Unit 1- Models of Neuron

Neuron - An information processing unit that is a fundamental component in a neural network.

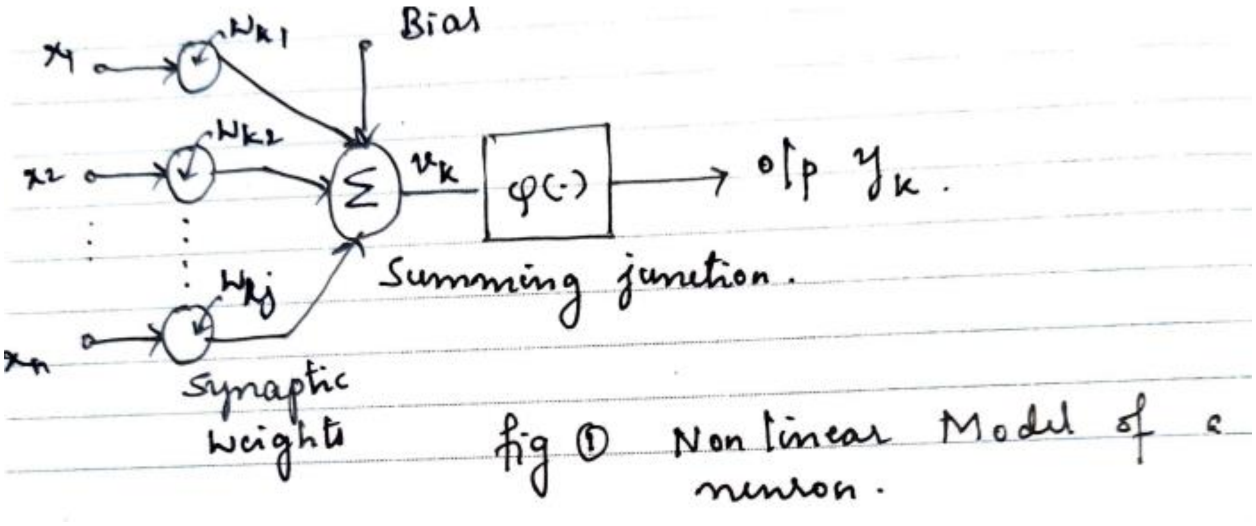


fig ① Non linear Model of a neuron.

The above block diagram shows the model of a neuron with 3 essential elements.

1. A set of synapses or connecting links:

Each is characterized by weight or strength of its own. Signal x_j at the input of synapse j connected to neuron k is multiplied by synaptic weight w_{kj}.

2. Adder: Summing the input signals, weighted by the respective synapses of the neuron. The operations described here contribute a linear combiner.
3. An activation function for limiting the amplitude of the output of neuron. It is also called a squashing function.

Typically, the normalized amplitude range of the output of a neuron is written as the closed unit interval **[0,1]** or alternatively as **[-1,1]**.

$\mathbf{b_k}$ → externally applied bias has the effect of increasing or lowering the net input of activation function depending on whether it is positive or negative.

Mathematically,

Neuron k defines as follows:

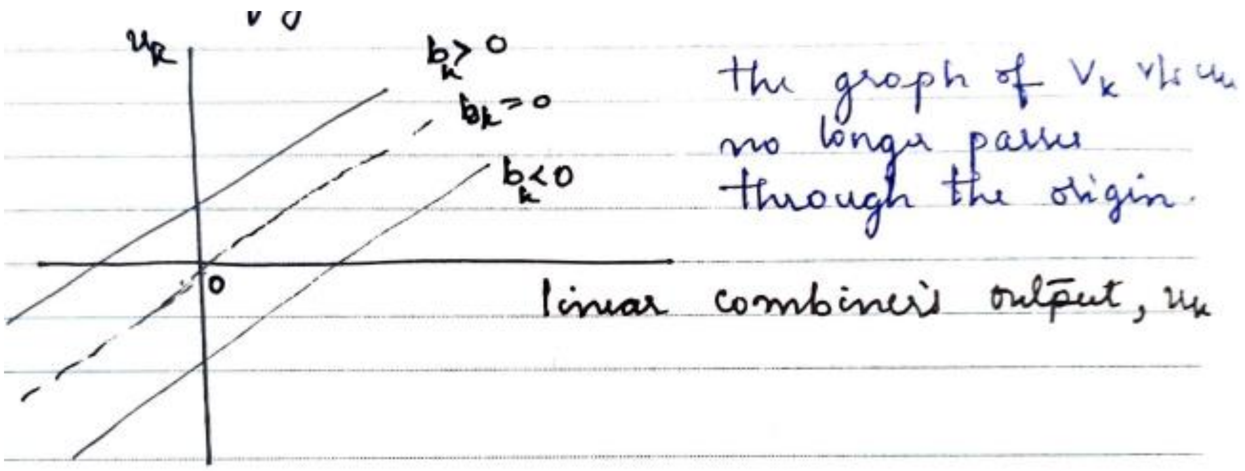
$u_k = \sum_{j=1}^{m} w_{kj} x_j$ (1)

$v_k = u_k + b_k$ (2)

$y_k = \Psi(u_k + b_k) = \Psi(v_k)$ (3)

The use of b_k has the effect of applying and transforming to the output v_k of the linear combiner in the model corresponding to equation (2).

Depending on whether b_k is positive or negative the relationship between the individual local field or activation potential v_k of neuron k and the linear combiner output u_k is modified as shown in the figure below.

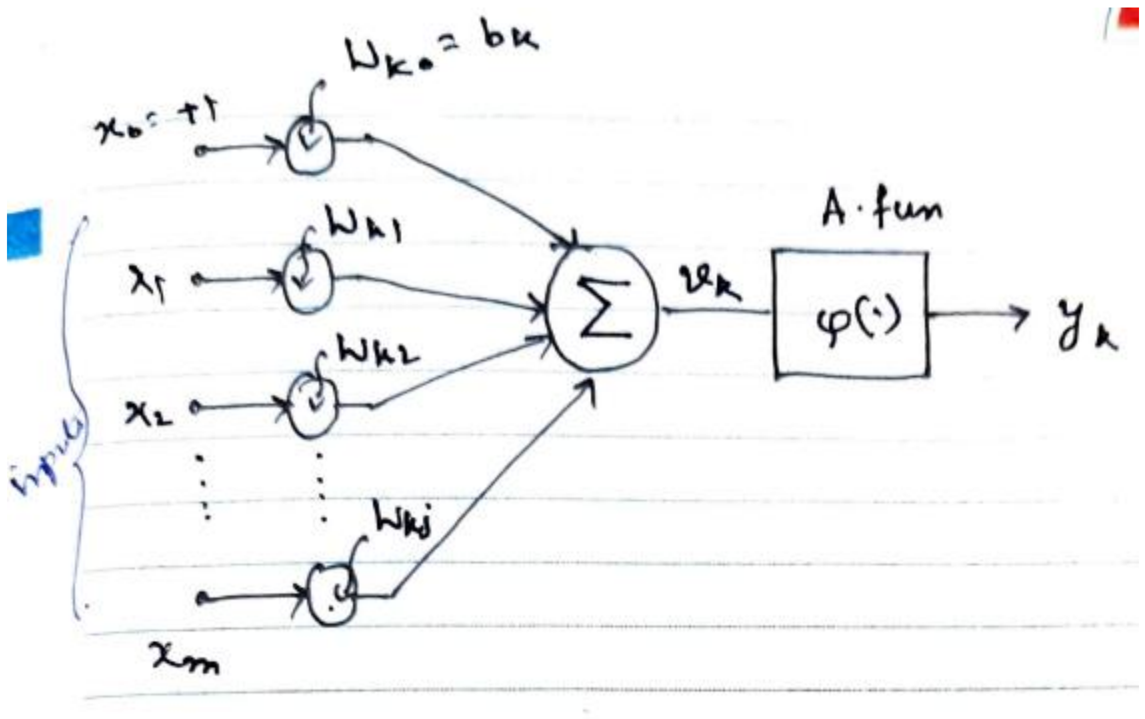

u_k
$b_k > 0$
$b_k = 0$
$b_k < 0$
0
the graph of V_k vs u_k no longer passes through the origin.
linear combiner's output, u_k

The graph of v_k vs u_k no longer passes through the origin.

$v_k = \sum_{j=0}^{m} w_{kj} x_j$

$y_k = \Psi(v_k)$

As we had a new synapse bk (wk0 = bk), its input is x0 = 1
After reformulating the model of neuron k as shown in figure b.



Types of activation function:

The activation function pi(v) defines the output of a neuron in terms of the induced local field u.
There are 3 different types of activation functions:

1. Threshold function: (McCulloch - Pitts model)

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

   It is also called the Heaviside function, correspondingly the output of neuron k

$$y_k = \begin{cases} 1 & \text{if } v_k \geq 0 \\ 0 & v_k < 0 \end{cases}$$

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k .$$

2. Piecewise - linear function:

$$\varphi(v) = \begin{cases} 1 & v \geq +\frac{1}{2} \\ v + \frac{1}{2} & \frac{1}{2} > v > -\frac{1}{2} \\ 0 & v \leq -\frac{1}{2} . \end{cases}$$

Where the amplification factor inside the linear region of operation is assumed to be unity.
This can be viewed as an approximation to a non-linear amplifier.
The following 2 situations may be viewed as special forms of the piecewise-linear function.
I. A linear combiner arises if the linear region of operation is maintained without running into saturation.
II. The piecewise - linear function reduces to a threshold function if the amplification factor of the linear region is made infinitely large.

3. Sigmoid Function: whose graph is s-shaped. It is defined as a strictly increasing function that exhibits a balance between linear and nonlinear behavior.

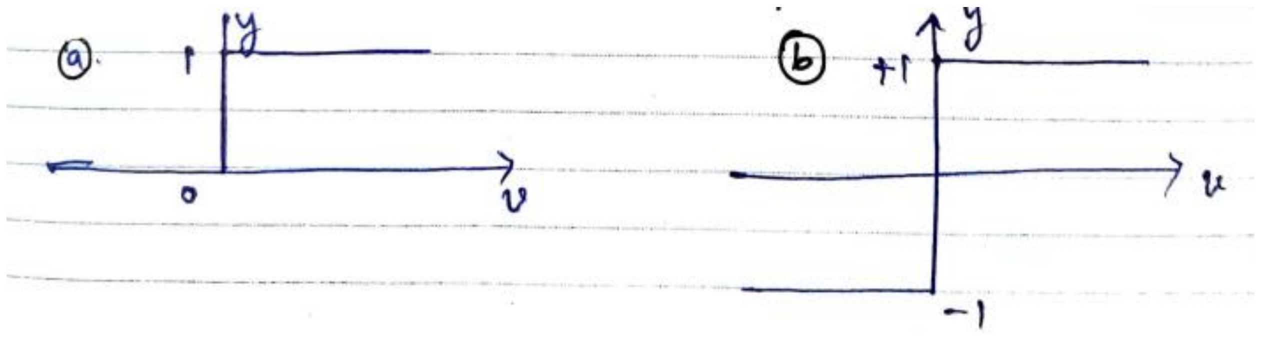
$$\varphi(v) = \frac{1}{1 + \exp(-av)} .$$

Where a is the slope parameter.
In fact, the slope at the origin is ………
If a → ∞, sigmoid function is equal to threshold function. Whereas **threshold function** assumes the value of 0 or 1, **sigmoid function** assumes a continuous range of values from 0 to 1.

Differentiability is an important step in neural network theory.

1. Sigmoid function is differentiable.
2. Threshold function is **not** differentiable.

a. Hard Limit / Threshold Function:



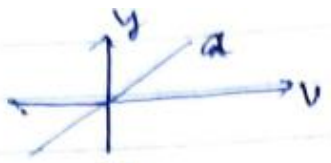
$$\varphi(v) = \begin{cases} 1 & v \geq 0 \\ -1 & v < 0 \end{cases}$$

or

$$\varphi(v) = \begin{cases} 1 & v > 0 \\ -1 & v \leq 0 \end{cases}$$

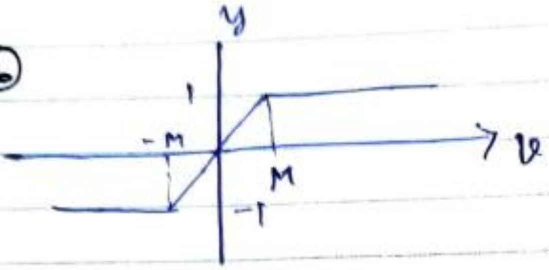
b. Piecewise linear function:

(a)



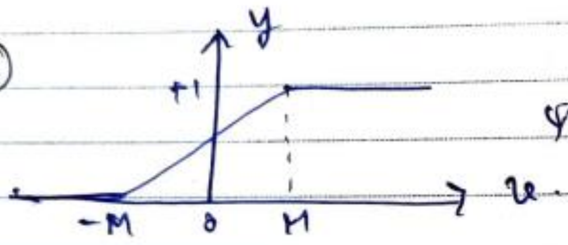
$$\varphi(v) = y = av$$

(b)



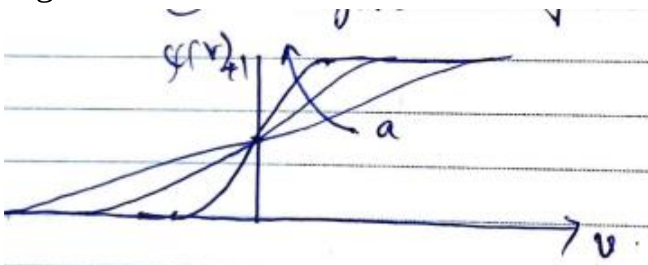
$$\varphi(v) = y = \begin{cases} -1 & v \le -M \\ x/M + 1 & -M < v < M \\ 1 & v \ge M \end{cases}$$

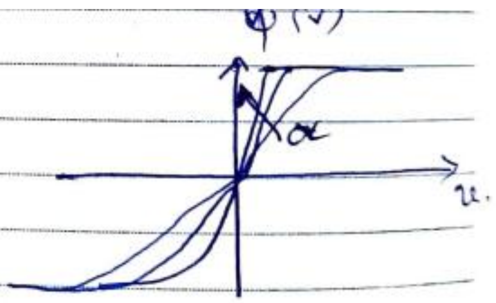
(c)



$$\varphi(v) = y = \begin{cases} 0 & v \le -M \\ x/2M & -M < v < M \\ 1 & v \ge M \end{cases}$$

c. Sigmoid Function:





logistic function

$$\frac{dy}{dv} = \frac{\alpha e^{-\alpha v}}{(1 + e^{-\alpha v})^2} = \alpha e^{-\alpha v} \cdot y^2$$

$$\frac{dy}{dv} = \alpha y(1 - y).$$

$$\frac{dy}{dv} = \alpha \left(\frac{1}{y} - 1\right) y^2$$

$$\frac{dy}{dv} = \alpha y - \alpha y^2$$

Other Activation Functions:

The amplitude of the above 3 types of activation function ranges from 0 to 1.
It is sometimes desirable to have the activation function range from -1 to 1.

Stochastic Model of a neuron:

For some applications of neural networks, it is desirable to base the analysis on a stochastic neuron model. In an analytically traceable approach, the activation function is given a probabilistic interpretation.
Specifically, a neuron is permitted to reside in only one of the two states +1 or -1.
The decision for a neuron to fire is probabilistic.

Let x - state of the neuron
p(v) - probability of firing
v - induced local field of a neuron
x is +1 with probability p(v) and -1 with probability 1-p(v)

The standard choice for p(v) is the sigmoid-shaped function.

$$P(v) = \frac{1}{1 + \exp(-v/T)}$$

T: pseudo temperature that is used to control the noise level and therefore the uncertainty in firing.

4. Gaussian Function
5. Squashing Function

Neural networks defined as Directed Graphs:

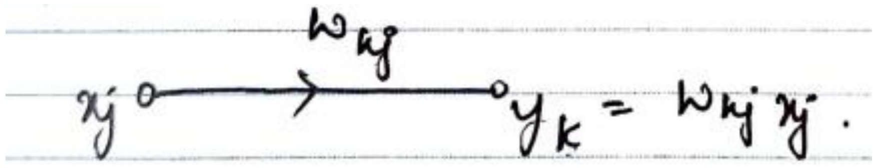
Signal flow graphs with a well-defined set of rules were originally developed by Mason for linear networks.
The presence of nonlinearity in the model of a neuron limits the scope of their application to neuron networks.
A signal flow graph is a network of directed links that are interconnected at certain points called nodes. A typical node j has an associated node signal x_j. A typical directed link originates at node j and terminates n node k.
It has an associated transfer function that specifies the manner in which the signal y_k at node k depends on the signal x_j at node j.

Three basic rules:

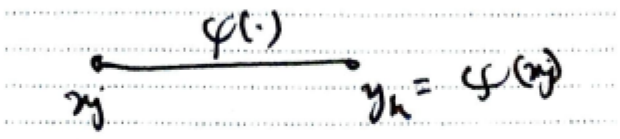
Rule 1: A signal flows along a link only n the direction defined by the arrow on the link.

Types of links:

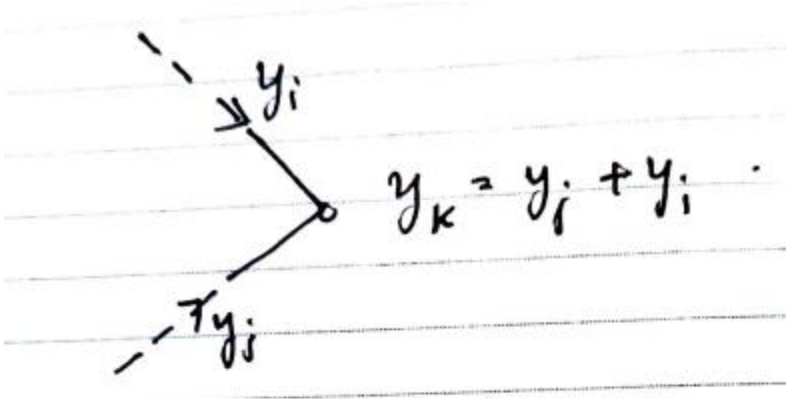
1. Synaptic link: Whose behaviour is governed by a linear input-output relation



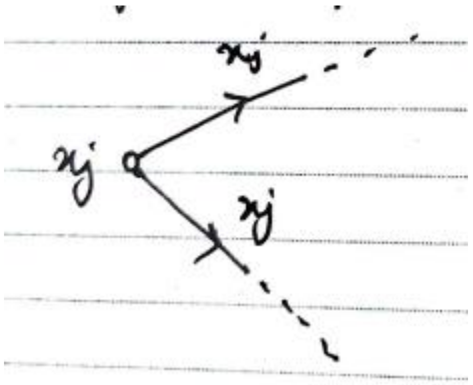
2. Activation links: Whose behaviour is governed by a nonlinear input-output relation.

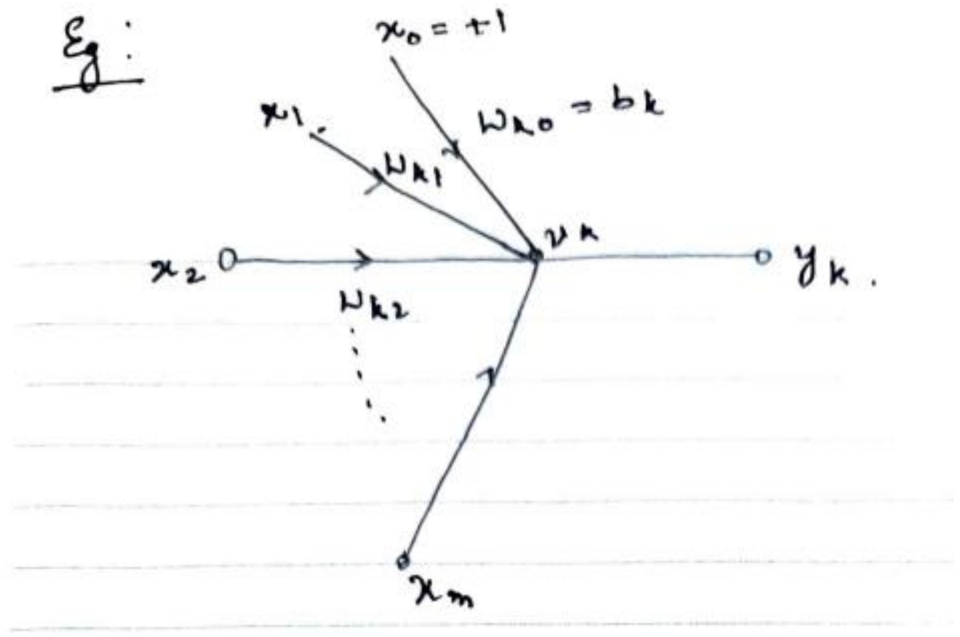


Rule 2: A node signals equals the algebraic sum of all signals entering the pertinent node via the incoming links.



Rule 3: The signal at a node is transmitted to each outgoing link originating from that node with the transmission being entirely independent of transfer function of outgoing links.





Mathematical Definition of a neural network:

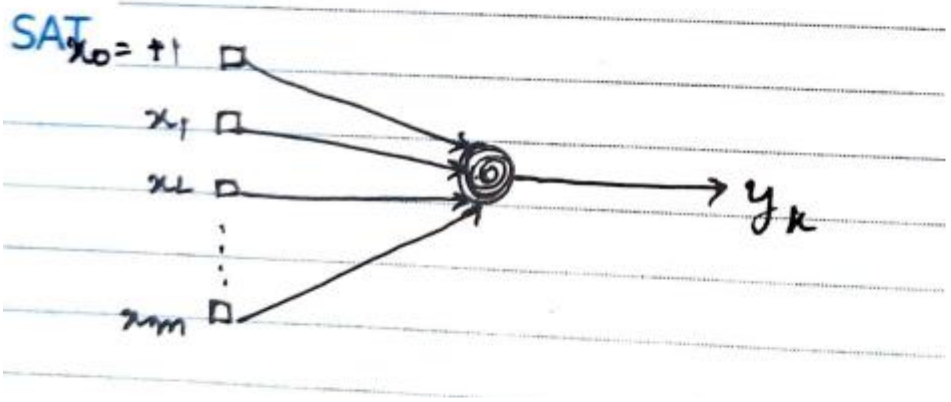
A neural network is a directed graph consisting of nodes with interconnecting synaptic and activation links and is characterized by four properties:

1. Each neuron is represented by a set of linear synaptic links an externally applied bias, and a possibly nonlinear activation link. The bias is represented by a synaptic link connected to an input fixed at +1.
2. Synaptic links of a neuron weight their respective input signals.
3. The weighted sum of the input signals defines the induced local field of the neuron in question
4. The activation link squashes the induced local field of the neuron to produce an output.

Partially Complete:

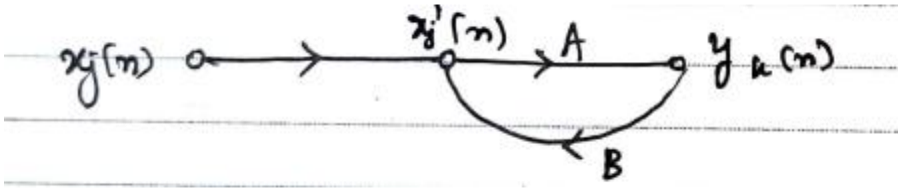
1. Source nodes supply input signals to the graph.
2. Each neuron is represented by a single node called computation node.
3. The communication link interconnecting the source and computation nodes of the graph carry no weight, they merely provide the directions of signal flow in the graph.

A partially complete directed graph defined in the way is referred to as an architectural graph describing the layout of a neural network.



Feedback:

It is said to exist in a dynamic system whenever the output of an element in the system influences in part the input applied to that particular element, thereby giving rise to one or more closed paths of transmission of signals around the system.
It plays a major role in special class of neural networks known as recurrent neural networks.



$y_k(n) = A[x_j'(n)]$

$x_j'(n) = x_j(n) + B[y_k(n)]$

$y_k(n) = A[x_j(n) + B[y_k(n)]]$

$y_k(n)\,[1-AB] = Ax_j(n)$

$y_k(n) = \frac{A}{1-AB}[x_j(n)]$

Where $\frac{A}{1-AB}$ is a closed loop operator of the system and AB is an open loop operator which is non communicative that is AB is not equal to BA.

Example:

$$\begin{tikzcd}
x_j(n) \arrow[r] & x_j'(n) \arrow[r, "w"] & y_k(n) \arrow[l, bend left, "z^{-1}"]
\end{tikzcd}$$

$$\frac{A}{1-AB} = \frac{w}{1-wz^{-1}}$$

$$= w\,(1-wz^{-1})^{-1}$$

4 WED

using binomial expansion for $(1-wz^{-1})^{-1}$

$$\frac{A}{1-AB} = w\sum_{l=0}^{\infty} w^{l} z^{-l}$$

$$y_k(n) = w\sum_{l=0}^{\infty} w^{l} z^{-l}\,[x_j(n)]$$

where z^{-1} is an operator.

$$z^{-1}[x_j(n)] = x_j(n-1).$$

$$\Rightarrow \quad y_k(n) = \sum_{l=0}^{\infty} w^{l+1} x_j(n-l).$$

Two specific cases :

1. $|w| < 1$ for which the output signal $y_k(n)$ is exponentially convergent, i.e., the system is stable.
2. $|w| \geq 1$ for which the output signal $y_k(n)$ is diverent, i.e., the system is unstable.

   $|w| = 1$, the divergence is linear
   $|w| > 1$, the divergence is exponential

Network Architecture:

Different classes of Network architecture

1. **<u>Single – layer feedforward Networks:</u>**
   - In a layered neural network the neurons are organised in the form of layers.
   - It consists of input layer of source node that projects onto an output layer of neurons (computation nodes) but not **vice versa**.
   - This network is strictly a **feedforward** or an **acyclic type**.
   - It is illustrated in **fig.** for the case of 4 nodes in both input and output layers: Single Layer Network.
   - Single layer referring to the output layer of computation nodes(neuron).
   - We do not count the input layer of source nodes because no computation is performed there.

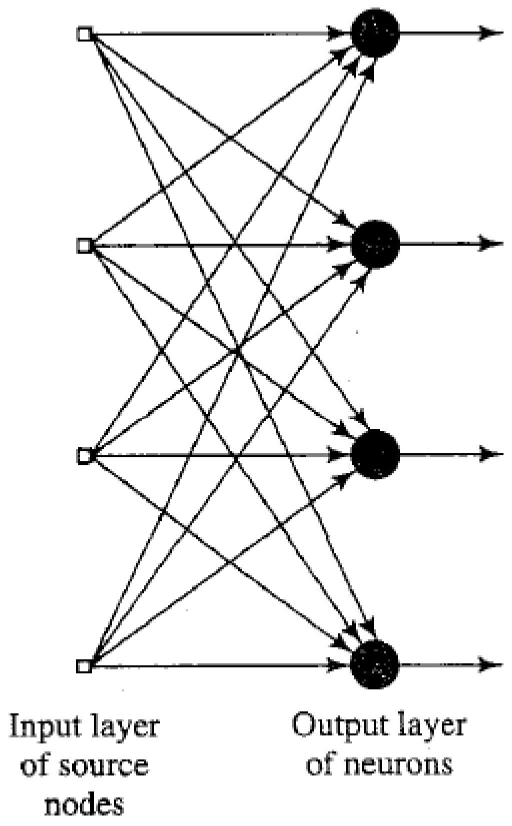


Fig.

2. **<u>Multilayer feedforward Networks:</u>**
   - The second class of a feedforward neural network distinguishes itself by the presence of one or more hidden layers whose computation nodes are corresponding called hidden neurons or hidden units.

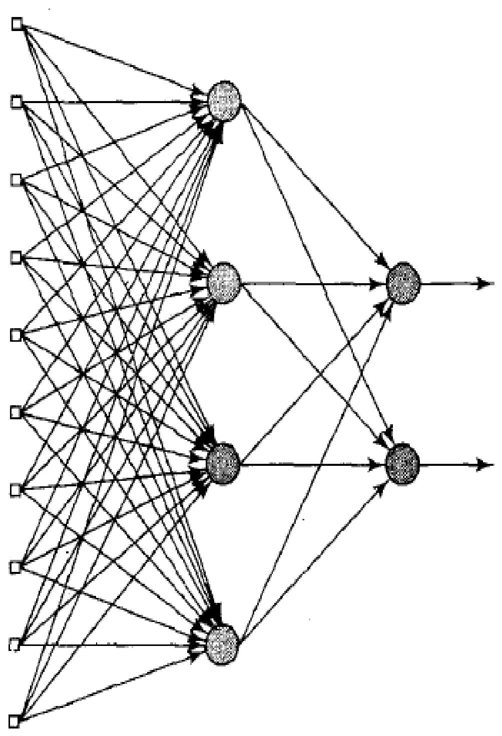


Fig.

3. **Recurrent Networks:**

- It distinguishes itself from a feedforward neural network in that it has at least one feedback loop.
- The presence of feedback loops whether in the recurrent structure with hidden or without hidden layers has a profound impact on the learning capability of the network and on its performance.
- The feedback loop involves the use of particular branches composed of unit delay elements; which result in a non-linear dynamical behaviour, assuming that the neural network contains NL units.

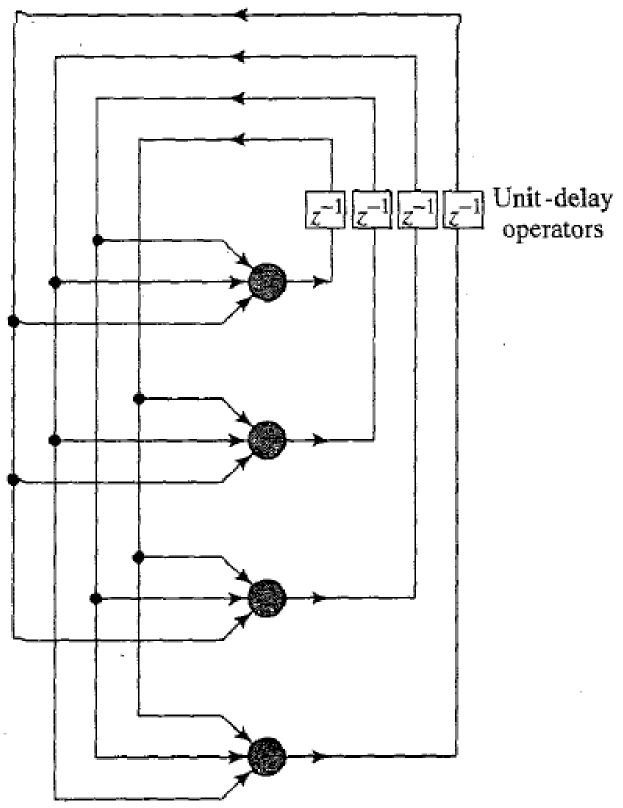
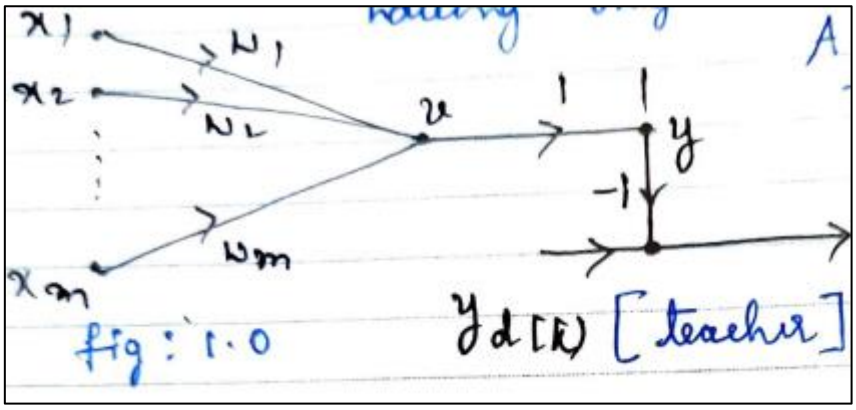


Fig.

ADALINE: Adaptive Linear Neuron:

It is a simple 2 layer network with only input layer and output layer, having only output neuron, All neurons in the network are linear and network uses LMS algorithm to learn.



$$v = \sum_{i=1}^{m} w_i x_i = W^T X$$

y =v as it is a linear neuron.

Function Approximation:

Let $\{X(k), y_d(k)\}_{k=1}$, $X(k) \in \mathbb{R}^m$, $y_d(k) \in \mathbb{R}$.

$$y_d(k) = \sin\big(X(k)\big)$$

$$e(k) = y_d(k) - y(k)$$

$$= \; y_d(k) - W^T X$$

$$\big(e(1)e(2)\ldots..e(N)\big) = (\underbrace{y_d(-1), y_d(2), \ldots.., y_d(N)}_{Y_d} - \; W^T(\underbrace{X(1)X(2)\ldots.X(N)}_{X})$$

$$\boldsymbol{E_{1XN} = \; Y_{d\,1XN} \; - W^T X_{1XM}}$$

- It was developed by Prof. Bernard Widrow.
- The difference between the ADALINE and standard perceptron is that in the learning phase, weights are adjusted according to the weighted sum of the inputs.
- In standard perceptron, the net is passed to the activation function and the function and the function's output is used for adjusting weights.
- Multilayer Network of ADALINE: **MADALINE**
- Application: Pattern Recognition, Data Filtering or to approximate linear function.

Note that this network can be applied only to linear problems.

The structure of ADALINE is shown in fig 1.0.

Linear Least Squares Filter:

Adaptive Filter:

Consider again, $\qquad E \;=\; Y_d - Y \;=\; Y_d - \; W^T X$

By setting $E \;=0$,

$$Y_d \;=\; W^T X$$

Take transpose on both sides,

$$X^T W \;=\; Y_d^T$$

Multiply X on both sides,

$$X X^T W \;=\; \; X Y_d^T$$

X is fat matrix of order mXN where $N > m$. So, to find W, we need X^T inverse. Hence, pre-multiplied by X on both sides. Hence, XX^T becomes square matrix which has order mXm. $[XX^T]_{mXm}$ and its $\rho(XX^T) = m$.

If rank of XX^T is m the XX^T is non singular matrix, their exists inverse.

$$W = (XX^T)^{-1} XY_d^T$$

$$W = X(XX^T)^{-1} \; Y_d^T$$

$$\boldsymbol{W = X^+ Y_d^T}$$

Here, X^+ is known as the **Moore penrose pseudo inverse.**

Moore penrose pseudoinverse is to compute a best fit (least square) solution to a system of linear equation that lacks a unique solution. It can be computed using SVD.

When A has linearly independent rows, A^+can be computed as,

$$A^+ = A^*(AA^*)^{-1} \text{ (Right Inverse)}$$

When A has linearly independent column,

$$A^+ = (A^*A)^{-1}A^* \text{ (Left Inverse)}$$

$(A^* = A^T)$

<u>Performance Index:</u>

$$J = \frac{1}{2}\sum_{i=1}^{m} e^2(i)$$

$$= \frac{1}{2} \begin{bmatrix} e(1) & e(2) & \dots & e(m) \end{bmatrix} \begin{bmatrix} e(1) \\ e(2) \\ \vdots \\ e(m) \end{bmatrix}$$

If energy is deterministic, $J = Y_2 E E^T$

Aim: To minimise the error.

$$J = Y_2(Y_d - W^TX)(Y_d - W^TX)^T$$

$$= Y_2[Y_dY_d^T - W^TXY_d^T - Y_dX^TW + W^TXX^TW^T]$$

$$J = Y_2[Y_dY_d^T - 2W^TXY_d^T + W^TXX^TW^T]$$

For error to be minimum,

$$\nabla_{\mathrm{W}}J = 0$$

$$\nabla J = Y_2[-2Y_dX^T + W^TXX^T + XX^TW]$$

$$\nabla J = Y_2[-2Y_dX^T + 2XX^TW] = 0$$

$$XX^TW = Y_dX^T$$

$$\boldsymbol{W} = (\boldsymbol{XX^T})^{-1}\boldsymbol{XY_d^T}$$

This is the Linear Least Square solution. If XX^T is singular, i.e., rank of

$$XX^T \neq m$$

If it is $< m$ the customary practice is to add diagonal matrix δI.

Where I is the identity matrix.

Using performance index:

$$J = Y_2EE^T + \frac{\delta}{2}\,||W||^2$$

$$J = Y_2EE^T + \frac{\delta}{2}\,W^TW$$

$$J = Y_2[Y_dY_d^T - 2W^TXY_d^T + W^TXX^TW^T] + \frac{\delta}{2}\,W^TW$$

$$\nabla_{\mathrm{W}}J = -XY_d^T + XX^TW + \delta W$$

$$\nabla_{\mathrm{W}}J = 0 \text{ (To minimise error)}$$

$$(XX^T + \delta)W = XY_d^T$$

$$E = [e(1) \quad e(2) \quad \cdots \quad e(N)]$$

$$J_E = (\nabla_{\mathrm{W}}e(1)\ \ \nabla_{\mathrm{W}}e(2) \cdots \nabla_{\mathrm{W}}e(N))^T$$

We have,

$$E = Y_d - W^TX$$

$$\nabla_{\mathrm{W}} E = -X^T$$

$$\therefore\ J_E = -(X(1)\ X(2)\ \cdots X(N))^T$$

$$J_E = -X^T$$

We have,

$$W = (XX^T)^{-1} X Y_d^T$$

$$\boxed{\therefore \boldsymbol{W} = (\boldsymbol{J_E^T J_E})^{-1} \boldsymbol{J_E^T Y_d^T}}$$

This is the Jacobian Form of the Linear Least Square solution.

That is why we calculate the ∇ of Cost Function at this point $W = W(n)$ at the end of n observations.

When we operate around $W = W(n)$, we can express it as e' which is the updated error.

<u>Wiener Filter:</u>

It is the limiting form of the Linear Least Squares Filter for an Ergodic environment.

Till now the data was deterministic, hence, we used energy,

$$J = Y_2 \sum_{i=1}^{N} e^2(i)$$

If the data is from a stochastic process, then error signal also becomes stochastic. For this kind we use mean square error.

$$\xi = E[e^2(k)]$$

$$e^2(k) = \{\underset{\text{(1x1)}}{y_d(k)} - \underset{\text{(1xm)}}{W^T}\underset{\text{(mx1)}}{X(k)}\}^2$$

$$= y_d^2(k) + \{W^T X(k)\}^2 - 2y_d(k)x(k)W^T(k)$$

Take Expectation on both sides,

$$E\{e^2(k)\} = E\{y_d^2(k)\} + E\{(W^TX(k))^2W\} - 2W^TE\{Y_dX\}$$

$$\xi = r_d(0) + W^TRW - 2W^Tr_{dx}(k)$$

Where r_d = autocorrelation of Y_d.

R = autocorrelation of $X(k)$.

r_{dx} = Cross Correlation of X, Y_d.

$$\nabla_W\xi = 0 + 2RW - 2r_{dx}(k) = 0$$

$$RW = r_{dx}$$

$$\boxed{W = R^{-1}r_{dx}}$$

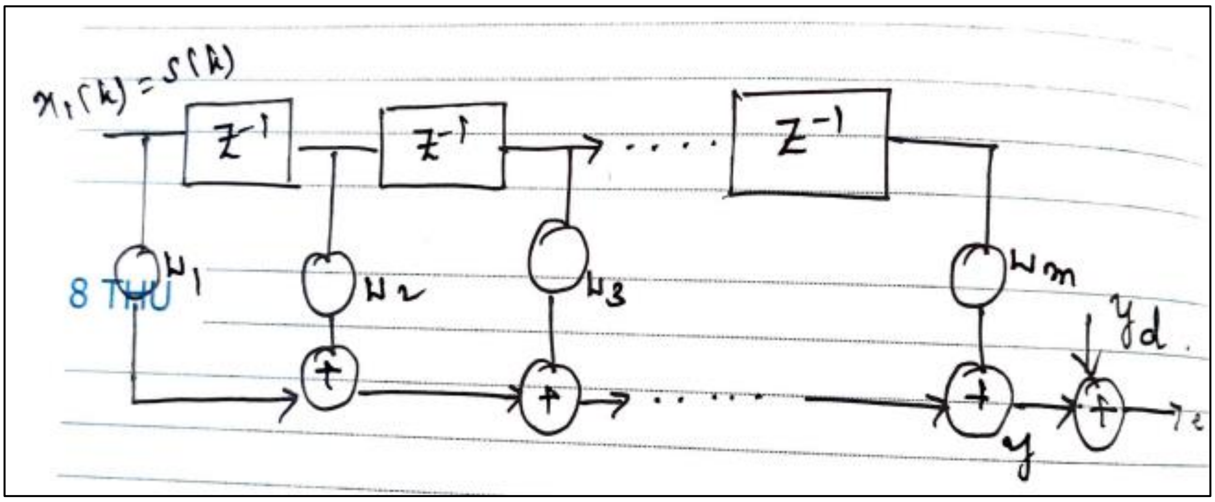
If $|R| \neq 0$ (non singular)

This is referred as Wiener solution to the optimum filtering problem.

For an ergodic process, the Linear Least Squares filters asymptotically approaches the Wiener filter as the number of observations approaches infinity.

Let us assume that $x_1(k) = s(k), x_2(k) = s(k-1), x_3(k) = s(k-3), \dots\dots$

$x_m(k) = s(k-m+1)$



$$r_{dx}(k) = E[Y_d(k)x(k)]$$

$$= E[\underset{\substack{\uparrow \\ (1\text{x}N)}}{y_d}(k)\begin{bmatrix} s(k) \\ \vdots \\ s(k-m+1) \end{bmatrix}]$$

$$= \begin{bmatrix} r_{ds}(0) \\ r_{ds}(1) \\ \vdots \\ r_{ds}(km-1) \end{bmatrix}$$

$$R_X = E[\begin{bmatrix} s(k) \\ s(k-1) \\ \vdots \\ s(k-m+1) \end{bmatrix}[s(k) \quad \cdots \quad s(k-m+1)]$$

$$= E\begin{bmatrix} s^2(k) & s(k)\,s(k-1) & \dots & s(k)\,s(k-m+1) \\ s(k-1)\;\; s(k) & s(k-1)\,s(k-1) & \dots & s(k-1)\;\; s(k-m+1) \\ \vdots & \vdots & \vdots & \vdots \\ s(k-m+1) & \dots & \dots & s(k-m+1)\,s(k-m+1) \end{bmatrix}$$

$$R_X = \begin{bmatrix} r_s(0) & r_s(1) & \dots & r_s(m-1) \\ r_s(-1) & r_s(0) & \dots & r_s(m-2) \\ \vdots & \vdots & \vdots & \vdots \\ r_s(1-m) & r_s(2-m) & \dots & r_s(0) \end{bmatrix}$$

Property of auto correlation is $r_s(-k) = r_s(k)$.

$$R_X = \begin{bmatrix} r_s(0) & r_s(1) & \dots & r_s(m-1) \\ r_s(1) & r_s(0) & \dots & r_s(m-2) \\ \vdots & \vdots & \vdots & \vdots \\ r_s(m-1) & r_s(m-2) & \dots & r_s(0) \end{bmatrix}$$

$$W = R_X^{-1} r_{dx}$$

$$\boxed{\boldsymbol{W_{op} = R_X^{-1} r_{dx}}}$$

This kind of filter is called a **Weiner Filter**.

It’s an optimal filter which reduces the mean square error.

- For an ergodic process, the linear least-squares filter asymptotically approaches the Wiener filter as the number of observations approaches infinity.
- Proof:

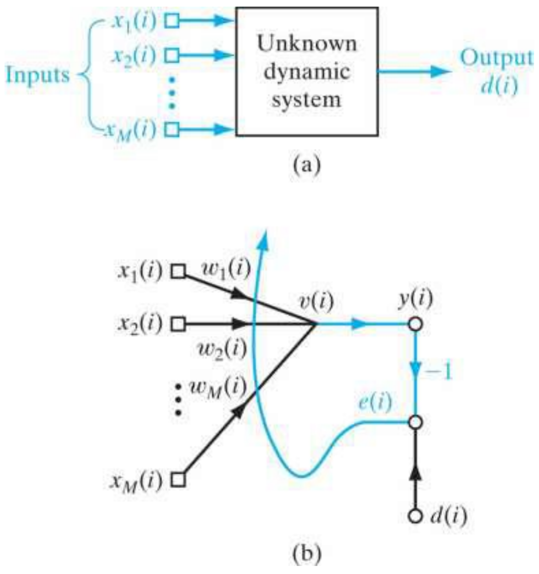
$$
\begin{aligned}
w_0 &= \lim_{n\to\infty} w(n+1) \\
&= \lim_{n\to\infty} (X^T(n)X(n))^{-1} X^T(n)d(n) \\
&= \lim_{n\to\infty} \frac{1}{n}(X^T(n)X(n))^{-1} \lim_{n\to\infty} \frac{1}{n} X^T(n)d(n) \\
&= R_x^{-1} r_{dx}
\end{aligned}
$$

Adaptive Filtering System

Consider dynamical system whose characterization is unknown.

Only available data is input and output data.

When m dim input stimulus X(i) is applied across m input nodes of the system, s/m produces an scalar output d(i)



Unknown dynamic system described by:

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

Where,

$$x(i) = [x_1(i), x_2(i), \dots, x_m(i)]^T$$

m is the input dimensionality.

The stimulus vector x(i) can arise in either way of the following:

• Spatial: The m input elements of x originate at different point in space, Snapshot data

- Temporal data, x(i) is uniformly spaced in time, The m input elements of x represent the set of present and (m-1) past values of some excitation.

"Design a multi-input-single-output model of the unknown dynamical system by building it around a single linear neuron"

Adaptive filtering:

- Start from an arbitrary setting of the adjustable weights.
- Adjustment of weights are made on continuous basis.
- Computation of adjustments to the weight are completed inside one interval that is one sampling period long.

Adaptive filter consists of two continuous processes:

- Filtering process: computation of the output signal y(i) and the error signal e(i).
- Adaptive process: The automatic adjustment of the weights

according to the error signal.

Since the neuron is linear,

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

In matrix form

$$y(i) = X^T(i)W(i)$$

Where, $W(i) = [w_1(i), w_2(i),\ldots, w_m(i)]$

Error, e(i) = d(i) – y(i)

The method of adjusting the synaptic weight with the help of error signal is determined by the cost function, which is related to the optimization problem.

LINEAR NEURON

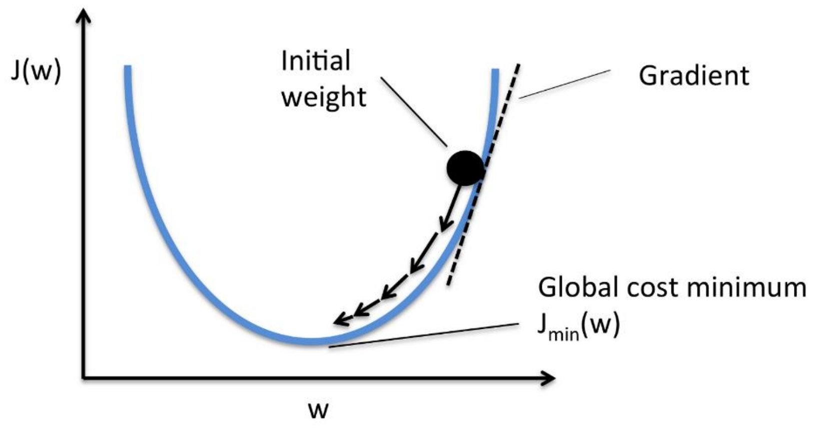
This is the measure of how to choose the w of adaptive filtering algorithm so that it behaves in optimum manner.

Unconstrained Optimization

- Consider the cost function E (w) that is continuously differentiable function of unknown weights w.
- Goal: find the optimal solution w* that satisfies: $E(w^*) \leq E(w) \Rightarrow$ Minimize E(w) with respect to w.
- The necessary condition for optimality:$\nabla E(w^*)$ = 0

- $\nabla = [\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \dots \frac{\partial}{\partial w_m}]^T \Rightarrow \nabla E(w) = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots \frac{\partial E}{\partial w_m}\right]^T$
- A class of unconstrained optimization algorithm, based on the idea of local iterative descent:
- Starting with a guess w(0), generate a sequence of weight vectors w(1), w(2), …, such that: E(w) is reduced at each iteration of the algorithm
- Eventually the algorithm converges to optimal solution w*

UNCONSTRAINED OPTIMIZATION TECHNIQUE



<u>Method of Steepest Descent:</u>

It is also referred as gradient descent technique.

It is used in determining minima for the function, in this case it is cost function

The idea is that start with some initial guess, find the gradient

The successive adjustments applied to the weight vector w are in the direction of steepest descent; that is in a direction opposite to the gradient of E(W). (In fig. E(W) is considered as J (w))

If g = ∇ E(w), then the steepest descent algorithm is

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

η is a positive constant called the step size, or learning rate parameter. In each step, the algorithm applies the correction:

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

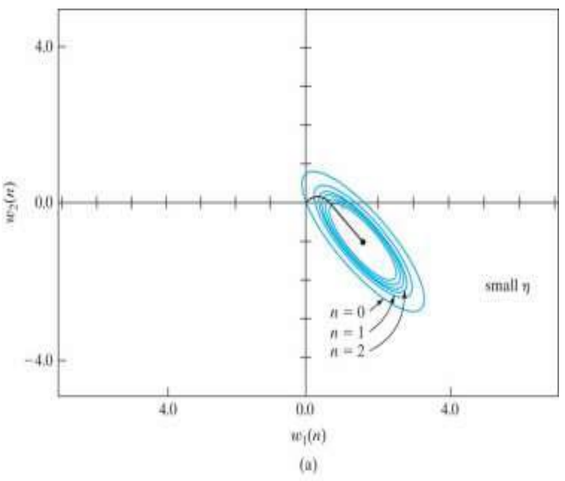
- When η is small, the transient response of the algorithm is overdamped, i.e., w(n) follows smooth path.
- When η is large, the transient response of the algorithm is underdamped, i.e., w(n) follows zig zagging (oscillatory) path.

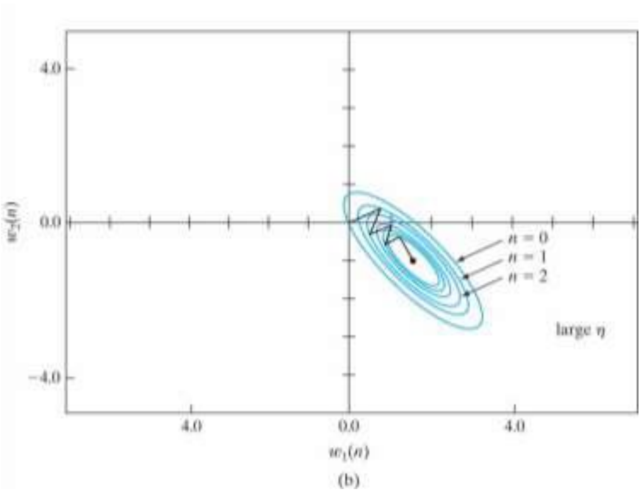
- When η exceeds critical value, the algorithm becomes unstable(diverges).

The SD converges slowly, but eta has significant influence on the convergence.

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.





For example, let us consider X(i) and D(i) are from stochastic process(ergodic), then error signal will also be stochastic.

Cost Function, $\xi \triangleq E[e^2(k)]$,

E - Expectation

In matrix form, $e(k) = D - W^T X$

$$e^2(k) = (D - W^T X)(D - W^T X)^T$$

$$= DD^T - DX^T W - W^T X D^T + W^T X X^T W$$

$$\Rightarrow \xi = E[e^2(k)] = E[DD^T - DX^T W - W^T X D^T + W^T X X^T W]$$

$$= r_d - r_{dx} W - W^T r_{xd} + W^T R_x W$$

Where, $r_d = E[DD^T]$, $R_x = E[XX^T]$

$$r_{dx} = E[DX^T] = r_{xd} = E[XD^T]$$

Let $a, b \in \mathbb{R}^n$, then

$$\frac{\partial(a^T b)}{\partial a} = \frac{\partial(b^T a)}{\partial a} \triangleq \begin{pmatrix} \frac{\partial(b^T a)}{\partial a_1} \\ \vdots \\ \frac{\partial(b^T a)}{\partial a_n} \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = b$$

And Let P = $P^T \in \mathbb{R}^{nxn}$

$$\frac{\partial(a^T P a)}{\partial a} = 2Pa$$

$$\Rightarrow \frac{\partial \xi}{\partial W} = -r_{dx} - r_{xd} + W^T R_x + R_x W$$

$$= -2r_{dx} + 2R_x W$$

WKT, $w(n+1) = w(n) + \Delta w(n)$

$$\Delta w(n) = -\eta \frac{1}{2} \frac{\partial \xi}{\partial W}$$

$$w(n+1) = w(n) - \eta(R_x W - r_{dx})$$

$$= (I - \eta R_x) w(n) + \eta r_{dx}$$

$$w(n+1) = w(n) - \eta(R_x W - r_{dx})$$

$$\boxed{= (I - \eta R_x) w(n) + \eta r_{dx}}$$

With affine transformation and rotation, the above equation is transformed in the form of x(k) = α^kx(0), which is stable and converge to zero if $|\alpha| < 1$

With similar analysis, $|1 - \eta\lambda_i| < 1 \Rightarrow 0 < \eta\lambda_i < 2$

Where λ_i - maximum eigen value of R_x

Final equation represents the state equation of discrete time system.

Newton's Method

- The idea is to minimize the quadratic approximation of the cost function E (w) around the current point w(n).
- This minimization is performed at each iteration, using 2nd order Taylor series expansion of E(w) around the point w(n)
- $E(w(n) + \Delta w(n)) = E\big(w(n)\big) + E'(w(n))\Delta w(n) + \frac{1}{2}\Delta w^T(n)E''\big(w(n)\big)\Delta w(n) + \cdots$

$$f(\mathbf{x}_k + \Delta\mathbf{x}) \approx f(\mathbf{x}_k) + \mathbf{g}_k^T\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T Q_k \Delta\mathbf{x}_k$$

,where

$$\mathbf{g}_k = \mathbf{g}(\mathbf{x}_k) = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k}$$

- Note: Taylor series expansion

Δ w(n) = w(n+1)-w(n)

- Gradient of cost function i.e., diff. w.r.to $\Delta w(n)$
  - $\nabla E(w(n) + \Delta w(n)) = E'\big(w(n)\big) + \frac{1}{2}.2.H\big(w(n)\big)\Delta w(n) = 0$
  - $E'\big(w(n)\big) = g(n)$ =1st derivative

$$\mathbf{H} = \nabla^2\mathscr{E}(\mathbf{w})$$

$$= \begin{bmatrix} \frac{\partial^2\mathscr{E}}{\partial w_1^2} & \frac{\partial^2\mathscr{E}}{\partial w_1\partial w_2} & \cdots & \frac{\partial^2\mathscr{E}}{\partial w_1\partial w_m} \\ \frac{\partial^2\mathscr{E}}{\partial w_2\partial w_1} & \frac{\partial^2\mathscr{E}}{\partial w_2^2} & \cdots & \frac{\partial^2\mathscr{E}}{\partial w_2\partial w_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2\mathscr{E}}{\partial w_m\partial w_1} & \frac{\partial^2\mathscr{E}}{\partial w_m\partial w_2} & \cdots & \frac{\partial^2\mathscr{E}}{\partial w_m^2} \end{bmatrix}$$

- Gradient of cost function i.e., diff. w.r.to $\Delta w(n)$
    - $\nabla E(w(n) + \Delta w(n)) = E'\big(w(n)\big) + \frac{1}{2}.2.H\big(w(n)\big)\Delta w(n) = 0$

$\rightarrow H\big(w(n)\big)\Delta w(n) = -\,E'\big(w(n)\big)$

$\rightarrow \Delta w(n) = \ -H\big(w(n)\big)^{-1}E'\big(w(n)\big)$

$\rightarrow w(n+1) = w(n) - \eta H\big(w(n)\big)^{-1}E'\big(w(n)\big)$

$\rightarrow w(n+1) = w(n) - \eta H\big(w(n)\big)^{-1}E'\big(w(n)\big)$

$\rightarrow$ Where $H^{-1} = inverse\ of\ Hessian\, matrix$

$\rightarrow$ For Newton's method to work well H must be positive definite,

No guarantee that the H is p.d always, Wherever H is not p.d , necessary modification has to be incorporated

Newton's Method Example:

For example, let us consider X(i) and D(i) are from stochastic process(ergodic), then error signal will also be stochastic.

Cost Function, $\xi \triangleq E[e^2(k)]$, E - Expectation

In matrix form, $e(k) = D - W^T X$

$$e^2(k) = \ (D - W^T X)(D - W^T X)^T$$

$$= DD^T - DX^T W - W^T X D^T + W^T X X^T W$$

$$\Rightarrow\ \xi = E[e^2(k)] = E[DD^T - DX^T W - W^T X D^T +\ W^T X X^T W\,]$$

$$= r_d - r_{dx}W - W^T r_{xd} + W^T R_x W$$

Where, $r_d = E[DD^T]$, $R_x = E[XX^T]$

$$r_{dx} = E[DX^T] = r_{xd} = E[XD^T]$$

$$\Rightarrow\ \frac{\partial \xi}{\partial W} = -r_{dx} - r_{xd} + W^T R_x + R_x W$$

$$= -2r_{dx} + 2R_x W = E'\big(w(n)\big)$$

$$\frac{\partial^2 \xi}{\partial W^2} = 2R_x = H\big(w(n)\big)$$

$$w(n+1) = w(n) - \eta H\big(w(n)\big)^{-1} E'\big(w(n)\big)$$

$$w(n+1) = w(n) - \eta \tfrac{1}{2} R_x^{-1} (-2r_{dx} + 2R_x W)$$

$$w(n+1) = w(n) - \eta\big(\mathrm{w(n)} - w_{op}\big)$$

$$\boxed{= (1-\eta)w(n) + \eta w_{op}}$$

Final equation represents the state equation of discrete time system.

With affine transformation and rotation, the above equation is transformed in the form of x(k) = $\alpha^k x(0),$ which is stable and converge to zero if $|\alpha| < 1$

With similar analysis, $|1-\eta| < 1 \Rightarrow 0 < \eta < 2$

Generally, *Newton's* method *converges quickly* and *does not exhibit* the zigzagging behavior of the method of *steepest descent*.

However, *Newton's* method has *two* main *disadvantages*:

- the *Hessian* matrix H(n) has to be a *positive definite* matrix for all n, which is *not guaranteed* by the algorithm.
  - This is solved by the *modified Gauss-Newton* method.
- It has high *computational complexity*

Gauss-Newton Method:

Applicable to the cost function of sum of error squares, Let

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

e(i) is function of w, which is fixed over observation interval $1 \le i \le n$

e(i) is function of w, which is fixed over observation interval $1 \le i \le n$. Given an operating point w(n), linearize the dependence of e(i) on w as

$$e'(i,w) = e(i) + \left[\frac{\partial e(i)}{\partial w}\right]^T \big(w - w(n)\big),\ i = 1,2,\dots,n\ \ at\, w = w(n)$$

In matrix form

$$e'(n,w) = e(n) + J(n)\big(w - w(n)\big)$$

where, e(n) is the error vector, e(n) =[e(1), e(2),…,e(n)]^{T}and J(n) is the Jacobian matrix, which is defined as

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

The gradient of error vector

$\nabla e(n) = [\nabla e(1), \nabla e(2), \dots, \nabla e(n)]$ m x n

But, J(n) is the Jacobian matrix of dimension n x m, therefore

J(n) = $\nabla e(n)^T$

The updated weight w(n+1) is defined by

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

The squared Euclidean norm of error vector

$$\frac{1}{2}\|e'(n,w)\|^2 = \frac{1}{2}\|e\,(n)\|^2 + e^T(n)J(n)\big(w - w(n)\big) + \frac{1}{2}\big(w - w(n)\big)^T J^T(n)J(n)(w - w(n))$$

Diff. this eq. with w and setting it to zero, we get

$$J^T(n)e(n) + J^T(n)J(n)\big(w - w(n)\big) = 0$$

Let $\mathrm{a}, \mathrm{b} \in \mathbb{R}^n$, then

$$\frac{\partial(a^T b)}{\partial a} = \frac{\partial(b^T a)}{\partial a} \triangleq \begin{pmatrix} \frac{\partial(b^T a)}{\partial a_1} \\ \vdots \\ \frac{\partial(b^T a)}{\partial a_n} \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = b$$

And Let P = $P^T \in \mathbb{R}^{nxn}$

$$\frac{\partial(a^T P a)}{\partial a} = 2Pa$$

The squared Euclidean norm of error vector

$$\frac{1}{2}\|e'(n,w)\|^2$$
$$= \frac{1}{2}\|e\,(n)\|^2 + e^T(n)J(n)\big(w - w(n)\big)$$
$$+ \frac{1}{2}\big(w - w(n)\big)^T J^T(n)J(n)(w - w(n))$$

Diff. this eq. with w and setting it to zero, we get

$$J^T(n)e(n) + J^T(n)J(n)\big(w - w(n)\big) = 0$$

$$J^T(n)e(n) + J^T(n)J(n)\big(w - w(n)\big) = 0$$

$$\Rightarrow w - w(n) = -\big(J^T(n)J(n)\big)^{-1}J^T(n)e(n)$$

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

Gauss-Newton Method *does not require Hessian matrix as that* of the *Newton's* method instead it uses Jacobian of error matrix

However, Gauss-Newton Method has a *disadvantage*:

- $J^T(n)J(n)$ matrix has to be nonsingular, which is *not guaranteed* always
    - This is solved by adding δI to $J^T(n)J(n)$ matrix
- Where δ is a positive constant, added to ensure
- $J^T(n)J(n) + \delta I$ to be positive definite for all n

Therefore, the modified weight update equation Gauss-Newton Method as

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

<u>LEAST-MEANS-SQUARE(LMS) ALGORITHM:</u>

- In Least-Mean Square (LMS), developed by Widrow and Hoff (1960), was the first linear adaptive filtering algorithm (inspired by the perceptron) for solving problems such as prediction
- Some features of the LMS algorithm:
    - Linear computational complexity with respect to adjustable parameters.
    - Robust with respect to external disturbance

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

The aim of the LMS algorithm is to minimize the instantaneous value of the cost function E (w):

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

where e(n) is the error signal measured at time n.

Differentiation of E (w), with respect to w, yields:

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

As with the least-square filters, the LMS operates on linear neuron, we can write:

$$e(n) = d(n) - X^T(n)w(n)$$

$$\frac{\partial e(n)}{\partial w} = -X(n)$$

$$g(n) = \frac{\partial E(w)}{\partial w} = -e(n)X(n)$$

Weight update equation in steepest descent algorithm

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

Therefore,

$$w(n+1) = w(n) + \eta e(n)X(n)$$

Where η is the learning rate parameter

The **inverse** of the **learning-rate** acts as a **memory** of the LMS algorithm. The smaller the learning-rate η , the longer the memory span over the past data, which leads to more accurate results but with slow convergence rate.

In the **steepest-descent** algorithm the weight vector w(n) follows a **well-defined** trajectory in the weight space for a prescribed η.

The feedback loop around the weight vector

w(n) in the LMS algorithm behaves like a low-pass filter, passing the low frequency

components of the error signal and attenuating its high frequency components

(Haykin, 1996). The average time constant of this filtering action is inversely proportional

to the learning-rate parameter “

Hence, by assigning a small value to eta, the

adaptive process will progress slowly. More of the past data are then remembered by

the LMS algorithm, resulting in a more accurate filtering action.

In contrast, in the **LMS** algorithm, the weight vector w(n) traces a **random** trajectory. For this reason, the LMS algorithm is sometimes referred to as “**stochastic gradient algorithm**.”

Unlike the steepest-descent, the LMS algorithm **does not** require knowledge of the **statistics** of the **environment**. It produces an **instantaneous estimate** of the weight vector.

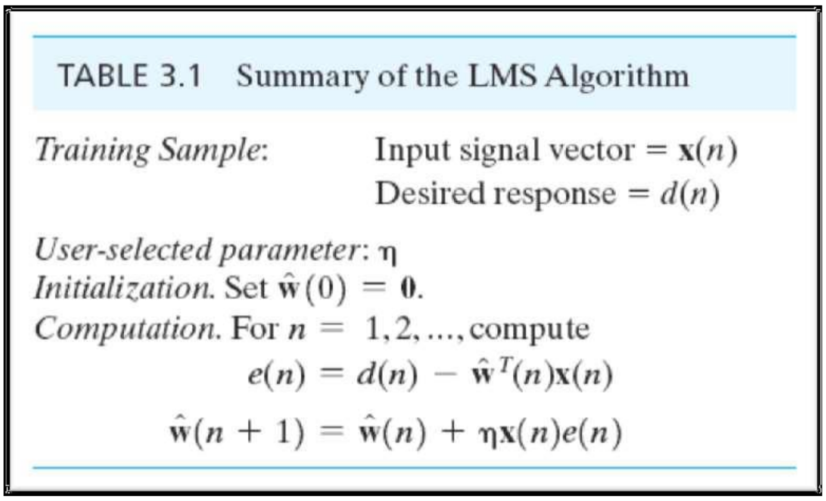


TABLE 3.1 Summary of the LMS Algorithm

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

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

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

Virtues and Limitation of the LMS Algorithm:

Computational Simplicity and Efficiency:

- The Algorithm is very simple to code, only two or three line of code.
- The computational complexity of the algorithm is linear in the adjustable parameters.

Robustness

- Since the LMS is model independent, therefore it is robust with respect to disturbance, (small model uncertainty and small disturbances (i.e., disturbances with small energy) can only result in small estimation errors (error signals)).

The primary limitations of the LMS algorithm are:

- Its **slow rate** of convergence (which become serious when the dimensionality of the input space becomes **high**)
- Its **sensitivity** to variation in the eigen structure of the input. (it typically requires a number of iterations equal to about 10 times the dimensionality of the input data space for it to converge to a stable solution)

• The sensitivity to changes in environment become particularly acute when the condition number of the LMS algorithm is high.

• The condition number,

$$\chi(R) = {}^{\lambda_{\max}}/_{\lambda_{\min}}$$

Where $\lambda_{\max}$ and $\lambda_{\min}$ are the maximum and minimum eigenvalues of the correlation matrix, R_x.

Convergence of the LMS Algorithm:

- Convergence is influenced by the statistical characteristics of the input vector x(n) and the value assigned to the learning-rate parameter η
- By invoking the elements of independence theory and assuming the learning- rate parameter η is sufficiently small, it is shown in Haykin (1996) that 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 eigenvalue of the correlation matrix R_x,

In typical applications of the LMS algorithm, knowledge of λ_{max} is not available. To overcome this difficulty, the trace of R_x, may be taken as a conservative estimate forλ_{max}, the condition is reformulated as

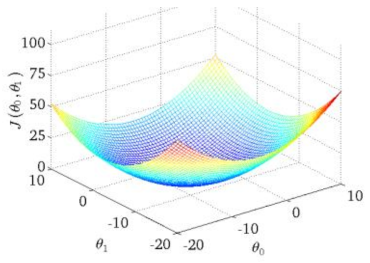
$$0 < \eta < \frac{2}{tr[R_x]}$$

Where, λ_{max} is the largest eigenvalue of the correlation matrix R_x.

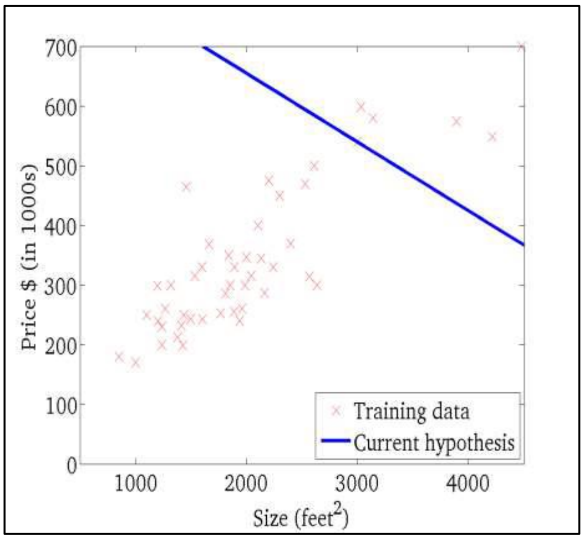
By definition, the trace of a square matrix is equal to the sum of its diagonal elements. Since each diagonal element of the correlation matrix R_x equals the mean-square value of the corresponding sensor input

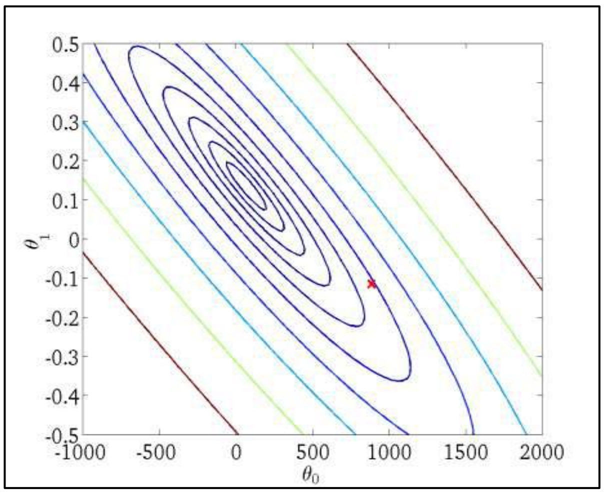
$$0 < \eta < \frac{2}{\text{sum of mean-square values of the sensor inputs}}$$

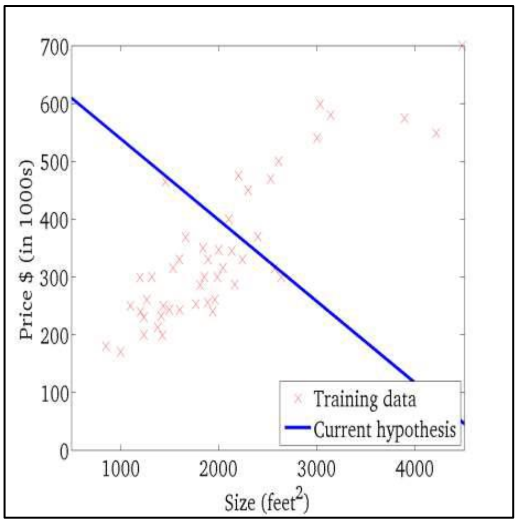
Computational Example:

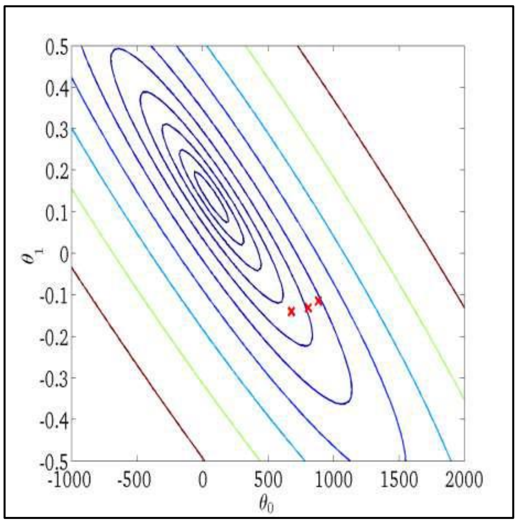


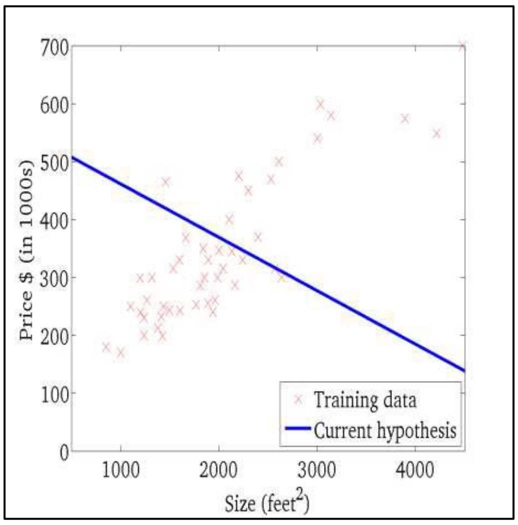
Usually, *linear models* produce concave cost functions

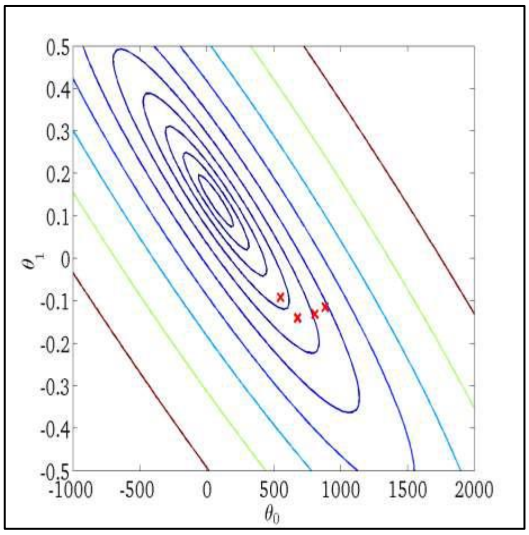


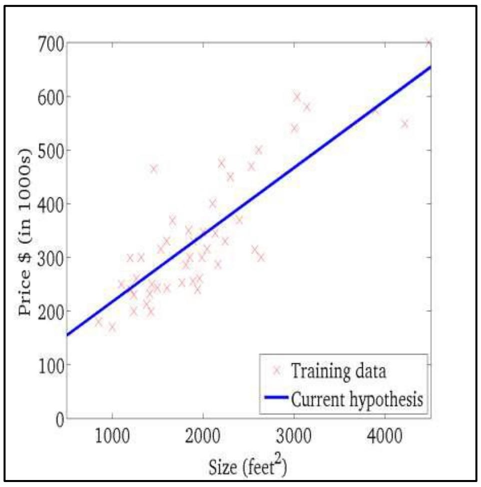


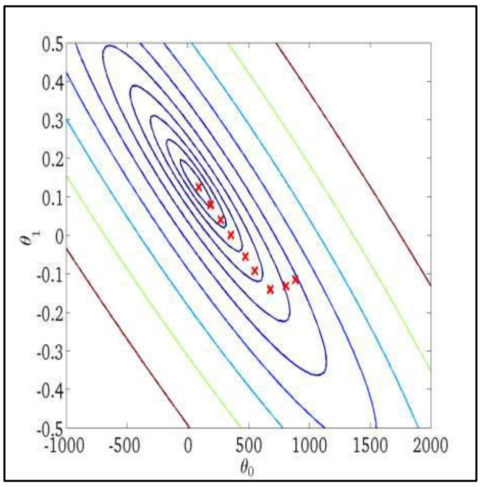

Training data
Current hypothesis
Price $ (in 1000s)
Size (feet^2)


θ_0
θ_1


Training data
Current hypothesis
Price $ (in 1000s)
Size (feet^2)


θ_0
θ_1


Training data
Current hypothesis
Price $ (in 1000s)
Size (feet^2)


θ_0
θ_1