Least-Mean-Square Algorithm

Sanjay Singh[⋆†]

[⋆]Department of Information and Communication Technology
Manipal Institute of Technology, MAHE
Manipal-576104, INDIA
sanjay.singh@manipal.edu

[†]Centre for Artificial and Machine Intelligence (CAMI)
MAHE, Manipal-576104, INDIA

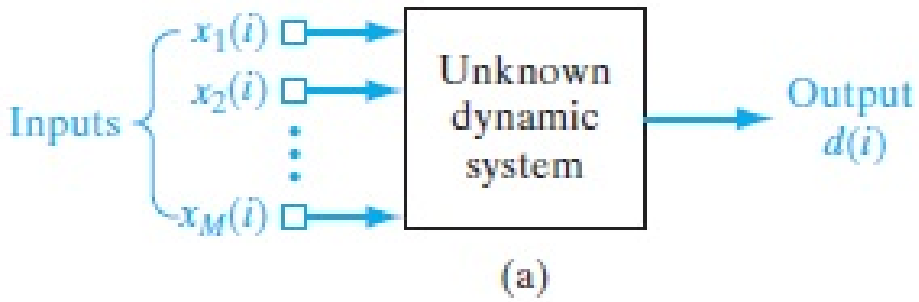
February 20, 2019

- Perceptron is the first learning algorithm for solving a linearly separable pattern-classification problem
- The least-mean-square (LMS) algorithm, developed by Widrow and Hoff, is the first adaptive-filtering algorithm for solving problem such as prediction and communication-channel equalization
- LMS was inspired by the perceptron algorithm

Multiple Faces of a Single Neuron

What a single neuron does can be viewed from different perspective:

- Classifier: as in perceptron
- Adaptive filter: as in signal processing



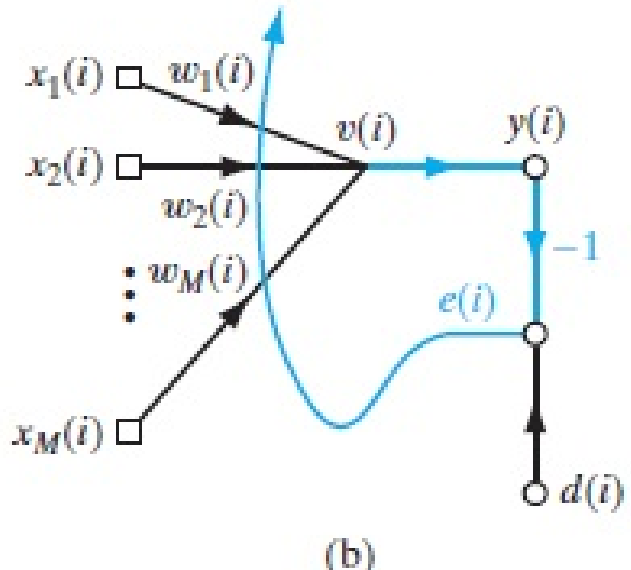


Figure 1: (a) Unknown dynamical system. (b) SFG of adaptive model

- Consider an unknown **dynamical system**, that takes m inputs and generates one output
- Behavior of the system described as its input/output pair:

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

where

$$\mathbf{x}(i) = [x_1(i), x_2(i), \ldots, x_m(i)]^T$$

- Stimulus $\mathbf{x}(i)$ can arise in one of two different ways: spatial and other temporal
  - The m elements of $\mathbf{x}(i)$ originate at different points in space; in this case we say $\mathbf{x}(i)$ as a snapshot of data
  - The m elements of $\mathbf{x}(i)$ represent the set of present and $(m-1)$ past values of some excitation that are uniformly spaced in time.

The problem we address is how to design a multiple-input-single-output model of the unknown dynamic system by building it around a single linear neuron

We keep following points in mind:

- The algorithm starts from an arbitrary setting of the neuron's synaptic weights
- Adjustments to the synaptic weights in response to statistical variation in the system's behavior are made on a continuous basis
- Computations of adjustments to the synaptic weights are completed within one sampling period

There are two important processes in adaptive filtering

- **Filtering process**-computation of two signals
  1. Output $y(i)$ is produced in response to the m elements of stimulus vector $\mathbf{x}(i)$, where $\mathbf{x}(i) \in \mathbb{R}^m$
  2. An error signal, $e(i)$ obtained by comparing output $y(i)$ with corresponding output $d(i)$ produced by the unknown system
- **Adaptive process** involves the automatic adjustment of the synaptic weights of the neuron in accordance with error signal $e(i)$

- Since the neuron is linear, output $y(i)$ is same as the induced local field $v(i)$, i.e,

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

- In matrix form, $y(i) = \mathbf{x}^T(i)\mathbf{w}(i)$, where $\mathbf{w}(i) = [w_1(i), w_2(i), \ldots, w_m(i)]^T$
- Error signal, $e(i) = d(i) - y(i)$
- Manner in which $e(i)$ controls the adjustment of synaptic weight is determined by the cost function
- Issue is closely related to optimization

Optimization

- Constrained optimization

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq b_i, \ i = 1, \ldots, m. \end{aligned}$$

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & \text{trace}(X) \\ \text{subject to} \quad & X_{ij} = M_{ij}, \ (i,j) \in \Omega, \\ & X \succeq 0. \end{aligned}$$

- Unconstrained optimization

$$\max_x f(x) \ or \ \min_x f(x)$$

Unconstrained Optimization Technique

- How can we adjust $\mathbf{w}(i)$ to gradually minimize $e(i)$?
  Note that $e(i) = d(i) - y(i) = d(i) - \mathbf{x}^T(i)\mathbf{w}(i)$, $d(i)$ and $\mathbf{x}(i)$ are fixed, only the change in $\mathbf{w}(i)$ can change $e(i)$
- We want to minimize the cost function $\mathcal{E}(w)$ w.r.t weight vector $\mathbf{w}$: find the optimal solution $\mathbf{w}^*$
- **Necessary condition** for optimality

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

  where the gradient operator ∇ is defined as

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

- Gradient vector of the cost function

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

A class of unconstrained optimization algorithm that is well suited for the design of adaptive filters is based on the idea of local iterative descent

Starting with an initial guess denoted by $\mathbf{w}(0)$, generate a sequence of weight vectors $\mathbf{w}(1), \mathbf{w}(2), \ldots$, such that the cost function $\mathcal{E}(\mathbf{w})$ is reduced at each iteration of the algorithm, as shown by

$$\mathcal{E}(\mathbf{w}(n+1)) < \mathcal{E}(\mathbf{w}(n))$$

where $\mathbf{w}(n)$ is the old value of the weight vector and $\mathbf{w}(n+1)$ is its updated value.

We'll consider following unconstrained-optimization methods that rely on the idea of iterative descent

- Method of steepest descent
- Newton's method
- Gauss-Newton method

Method of steepest descent

- Here, successive adjustments applied to the weight vector $\mathbf{w}$ are in the direction of steepest descent, i.e, in a direction opposite to the gradient vector $\nabla\mathcal{E}(\mathbf{w})$
- Define, $\mathbf{g} = \nabla\mathcal{E}(\mathbf{w})$
- Steepest descent algorithm is formally described as

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

  , where η is called step size or learning-rate parameter
- From iteration n to $n+1$ the algorithm applies the correction

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

Sanjay Singh Least-Mean-Square Algorithm

- To show that the formulation of steepest descent algorithm satisfies the condition

$$\mathcal{E}(\mathbf{w}(n+1)) < \mathcal{E}(\mathbf{w}(n))$$

, we use a first order [1]Taylor series around $\mathbf{w}(n)$ to approximate $\mathcal{E}(\mathbf{w}(n+1))$

- Here $x = \mathbf{w}(n+1)$ and $a = \mathbf{w}(n)$, now

$$\begin{aligned}\mathcal{E}(\mathbf{w}(n+1)) &= \mathcal{E}(\mathbf{w}(n)) + \frac{\mathbf{w}(n+1)-\mathbf{w}(n)}{1!}\mathcal{E}'(\mathbf{w}(n)) + \dots \\ &= \mathcal{E}(\mathbf{w}(n)) + \frac{\mathbf{w}(n+1)-\mathbf{w}(n)}{1!}\nabla_{\mathbf{w}}\mathcal{E}(\mathbf{w}(n)) \\ &= \mathcal{E}(\mathbf{w}(n)) + \mathbf{g}^T(n)\Delta\mathbf{w}(n)\end{aligned}$$

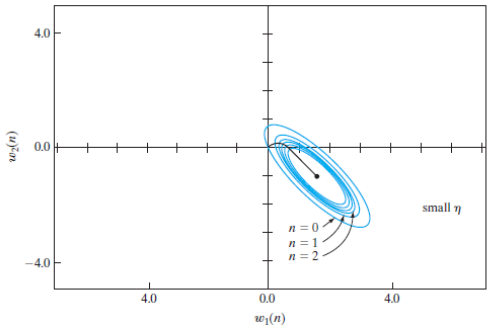
[1] $f(x) = f(a) + \frac{(x-a)}{1!}f'(a) + \frac{(x-a)^2}{2!}f''(a) + \dots$

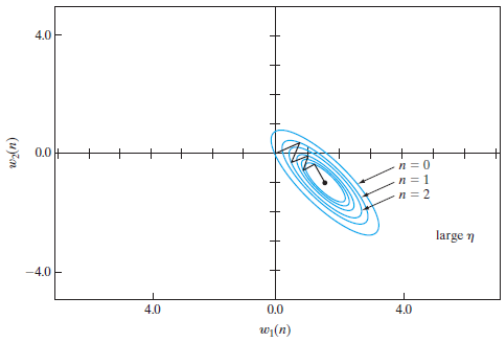
- Since $\Delta\mathbf{w}(n) = -\eta\mathbf{g}(n)$,
$\mathcal{E}(\mathbf{w}(n+1)) = \mathcal{E}(\mathbf{w}(n)) + \mathbf{g}^T(n)\Delta\mathbf{w}(n)$ becomes

$$\begin{aligned}\mathcal{E}(\mathbf{w}(n+1)) &= \mathcal{E}(\mathbf{w}(n)) - \eta\mathbf{g}^T(n)\mathbf{g}(n) && (1)\\ &= \mathcal{E}(\mathbf{w}(n)) - \eta||\mathbf{g}(n)||^2 && (2)\end{aligned}$$

- For positive learning-rate η

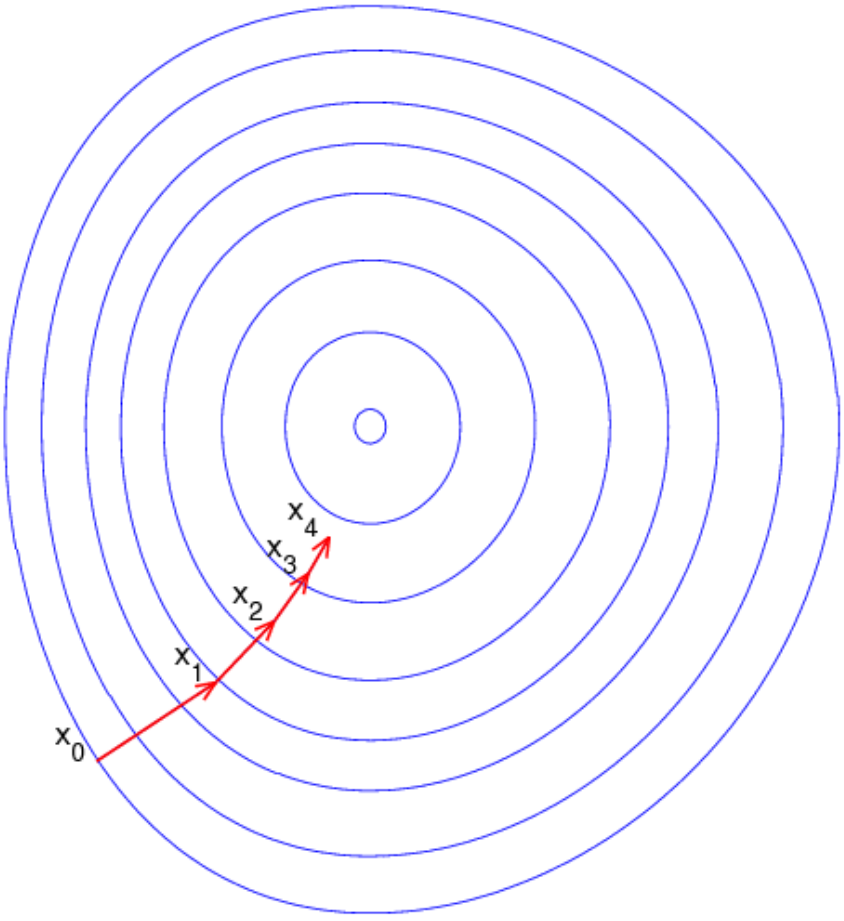
$$\mathcal{E}(\mathbf{w}(n+1)) < \mathcal{E}(\mathbf{w}(n))$$





Learning-rate η has profound influence on convergence of steepest descent algorithm

- Small η: overdamped, smooth trajectory
- Large η: underdamped, zigzagging (oscillatory) path
- η too large: algorithms becomes unstable



Newton's Method

- Newton's method is an extension of steepest descent, where the second-order term in the Taylor series expansion is used
- $$\mathcal{E}(\mathbf{w}(n+1)) = \mathcal{E}(\mathbf{w}(n)) + \frac{\mathbf{w}(n+1)-\mathbf{w}(n)}{1!}\mathcal{E}'(\mathbf{w}(n)) + \frac{(\mathbf{w}(n+1)-\mathbf{w}(n))^2}{2!}\mathcal{E}''(\mathbf{w}(n)) + \dots$$
$$= \mathcal{E}(\mathbf{w}(n)) + \frac{\mathbf{w}(n+1)-\mathbf{w}(n)}{1!}\nabla\mathcal{E}(\mathbf{w}(n)) + \frac{(\mathbf{w}(n+1)-\mathbf{w}(n))^2}{2!}\nabla^2\mathcal{E}(\mathbf{w}(n))$$
$$\Delta\mathcal{E}(\mathbf{w}(n)) = \mathbf{g}^T(n)\Delta\mathbf{w}(n) + \frac{1}{2}\Delta\mathbf{w}^T(n)\mathbf{H}(n)\Delta\mathbf{w}(n)$$
- $\mathbf{H}(n)$ is $m \times m$ Hessian matrix of $\mathcal{E}(\mathbf{w})$ and defined as $\mathbf{H} = \nabla^2\mathcal{E}(\mathbf{w})$
$$= \begin{bmatrix} \frac{\partial^2\mathcal{E}}{\partial w_1^2} & \frac{\partial^2\mathcal{E}}{\partial w_1 \partial w_2} & \dots & \frac{\partial^2\mathcal{E}}{\partial w_1 \partial w_m} \\ \frac{\partial^2\mathcal{E}}{\partial w_2 \partial w_1} & \frac{\partial^2\mathcal{E}}{\partial w_2^2} & \dots & \frac{\partial^2\mathcal{E}}{\partial w_2 \partial w_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2\mathcal{E}}{\partial w_m \partial w_1} & \frac{\partial^2\mathcal{E}}{\partial w_m \partial w_2} & \dots & \frac{\partial^2\mathcal{E}}{\partial w_m^2} \end{bmatrix}$$

Newton's Method

- $\Delta\mathcal{E}(\mathbf{w}(n)) = \mathbf{g}^T(n)\Delta\mathbf{w}(n) + \frac{1}{2}\Delta\mathbf{w}^T(n)\mathbf{H}(n)\Delta\mathbf{w}(n)$, differentiating this eqn w.r.t $\Delta\mathbf{w}(n)$ and equating with zero, we get
$$\mathbf{g}(n) + \mathbf{H}(n)\Delta\mathbf{w}(n) = 0$$
. Solving this equation for $\Delta\mathbf{w}(n)$ yields
$$\Delta\mathbf{w}(n) = -\mathbf{H}^{-1}(n)\mathbf{g}(n)$$
- $\mathbf{w}(n+1) = \mathbf{w}(n) + \Delta\mathbf{w}(n) = \mathbf{w}(n) - \mathbf{H}^{-1}(n)\mathbf{g}(n)$
- Newton's method converges quickly, and does not exhibit the zigzagging behavior
- For Newton's method to work, $\mathbf{H}(n)$ has to be **positive definitive matrix**
- Positive definitive of $\mathbf{H}(n)$ at every iteration can not be guaranteed

Some definitions

- A symmetric matrix $\mathbf{A} \in \mathbb{S}^n$ is **positive definite** (PD) if for all non-zero vectors $\mathbf{x} \in \mathbb{R}^n, \mathbf{x}^T\mathbf{A}\mathbf{x} > 0$
- A symmetric matrix $\mathbf{A} \in \mathbb{S}^n$ is **positive semidefinite** (PSD) if for all vectors $\mathbf{x} \in \mathbb{R}^n, \mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$
- A symmetric matrix $\mathbf{A} \in \mathbb{S}^n$ is **negative definite** (ND) if for all vectors $\mathbf{x} \in \mathbb{R}^n, \mathbf{x}^T\mathbf{A}\mathbf{x} < 0$
- A symmetric matrix $\mathbf{A} \in \mathbb{S}^n$ is **negative semidefinite** (NSD) if for all vectors $\mathbf{x} \in \mathbb{R}^n, \mathbf{x}^T\mathbf{A}\mathbf{x} \leq 0$
- Finally, a symmetric matrix $\mathbf{A} \in \mathbb{S}^n$ is **indefinite** if it is neither PSD nor NSD.

Gauss-Newton Method

- It deals with the computational complexity of Newton's method without compromising its convergence behavior
- Applicable to a cost function that is expressed as sum of error squares

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

where e_i is the error in the ith trial and it is function of weight vector $\mathbf{w}$

- Given an operating point $\mathbf{w}(n)$, we linearize the dependence of $e(i)$ on $\mathbf{w}$ by writing

$$e'(i, \mathbf{w}) = e(i) + \left[\frac{\partial e_i}{\partial \mathbf{w}}\right]^T_{\mathbf{w}=\mathbf{w}(n)} (\mathbf{w} - \mathbf{w}(n)), \quad i = 1, 2, \ldots, n$$

- In matrix notation, $\mathbf{e}'(n, \mathbf{w}) = \mathbf{e}(n) + J(n)(\mathbf{w} - \mathbf{w}(n))$ where $\mathbf{e}(n)$ is the error vector

$$\mathbf{e}(n) = [e(1), e(2), \ldots, e(n)]^T$$

Gauss-Newton Method

- $\mathbf{J}(n)$ is the $n \times m$ Jacobian matrix of $\mathbf{e}(n)$:

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

- Jacobian $\mathbf{J}(n)$ is the transpose of the $m \times n$ gradient matrix $\nabla \mathbf{e}(n)$ where $\nabla \mathbf{e}(n) = [\nabla e(1), \nabla e(2), \ldots \nabla e(n)]$

Example: Jacobian Matrix

- Given

$$e(x,y) = \begin{bmatrix} e_1(x,y) \\ e_2(x,y) \end{bmatrix} = \begin{bmatrix} x^2 + y^2 \\ cos(x) + sin(y) \end{bmatrix}$$

- Jacobian of $e(x,y)$ becomes

$$J(x,y) = \begin{bmatrix} \frac{\partial e_1(x,y)}{\partial x} & \frac{\partial e_1(x,y)}{\partial y} \\ \frac{\partial e_2(x,y)}{\partial x} & \frac{\partial e_2(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x & 2y \\ -sin(x) & cos(y) \end{bmatrix}$$

- For $(x,y) = (0.5\pi, \pi)$ we get

$$J(0.5\pi, \pi) = \begin{bmatrix} \pi & 2\pi \\ -sin(0.5\pi) & cos(\pi) \end{bmatrix} = \begin{bmatrix} \pi & 2\pi \\ -1 & -1 \end{bmatrix}$$

- Starting with $\mathbf{e}'(n, \mathbf{w}) = \mathbf{e}(n) + \mathbf{J}(n)(\mathbf{w} - \mathbf{w}(n))$, we want to set $\mathbf{w}$ such that error approaches 0, i.e

$$\mathbf{w}(n+1) = \arg\min_{\mathbf{w}} \left\{ \frac{1}{2} ||\mathbf{e}'(n, \mathbf{w})||^2 \right\}$$

- $\frac{1}{2}||\mathbf{e}'(n, \mathbf{w})||^2 = \frac{1}{2}(\mathbf{e}(n) + \mathbf{J}(n)(\mathbf{w} - \mathbf{w}(n)))^2$

$$= \frac{1}{2}||\mathbf{e}(n)||^2 + \mathbf{e}^T(n)\mathbf{J}(n)(\mathbf{w} - \mathbf{w}(n)) + \frac{1}{2}(\mathbf{w} - \mathbf{w}(n))^T \mathbf{J}^T(n)\mathbf{J}(n)(\mathbf{w} - \mathbf{w}(n))$$

- Differentiating w.r.t $\mathbf{w}$ and equating with 0 yields
  $\mathbf{J}^T(n)\mathbf{e}(n) + \mathbf{J}^T(n)\mathbf{J}(n)(\mathbf{w} - \mathbf{w}(n)) = 0$
  $\mathbf{w}(n+1) = \mathbf{w}(n) - (\mathbf{J}^T(n)\mathbf{J}(n))^{-1}\mathbf{J}^T(n)\mathbf{e}(n)$
- $\mathbf{J}^T(n)\mathbf{J}(n)$ needs to be non-singular
- So, $\mathbf{w}(n+1) = \mathbf{w}(n) - (\mathbf{J}^T(n)\mathbf{J}(n) + \delta\mathbf{I})^{-1}\mathbf{J}^T(n)\mathbf{e}(n)$

Linear Least Square Problem

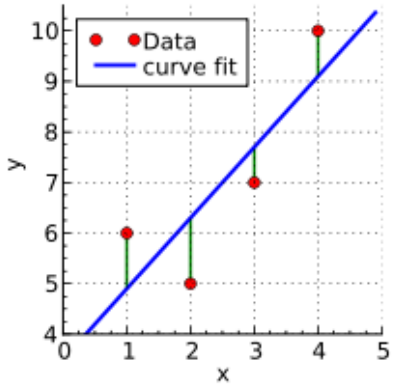


Figure 2: $y = a + bx$

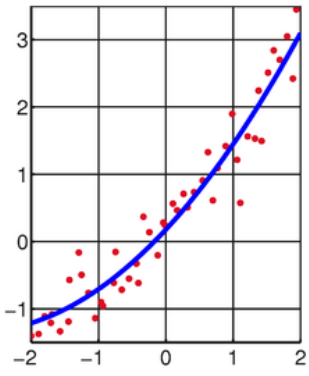


Figure 3: $y = a + bx + cx^2$

Linear Least Square Filter

- Here neuron is linear
- Cost function $\mathcal{E}(\mathbf{w})$ used to design filter is sum of error squares
- We may express the error vector $\mathbf{e}(n)$ as follows:

$$\mathbf{e}(n) = \mathbf{d}(n) - [\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(n)]^T \mathbf{w}(n) = \mathbf{d}(n) - \mathbf{X}(n)\mathbf{w}(n)$$

, where $\mathbf{d}(n)$ is $n \times 1$ desired response vector $\mathbf{d}(n) = [\mathbf{d}(1), \mathbf{d}(2), \ldots, \mathbf{d}(n)]^T$ and $\mathbf{X}(n)$ is $n \times m$ data matrix $\mathbf{X}(n) = [\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(n)]^T$

- Differentiating $\mathbf{e}(n)$ w.r.t $\mathbf{w}$, we get

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

So the Jacobian becomes $\mathbf{J}(n) = (\nabla \mathbf{e}(n))^T = -\mathbf{X}$

- Plugging this in to the Gauss-Newton equation $\mathbf{w}(n+1) = \mathbf{w}(n) - (\mathbf{J}^T(n)\mathbf{J}(n))^{-1}\mathbf{J}^T(n)\mathbf{e}(n)$, we get

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

- $\mathbf{w}(n+1) = \mathbf{X}^+(n)d(n)$
- Gauss-Newton method converges in single iteration
- The weight vector $\mathbf{w}(n+1)$ solves the linear least-squares problem, defined over an observation interval of duration n, as the product of two terms: the pseudoinverse $\mathbf{X}^+(n)$ and the desired response vector $\mathbf{d}(n)$

Some Definitions

- **Stochastic Process**-A stochastic process is a random process evolving with time. More specifically, in probability theory, a stochastic process is a time sequence representing the evolution of some system represented by a variable whose change is subject to a random variation
- **Ergodic**-A stochastic process is said to be ergodic if its statistical properties (such as mean and variance) can be deduced from a single, sufficiently long sample (realization) of the process
- **Stationary ergodic process**-this implies that the random process will not change its statistical properties with time and that its statistical properties can be deduced from a single sufficiently long sample of the process

Wiener Filter

Limiting form of Least-Square Filter for Ergodic Environment

- Let $\mathbf{w}_o$ denotes the limiting form of least-square filter as $n \to \infty$, i.e.,

$$\begin{aligned}\mathbf{w}_o &= \lim_{n\to\infty} \mathbf{w}(n+1) \\ &= \lim_{n\to\infty} (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)\mathbf{d}(n) \\ &= \lim_{n\to\infty} \left(\frac{1}{n}\mathbf{X}^T(n)\mathbf{X}(n)\right)^{-1} \times \lim_{n\to\infty} \frac{1}{n}\mathbf{X}^T(n)\mathbf{d}(n)\end{aligned}$$

- Consider $x(i)$ and $d(i)$ are drawn from a stationary ergodic environment
- Such environment is partially described by second order statistics
  - Correlation matrix of the input vector $\mathbf{x}(i)$, defined as $\mathbf{R}_{xx} = \mathbb{E}[\mathbf{x}(i)\mathbf{x}^T(i)]$
  - Cross-correlation vector between $\mathbf{x}(i)$ and $d(i)$; it is defined as $\mathbf{r}_{xd} = \mathbb{E}[\mathbf{x}(i)d(i)]$

- Under the ergodicity assumption we can write,

$$\mathbf{R}_{xx} = \lim_{n\to\infty} \frac{1}{n} X(n)X^T(n)$$

, and

- $\mathbf{r}_{xd} = \lim_{n\to\infty} \frac{1}{n} \mathbf{X}^T(n)\mathbf{d}(n)$
- We may reformulate the linear least-squares solution as follows:

$$\begin{aligned}
\mathbf{w}_o &= \lim_{n\to\infty} \mathbf{w}(n+1) \\
&= \lim_{n\to\infty} (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)\mathbf{d}(n) \\
&= \lim_{n\to\infty} (\frac{1}{n}\mathbf{X}^T(n)\mathbf{X}(n))^{-1} \lim_{n\to\infty} \frac{1}{n}\mathbf{X}^T(n)\mathbf{d}(n) \\
&= R_{xx}^{-1}\mathbf{r}_{xd}
\end{aligned}$$

- $\mathbf{w}_o$ is called the Wiener solution to the linear optimum filtering problem
- *For an ergodic process, the linear least-squares filter asymptotically approaches the Wiener filter as the number of observation approaches infinity*

- Designing Wiener filter requires second order statistics
- In most of the cases it is not available
- One can deal with an unknown environment by using a linear adaptive filter
- Filter is able to adjust its free parameters in response to statistical variations in the environment
- One such algorithm is least-mean-square

Least-Mean-Square (LMS) Algorithm

- LMS is configured to minimize the instantaneous value of the cost function

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2}e^2(n)$$

- Differentiating $\mathcal{E}(\mathbf{w})$ w.r.t $\mathbf{w}$ yields

$$\frac{\partial \mathcal{E}(\mathbf{w})}{\partial \mathbf{w}} = e(n)\frac{\partial e(n)}{\partial \mathbf{w}}$$

- Similar to linear least-square, LMS algorithm operates with a linear neuron, so the error signal is

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

- Hence, $\frac{\partial e(n)}{\partial \mathbf{w}} = -\mathbf{x}(n)$ and

$$\frac{\partial \mathcal{E}(\mathbf{w})}{\partial \mathbf{w}} = -\mathbf{x}(n)e(n)$$

Least-Mean-Square (LMS) Algorithm

- Using this result as an estimate for the gradient vector, we may write

$$\hat{\mathbf{g}}(n) = -\mathbf{x}(n)e(n)$$

- For the method of steepest descent, we may formulate the LMS algorithm as

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

- Note that the inverse of learning-rate parameter η acts as a measure of the memory of LMS algorithm
- The smaller we make η, the longer the memory over which LMS algorithm remembers past data

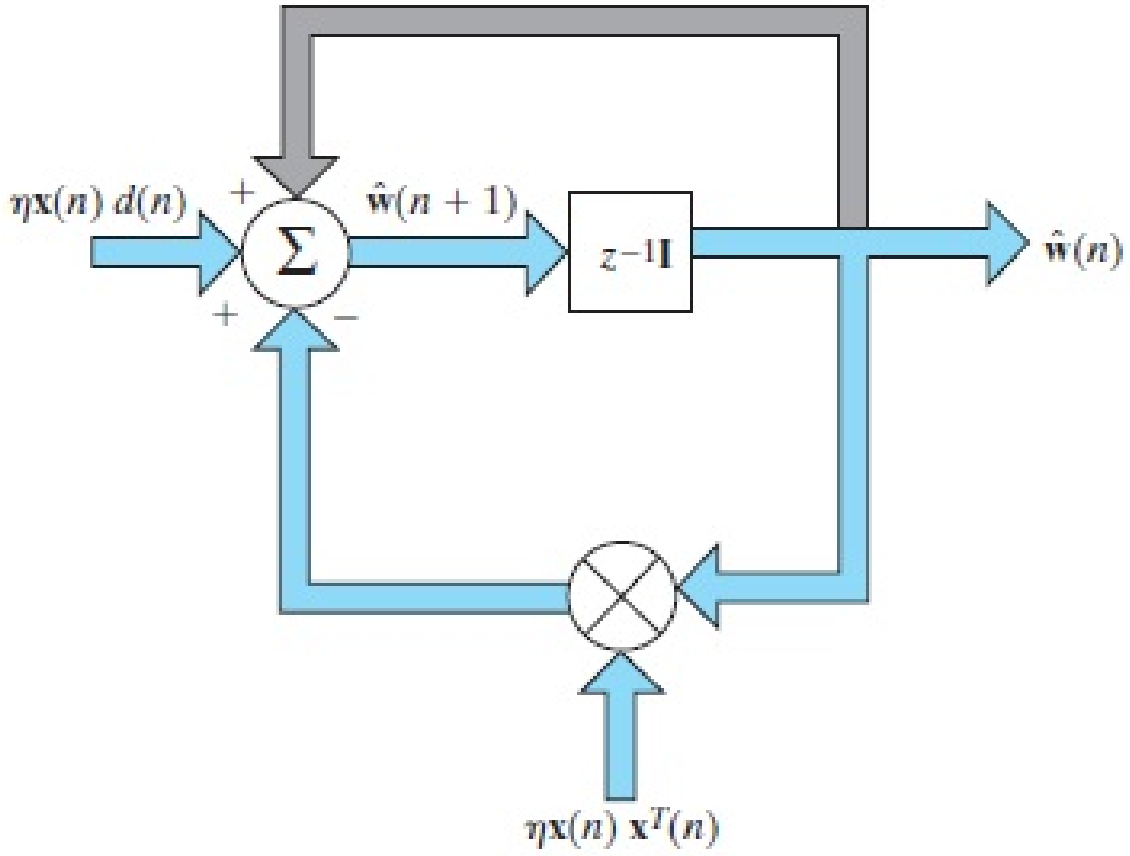
- Here we use $\hat{\mathbf{w}}(n)$ in place of $\mathbf{w}(n)$ to emphasize that LMS algorithm produces an estimate of weight vector that would result from the use of steepest descent
- In steepest descent algorithm the weight vector follows a well-defined trajectory in weight space for a prescribed η
- In LMS algorithm the weight vector $\hat{\mathbf{w}}(n)$ traces a random trajectory, that is why it is also known as **stochastic gradient algorithm**
- As $n \to \infty$ $\hat{\mathbf{w}}(n)$ performs a [2]random walk about the Wiener solution $\mathbf{w}_o$
- Unlike the method of steepest descent, the LMS algorithm does not require knowledge of the statistics of the environment

[2]The movements of an object or changes in a variable that follow no discernible pattern or trend. A random walk is a mathematical formalization of a path that consists of a succession of random steps.

Least-Mean-Square (LMS) Algorithm

Table 1: Summary of LMS Algorithm

Training Sample	Input signal vector= $\mathbf{x}(n)$
	Desired response=$d(n)$
User-selected parameter	η
Initialization	Set $\hat{\mathbf{w}}(0) = 0$
Computation	For $n = 1, 2, \ldots$ compute
	$e(n) = d(n) - \hat{\mathbf{w}}^T(n)\mathbf{x}(n)$
	$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)e(n)$



Convergence of LMS

- Convergence in the mean square i.e

$$\mathbb{E}[e^2(n)] \rightarrow \text{constant as } n \rightarrow \infty$$

- Following assumptions are made for convergence analysis of LMS
  - Successive input vectors $\mathbf{x}(1), \mathbf{x}(2), \ldots$ are statistically independent of each other
  - At time step n, the input vector $\mathbf{x}(n)$ is statistically independent of all previous samples of the desired response, namely $d(1), d(2), \ldots, d(n-1)$
  - At time step n $d(n)$ is dependent on $\mathbf{x}(n)$, but statistically independent of all previous values of the desired response
  - $\mathbf{x}(n)$ and $d(n)$ are drawn from Gaussian distributed populations

- As per the independence theory and assuming that η is sufficiently small, the LMS is convergent in the mean square provided that η satisfies the condition

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

where λ_{max} is the largest eigen value of correlation matrix R_x
- In many application knowledge of λ_{max} is not available, in that case we consider [3]trace of R_x, hence now condition becomes

$$0 < \eta < \frac{2}{tr[R_x]}$$

- Since each diagonal element of R_x equals the mean-square value of the corresponding sensor input, we can also say that

$$0 < \eta < \frac{2}{\text{sum of mean-square values of the sensor inputs}}$$

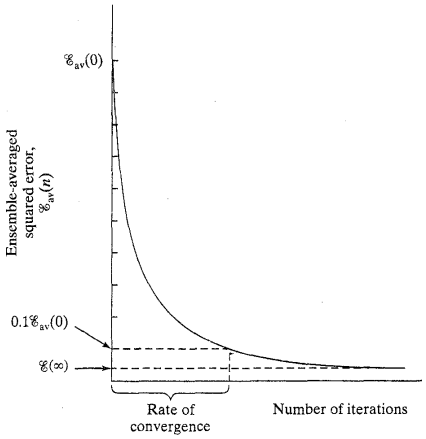
[3]Matrix trace:sum of diagonal elements

Virtues and Limitations of LMS

- LMS algorithm is simple, model-independent and thus robust
- Slow convergence is an issue
- LMS is sensitive to the input correlation matrix's [4]condition number or eigenvalue spread
- Condition number of R_x denoted by $\chi(R_x)$ is defined as $\chi(R_x) = \frac{\lambda_{max}}{\lambda_{min}}$
- A problem with a low condition number is said to be **well-conditioned**, while a problem with a high condition number is said to be **ill-conditioned**

[4]In the field of numerical analysis, the condition number of a function with respect to an argument measures how much the output value of the function can change for a small change in the input argument. This is used to measure how sensitive a function is to changes or errors in the input, and how much error in the output results from an error in the input

Learning Curves



- Learning curve is a plot of mean-square value of the estimation error, $\mathcal{E}_{av}$, vs the number of iterations
- Learning curve is used to examine the convergence behavior of LMS or an adaptive filter
- Rate of convergence of the adaptive filter is defined as the number of iterations n, required for $\mathcal{E}_{av}$ to decrease to some arbitrary chosen value such as 10% of initial value $\mathcal{E}_{av}(0)$

- Let $\mathcal{E}_{min}$ is the minimum mean-square error produced by the Wiener filter
- **Misadjustment** $\mathcal{M}$ is a dimensionless quantity, provides a measure of how close the adaptive filter is to optimality in the mean-square error sense, and defined as

$$\mathcal{M} = \frac{\mathcal{E}(\infty) - \mathcal{E}_{min}}{\mathcal{E}_{min}} = \frac{\mathcal{E}(\infty)}{\mathcal{E}_{min}} - 1$$

- Smaller $\mathcal{M}$ is compared to unity, the more accurate is the adaptive filtering action of the algorithm
- Expressed as percentage. For example, a misadjustment of 10% means that the adaptive filter produces a mean-square error that is 10% greater than the minimum mean-square error $\mathcal{E}_{min}$ produced by the corresponding Wiener filter (considered to be satisfactory)
- Settling time-average time constant τ_{avg} as a rough measure of settling time
- Smaller the value of τ_{avg}, the faster the settling time (i.e.,the faster the LMS algorithm will converge to steady state)

- To a good degree of approximation
  - $\mathcal{M} \propto \eta$
  - $\tau_{avg} \propto \frac{1}{\eta}$
- We have conflicting results, careful attention must be given to the choice of η in the design of LMS algorithm to produce a satisfactory overall performance

Learning-Rate Annealing Schedules

- The main problem arises because of the fixed η
- One solution:use a time-varying learning rate, $\eta(n) = c/n$ as in stochastic optimization theory, where c is a constant
- A better alternative: use a hybrid method called search-then-converge

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

where η_0 and τ are user-selected constants
  - When $n < \tau$, performance is similar to standard LMS
  - When $n > \tau$, it behaves like stochastic optimization

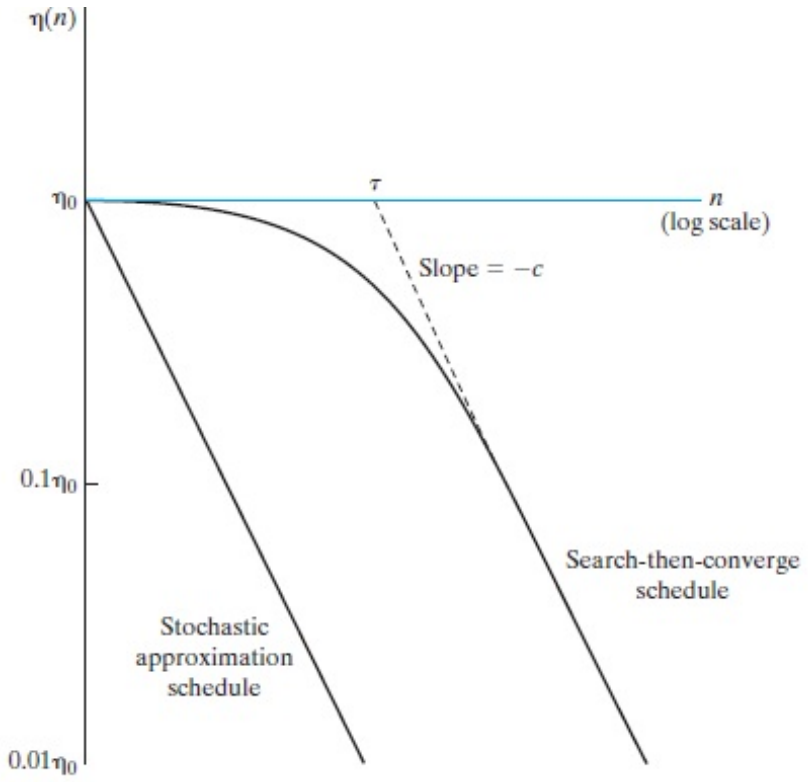


Figure 4: $\eta(n) = \frac{\eta_0}{n}$ vs. $\eta(n) = \frac{\eta_0}{1 + (n/\tau)}$