Chapter 5

Supervised Learning I: Perceptrons and LMS

Neural Networks: A Classroom Approach

Satish Kumar
Department of Physics & Computer Science
Dayalbagh Educational Institute (Deemed University)

Two Fundamental Learning Paradigms

- Non-associative
  - an organism acquires the properties of a single repetitive stimulus.
- Associative
  - an organism acquires knowledge about the relationship of either one stimulus to another, or one stimulus to the organism's own behavioural response to that stimulus.

Examples of Associative Learning

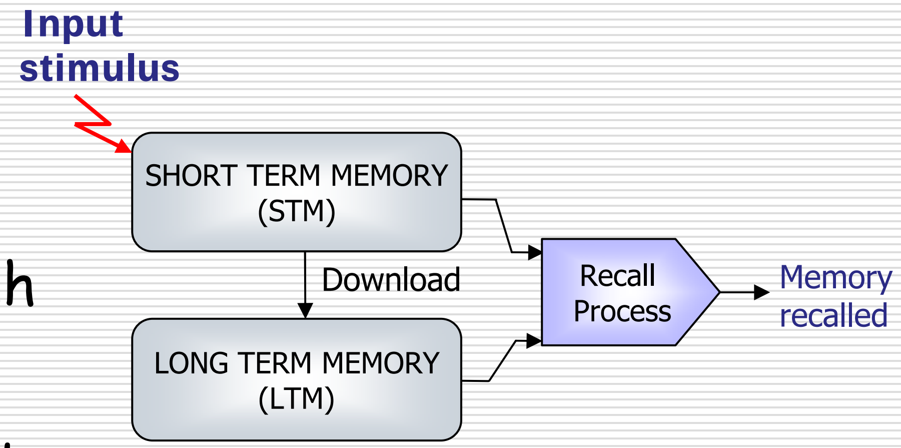
- Classical conditioning
  - Association of an unconditioned stimulus (US) with a conditioned stimulus (CS).
  - CS's such as a flash of light or a sound tone produce weak responses.
  - US's such as food or a shock to the leg produce a strong response.
  - Repeated presentation of the CS followed by the US, the CS begins to evoke the response of the US.
  - Example: If a flash of light is always followed by serving of meat to a dog, after a number of learning trials the light itself begins to produce salivation.
- Operant conditioning
  - Formation of a predictive relationship between a stimulus and a response.
  - Example: Place a hungry rat in a cage which has a lever on one of its walls. Measure the spontaneous rate at which the rat presses the lever by virtue of its random movements around the cage. If the rat is promptly presented with food when the lever is pressed, the spontaneous rate of lever pressing increases!

Reflexive and Declarative Learning

- Reflexive learning
  - repetitive learning is involved and recall does not involve any awareness or conscious evaluation.
- Declarative learning
  - established by a single trial or experience and involves conscious reflection and evaluation for its recall.
- Constant repitition of declarative knowledge often manifest itself in reflexive form.

Important Aspects of Human Memory

- Two distinct stages:
  - short-term memory (STM)
  - long-term memory (LTM)
- Inputs to the brain are processed into STMs which last at the most for a few minutes.
- Information is downloaded into LTMs for more permanent storage: days, months, and years.
- Capacity of LTMs is very large.



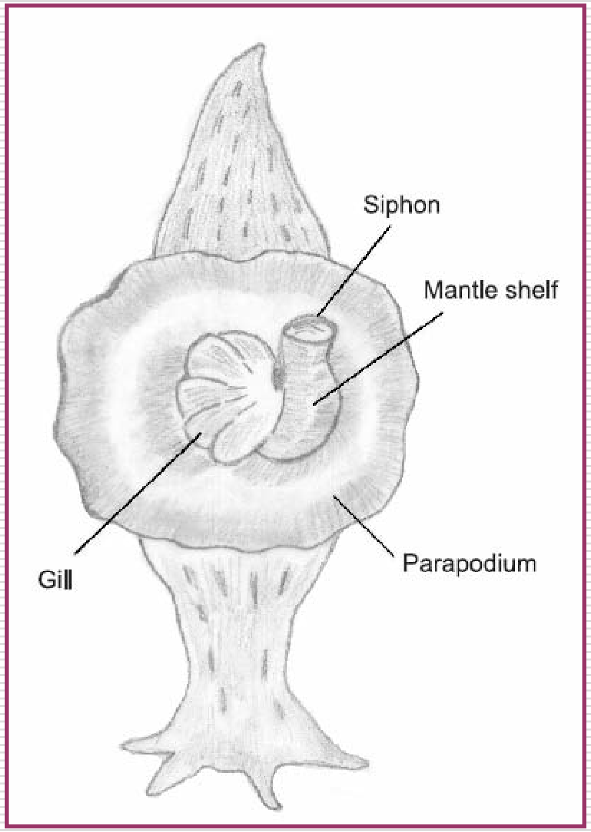
Recall process of these memories is distinct from the memories themselves.

Important Aspects of Human Memory

- Recall of recent memories is more easily disrupted than that of older memories.
- Memories are dynamic and undergo continual change and with time all memories fade.
- STM results in
  - physical changes in sensory receptors.
  - simultaneous and cohesive reverberation of neuron circuits.
- Long-term memory involves
  - plastic changes in the brain which take the form of strengthening or weakening of existing synapses
  - the formation of new synapses.
- Learning mechanism distributes the memory over different areas
  - Makes robust to damage
  - Permits the brain to work easily from partially corrupted information.
- Reflexive and declarative memories may actually involve different neuronal circuits.

From Synapses to Behaviour: The Case of *Aplysia*

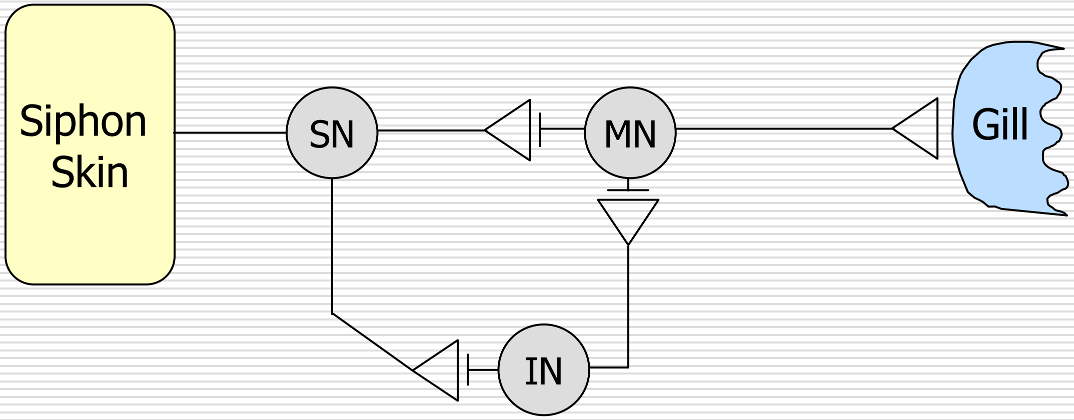
- *Aplysia* has a respiratory gill that is housedin the mantle cavity (which is a respiratory chamber covered by the mantle shelf) on the dorsal side of the mollusc.
- The mantle forms a spout at the dorsal end which usually protrudes between the parapodia which are wing-like extensions of the body wall.



Gill-Siphon Withdrawal (GSW) Reflex

- When a tactile stimulus is applied to the siphon or the mantle shelf, two reflexes are initiated:
  - the siphon contracts behind the parapodia
  - the gill is withdrawn into the mantle cavity.
- The gill-siphon withdrawal (GSW) reflex is an example of an innate defensive reflex.

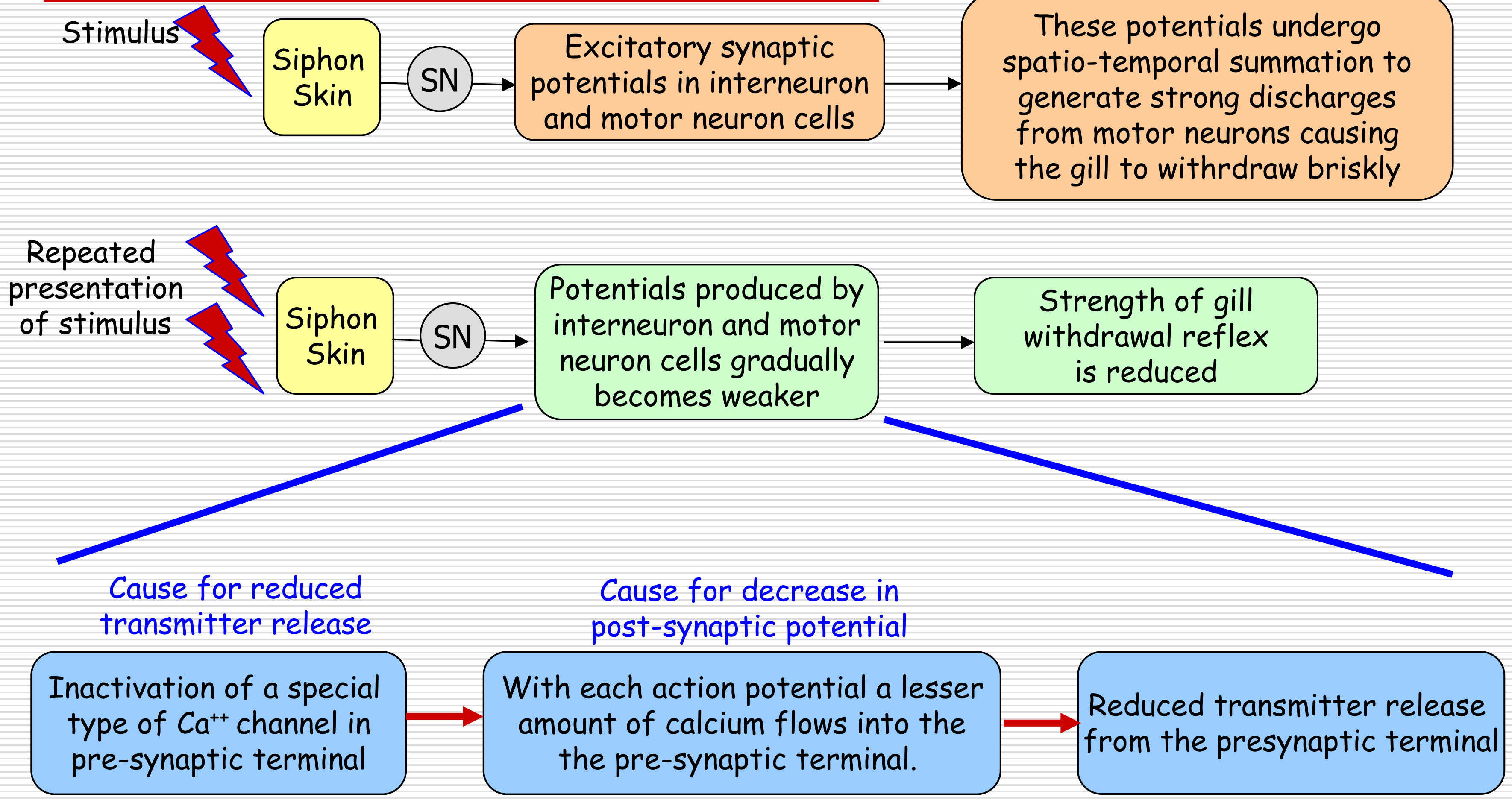
Cellular Mechanism of GSW Habituation



MN:	Mentor Neuron
IN:	Interneuron
SN:	Sensory Neuron

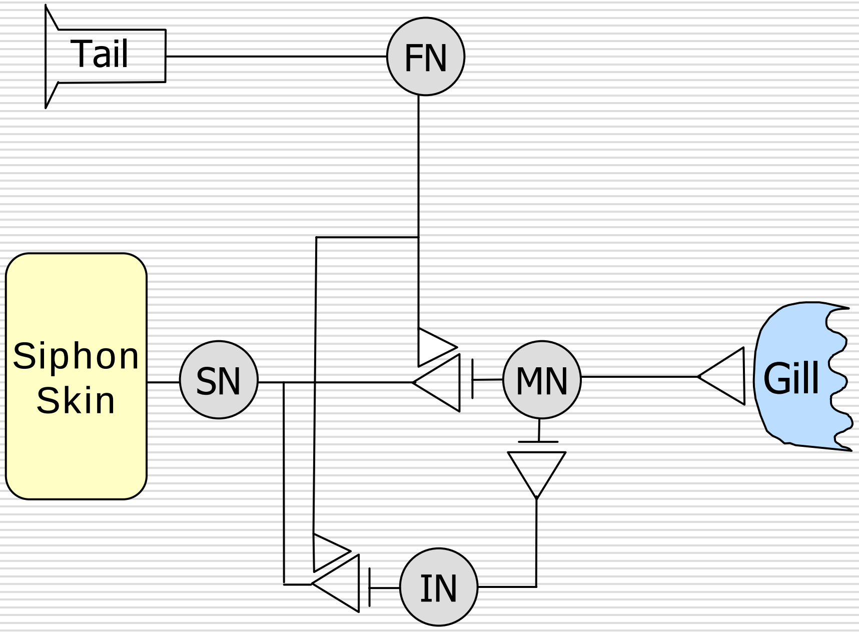
- Two dozen sensory neurons are embedded in the siphon skin
- One of such sensory neurons is illustrated in the above figure
- A sensory neuron terminates on a cluster of six motor neurons that activate the gill. Sensory neurons also excite motor neuron via an inter-neuronal pathway.

Habituation Cascade


Stimulus
Siphon Skin
SN
Excitatory synaptic potentials in interneuron and motor neuron cells
These potentials undergo spatio-temporal summation to generate strong discharges from motor neurons causing the gill to withrdraw briskly
Repeated presentation of stimulus
Siphon Skin
SN
Potentials produced by interneuron and motor neuron cells gradually becomes weaker
Strength of gill withdrawal reflex is reduced
Cause for reduced transmitter release
Cause for decrease in post-synaptic potential
Inactivation of a special type of Ca++ channel in pre-synaptic terminal
With each action potential a lesser amount of calcium flows into the the pre-synaptic terminal.
Reduced transmitter release from the presynaptic terminal

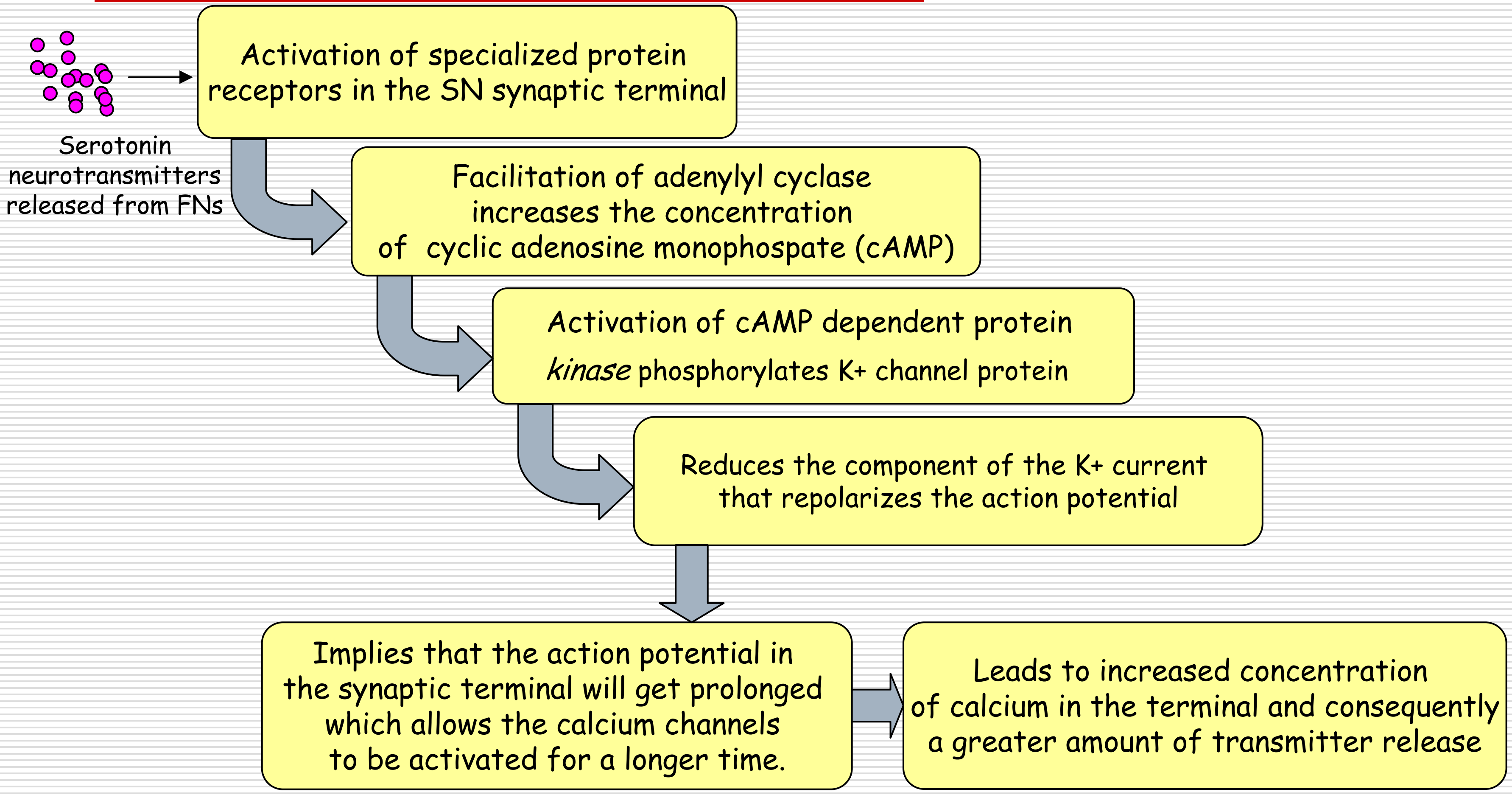
Cellular Mechanism of GSW Sensitization

- Sensitization is brought about by a sudden enhancement in synaptic transmission.
- Example:
  - Repeated presentations of tactile stimuli to the siphon leads to habituation
  - The delivery of a brief electrical stimulus to the tail rapidly facilitates excitatory postsynaptic potentials produced in motor neurons on sensory stimulation.
- The same set of synapses that are depressed by habituation, are enhanced by sensitization.
- The sensitizing stimulus activates a group of facilitating inter neurons that synapse axo-axonically on the terminals of sensory neurons.



MN: Motor Neuron
FN: Facilitatory Neuron
SN: Sensor Neuron
IN: Interneuron

Sensitization Cascade Involves Presynaptic Facilitation



Serotonin neurotransmitters released from FNs

Activation of specialized protein receptors in the SN synaptic terminal

Facilitation of adenylyl cyclase increases the concentration of cyclic adenosine monophospate (cAMP)

Activation of cAMP dependent protein *kinase* phosphorylates K+ channel protein

Reduces the component of the K+ current that repolarizes the action potential

Implies that the action potential in the synaptic terminal will get prolonged which allows the calcium channels to be activated for a longer time.

Leads to increased concentration of calcium in the terminal and consequently a greater amount of transmitter release

Learning Algorithms

- Define an architecture-dependent procedure to *encode pattern information* into weights
- Learning proceeds by modifying connection strengths.
- Learning is data driven:
  - A set of input–output patterns derived from a (possibly unknown) probability distribution.
    - Output pattern might specify a desired system response for a given input pattern
    - Learning involves approximating the unknown function as described by the given data.
  - Alternatively, the data might comprise patterns that naturally cluster into some number of unknown classes
    - Learning problem involves generating a suitable classification of the samples.

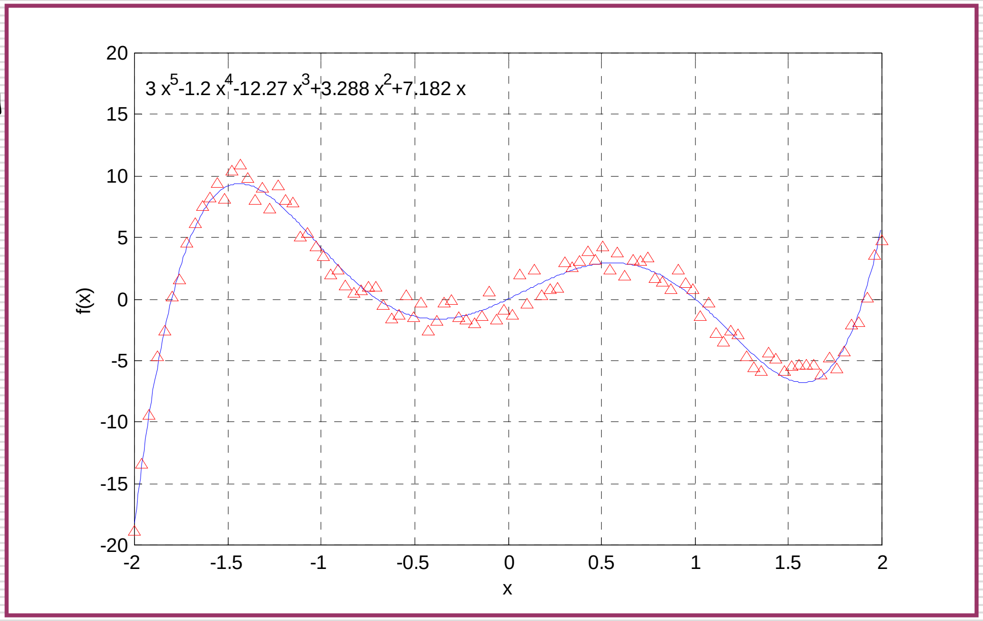
Supervised Learning

- Data comprises a set of discrete samples drawn from the pattern space where each sample relates an input vector $X_k \in R^n$ to an output vector $D_k \in R^p$.

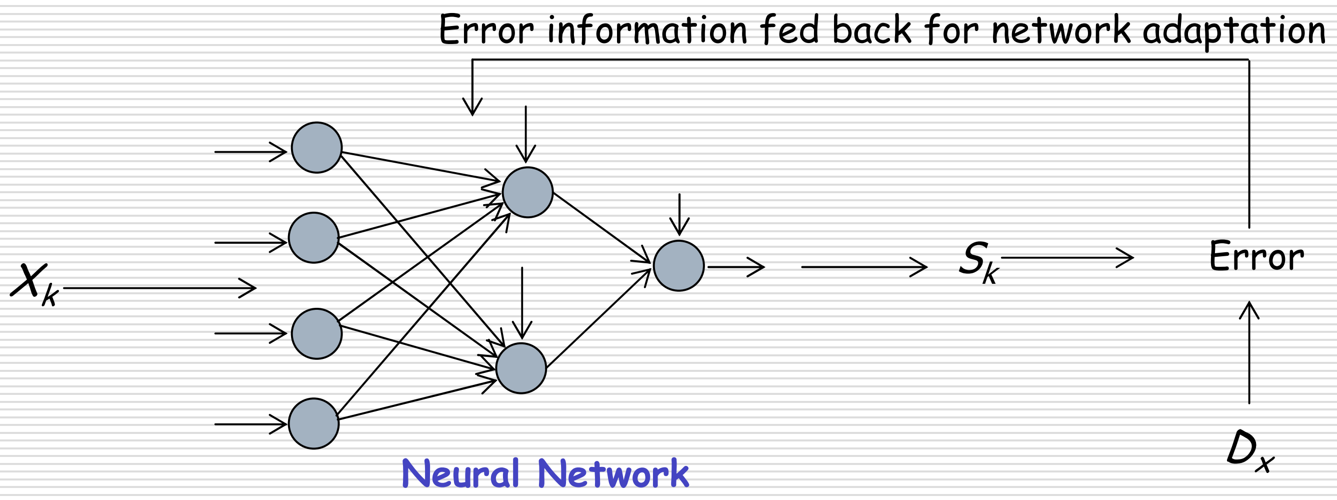
$$T = \{(X_k, D_k)\}_{k=1}^{Q}$$

- The set of samples describe the behaviour of an unknown function $f : R^n \to R^p$ which is to be characterized.

An example function described by a set of noisy data points



The Supervised Learning Procedure



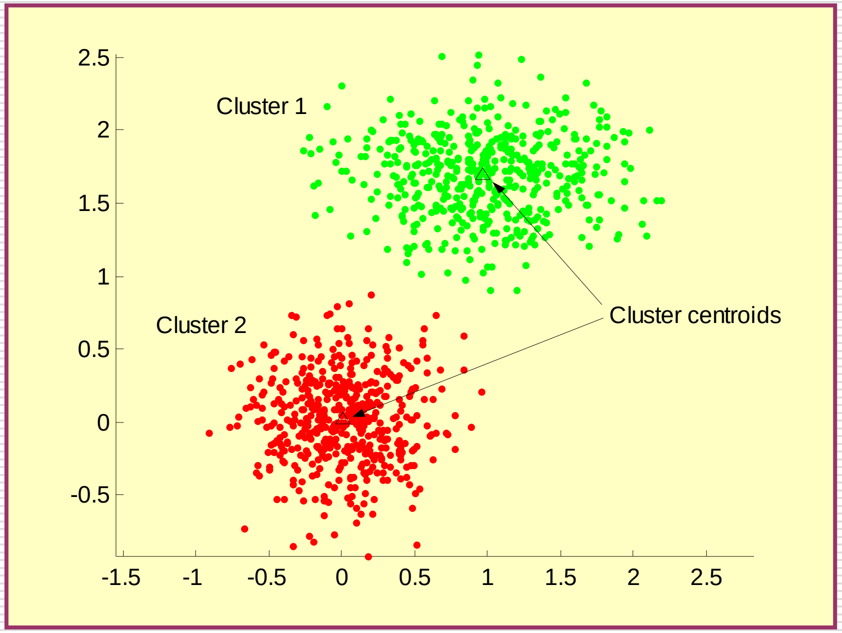
- We want the system to generate an output D_k in response to an input X_k, and we say that the system has learnt the underlying map if a stimulus X_k' close to X_k elicits a response S_k' which is sufficiently close to D_k. The result is a continuous function estimate.

Unsupervised Learning

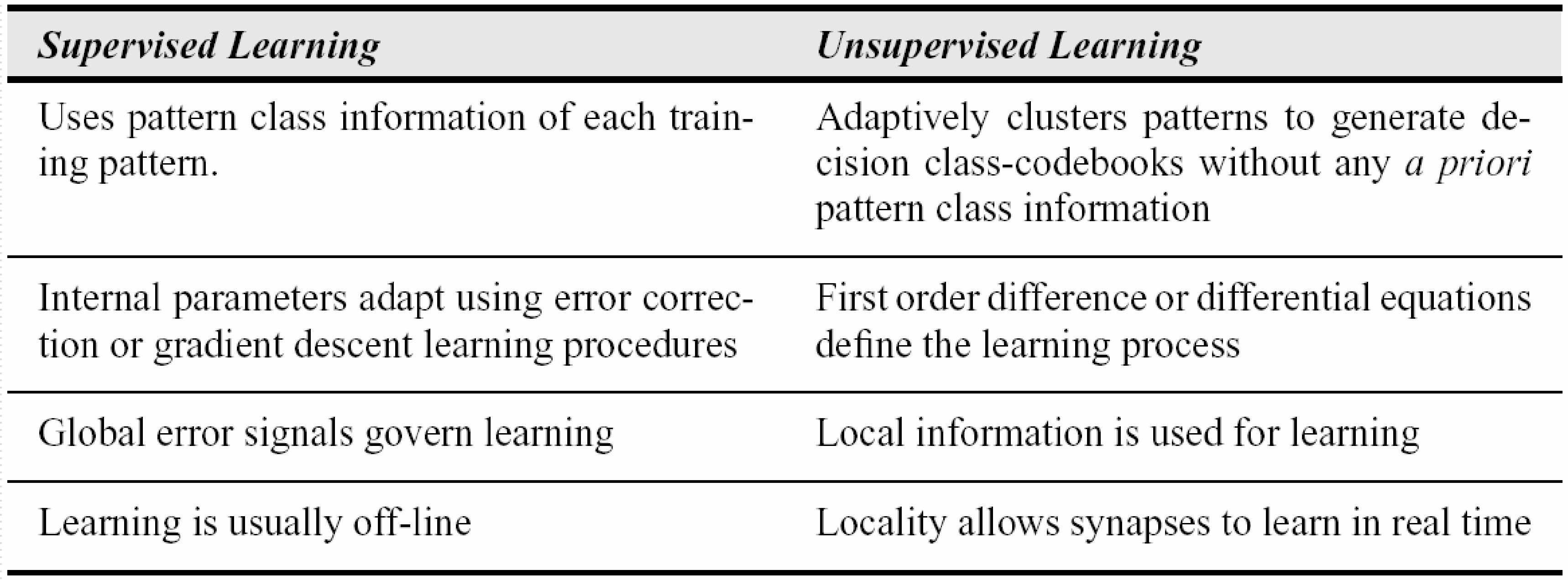
- Unsupervised learning provides the system with an input X_k, and allow it to *self-organize* its weights to generate internal prototypes of sample vectors.
  Note: There is no teaching input involved here.
- The system attempts to represent the entire data set by employing a small number of prototypical vectors—enough to allow the system to retain a desired level of discrimination between samples.
- As new samples continuously buffer the system, the prototypes will be in a state of constant flux.
- This kind of learning is often called *adaptive vector quantization*

Clustering and Classification

- Given a set of data samples $\{X_i\}$, X_i ∈ R^n, is it possible to identify well defined "clusters", where each cluster defines a class of vectors which are similar in some broad sense?
- Clusters help establish a classification structure within a data set that has no categories defined in advance.
- Classes are derived from clusters by appropriate labelling.
- The goal of pattern classification is to assign an input pattern to one of a finite number of classes.
- Quantization vectors are called codebook vectors.



Characteristics of Supervised and Unsupervised Learning



Supervised Learning	*Unsupervised Learning*
Uses pattern class information of each training pattern.	Adaptively clusters patterns to generate decision class-codebooks without any *a priori* pattern class information
Internal parameters adapt using error correction or gradient descent learning procedures	First order difference or differential equations define the learning process
Global error signals govern learning	Local information is used for learning
Learning is usually off-line	Locality allows synapses to learn in real time

General Philosophy of Learning: *Principle of Minimal Disturbance*

- Adapt to reduce the output error for the current training pattern, with minimal disturbance to responses already learned.

Error Correction and Gradient Descent Rules

- *Error correction rules* alter the weights of a network using a linear error measure to reduce the error in the output generated in response to the present input pattern.

- *Gradient rules* alter the weights of a network during each pattern presentation by employing gradient information with the objective of reducing the mean squared error (usually averaged over all training patterns).

Learning Objective for TLNs

- Augmented Input and Weight vectors

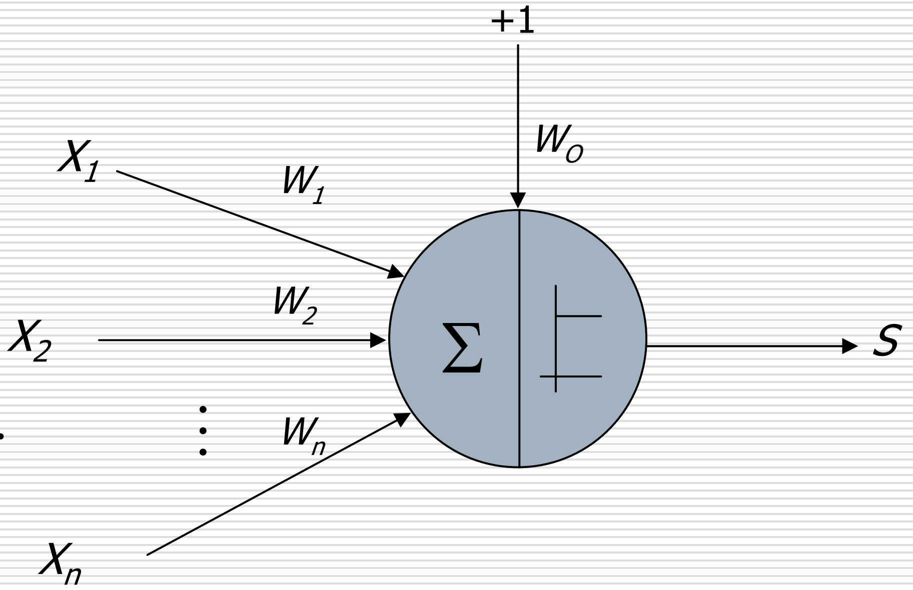
$$X_k = \left(x_0, x_1^k, \ldots, x_n^k\right)^T \; X_k \in R^{n+1}$$

$$W_k = \left(w_0^k, w_1^k, \ldots, w_n^k\right)^T \; W_k \in R^{n+1}$$

- Objective: To design the weights of a TLN to correctly classify a given set of patterns.
- Assumption: A training set of following form is given

$$T = \{(X_k, D_k)\}_{k=1}^{Q} \quad X_K \in R^{n+1}, \; d_k \in \{0,1\}$$

- Each pattern X_k is tagged to one of two classes C_0 or C_1 denoted by the desired output d_k being 0 or 1 respectively.



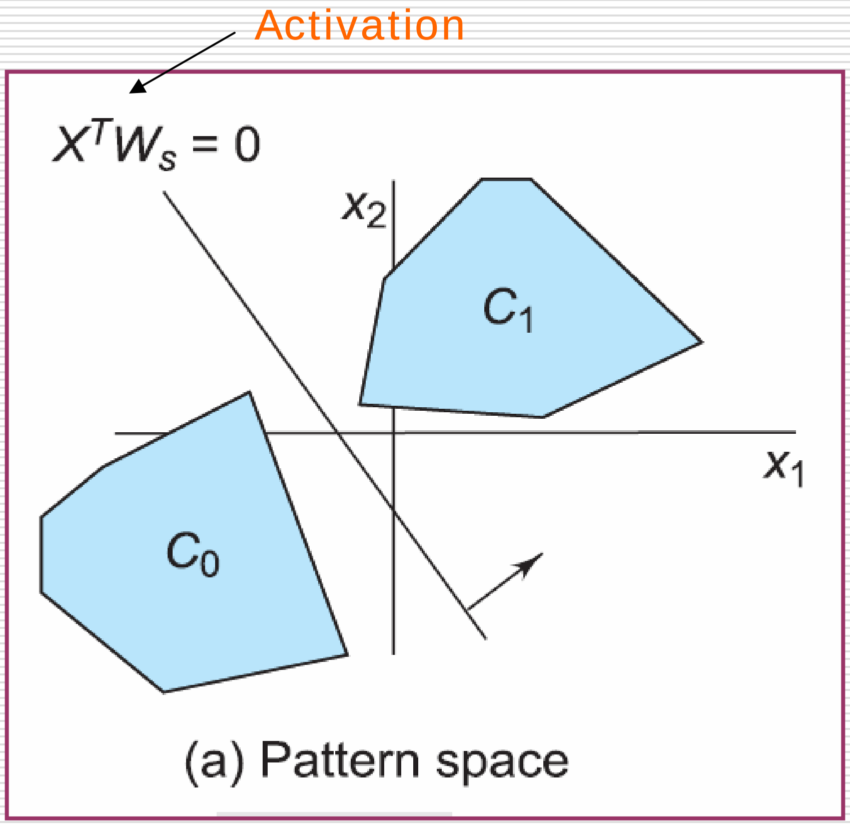
$$\mathcal{S}(y_k) = \begin{cases} 1 & y_k > 0 \\ 0 & y_k < 0 \end{cases}$$

Learning Objective for TLNs (contd.)

- Two classes identified by two possible signal states of the TLN
  - C_0 by a signal $S(y_k) = 0$, C_1 by a signal $S(y_k) = 1$.
- Given two sets of vectors X_0 and X_1 belonging to classes C_0 and C_1 respectively the learning procedure searches a solution weight vector W_S that correctly classifies the vectors into their respective classes.
- Context: TLNs
  - Find a weight vector W_S such that for all $X_k \in X_1$, $S(y_k) = 1$; and for all $X_k \in X_0$, $S(y_k) = 0$.
  - Positive inner products translate to a +1 signal and negative inner products to a 0 signal
  - Translates to saying that for all $X_k \in X_1$, $X_k^T W_S > 0$; and for all $X_k \in X_0$, $X_k^T W_S < 0$.

Pattern Space

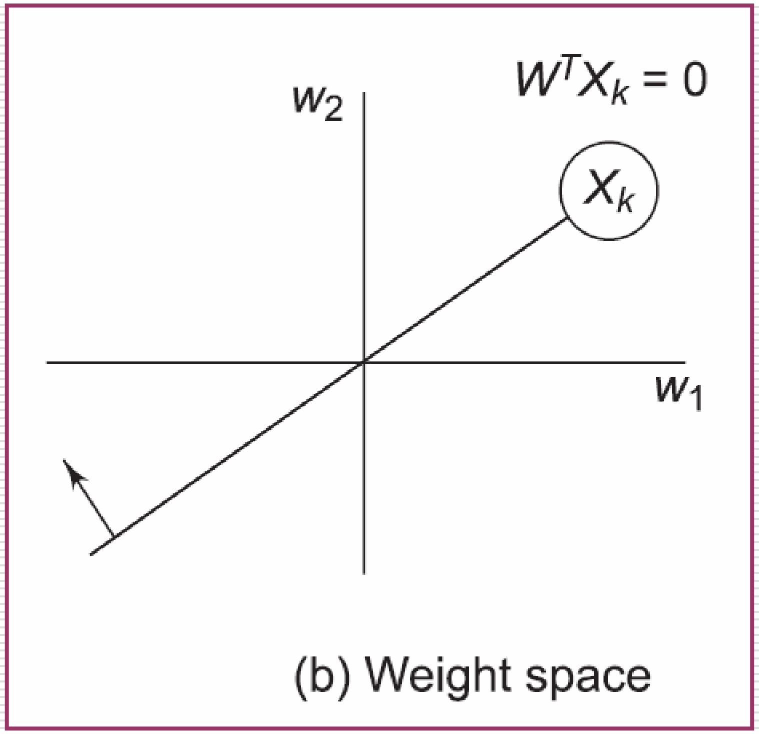
- Points that satisfy $X^T W_S = 0$ define a separating hyperplane in pattern space.
- Two dimensional case:
  - Pattern space points on one side of this hyperplane (with an orientation indicated by the arrow) yield positive inner products with W_S and thus generate a +1 neuron signal.
  - Pattern space points on the other side of the hyperplane generate a negative inner product with W_S and consequently a neuron signal equal to 0.
- Points in C_0 and C_1 are thus correctly classified by such a placement of the hyperplane.



(a) Pattern space

A Different View: Weight Space

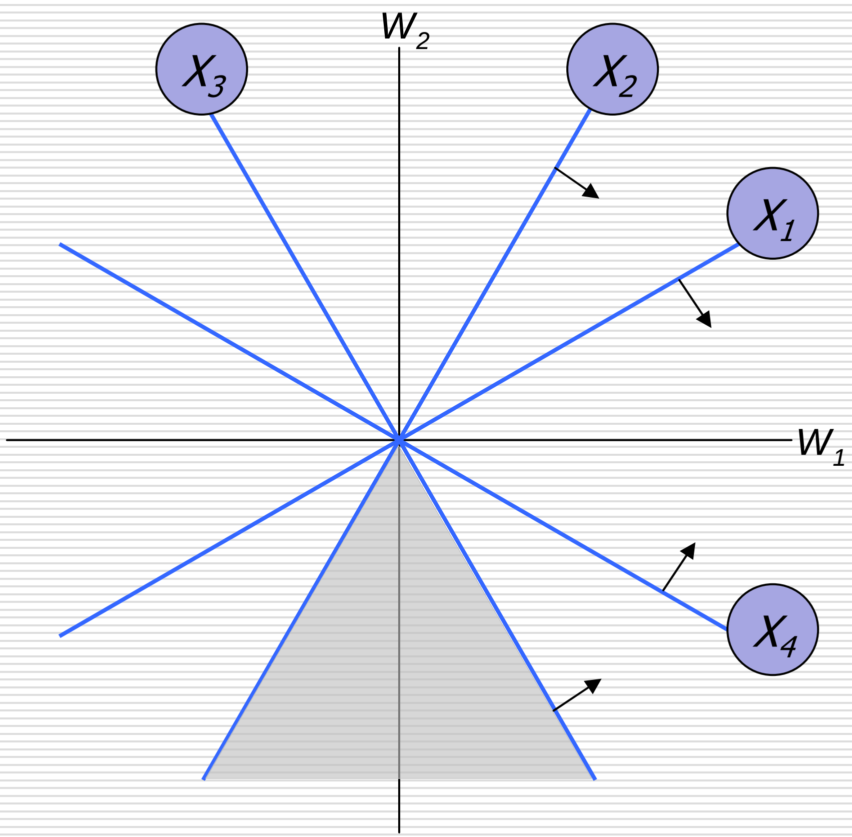
- Weight vector is a variable vector.
- $W^TX_k = 0$ represents a hyperplane in weight space
- Always passes through the origin since $W = 0$ is a trivial solution of $W^TX_k = 0$.
- Called the pattern hyperplane of pattern X_k.
- Locus of all points W such that $W^TX_k = 0$.
- Divides the weight space into two parts: one which generates a positive inner product $W^TX_k > 0$, and the other a negative inner product $W^TX_k < 0$.



(b) Weight space

Identifying a *Solution Region* from Orientated Pattern Hyperplanes

- For each pattern X_k in pattern space there is a corresponding hyperplane in weight space.
- For every point in weight space there is a corresponding hyperplane in pattern space.
- A solution region in weight space with four pattern hyperplanes
  - $\chi_1 = \{X_1, X_2\}$
  - $\chi_0 = \{X_3, X_4\}$



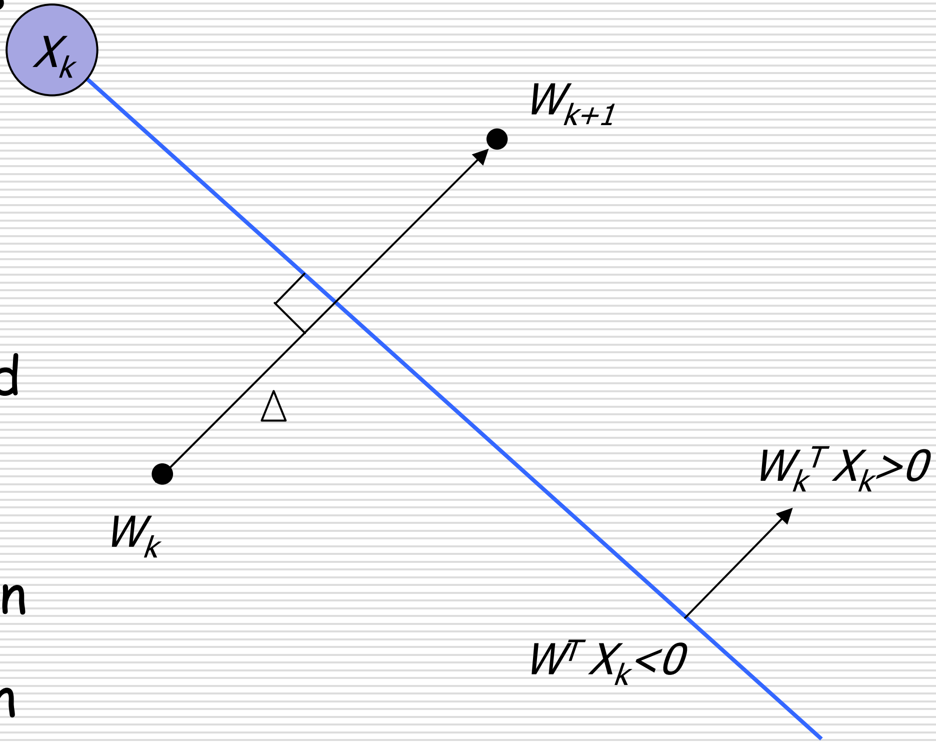
Solution region

Requirements of the Learning Procedure

- Linear separability guarantees the existence of a solution region.
- Points to be kept in mind in the design of an automated weight update procedure :
  - It must consider each pattern in turn to assess the correctness of the present classification.
  - It must subsequently adjust the weight vector to eliminate a classification error, if any.
  - Since the set of all solution vectors forms a *convex cone*, the weight update procedure should terminate as soon as it penetrates the boundary of this cone (solution region).

Design in Weight Space

- Assume: X_k X_1 and $W_k^T X_k$ as erroneously non-positive.
- For correct classification, shift the weight vector to some position W_{k+1} where the inner product is positive.
- The smallest perturbation in W_k that produces the desired change is, the perpendicular distance from W_k onto the pattern hyperplane.
- In weight space, the direction perpendicular to the pattern hyperplane is none other than that of X_k itself.



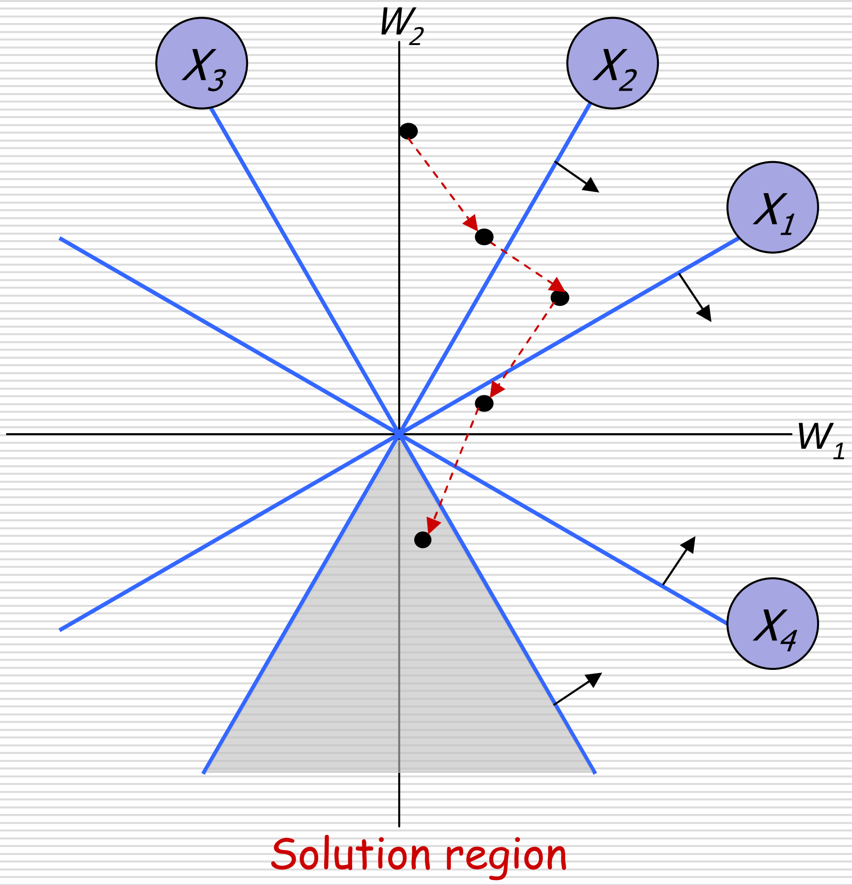
Simple Weight Change Rule: Perceptron Learning Law

$$W_{k+1} = \begin{cases} W_k + \eta_k X_k & \text{if} \quad X_k \in \mathcal{X}_1 \quad \text{and} \quad W_k^T X_k \leq 0 \\ W_k - \eta_k X_k & \text{if} \quad X_k \in \mathcal{X}_0 \quad \text{and} \quad W_k^T X_k \geq 0 \end{cases}$$

- If X_k X_1 and $W_k^T X_k < 0$ add a fraction of the pattern to the weight W_k if one wishes the inner product $W_k^T X_k$ to increase.
- Alternatively, if X_k X_0, and $W_k^T X_k$ is erroneously non-negative we will subtract a fraction of the pattern from the weight W_k in order to reduce this inner product.

Weight Space Trajectory

- The weight space trajectory corresponding to the sequential presentation of four patterns with pattern hyperplanes as indicated:
  - $\chi_1 = \{X_1, X_2\}$ and $\chi_0 = \{X_3, X_4\}$



Linear Containment

- Consider the set X_0' in which each element X_0 is negated.
- Given a weight vector W_k, for any $X_k \square X_1 \square X_0'$, $X_k^T W_k > 0$ implies correct classification and $X_k^T W_k < 0$ implies incorrect classification.
- $X' = X_1 \square X_0'$ is called the adjusted training set.
- Assumption of linear separability guarantees the existence of a solution weight vector W_S, such that $X_k^T W_S > 0 \square X_k \square X$
- We say X' is a linearly contained set.

Recast of Perceptron Learning with Linearly Contained Data

$$W_{k+1} = \begin{cases} W_k + \eta_k X_k & \text{if } X_k^T W_k \leq 0 \\ W_k & \text{if } X_k^T W_k > 0 \end{cases}$$

- Since X_k X', a misclassification of X_k will add $\eta_k X_k$ to W_k.
- For X_k X_0', X_k actually represents the negative of the original vector.
- Therefore addition of $\eta_k X_k$ to W_k actually amounts to subtraction of the original vector from W_k.

Perceptron Algorithm: Operational Summary

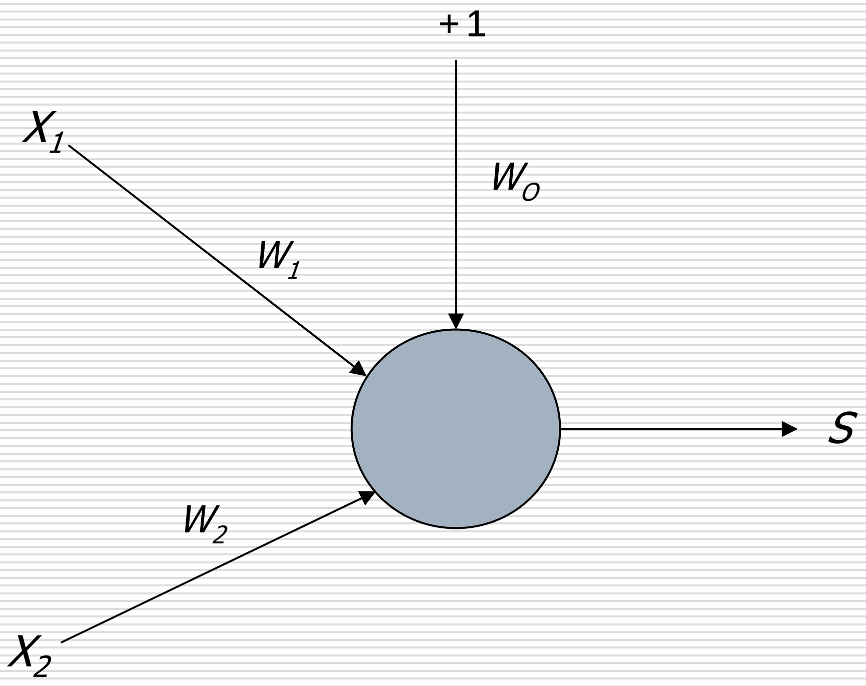
Given	An adjusted linearly contained training set $\mathcal{X}' = \mathcal{X}_1 \cup \mathcal{X}_0'$ comprising augmented vectors $X_k \in \mathbb{R}^{n+1}$. (Note that the corresponding desired classes d_k have been used in the augmentation process. Their use is implicit.)
Initialize	$\hookrightarrow$ $W \in \mathbb{R}^{n+1}$ to $W_1 = 0$ or some small random vector.
Iterate	$\circlearrowleft$ Repeat (on an epoch by epoch basis)

```
{
    $\rightsquigarrow$ Select an $X_k \in \mathcal{X}'$ (in any sequence)
    $\rightsquigarrow$ Compute $X_k^T W_k$
    $\rightsquigarrow$ If $X_k^T W_k \leq 0$
            $W_{k+1} = W_k + \eta X_k$
} until $(X_i^T W_k > 0$ for all $X_i \in \mathcal{X}')$
```

Perceptron Convergence Theorem

- Given: A linearly contained training set X' and any initial weight vector W_1.
- Let S_W be the weight vector sequence generated in response to presentation of a training sequence S_X upon application of Perceptron learning law. Then for some finite index k_0 we have: $W_{k0} = W_{k0}+1 = W_{k0}+2 =$ □ □ □ $=$ W_S as a solution vector.
- See the text for detailed proofs.

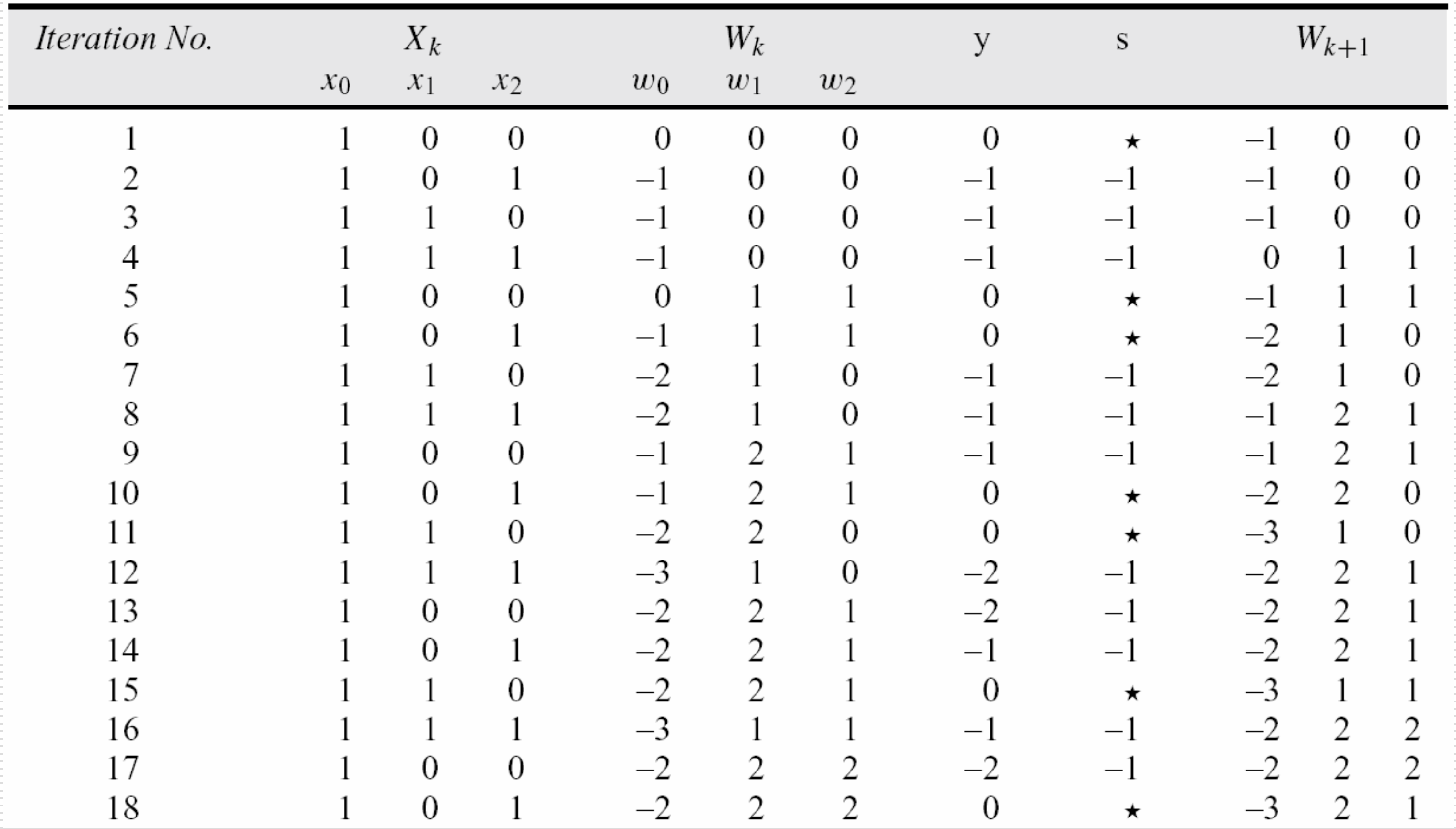
Hand-worked Example



Binary threshold neuron

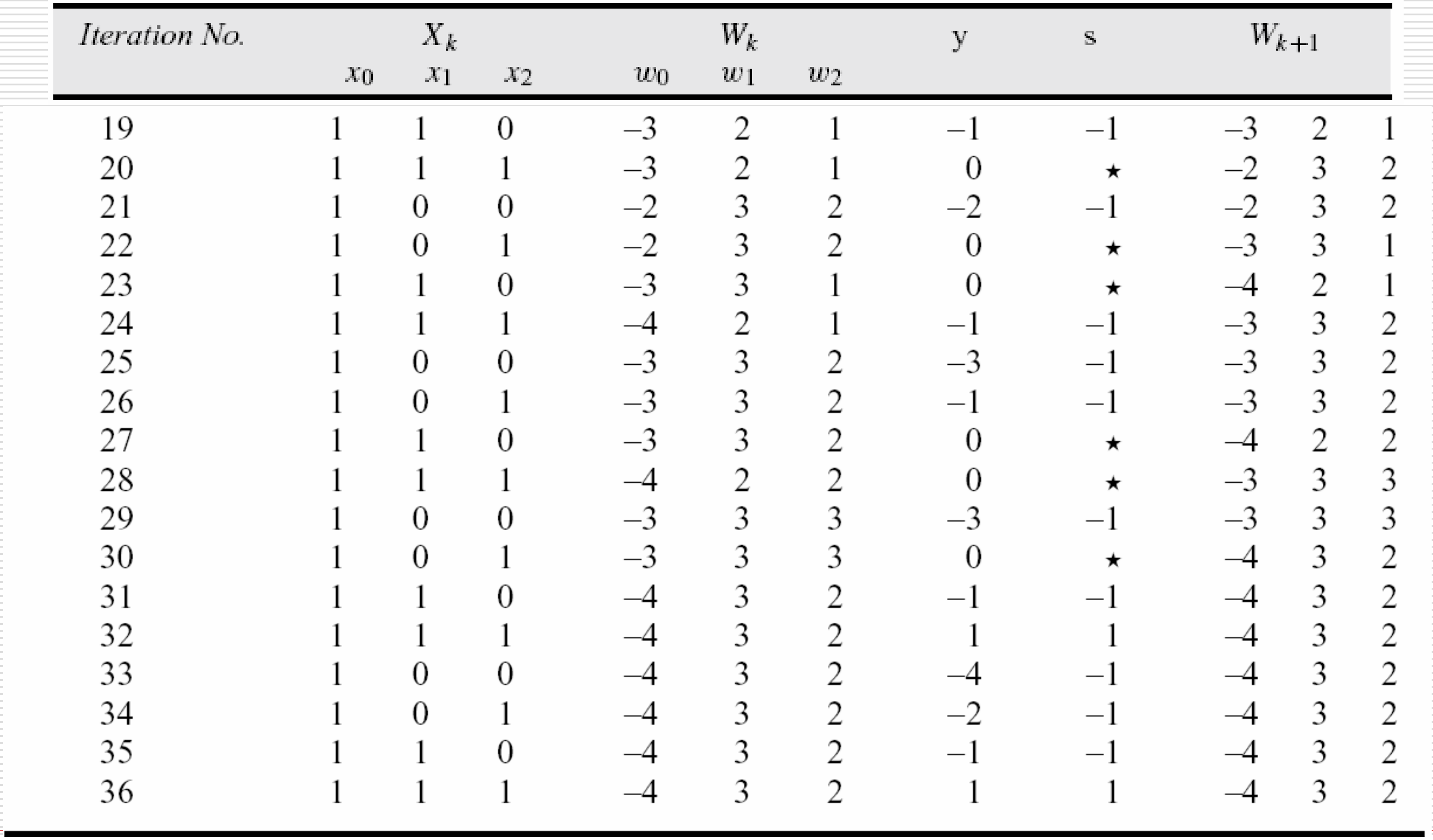
Pattern	x_0	x_1	x_2	d
1	1	0	0	0
2	1	0	1	0
3	1	1	0	0
4	1	1	1	1

Classroom Exercise



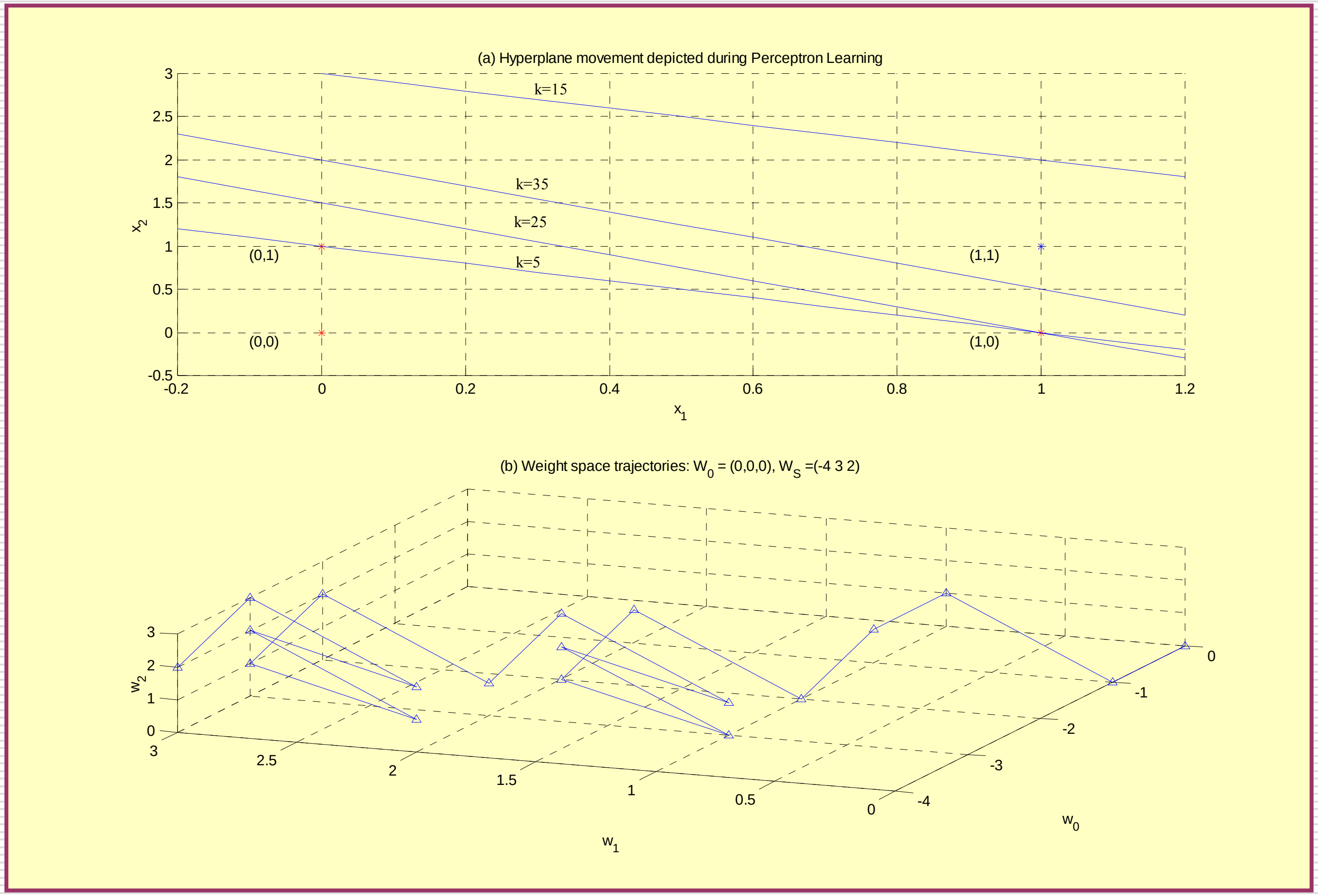
Iteration No.	X_k			W_k			y	s	W_{k+1}		
	x_0	x_1	x_2	w_0	w_1	w_2					
1	1	0	0	0	0	0	0	⋆	–1	0	0
2	1	0	1	–1	0	0	–1	–1	–1	0	0
3	1	1	0	–1	0	0	–1	–1	–1	0	0
4	1	1	1	–1	0	0	–1	–1	0	1	1
5	1	0	0	0	1	1	0	⋆	–1	1	1
6	1	0	1	–1	1	1	0	⋆	–2	1	0
7	1	1	0	–2	1	0	–1	–1	–2	1	0
8	1	1	1	–2	1	0	–1	–1	–1	2	1
9	1	0	0	–1	2	1	–1	–1	–1	2	1
10	1	0	1	–1	2	1	0	⋆	–2	2	0
11	1	1	0	–2	2	0	0	⋆	–3	1	0
12	1	1	1	–3	1	0	–2	–1	–2	2	1
13	1	0	0	–2	2	1	–2	–1	–2	2	1
14	1	0	1	–2	2	1	–1	–1	–2	2	1
15	1	1	0	–2	2	1	0	⋆	–3	1	1
16	1	1	1	–3	1	1	–1	–1	–2	2	2
17	1	0	0	–2	2	2	–2	–1	–2	2	2
18	1	0	1	–2	2	2	0	⋆	–3	2	1

Classroom Exercise



Iteration No.	X_k			W_k			y	s	W_{k+1}		
	x_0	x_1	x_2	w_0	w_1	w_2					
19	1	1	0	–3	2	1	–1	–1	–3	2	1
20	1	1	1	–3	2	1	0	⋆	–2	3	2
21	1	0	0	–2	3	2	–2	–1	–2	3	2
22	1	0	1	–2	3	2	0	⋆	–3	3	1
23	1	1	0	–3	3	1	0	⋆	–4	2	1
24	1	1	1	–4	2	1	–1	–1	–3	3	2
25	1	0	0	–3	3	2	–3	–1	–3	3	2
26	1	0	1	–3	3	2	–1	–1	–3	3	2
27	1	1	0	–3	3	2	0	⋆	–4	2	2
28	1	1	1	–4	2	2	0	⋆	–3	3	3
29	1	0	0	–3	3	3	–3	–1	–3	3	3
30	1	0	1	–3	3	3	0	⋆	–4	3	2
31	1	1	0	–4	3	2	–1	–1	–4	3	2
32	1	1	1	–4	3	2	1	1	–4	3	2
33	1	0	0	–4	3	2	–4	–1	–4	3	2
34	1	0	1	–4	3	2	–2	–1	–4	3	2
35	1	1	0	–4	3	2	–1	–1	–4	3	2
36	1	1	1	–4	3	2	1	1	–4	3	2

MATLAB Simulation



Perceptron Learning Algorithm: MATLAB Code

```
p = [1 0 0
     1 0 1
     1 1 0
     1 1 1];
d =[0 0 0 1];
w =[0 0 0];
eta =1;
update =1;
while update==1
    for i=1:4
      y =p(i,:)*w';
      if y >=0 & d(i) ==0
         w =w - eta*p(i,:);
         up(i) =1;
      elseif y<=0 & d(i) == 1
         w =w + eta*p(i,:);
         up(i) =1;
      else
         up(i)=0;
      end
    end
    number_of_updates =up * up';
    if number_of_updates > 0
      update =1;
    else
      update =0;
    end
end
```

Perceptron Learning and Non-separable Sets

- Theorem:
  - Given a finite set of training patterns X, there exists a number M such that if we run the Perceptron learning algorithm beginning with any initial set of weights, W_1, then any weight vector W_k produced in the course of the algorithm will satisfy $W_k \leq W_1 + M$

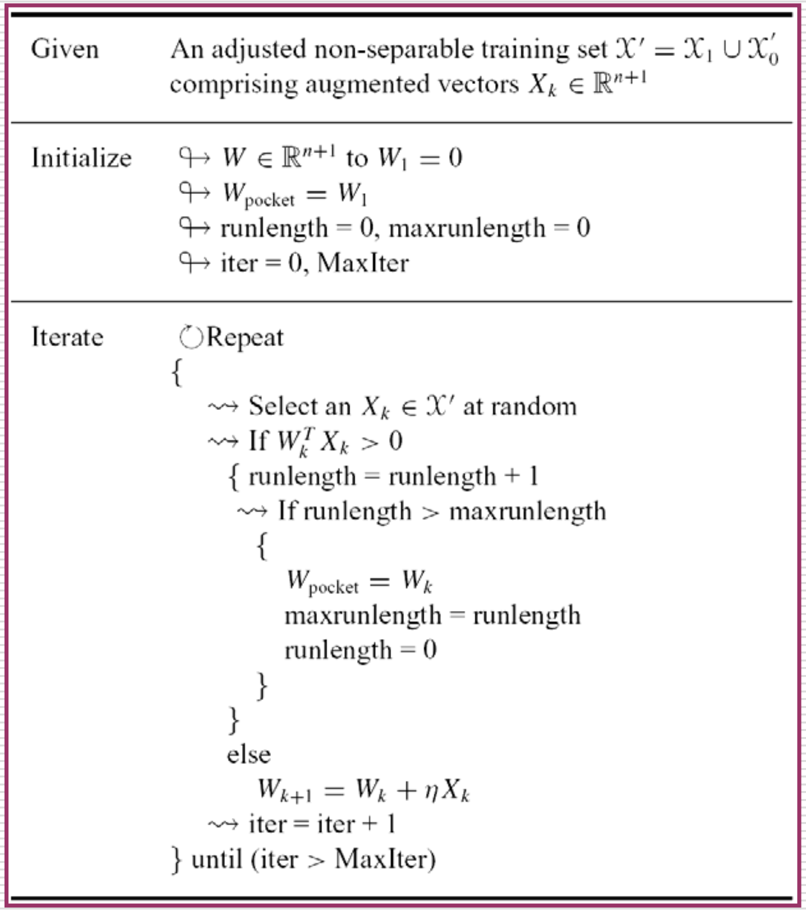
Two Corollaries

- If, in a finite set of training patterns X, each pattern X_k has integer (or rational) components x_i^k, then the Perceptron learning algorithm will visit a finite set of distinct weight vectors W_k.
- For a finite set of training patterns X, with individual patterns X_k having integer (or rational) components x_i^k the Perceptron learning algorithm will, in finite time, produce a weight vector that correctly classifies all training patterns iff X is linearly separable, or leave and re-visit a specific weight vector iff X is linearly non-separable.

Handling Linearly Non-separable Sets: The Pocket Algorithm

- Philosophy: Incorporate positive reinforcement in a way to reward weights that yield a low error solution.
- Pocket algorithm works by remembering the weight vector that yields the largest number of correct classifications on a consecutive run.
- This weight vector is kept in the “pocket”, and we denote it as W_{pocket} .
- While updating the weights in accordance with Perceptron learning, if a weight vector is discovered that has a longer run of consecutively correct classifications than the one in the pocket, it replaces the weight vector in the pocket.

Pocket Algorithm: Operational Summary



Given An adjusted non-separable training set $\mathcal{X}' = \mathcal{X}_1 \cup \mathcal{X}_0'$ comprising augmented vectors $X_k \in \mathbb{R}^{n+1}$

Initialize

```
↪ $W \in \mathbb{R}^{n+1}$ to $W_1 = 0$
↪ $W_{\text{pocket}} = W_1$
↪ runlength = 0, maxrunlength = 0
↪ iter = 0, MaxIter
```

Iterate

```
⟳Repeat
{
    ⇝ Select an $X_k \in \mathcal{X}'$ at random
    ⇝ If $W_k^T X_k > 0$
      { runlength = runlength + 1
        ⇝ If runlength > maxrunlength
          {
              $W_{\text{pocket}} = W_k$
              maxrunlength = runlength
              runlength = 0
          }
      }
      else
          $W_{k+1} = W_k + \eta X_k$
    ⇝ iter = iter + 1
} until (iter > MaxIter)
```

Pocket Convergence Theorem

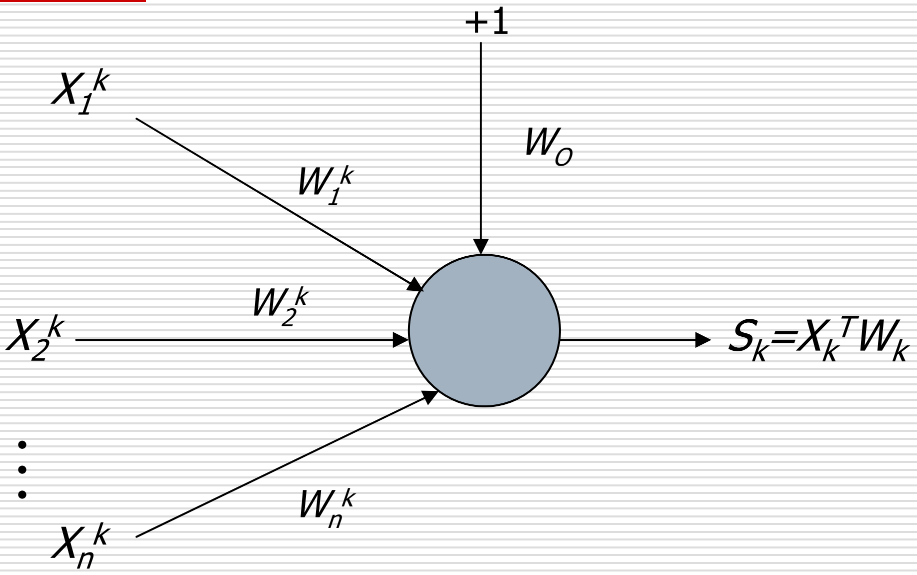
- Given a finite set of training examples, X, and a probability $p < 1$, there exists an integer k_0 such that after any $k > k_0$ iterations of the pocket algorithm, the probability that the pocket weight vector W_{pocket} is optimal exceeds p.

Linear Neurons and Linear Error

- Consider a training set of the form $T = \{X_k, d_k\}$, $X_k \in R^{n+1}$, $d_k \in R$.
- To allow the desired output to vary smoothly or continuously over some interval consider a linear signal function: $s_k = y_k = X_k^T W_k$
- The linear error e_k due to a presented training pair (X_k, d_k), is the difference between the desired output d_k andthe neuronal signal s_k: $e_k = d_k - s_k = d_k - X_k^T W_k$

Operational Details of α-LMS

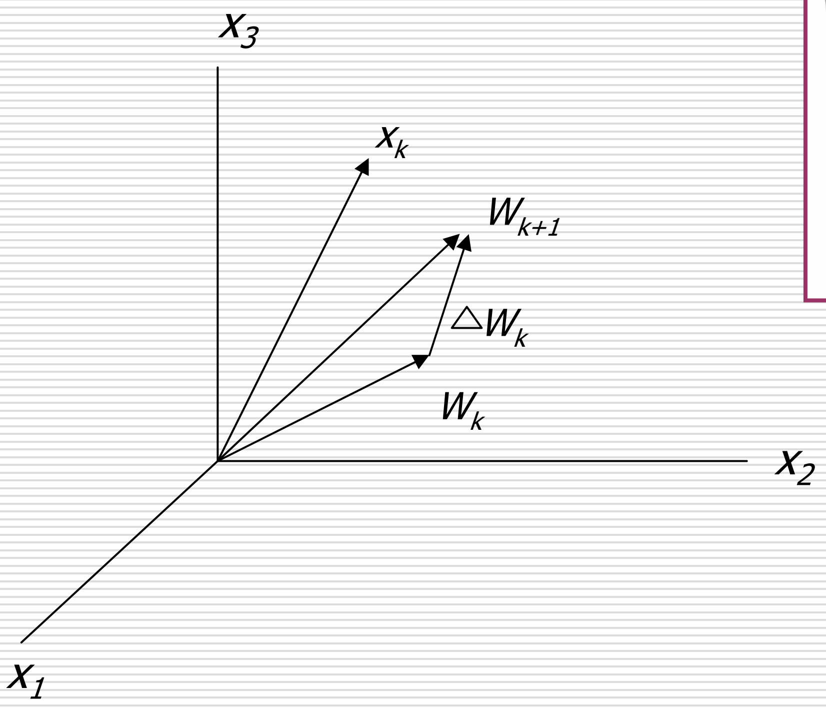
- α-LMS error correction is proportional to the error itself
- Each iteration reduces the error by a factor of η.
- η controls the stability and speed of convergence.
- Stability ensured if $0 < \eta < 2$.



$$W_{k+1} = W_k + \eta e_k \frac{X_k}{\|X_k\|^2}$$

$$\Delta W_k = \frac{\eta e_k}{\|X_k\|} \frac{X_k}{\|X_k\|} = \frac{\eta}{\|X_k\|} e_k \hat{X}_k = \hat{\eta}_k e_k \hat{X}_k$$

α-LMS Works with Normalized Training Patterns



$$W_{k+1} = W_k + \eta\left(\frac{d_k}{\|X_k\|} - W_k^T\frac{X_k}{\|X_k\|}\right)\frac{X_k}{\|X_k\|}$$
$$= W_k + \eta\left(\hat{d}_k - W_k^T\hat{X}_k\right)\hat{X}_k$$
$$= W_k + \eta\hat{e}_k\hat{X}_k$$

$$\hat{d}_k = \frac{d_k}{\|X_k\|}$$
$$\hat{X}_k = \frac{X_k}{\|X_k\|}$$

α-LMS: Operational Summary

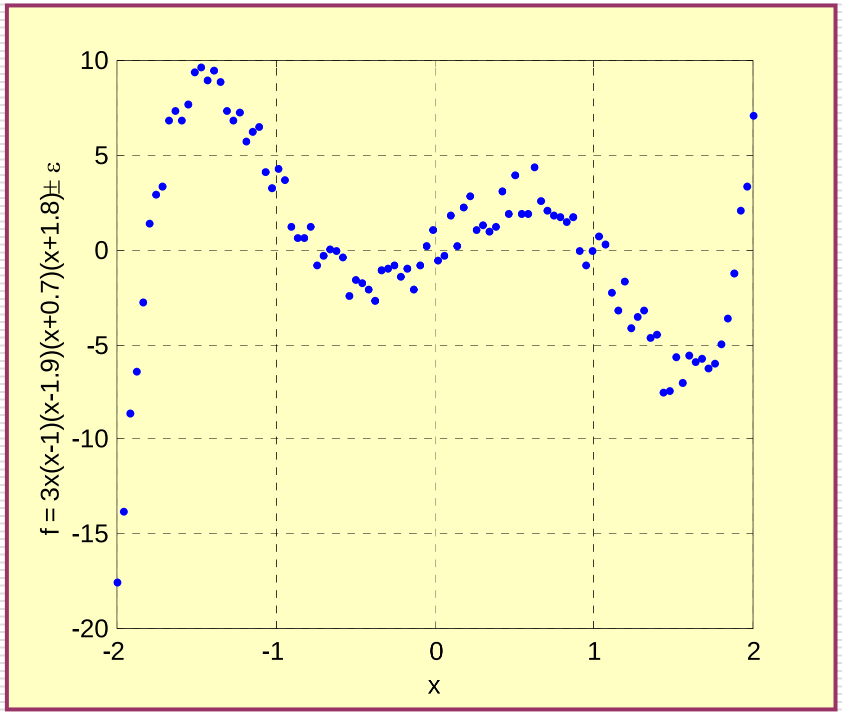
Given	A training set $\mathcal{T}$ comprising augmented vectors $X_k \in \mathbb{R}^{n+1}$ and corresponding desired real valued outputs $d_k \in \mathbb{R}$
Initialize	$\looparrowright$ $W \in \mathbb{R}^{n+1}$ some small random vector. $\looparrowright$ a weight change tolerance τ
Iterate	see below

Iterate

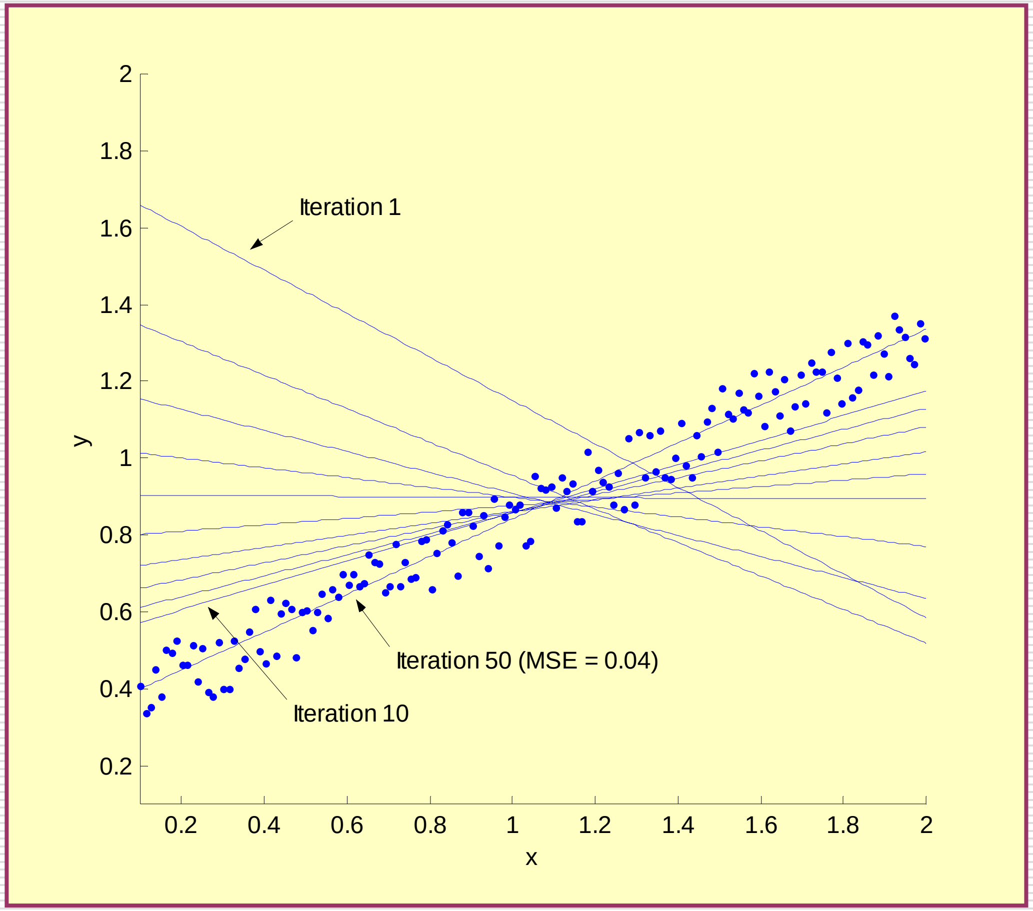
```
⟲Repeat (on an epoch by epoch basis)
{
    ⇝ Select an $X_k \in \mathcal{T}$ (in any sequence)
    ⇝ Compute $X_k^T W_k$
    ⇝ Compute $e_k = d_k - s_k$
    ⇝ Update weights $W_{k+1} = W_k + \eta e_k (X_k / \|X_k\|^2)$
} until (the average of $\Delta W_k$ over an epoch is less than $\tau$)
```

MATLAB Simulation Example

- Synthetic data set shown in the figure is generated by artificially scattering points around a straight line: $y = 0.5x + 0.333$ and generate a scatter of 200 points in a ±0.1 interval in the y direction.
- This is achieved by first generating a random scatter in the interval [0,1].
- Then stretching it to the interval [–1, 1], and finally scaling it to ±0.1



Computer Simulation of α-LMS Algorithm



MATLAB Program for α-LMS Learning

```
max_points =200;                     % Assume 200 data points
x =linspace(0,2.5,max_points); % Generate the x linspace
y =.5*x + 0.333;                     % Define a straight line
scatter =rand(1,max_points); % Generate scatter vector
ep =.1;                              % Compress scatter to 0.1
d =((2*scatter-1)*ep) + y;           % Set up desired values
eta =.01;                            % Set learning rate
w =3*(2*rand(1,2) - 1);              % Randomize weights
```

MATLAB Program for α-LMS Learning

```
for loop =1:50                     % Train for 50 epochs
   randindex =randperm(200); % Randomize order
   for j =1: max_points            % For each data point
      i =randindex(j);             % Get the index
      s(i) =w(1) + w(2)*x(i);      % Compute signal value
      err(i) =d(i) - s(i);         % Compute pattern error
      w(1) =w(1) + eta*err(i)/(1+x(i)^2);% Change the weights
     w(2) =w(2) + eta*err(i)*x(i)/(1+x(i)^2);
  end
end
```

A Stochastic Setting

- Assumption that the training set T is well defined in advance is incorrect when the setting is stochastic.
- In such a situation, instead of deterministic patterns, we have a *sequence* of samples $\{(X_k, d_k)\}$ assumed to be drawn from a *statistically stationary* population or process.
- For adjustment of the neuron weights in response to some pattern-dependent error measure, the error computation has to be based on the expectation of the error over the ensemble.

Definition of Mean Squared Error (MSE)

- We introduce the square error on a pattern X_k as

$$\varepsilon_k = \frac{1}{2}\left(d_k - X_k^T W_k\right)^2 = \frac{1}{2}e_k^2 \qquad (1)$$

$$= \frac{1}{2}\left(d_k^2 - 2d_k X_k^T W_k + W_k^T X_k X_k^T W_k\right) \qquad (2)$$

- Assumption: The weights are held fixed at W_k while computing the expectation.
- The mean-squared error can now be computed by taking the expectation on both sides of (2):

$$\varepsilon = E[\varepsilon_k] = \frac{1}{2}E\left[d_k^2\right] - E\left[d_k X_k^T\right]W_k + \frac{1}{2}W_k^T E\left[X_k X_k^T\right]W_k \qquad (3)$$

Our problem…

- To find optimal weight vector that minimizes the mean-square error.

Cross Correlations

- For convenience of expression we define the pattern vector P as the cross-correlation between the desired scalar output, d_k, and the input vector, X_k

$$P^T \triangleq \mathrm{E}\left[\mathrm{d_k} X_k^T\right] = E\left[\left(d_k, d_k x_1^k, \cdots, d_k x_n^k\right)\right] \qquad (4)$$

- and the input correlation matrix, R, as

$$R \triangleq \mathrm{E}\left[X_\mathrm{k} X_k^T\right] = E\begin{bmatrix} 1 & x_1^k & \cdots & x_n^k \\ x_1^k & x_1^k x_1^k & \cdots & x_1^k x_n^k \\ \vdots & \vdots & \ddots & \vdots \\ x_n^k & x_n^k x_1^k & & x_n^k x_n^k \end{bmatrix} \qquad (5)$$

Cross Correlations (contd.)

- Using Eqns. (4)-(5), we rewrite the MSE expression of Eqn. (3) succinctly as

$$\varepsilon = E[\varepsilon_k] = \frac{1}{2}E[d_k^2] - P^T W_k + \frac{1}{2}W_k^T R W_k \qquad (6)$$

- Note that since the MSE is a quadratic function of the weights it represents a bowl shaped surface in the (n+1) x 1 weight—MSE cross space.

Finding the Minimum Error

- First compute the gradient by straightforward differentiation which is a linear function of weights

$$\nabla\varepsilon = \left(\frac{\partial\varepsilon}{\partial w_0}, \cdots, \frac{\partial\varepsilon}{\partial w_n}\right)^T = -P + RW \qquad (7)$$

- To find the optimal set of weights, $\hat{W}$, simply set $\nabla\varepsilon = 0$ which yields $R\hat{W} = P \qquad (8)$
- This system of equations (8) is called the *Weiner-Hopf system* and its solution is the Weiner solution or the Weiner filter

$$\hat{W} = R^{-1}P \qquad (9)$$

- is the point in weight space that represents the minimum mean-square error $\varepsilon_{\min}$

Computing the Optimal Filter

- First compute R^{-1} and P. Substituting from Eqn. (9) into Eqn. (6) yields:

$$\varepsilon_{\min} = \frac{1}{2}E\left[d_k^2\right] + \frac{1}{2}\hat{W}^T R\hat{W} - P^T\hat{W} \qquad (10)$$

$$= \frac{1}{2}E\left[d_k^2\right] + \frac{1}{2}\left(R^{-1}P\right)^T R\left(R^{-1}P\right) - P^T R^{-1}P \qquad (11)$$

$$= \frac{1}{2}E\left[d_k^2\right] - \frac{1}{2}P^T\hat{W} \qquad (12)$$

- For the treatment of weight update procedures we reformulate the expression for mean-square error in terms of the deviation $V = W - \hat{W}$, of the weight vector from the Weiner solution.

Computing R

- Substituting $W = V + \hat{W}$ into Eqn. (6)

$$\varepsilon = E\left[d_k^2\right] + \frac{1}{2}\left(V + \hat{W}\right)^T R\left(V + \hat{W}\right) - P^T\left(V + \hat{W}\right) \qquad (13)$$

$$= E\left[d_k^2\right] + \frac{1}{2}\left(V^T R V + \underbrace{V^T R\hat{W} + \hat{W}^T R V}_{2\hat{W}^T RV = 2P^T R^{-1} RV = 2P^T V} + \hat{W}^T R\hat{W}\right) - P^T V - P^T \hat{W}$$

$$(14)$$

$$= \varepsilon_{\min} + \frac{1}{2} V^T R V \qquad (15)$$

$$= \varepsilon_{\min} + \frac{1}{2}\left(W - \hat{W}\right)^T R\left(W - \hat{W}\right)^T R\left(W - \hat{W}\right) \qquad (16)$$

- Note that since the mean-square error ε is non-negative, we must have $V^T R V \geq 0$. This implies that R is positive semi-definite. Usually however, R is positive definite.

Diagonalization of R

- Assume that R has distinct eigenvalues λ_i. Then we can construct a matrix Q whose columns are corresponding eigenvectors η_i of R.

$$Q = \begin{pmatrix} \eta_0 & \eta_1 & \cdots & \eta_n \end{pmatrix} \qquad (17)$$

- R can be diagonalized using an *orthogonal similarity transformation* as follows. Having constructed Q, and knowing that:

We have

$$RQ = \begin{pmatrix} \eta_0 & \eta_1 & \cdots & \eta_n \end{pmatrix} \begin{pmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & & \lambda_n \end{pmatrix} \qquad (18)$$

$$Q^{-1}RQ = D = diag\{\lambda_0, \lambda_1, \lambda_2, \cdots, \lambda_n\} \qquad (19)$$

Some Observations

- It is usual to choose the eigenvectors of R to be orthonormal $QQ^T = I$ and $Q^{-1} = Q^T$.
- Then $R = QDQ^T = QDQ^{-1}$
- From Eqn. (15) we know that the shape of ε is a bowl-shaped paraboloid with a minimum at the Weiner solution (V=0, the origin of V-space).
- Slices of ε parallel to the W space yield elliptic constant error contours which define the weights in weight space that the specific value of the square-error (say ε_c) at which the slice is made:

$$\frac{1}{2}V^T RV = \varepsilon_c - \varepsilon_{\min} = \text{constant} \qquad (20)$$

Eigenvectors of R

- Also note from Eqn. (15) that we can compute the MSE gradient in V-space as,

$$\nabla \varepsilon = RV \qquad (21)$$

which defines a family of vectors in V-space.

- Exactly n+1 of these pass through the origin of V-space and these are the principal axes of the ellipse.

- However, vectors passing through the origin must take the λV. Therefore, for the principal axes V',

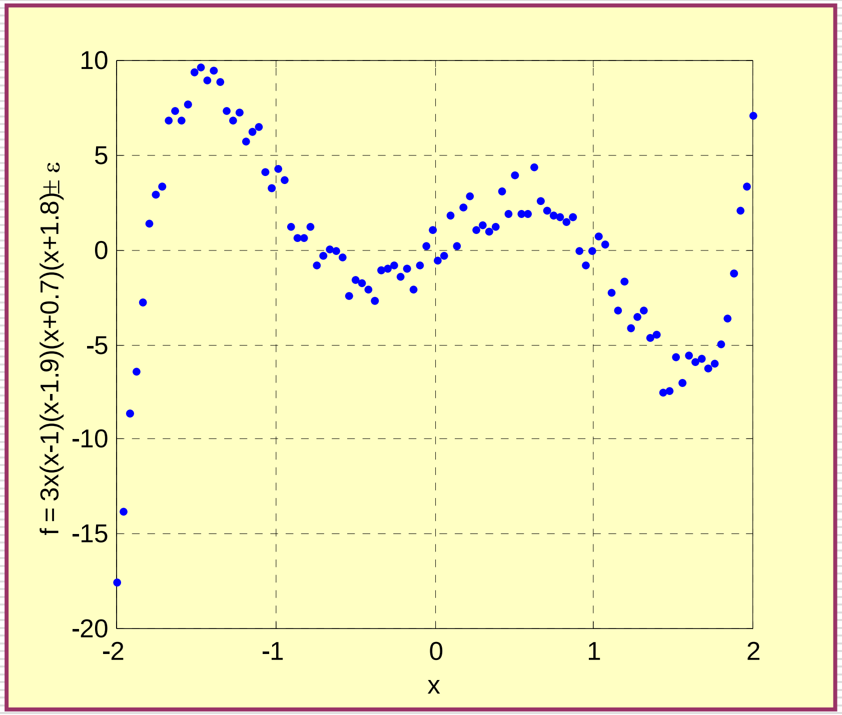
$$RV' = \lambda V' \qquad (22)$$

- Clearly, V' is an eigenvector of R.

MATLAB Simulation Example

- We use a data scatter of 100 sample points that describes a fifth order function
- The points are scattered around the function in a ±1 interval.

$$f(x) = 3x(x-1)(x-1.9)(x+0.7)(x+1.8) \qquad (23)$$



MATLAB Simulation: Computing the MSE Surface

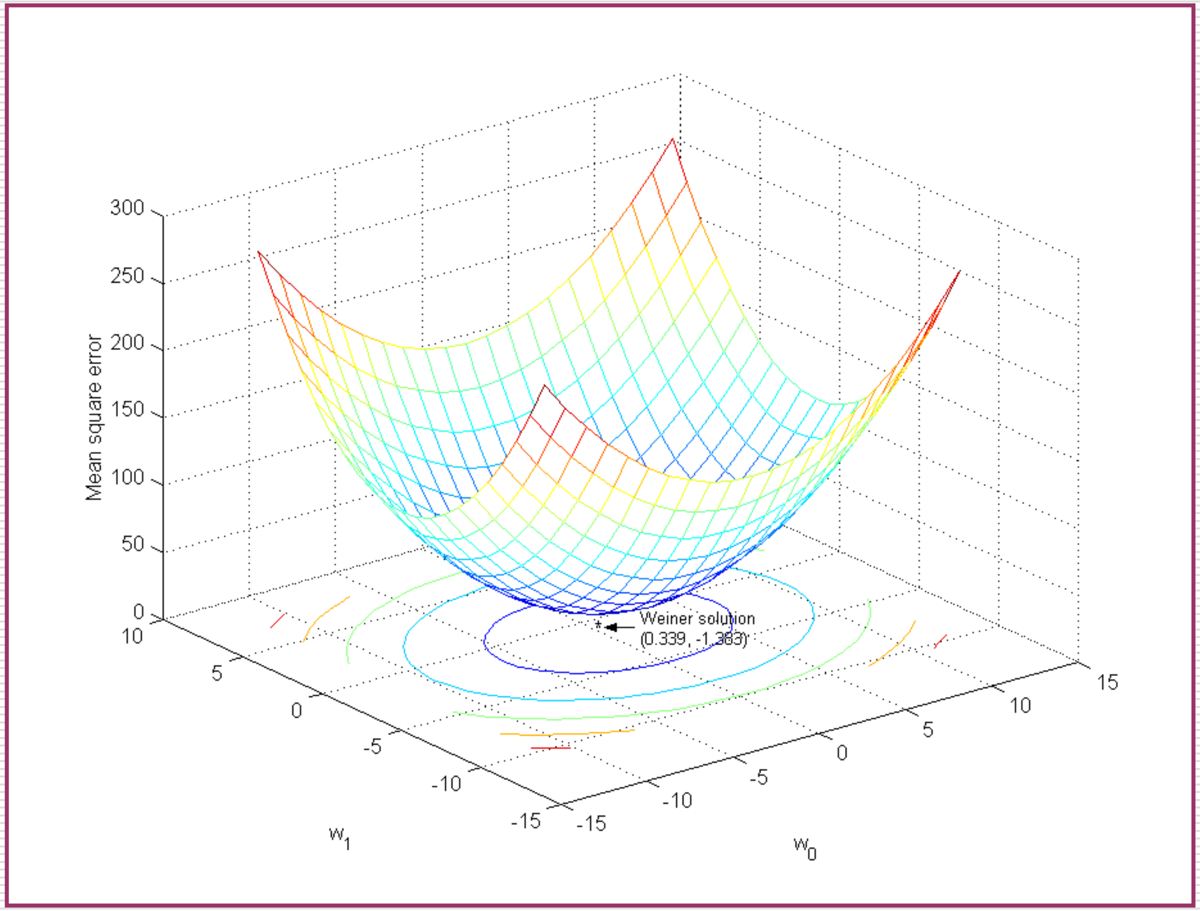
- First calculate the correlation matrix *R*, the cross-correlation P, $E\left[d^2\right]$, and the Weiner solution. These are straightforward to compute since the data set is deterministic.
- For the pattern set under consideration, $E\left[d^2\right] = 859.59$,and the Weiner solution can be computed to be

$$R = \begin{pmatrix} 1 & 0 \\ 0 & 1.36 \end{pmatrix} \qquad (24)$$

$$P = \begin{pmatrix} 0.3386 \\ -1.8818 \end{pmatrix} \qquad (25)$$

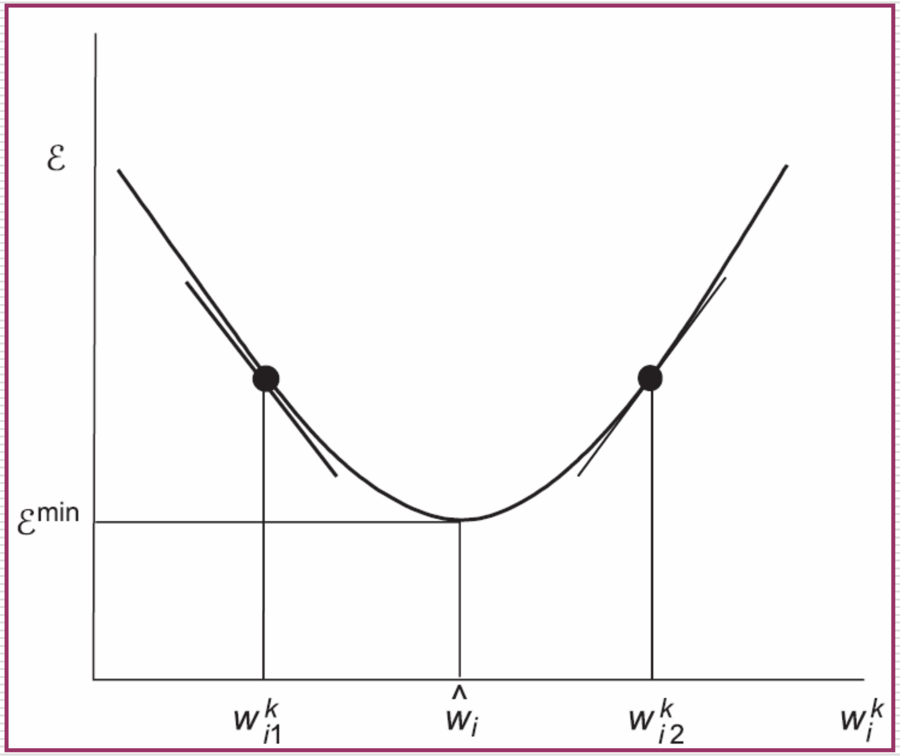
$$\hat{W} = \begin{pmatrix} 0.3386 \\ -1.3834 \end{pmatrix} \qquad (26)$$

MSE Surface in the Vicinity of Weiner Solution



Steepest Descent Search with Exact Gradient Information

- Steepest descent search uses exact gradient information available from the mean-square error surface to direct the search in weight space.
- The figure shows a projection of the square-error function on the $\varepsilon - w_i^k$ plane.



Steepest Descent Procedure Summary

- Provide an appropriate weight increment to w_i^k to push the error towards the minimum which occurs at $\hat{w}_i$.
- Perturb the weight in a direction that depends on which side of the optimal weight $\hat{w}_i$ the current weight value w_i^k lies.
- If the weight component w_i^k lies to the left of $\hat{w}_i$, say at w_{i1}^k, where the error gradient is negative (as indicated by the tangent) we need to increase w_i^k.
- If w_i^k is on the right of $\hat{w}_i$, say at w_{i2}^k where the error gradient is positive, we need to decrease $\hat{w}_i$.
- This rule is summarized in the following statement:

$$\text{If } \frac{\partial \varepsilon}{\partial w_i^k} > 0, \left(w_i^k > \hat{w}_i\right), \underline{\text{decrease}}\ w_i^k$$

$$\text{If } \frac{\partial \varepsilon}{\partial w_i^k} < 0, \left(w_i^k < \hat{w}_i\right), \underline{\text{increase}}\ w_i^k$$

Weight Update Procedure

- It follows logically therefore, that the weight component should be updated in proportion with the *negative* of the gradient:

$$w_i^{k+1} = w_i^k + \eta\left(-\frac{\partial \varepsilon}{\partial w_i^k}\right) \qquad \mathrm{i} = 0,1,\ldots,\mathrm{n} \qquad (27)$$

- Vectorially we may write

$$W_{k+1} = W_k + \eta\left(-\nabla \varepsilon\right) \qquad (28)$$

where we now re-introduced the iteration time dependence into the weight components:

$$\nabla \varepsilon = \left(\frac{\partial \varepsilon}{\partial w_0^k}, \ldots, \frac{\partial \varepsilon}{\partial w_n^k}\right)^T \qquad (29)$$

- Equation (28) is the steepest descent update procedure. Note that steepest descent uses exact gradient information at each step to decide weight changes.

Convergence of Steepest Descent - 1

- Question: What can one say about the stability of the algorithm? Does it converge for all values of η ?
- To answer this question consider the following series of subtitutions and transformations. From Eqns. (28) and (21)

$$
\begin{aligned}
W_{k+1} &= W_k + \eta(-\nabla\varepsilon) && (30)\\
&= W_k - \eta R V_k && (31)\\
&= W_k + \eta R(\hat{W} - W_k) && (32)\\
&= (1 - \eta R)W_k + \eta R \hat{W} && (33)
\end{aligned}
$$

- Transforming Eqn. (33) into V-space (the principal coordinate system) by substitution of $V_k + \hat{W}$ for W_k yields:

$$V_{k+1} = (I - \eta R)V_k \qquad (34)$$

Steepest Descent Convergence - 2

- Rotation to the principal axes of the elliptic contours can be effected by using $V = QV'$:

$$QV'_{k+1} = (I - \eta R) QV'_k \qquad (35)$$

or

$$V'_{k+1} = Q^{-1}(I - \eta R) QV'_k \qquad (36)$$

$$= (I - \eta D) V'_k \qquad (37)$$

- where D is the diagonal eigenvalue matrix. Recursive application of Eqn. (37) yields:

$$V'_k = (I - \eta D)^k V'_0 \qquad (38)$$

- It follows from this that for stability and convergence of the algorithm:

$$\lim_{k \to \infty} (I - \eta D)^k = 0 \qquad (39)$$

Steepest Descent Convergence - 3

This requires that $\lim_{k\to\infty}\left(1-\eta\lambda_{\max}\right)^k = 0$ (40)

or $$0 < \eta < \frac{2}{\lambda_{\max}} \quad (41)$$

$\lambda_{\max}$ being the largest eigenvalue of R.

If this condition is satisfied then we have

$$\lim_{k\to\infty} V_k' = \lim_{k\to\infty} Q^{-1}\left(W_k - \hat{W}\right) = 0 \quad (42)$$

or $$\lim_{k\to\infty} W_k = \hat{W} \quad (43)$$

- Steepest descent is guaranteed to converge to the Weiner solution as long as the learning rate is maintained within the limits defined by Eqn. (41).

Computer Simulation Example

- This simulation example employs the fifth order function data scatter with the data shifted in the y direction by 0.5. Consequently, the values of R,P and the Weiner solution are respectively:

$$R = \begin{pmatrix} 1 & 0.500 \\ 0.500 & 1.61 \end{pmatrix}, \; P = \begin{pmatrix} 0.8386 \\ -1.4625 \end{pmatrix} \qquad (44)$$

$$\hat{W} = \begin{pmatrix} 1.5303 \\ -1.3834 \end{pmatrix} \qquad (45)$$

- Exact gradient information is available since the correlation matrix R and the cross-correlation matrix P are known.
- The weights are updated using the equation:

$$W_{k+1} = W_k + 2\eta R\left(\hat{W} - W_k\right) \qquad (46)$$

MATLAB Simulation Example

```
eta = .01;                                  % Set learning rate
R=zeros(2,2);                               % Initialize correlation matrix
X = [ones(1,max_points);x];                 % Augment input vectors

P = (sum([d.*X(1,:); d.*X(2,:)],2))/max_points;  % Cross-correlations
D = (sum(d.^2))/max_points; %squares;            % target expectations

for k =1:max_points
  R = R + X(:,k)*X(:,k)';                   % Compute R
end

R = R/max_points;
weiner=inv(R)*P;                            % Compute the Weiner solution
errormin = D - P'*inv(R)*P;                 % Find the minimum error
```

MATLAB Simulation Example (contd.)

```
shift1 = linspace(-12,12, 21);             % Generate a weight space matrix
shift2 = linspace(-9,9, 21);
for i = 1:21                               % Compute a weight matrix about
  shiftwts(1,i) =  weiner(1)+shift1(i);    % Weiner solution
  shiftwts(2,i) =  weiner(2)+shift2(i);
end

for i=1:21                                 % Compute the error matrix
  for j = 1:21                             % to plot the error contours
    error(i,j) = sum((d - (shiftwts(1,i) + x.*shiftwts(2,j))).^2);
  end
end
error = error/max_points;

figure; hold on;                           % Plot the error contours
plot(weiner(1),weiner(2),'*k')             % Labelling statements no shown
```

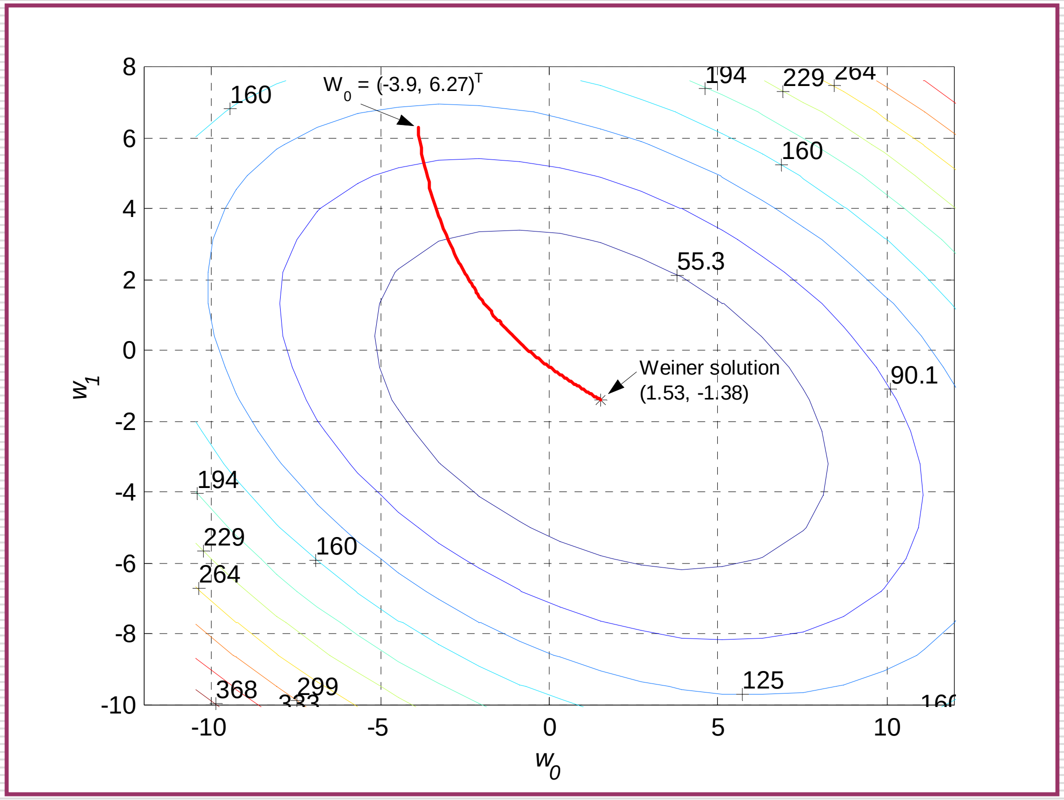
MATLAB Simulation Example (Contd.)

```
w = 10*(2*rand(2,1)-1);                  % Randomize weights
w0 = w;                                  % Remember the initial weights

for loop = 1:500                         % Perform descent for 500 iters
  w = w + eta*(-2*(R*w-P));
  wts1(loop)=w(1); wts2(loop)=w(2);
End

                                         % Set up weights for plotting
wts1=[w0(1) wts1]; wts2=[w0(2) wts2];

plot(wts1,wts2,'r')                      % Plot the weight trajectory
```

Smooth Trajectory towards the Weiner Solution

$$W_0 = (-3.9039, 6.2768)^T$$

- Steepest descent uses exact gradient information to search the Weiner solution in weight space.



μ-LMS: Approximate Gradient Descent

- The problem with steepest descent is that true gradient information is only available in situations where the data set is completely specified in advance.

- It is then possible to compute R and P exactly, and thus the true gradient at iteration
$$k : \nabla \varepsilon = RW_k - P$$

- However, when the data set comprises a random stream of patterns (drawn from a stationary distribution), R and P cannot be computed accurately. To find a correct approximation one might have to examine the data stream for a reasonably large period of time and keep averaging out.

- How long should we examine the stream to get reliable estimates of R and P ?

Definition

- The μ-LMS algorithm is *convergent in the mean* if the average of the weight vector W_k approaches the optimal solution W_{opt} as the number of iterations k, approaches infinity:
  - $E[W_k] \rightarrow W_{opt}$ as $k \rightarrow \infty$

μ-LMS employs ε_k for $\varepsilon = E[\varepsilon_k]$

- The gradient computation modifies to:

$$\tilde{\nabla}\varepsilon_k = \left(\frac{\partial \varepsilon_k}{\partial w_0^k}, \ldots \frac{\partial \varepsilon_k}{\partial w_n^k}\right)^T = e_k\left(\frac{\partial e_k}{\partial w_0^k}, \ldots, \frac{\partial e_k}{\partial w_n^k}\right)^T = -e_k X_k \qquad (47)$$

- where $e_k = (d_k - s_k)$, and $s_k = X_k^T W_k$ since we are dealing with linear neurons. Note therefore that the recursive update equation then becomes

$$W_{k+1} = W_k + \eta(-\tilde{\nabla}\varepsilon_k) = W_k + \eta(d_k - s_k)X_k \qquad (48)$$

$$= W_k + \eta e_k X_k \qquad (49)$$

- What value does the long term average of $\tilde{\nabla}\varepsilon_k$ converge to? Taking the Expectation of both sides of Eqn. (47):

$$E\left[\tilde{\nabla}\varepsilon_k\right] = -E\left[e_k X_k\right] = -E\left[d_k X_k - X_k X_k^T W\right] \qquad (50)$$

$$= RW - P \qquad (51)$$

$$= \nabla\varepsilon \qquad (52)$$

Some Observations

- Since the long term average of $\tilde{\nabla}\varepsilon_k$ approaches $\nabla\varepsilon$, we can safely $\tilde{\nabla}\varepsilon_k$ use as an unbiased estimate. That's what makes μ-LMS work!

- Since $\tilde{\nabla}\varepsilon_k$ approaches $\nabla\varepsilon$ in the long run, one could keep collecting $\tilde{\nabla}\varepsilon_k$ for a sufficiently large number of iterations (while keeping the weights fixed), and then make a weight change collectively for all those iterations together.

- If the data set is finite (deterministic), then one can compute $\nabla\varepsilon$ accurately by first collecting the different $\tilde{\nabla}\varepsilon_k$ gradients over all training patterns X_k for the same set of weights. This accurate measure of the gradient could then be used to change the weights. In this situation μ-LMS is identical to the steepest descent algorithm.

Observations Contd.

- Even if the data set is deterministic, we still use $\tilde{\nabla}\varepsilon_k$ to update the weights. After all if the data set becomes large, collection of all the gradients becomes expensive in terms of storage. Much easier to just go ahead and use $\tilde{\nabla}\varepsilon_k$!

- Be clear about the approximation made: we are estimating the true gradient (which should be computed from $E[\varepsilon_k]$) by a gradient computed from the instantaneous sample error ε_k . Although this may seem to be a rather drastic approximation, it works.

- In the deterministic case we can justify this as follows: if the learning rate η , is kept small, the weight change in each iteration will be small and consequently the weight vector W will remain "somewhat constant" over Q iterations where Q is the number of patterns in the training set.

Observations Contd.

- Of course this is provided that Q is a small number! To see this, observe the total weight change ∇W, over Q iterations from the k^{th} iteration:

$$\nabla W = -\sum_{i=0}^{Q-1} \frac{\partial \varepsilon_{k+i}}{\partial W_{k+i}} \tag{53}$$

$$= -Q\left(\frac{1}{Q} \sum_{i=0}^{Q-1} \frac{\partial \varepsilon_{k+i}}{\partial W_k} \right) \tag{54}$$

$$= -Q \frac{\partial}{\partial W_k} \left(\frac{1}{Q} \sum_{i=0}^{Q-1} \varepsilon_{k+i} \right) \tag{55}$$

$$= -Q \frac{\partial \varepsilon}{\partial W_k} \tag{56}$$

- Where ε denotes the mean-square error. Thus the weight updates follow the true gradient *on average.*

Observations Contd.

- Observe that steepest descent search is guaranteed to search the Weiner solution provided the learning rate condition (41) is satisfied.

- Now since $\tilde{\nabla}\varepsilon_k$ is an unbiased estimate of $\nabla\varepsilon$ one can expect that μ-LMS too will search out the Weiner solution. Indeed it does—but not smoothly. This is to be expected since we are only using an estimate of the true gradient for immediate use.

- Although α-LMS and μ- LMS are similar algorithms, α-LMS works on the normalizing training set. What this simply means is that α-LMS also uses gradient information, and will eventually search out the Weiner solution-of the normalized training set. However, in one case the two algorithms are identical: the case when input vectors are bipolar. (Why?)

μ-LMS Algorithm: Convergence in the Mean (1)

- **Definition 0.1** The μ-LMS algorithm is convergent in the mean if the average of the weight vector W_k approaches the optimal solution $\hat{W}$ as the number of iterations k, approaches infinity:

$$E[W_k] \to \hat{W} \text{ as } \mathrm{k} \to \infty \tag{57}$$

- **Definition 0.2** The μ-LMS algorithm is convergent in the mean square if the average of the squared error ε_k approaches a constant as the Number of iterations, k, approaches infinity:

$$E[\varepsilon_k] \to \text{constant as } k \to \infty \tag{58}$$

- Convergence in the mean square is a stronger criterion than convergence in the mean. In this section we discuss convergence in the mean and merely state the result for convergence in the mean square.

μ-LMS Algorithm: Convergence in the Mean (2)

- Consider the μ-LMS weight update equation:

$$W_{k+1} = W_k + \eta(d_k - s_k)X_k \qquad (59)$$
$$= W_k + \eta(d_k - X_k^T W_k)X_k \qquad (60)$$
$$= W_k + \eta d_k X_k - \eta X_k^T W_k X_k \qquad (61)$$
$$= W_k - \eta X_k X_k^T W_k + \eta d_k X_k \qquad (62)$$
$$= (\mathrm{I} - \eta X_k X_k^T) W_k + \eta d_k X_k \qquad (63)$$

- Taking the expectation of both sides of Eqn. (63) yields,

$$E[W_{k+1}] = (I - \eta E[X_k X_k^T])E[W_k] + \eta E[d_k X_k] \qquad (64)$$
$$= (I - \eta R)E[W_k] + \eta P \qquad (65)$$

- Where P and R are as already defined

μ-LMS Algorithm: Convergence in the Mean (3)

- Appropriate substitution yields:

$$\begin{aligned} E[W_{k+1}] &= \left(I-\eta QDQ^T\right)E[W_k]+\eta QDQ^T\hat{W} \\ &= \left(QQ^T-\eta QDQ^T\right)E[W_k]+\eta QDQ^T\hat{W} \qquad (66) \\ &= Q(I-\eta D)Q^T E[W_k]+\eta QDQ^T\hat{W} \end{aligned}$$

- Pre-multiplication throughout by Q^T results in:

$$Q^T E[W_{k+1}] = (I-\eta D)Q^T E[W_k]+\eta DQ^T\hat{W} \qquad (67)$$

- And subtraction of $Q^T\hat{W}$ from both sides gives:

$$Q^T\left(E[W_{k+1}]-\hat{W}\right) = (I-\eta D)Q^T E[W_k]-(I-\eta D)Q^T\hat{W} \qquad (68)$$

$$= (I-\eta D)Q^T\left(E[W_k]-\hat{W}\right) \qquad (69)$$

- We will re-write Eqn. (69) in familiar terms:

$$Q^T\tilde{V}_{k+1} = (I-\eta D)Q^T\tilde{V}_k \qquad (70)$$

μ-LMS Algorithm: Convergence in the Mean (4)

And

$$\tilde{V}'_{k+1} = (I - \eta D)\tilde{V}'_k \qquad (71)$$

Where $\tilde{V}_k = E[W_k] - \hat{W}$ and $\tilde{V}'_k = Q^T \tilde{V}_k$. Eqn. (71) represents a set of n+1 *Decoupled difference equations*:

$$\tilde{v}'^{k+1}_i = (1 - \eta\lambda_i)\tilde{v}'^{k}_i \qquad \text{i} = 0,1,\ldots,\text{n}. \qquad (72)$$

Recursive application of Eqn. (72) yields,

$$\tilde{v}'^{k}_i = (1 - \eta\lambda_i)^k \tilde{v}'^{0}_i \qquad \text{i} = 0,1,\ldots,\text{n} \qquad (73)$$

- To ensure convergence in the mean, $\tilde{v}'^{k}_i \rightarrow 0$ as $k \rightarrow \infty$ since this condition requires that the deviation of $E[W_k]$ from $\hat{W}$ should tend to 0.

μ-LMS Algorithm: Convergence in the Mean (5)

- Therefore from Eqn. (73):

$$|1-\eta\lambda_i|<1 \qquad \mathrm{i}=0,1,\ldots,\mathrm{n}. \qquad (74)$$

- If this condition is satisfied for the largest eigenvalue $\lambda_{\max}$ then it will be satisfied for all other eigenvalues. We therefore conclude that if

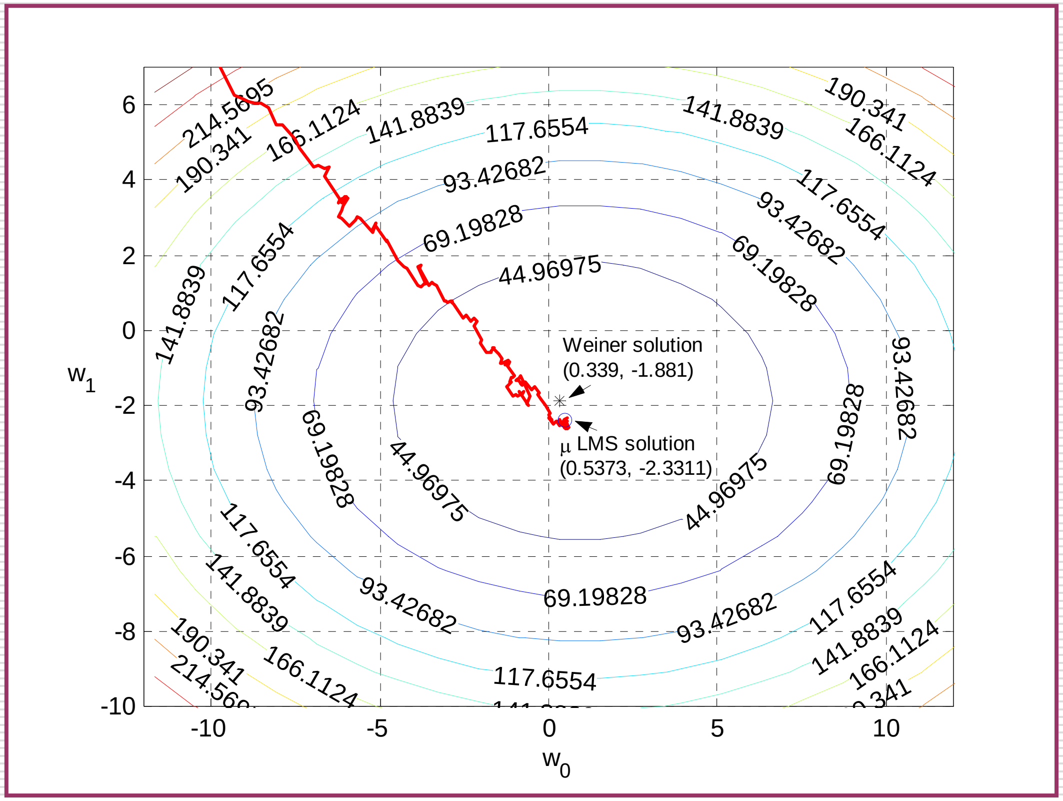
$$0<\eta<\frac{2}{\lambda_{\max}} \qquad (75)$$

  then the μ-LMS algorithm is convergent in the mean.
- Further, since $\mathrm{tr}[R] = \sum_{i=0}^{n}\lambda_i \ge \lambda_{\max}$ (where $\mathrm{tr}[R]$ is the trace of R) convergence is assured if

$$0<\eta<\frac{2}{tr[R]} \qquad (76)$$

Random Walk towards the Weiner Solution

- Assume the familiar fifth order function
- μ-LMS uses a local estimate of gradient to search the Weiner solution in weight space.



Computer Simulation Example

- μ-LMS learning assumes that the data stream is stochastic (with stationary properties).
- To simulate this data stream generate data points on the fly---within the weight update loop.
- The correlation and cross-correlation matrices R and P are computed by averaging over a sufficiently large stream of data points.
- The trajectory makes the expected *random walk* towards the Weiner solution starting out from .

$$W_0 = (-14.096, 13.611)^T$$

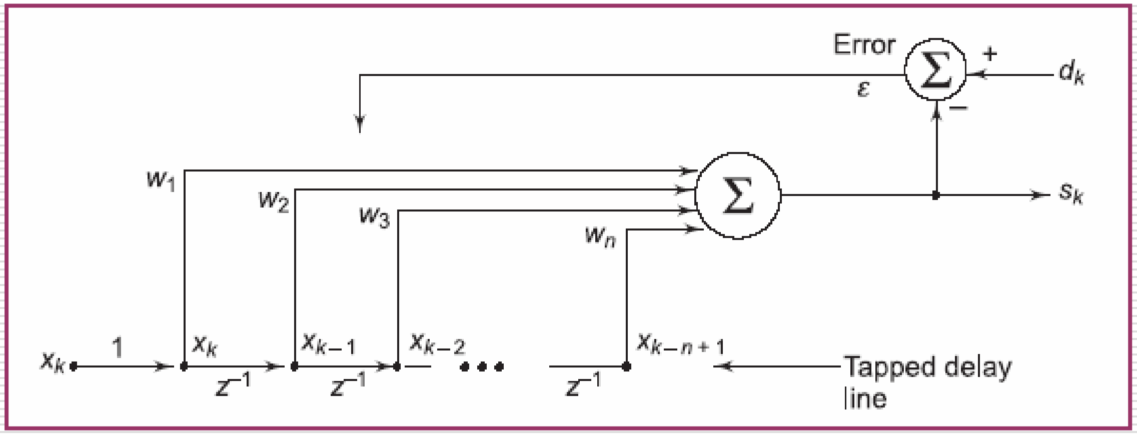
MATLAB Code Segment

```
for loop = 1:200
  x = (2*rand-1)*2;
  X = [1;x];
  y = 3.*x.^5 - 1.2.*x.^4 - 12.27.*x.^3 + 3.288.*x.^2
      + 7.182.*x;

  scatter = (2*rand-1)*eps;
  d = y + scatter;
  w = w + 2*eta*(d -X'*w)*X;
  wts1(loop)=w(1);
  wts2(loop)=w(2);
end
```

MATLAB Code Notes

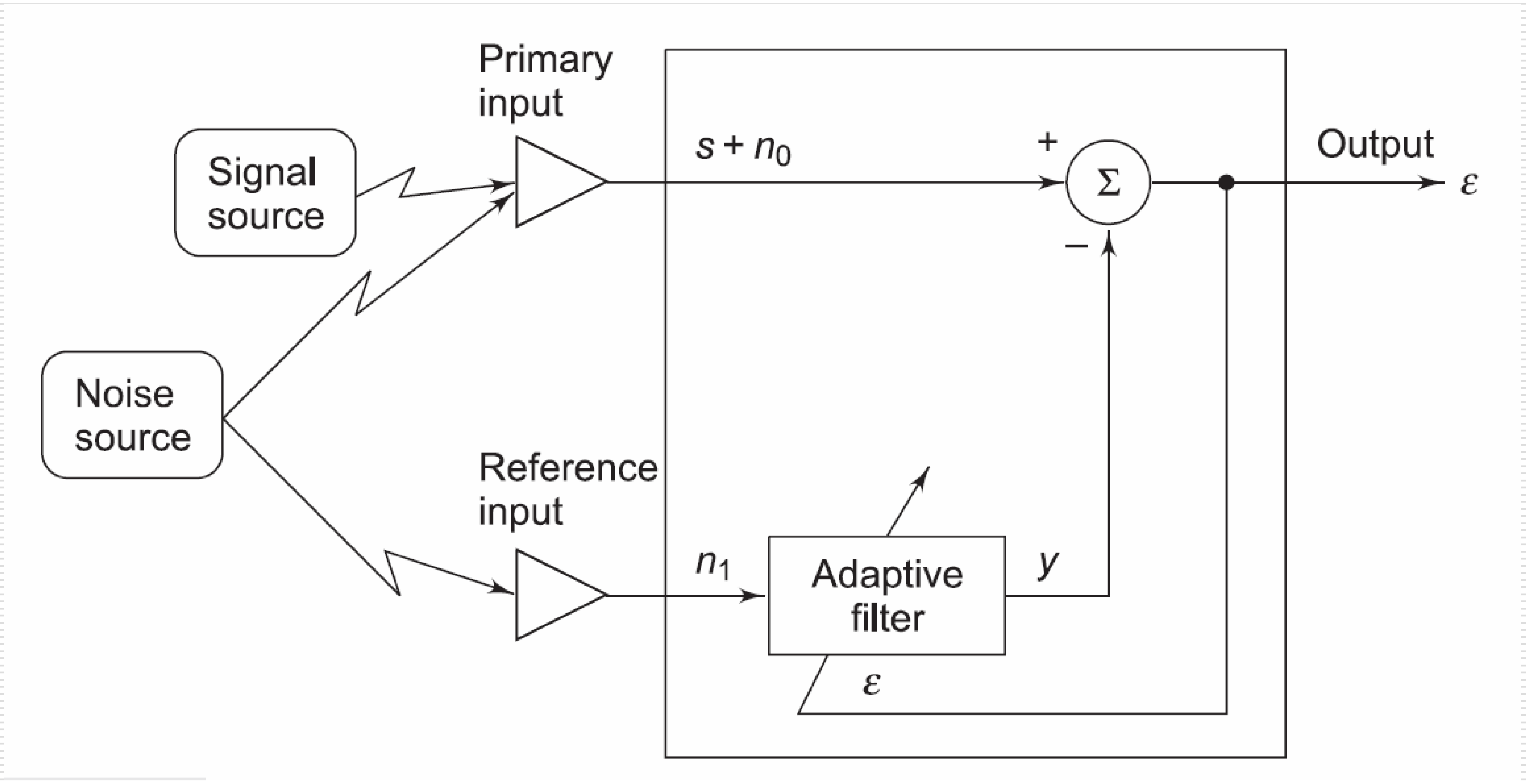
- The rest of the program is very similar to the code for steepest descent search.
- The source function for the steepest descent simulation example was shifted vertically by 0.5 (resulting in a change in the orientation of the error contours).
- Notice how the data points are now generated on-the-fly to simulate the stochastic stream of training data.
- Since the synthetic source function and the scatter generation functions are fixed, the stream has stationary stochastic properties.
- The values of R and P were pre-computed by averaging over a stream of 10,000 points.

Application of LMS Tapped Delay Line Filters



- Sampled input is delayed through a series of delay elements.
- These n signal samples (including the current one) are input to the *adaptive linear combiner* (ALC)
- Output is the inner product $y_k = X_k^T W_k$, where $X = (x_k, x_{k-1}, \ldots, x_{k-n+1})^T$ and $W = (w_1, \ldots, w_n)^T$.
- LMS procedure employed to adjust the weights so that the output matches the desired response.

Adaptive Noise Cancellation



Adaptive Noise Cancellation: Removal Of Noise n_0 From Signal s

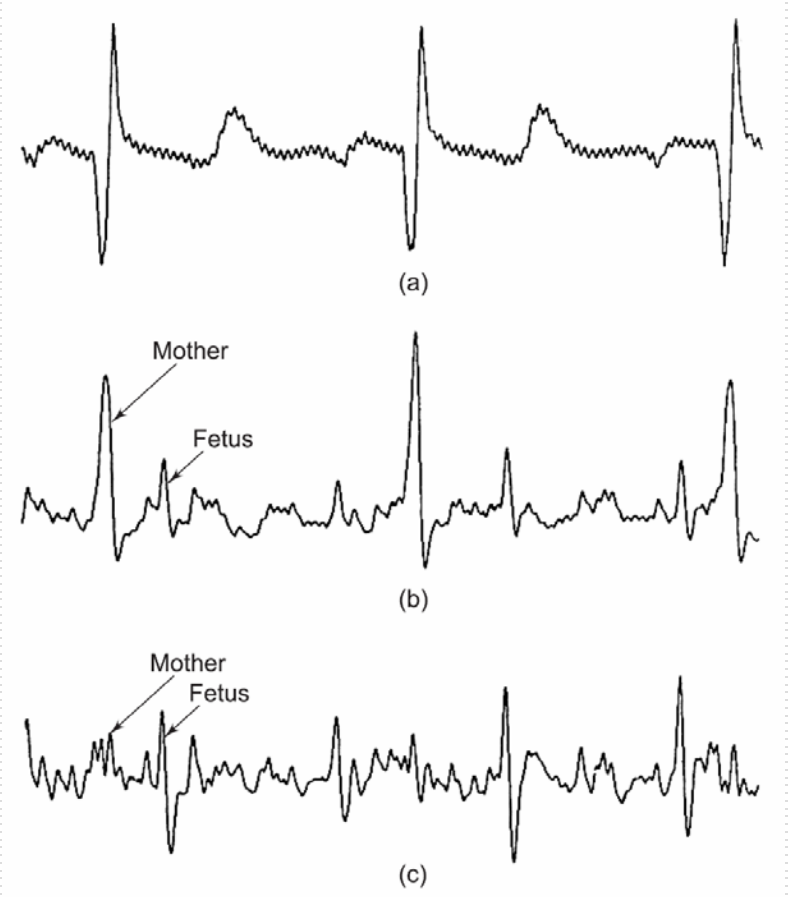
- Goal: To pass the signal and remove the noise.
- This approach can be used only if a reference signal is available that contains a noise component n_1 that is correlated with the noise n_0.
- The adaptive noise canceller subtracts the filtered reference signal from the noisy input, thereby making the output of the canceller an error signal.
- A simple argument shows that the filter can indeed adapt to cancel the noise rather easily.

Adaptive Noise Cancellation: Procedure

- If we assume that s, n_0, n_1, y are statistically independent and stationary with zero means, the analysis becomes tractable.
- For, $\epsilon = s + n_0 - y$ which on squaring yields $\epsilon^2 = s^2 + (n_0 - y)^2 + 2s(n_0 - y)$
- If we take expectations on both sides of above equation and assume that s is uncorrelated with n_0 and y, we have $E[\epsilon^2] = E[s^2] + E[(n_0 - y)^2] + 2E[s(n_0 - y)]$
- However, the last term tends to be zero due to the uncorrelated nature of s and$(n_0 - y)$, and so $E[\epsilon^2] = E[s^2] + E[(n_0 - y)^2]$

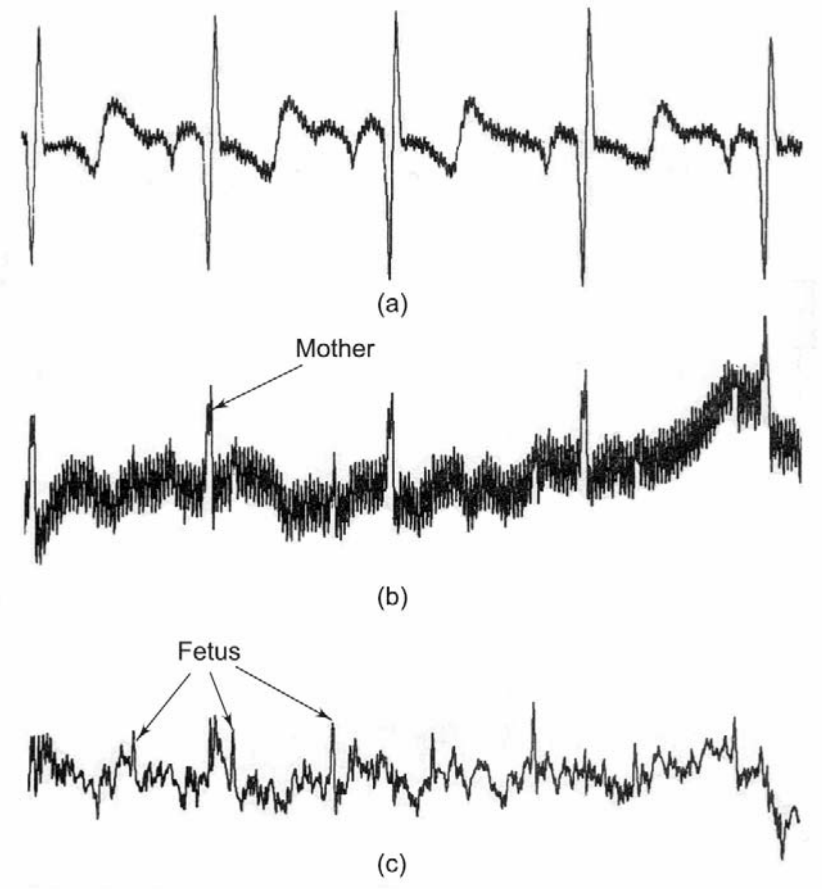
Noise cancellation in fetal ECG

- Recordings from the fetal ECG experiment at a bandwidth 3-35 Hz, sampling rate 256 Hz.
  - (a) Reference input.
  - (b) Primary input.
  - (c) Noise cancelled fetal ECG with interfering maternal ECG signal removed



Noise cancellation in fetal ECG

- Recordings from the fetal ECG experiment at a bandwidth of 0.3-75 Hz, sampling rate 512 Hz.
  - (a) Reference input.
  - (b) Primary input.
  - (c) Noise canceller output—fetal ECG with interfering maternal ECG signal removed


(a)
Mother
(b)
Fetus
(c)