end{aligned}$$

where,

$$\begin{aligned}p &= E\{d(n)x^T(n)\} \\ C_x &= E\{x(n)x^T(n)\}\end{aligned}$$

C_x is the covariance matrix for the input parameters, and p is the cross correlation vector between the desired response and the input patterns.

Continued…

To find the optimum condition, we set the gradient to zero,

$$\nabla\varepsilon(w) = \frac{\partial\varepsilon(w)}{\partial w} = -\mathrm{p} + C_x \mathrm{w(n)}$$

The optimal weights are obtained as,

$$w^* = {C_x}^{-1} p$$

This is the vector-matrix form of the Weiner-Hopf solution for the optimal weights of an adaptive filter.

Comments on the Weiner-Hopf Solution

Practically not a very useful result –

1. Computing the inverse of the covariance matrix is very computationally expensive.

2. Cannot perform online weight modification because the covariance matrix and the cross-correlation vector are not known a priori.

The Least-Mean-Squares Algorithm

- Widrow-Hoff developed the Least-Mean-Squares algorithm to circumvent these problems.
- The LMS algorithm uses a gradient descent approach to find the optimal weights.
- Uses the squared error function to approximate the steepest descent algorithm.
- Weights are modified in the negative gradient direction.
- To estimate the gradient we use the instantaneous estimate of the error surface.

LMS Weight Update Equation

The cost function is the instantaneous squared error signal of the linear neuron.

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

$$\frac{\partial \varepsilon(\hat{w})}{\partial \hat{w}} = e(n) \frac{\partial e(n)}{\partial w} \text{--------1}$$

$$e(n) = d(n) - \boldsymbol{w}^T(n)x(n) \text{-------2}$$

$$\frac{\partial e(n)}{\partial w} = -\mathrm{x(n)} \text{------3}$$

$$\frac{\partial \varepsilon(\hat{w})}{\partial \hat{w}} = -\mathrm{x(n)e(n)} \text{-----4}$$

$\frac{\partial \varepsilon(\hat{w})}{\partial \hat{w}}$ is the instantaneous estimate of the gradient.

LMS continued…

The LMS algorithm can be formulated as,

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

- Due to the instantaneous error values, the LMS algorithm performs a random walk on the weight surface
- No well defined trajectory like the steepest descent optimization
- Also called the Stochastic Gradient Descent Algorithm or the delta rule or the Widrow-Hoff learning algorithm

LMS Summary

TABLE 3.1 Summary of the LMS Algorithm

Training Sample: Input signal vector $= \mathbf{x}(n)$
Desired response $= d(n)$

User-selected parameter: η

Initialization. Set $\hat{\mathbf{w}}(0) = \mathbf{0}$.

Computation. For $n = 1, 2, \ldots,$ compute

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

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

From [1]

Comments on Learning Rate for the LMS Algorithm

- The learning rate parameter species the magnitude of the update step for the weights in the -ve gradient direction.
- Assuming that the x(n) and d(n) are jointly Gaussian, the algorithm is very robust for small values of .
- For small values of η , the algorithm modifies the weights very slowly.
- For large values of η , the weights begin to oscillate and the algorithm becomes unstable.

LMS- Upper Bounds on the Learning Rate

To ensure stability of the algorithm, the learning rate is bounded by,

$$0 < \eta < \frac{2}{\lambda_{max}}$$

λ_{max} is the largest eigenvalue of the input covariance matrix C_x.

Practically, we bound the learning rate as,

$$0 < \eta < \frac{2}{trace\{C_x\}}$$

trace function gives us the sum of the elements on the main diagonal of a square matrix.

LMS-Learning Rate Annealing Schedules

1. Simplest approach is to set η to a constant value.
   - Directly affects the accuracy of the weights
   - smaller values - painfully slow convergence, larger values -instability
   - A better approach is to vary η with time.
2. Stochastic approximation schedule
3. Search-then-converge approach
4. Adaptive normalization approach

Stochastic Approximation Schedule

We can compute η as,

$$\eta = \frac{k}{n}$$

k is some constant and n is the time step or iteration. The value of η decreases as n increases. If k is relatively small the algorithm is guaranteed to converge.

Search-then-Converge Approach

To make η decrease gradually, we can use,

$$\eta = \frac{\eta_0}{1 + \eta/\tau}$$

τ is the search time constant and is set $\tau >> 1$. The initial value of the learning parameter is $\eta_0 > 0$. Typically, $100 \leq \tau \leq 500$.

Search phase: η is relatively large and almost constant; decreases slowly

Converge phase: η decreases exponentially.

By selecting suitable values of η_0 and τ , the LMS algorithm can be made to converge much faster.

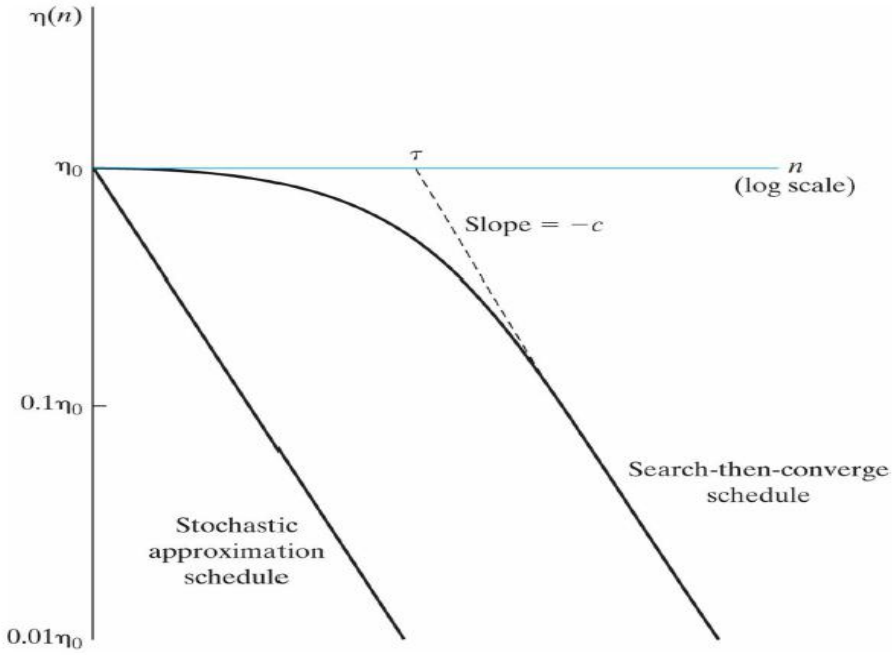
Adaptive Normalization Approach

We can adjust based on the input value at each iteration.

$$\eta(n) = \frac{\eta_0}{\|x(n)\|^2}$$

Stability is guaranteed if $0 < \eta_0 < 2$. To avoid stability issues, a more practical range is $0.1 \leq \eta_0 \leq 1$

Illustration of Learning Rate Annealing Schedules

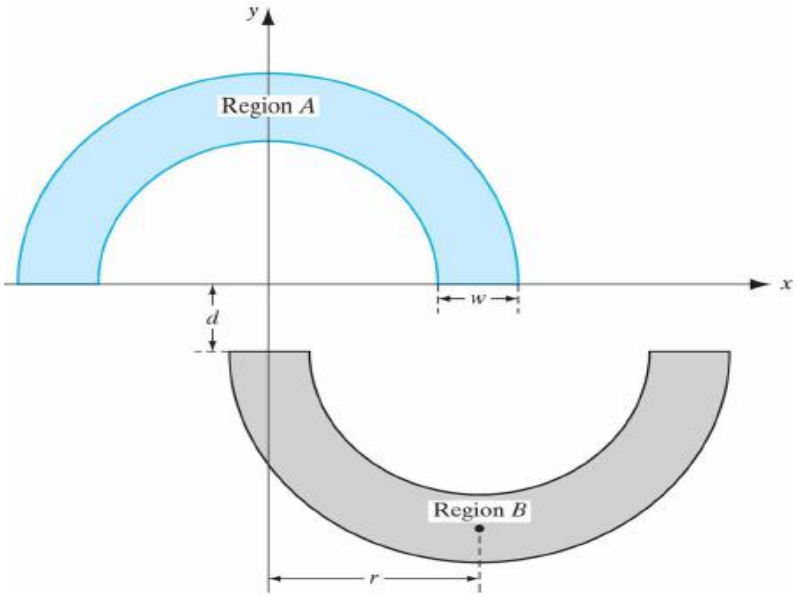


Learning-rate annealing schedules: The horizontal axis, printed in color, pertains to the standard LMS algorithm [1].

Examples: Computer Experiment

Objective: Apply the LMS algorithm to the double-moon classification problem

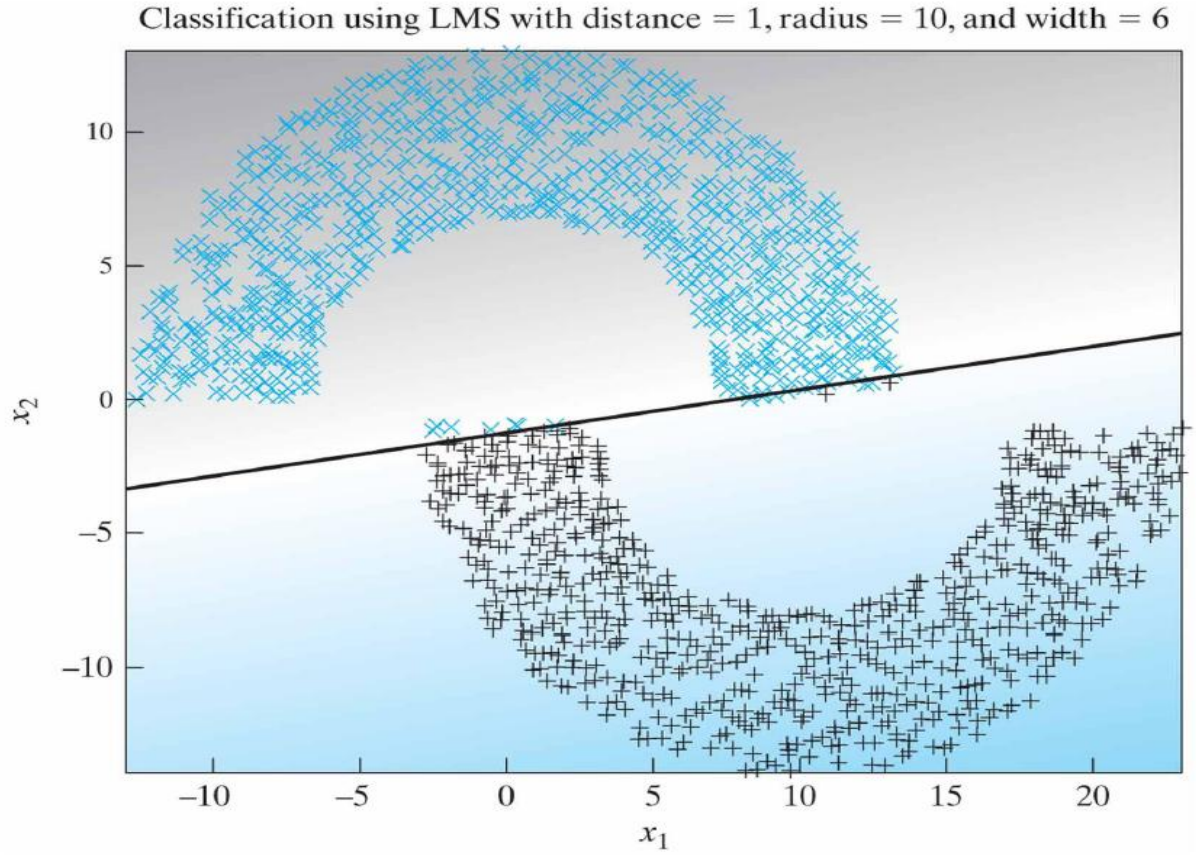
- radius of each moon, r = 10
- width of each moon, w = 6
- vertical distance separating
- the moons is variable, d



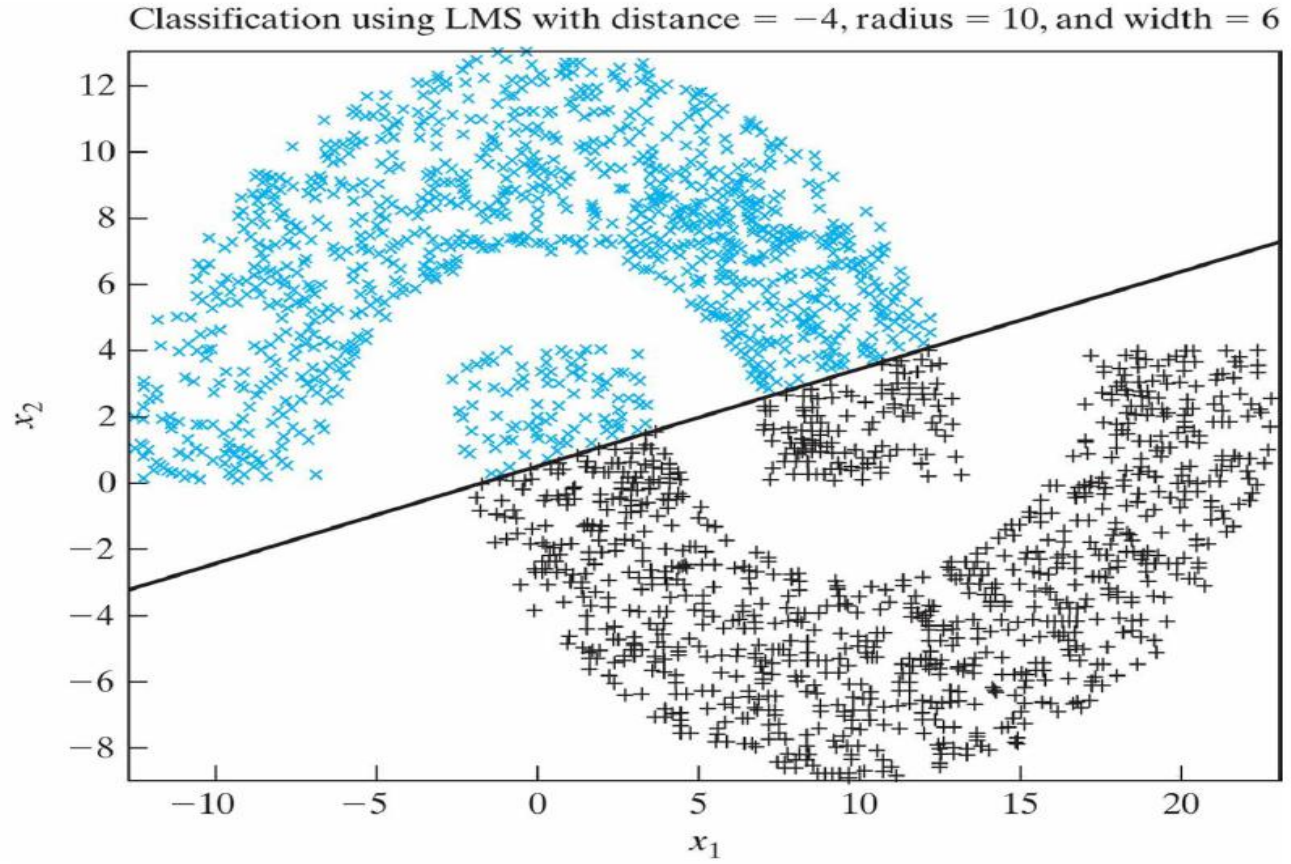
Simple Training and Testing Scheme

- Separate data into training and testing set
  - Train-test split, for example: 70%-30%
  - Cross-validation
- Initialize network
  - Initialize weights and biases
  - Select learning algorithm
  - Set network parameters such as learning rate and number of training epochs
- Train network until desired level of error is reached
- Test the trained network with the testing dataset
- Calculate network performance

LMS classification with distance 1



LMS classification with distance -4



Advantages and Limitations of the LMS Algorithm

- Very simple to code
- Computational complexity is linear in the number of adjustable parameters
- Robust to disturbances in the environment and to the initial weight vector
- Slow to converge
- Rate of convergence affected by dimensionality of the input space
  - requires approximately 10 times as many iterations to converge as the dimensions of the input space

For Next Week

- Homework 2 is due on Tuesday next week.
- Two power point slides describing your neural network application to an engineering problem.

References

[1] Simon Haykin, Neural Networks and Learning Machines, 3rd Ed, 2008.

[2] Ham and Kostanic, Principles of Neurocomputing for Science and Engineering, 2001.

[3] Hagan, Demuth and Beale, Neural Network Design, 1996, ISBN. 0-9717321-0-8.

[4] Raul Rojas, Neural Networks- A Systematic Introduction, Springer-Verlag, Berlin, New-York, 1996.