

## Overview of Unit 3

- **Basic Definitions and Key Concepts**
- **Supervised Learning of a Perceptron**
  - Definition and Architecture
  - Perceptron Learning Rule
  - The Perceptron Criterion Function
- **Gradient descent-based Learning Rules**
  - The Widrow-Hoff Learning Rule
  - The Delta Rules
- **Computation Capabilities of Perceptron**

## Basic Definitions and Key Concepts

**The criterion of success.** The criterion of success is what is measured in the performance evaluation. It thus acts as a *criterion relative to an external observer*. For example, the performance will be measured according to the error count made by the learner in the course of learning, or according to its error rate after learning. More generally, the measurement of performance can include factors independent of the adequacy to the data of learning and of very diverse natures. For example, *simplicity* of the learning result produced by the learning machine (LM), its *comprehensibility*, its *intelligibility* by an expert, its facility to the integration in a current theory, the low computational cost necessary to its obtaining, etc.

Here, it should be made an important remark. The criterion of success, measured by an external observer, is not necessarily identical to the *performance index* or to the *loss function* that is *intern to the LM* and used in the internal evaluation of the learning model. For example, an algorithm of learning of a connectionist network generally seeks to minimize a standard deviation between what it predicts on each example of learning and the desired exit.

**The protocol of learning.** The learning and its evaluation depend on the *protocol* that establishes the interactions between the LM and his environment, including the supervisor (the oracle). It is thus necessary to distinguish between the *batch learning*, in which all the data of learning are provided all at the start, and the *on-line learning* in which the data arrive in sequences and where the learner must deliberate and provide an answer after each entry or groups entries.

The protocol also stipulates the type of entries provided to be learned and the type of awaited exits. For example, a scenario can specify that at every moment the LM receive an observation  $\mathbf{x}_i$ , that it must provide an answer  $y_i$  and only then, the supervisor produces the correct answer  $u_i$ . One speaks then naturally about a *prediction task*. More, the tasks known as prediction are interested to envisage correctly a response in a precise point.

In contrast, in the *identification tasks* the goal is to find a total explanation among all those possible, which once known will make possible to make predictions whatever the question.

In addition, the LM can be more or less active. In the protocols described up to now, the LM receives passively the data without having influence on their selection. It is possible to consider scenarios in which the LM has a certain initiative in the search for information. In certain cases, this initiative is limited, for example when the LM, without having the total control of the choice of the learning sample, is simply able to direct its probability distribution; the *boosting* methods are an illustration of this case. In other cases, the LM can put questions about the class of membership of an observation, one speaks then of *learning by membership queries*, or to even organize experiments on the world, and one speaks then of *active learning*. The play of Mastermind, which consists in guessing a configuration of colors hidden pawns by raising questions according to certain rules, is a simple example of active learning in which the learner has the initiative of the questions.

According to Kant, any scientific theory contains three elements:

- (1) The setting of the problem,
- (2) The resolution of the problem, and
- (3) Proofs.

For the empirical inference problem, these three elements are clearly defined:

SETTING.

The setting of the Inference problems is based on the risk minimization model:

Minimize the risk functional

$$R(\alpha) = \int Q(y, f(x, \alpha)) dP(y, x), \quad \alpha \in \Lambda$$

in the situation when the probability measure is unknown, but i.i.d. data  $(y_1, x_1), \dots, (y_k, x_k)$  are given.

## RESOLUTION.

Two different resolutions of solving this problem were suggested:

(1) Aizerman, Braverman, Rozonoer, and Tsypkin from Russia and Amari from Japan suggested using methods based on gradient-type procedures

$$\alpha_n = \alpha_{n-1} + \gamma_n \text{grad}_{\alpha} Q(y_n, f(x_n, \alpha_{n-1}))$$

(2) Vapnik and Chervonenkis suggested methods that use the empirical risk minimization principle under the condition of uniform convergence.

## PROOFS.





Proofs that justify these resolutions are based on:

- (1) The theory of stochastic approximation for gradient based procedures, and
- (2) The theory of uniform convergence for the empirical risk minimization principle (1968, 1971). In 1989, Vapnik and Chervonenkis proved that (one-sided) uniform convergence defines the necessary condition for consistency not only for the empirical risk minimization method, but for any method that selects one function from a given set of admissible functions.

## Supervised Learning of a Perceptron



### 4 Parts of a Typical Nerve Cell

-  Dendrites: Accept inputs
-  Soma: Process the inputs
-  Axon: Turn the processed inputs into outputs
-  Synapses: The electrochemical contact between neurons



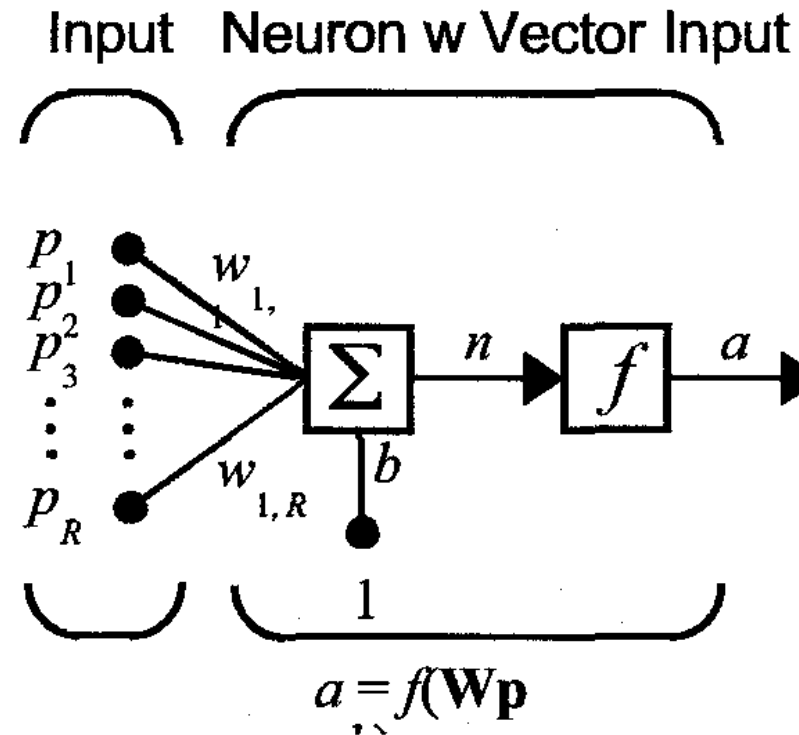
A present-day outline of the characteristics of a mammalian neuron is provided below. These points support the subsequent mathematical model (or caricature) of individual neurons and their interconnections.

1. A neuron has a **single output** conducted through its axon and **multiple inputs** conducted through dendrites.
  - a. The single neuronal output, produced by a pulse known as the action potential, can be “**fanned out**” to connect to many other neurons. Signal flow is **unilateral** at junctions.
  - b. The dendrites (and a neuron may have as many as 150,000 of them) are the major source of inputs. Hence, the model has **multiple inputs**.
2. The **neuronal response depends upon a summation of inputs**. If two subliminal volleys are sent in over the same nerve, each volley produces an effect which is manifested by an EPSP [excitatory postsynaptic potential]. The EPSPs will then sum, and if the critical level of depolarization is reached, an impulse will be set off.

3. The **neuronal response is “all or none”** in that the characteristics of the pulse (referred to as the **action potential**) do not depend on detailed characteristics of the input i.e., the nerve fiber under a given set of conditions gives a maximal response or no response at all. Thus, the propagated disturbance established in a single nerve fiber cannot be varied by grading the intensity or duration of the stimulus.

Hence, our model should exhibit thresholding and a limited range set of possible outputs. The single pulse case is that there are only two possible outputs with “1” representing the action potential and “0” or “-1” representing the quiescent state.

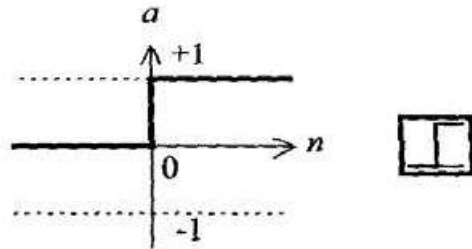
4. The neuron, over an appropriately chosen time scale, is a **memoryless mapping from excitation to response**.
5. It is known that mechanisms for changing the response of a neural network include the use of chemicals like calcium and neurotransmitters to change the characteristics of a neuron (open and close selected ion channels) as well as **changing the synaptic strengths**.



$$n = w_{1,1}p_1 + w_{1,2}p_2 + \dots + w_{1,R}p_R + b$$

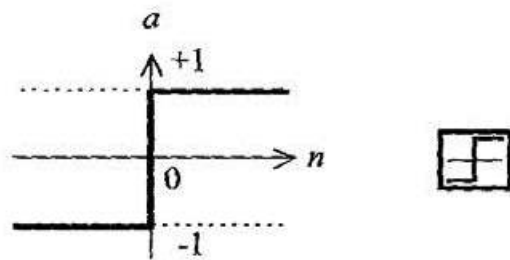
$$n = W^*p + b$$

## Transfer Function



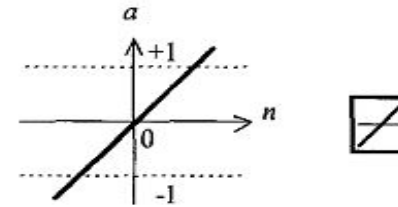
$$a = \text{hardlim}(n)$$

Hard-Limit Transfer Function



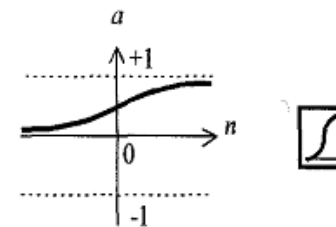
$$a = \text{hardlims}(n)$$

Symmetric Hard-Limit Trans. Funct.



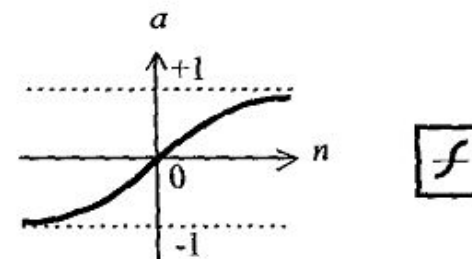
$$a = \text{purelin}(n)$$

Linear Transfer Function



$$a = \text{logsig}(n)$$

Log-Sigmoid Transfer Function

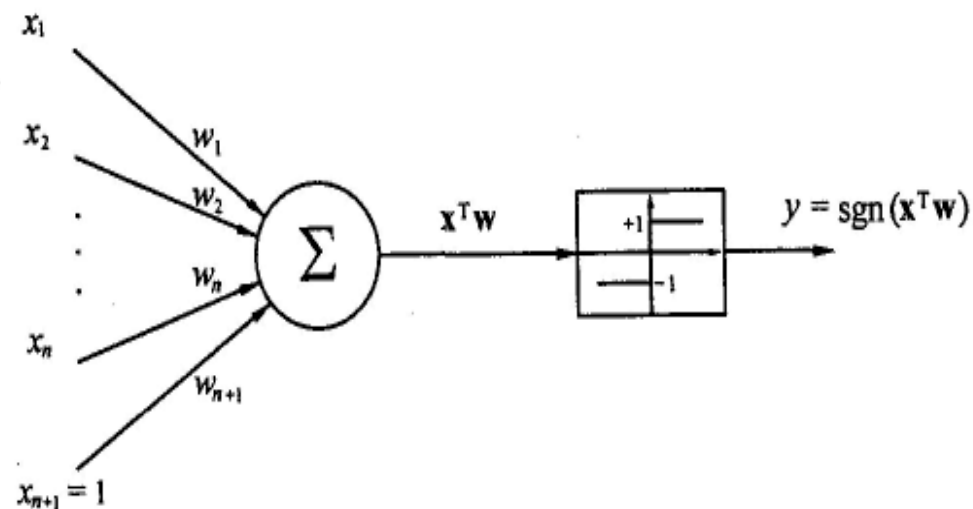


$$a = \text{tansig}(n)$$

Tan-Sigmoid Transfer Function

## Perceptron Learning Rule

Consider the linear threshold gate –LTG– shown in Figure 1, which will be referred to as the *perceptron*. The perceptron maps an input vector  $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_{n+1}]^T$  to a bipolar binary output  $y$ , and thus it may be viewed, as a simple two-class classifier. The input signal  $x_{n+1}$  is usually set to 1 and



plays the role of a bias to the perceptron. We will denote by  $\mathbf{w}$  the vector  $\mathbf{w} = [w_1 \ w_2 \ \dots \ w_{n+1}]^T \in R^{n+1}$  consisting of the free parameters (weights) of the perceptron. The input/output relation for the perceptron is given by  $y = \text{sgn}(\mathbf{x}^T \mathbf{w})$ , where  $\text{sgn}$  is the "sign" function, which returns +1 or –1 depending on whether the sign of its scalar argument is positive or negative, respectively.

Assume we are training this perceptron to load (learn) the training pairs  $\{\mathbf{x}^1, d^1\}, \{\mathbf{x}^2, d^2\}, \dots, \{\mathbf{x}^m, d^m\}$ , where  $\mathbf{x}^k \in R^{n+1}$  is the  $k$ th input vector and  $d^k \in \{-1, +1\}$ ,  $k = 1, 2, \dots, m$ , is the desired target for the  $k$ th input vector (usually the order of these training pairs is random). The entire collection of these pairs is called the *training set*.

The goal, then, is to design a perceptron such that for each input vector  $\mathbf{x}^k$  of the training set, the perceptron output  $y^k$  matches the desired target  $d^k$ ; that is, we require  $y^k = \text{sgn}(\mathbf{w}^T \mathbf{x}^k) = d^k$ , for each  $k = 1, 2, \dots, m$ . In this case we say that the perceptron correctly classifies the training set. Of course, "designing" an appropriate perceptron to correctly classify the training set amounts to determining a weight vector  $\mathbf{w}^*$  such that the following relations are satisfied:

$$\begin{cases} (\mathbf{x}^k)^T \mathbf{w}^* > 0 & \text{if } d^k = +1 \\ (\mathbf{x}^k)^T \mathbf{w}^* < 0 & \text{if } d^k = -1 \end{cases} \quad (2.1)$$

Recall that the set of all  $\mathbf{x}$  which satisfy  $\mathbf{x}^T \mathbf{w}^* = 0$  defines a hyperplane in  $R^n$ . Thus, in the context of the preceding discussion, finding a solution vector  $\mathbf{w}^*$  to Equation (2.1) is equivalent to finding a separating hyperplane that correctly classifies all vectors  $\mathbf{x}^k$ ,  $k = 1, 2, \dots, m$ . In other words, we desire a hyperplane  $\mathbf{x}^T \mathbf{w}^* = 0$  that partitions the input space into two distinct regions, one containing all points  $\mathbf{x}^k$  with  $d^k = +1$  and the other region containing all points  $\mathbf{x}^k$  with  $d^k = -1$ .

One possible incremental method for arriving at a solution  $\mathbf{w}^*$  is to invoke the **perceptron learning rule** (Rosenblatt, 1962):

$$\begin{cases} \mathbf{w}^1 \text{ arbitrary} \\ \mathbf{w}^{k+1} = \mathbf{w}^k + \rho(d^k - y^k)\mathbf{x}^k \quad k = 1, 2, \dots \end{cases} \quad \text{(the on-line form) (2.2)}$$

where  $\rho$  is a positive constant called the *learning rate*.

The incremental learning process given in Equation (2.2) proceeds as follows: First, an initial weight vector  $\mathbf{w}^1$  is selected (usually at random) to begin the process. Then, the  $m$  pairs  $\{\mathbf{x}^k, d^k\}$  of the training set are used to successively update the weight vector until (hopefully) a solution  $\mathbf{w}^*$  is found that correctly classifies the training set. This process of sequentially presenting the training patterns is usually referred to as *cycling* through the training set, and a complete presentation of the  $m$  training pairs is referred to as a *cycle* (or *pass*) through the training set. In general, more than one cycle through the training set is required to determine an appropriate solution vector



Notice that for  $\rho = 0.5$ , the perceptron learning rule can be written as

$$\begin{cases} \mathbf{w}^1 & \text{arbitrary} \\ \mathbf{w}^{k+1} = \mathbf{w}^k + \mathbf{z}^k & \text{if } (\mathbf{z}^k)^T \mathbf{w}^k \leq 0 \\ \mathbf{w}^{k+1} = \mathbf{w}^k & \text{otherwise} \end{cases} \quad (2.3)$$

where 
$$\mathbf{z}^k = \begin{cases} +\mathbf{x}^k & \text{if } d^k = +1 \\ -\mathbf{x}^k & \text{if } d^k = -1 \end{cases} \quad (2.4)$$

That is, a correction is made if and only if a misclassification, indicated by

$$(\mathbf{z}^k)^T \mathbf{w}^k \leq 0 \quad (2.5)$$

occurs.

In an analysis of any learning algorithm, and in particular the perceptron learning algorithm of Equation (2.2), there are two main issues to consider:

- (1) the existence of solutions and
- (2) convergence of the algorithm to the desired solutions (if they exist).

**In the case of the perceptron, it is clear that a solution vector (i.e., a vector  $w^*$  that correctly classifies the training set) exists if and only if the given training set is linearly separable. (Novikoff, 1962; Nilsson, 1965)**

In general, **a linearly separable problem admits an infinite number of solutions**. (the cardinal of the class of linearly indicator functions is infinity but its VC dimension is  $n+1$ ).

The perceptron learning rule in Equation (2.2) converges to one of these solutions. This solution, though, is sensitive to the value of the learning rate  $\rho$  used and to the order of presentation of the training pairs.

This sensitivity is responsible for the varying quality of the perception-generated separating surface observed in simulations.

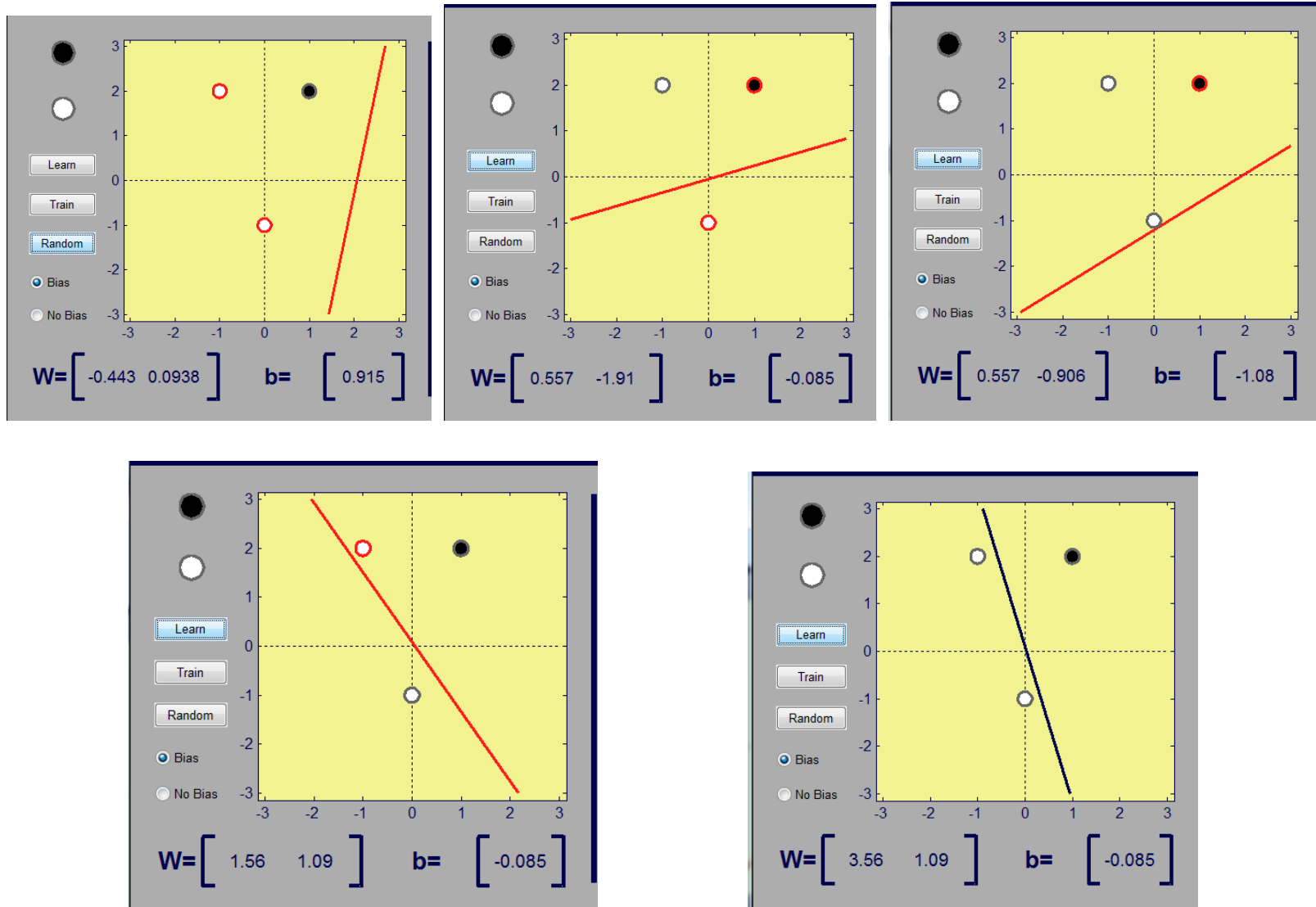
The bound on the number of corrections  $k_0$  depends on the choice of the initial weight vector  $\mathbf{w}^1$ .

If  $\mathbf{w}^1 = 0$ , we get

$$k_0 = \frac{\alpha^2 \|\mathbf{w}^*\|^2}{\beta^2} = \frac{\beta^2 \|\mathbf{w}^*\|^2}{\gamma^2} \text{ or } k_0 = \frac{\max_i \|\mathbf{x}^i\|^2 \|\mathbf{w}^*\|^2}{\left[ \min_i (\mathbf{x}^i)^T \mathbf{w}^* \right]^2} \quad (2.15)$$

Here,  $k_0$  is a function of the initially unknown solution weight vector  $\mathbf{w}^*$ . Therefore, Equation (2.15) is of no help for predicting the maximum number of corrections. However, the denominator of Equation (2.15) implies that the difficulty of the problem is essentially determined by the samples most nearly orthogonal to the solution vector.

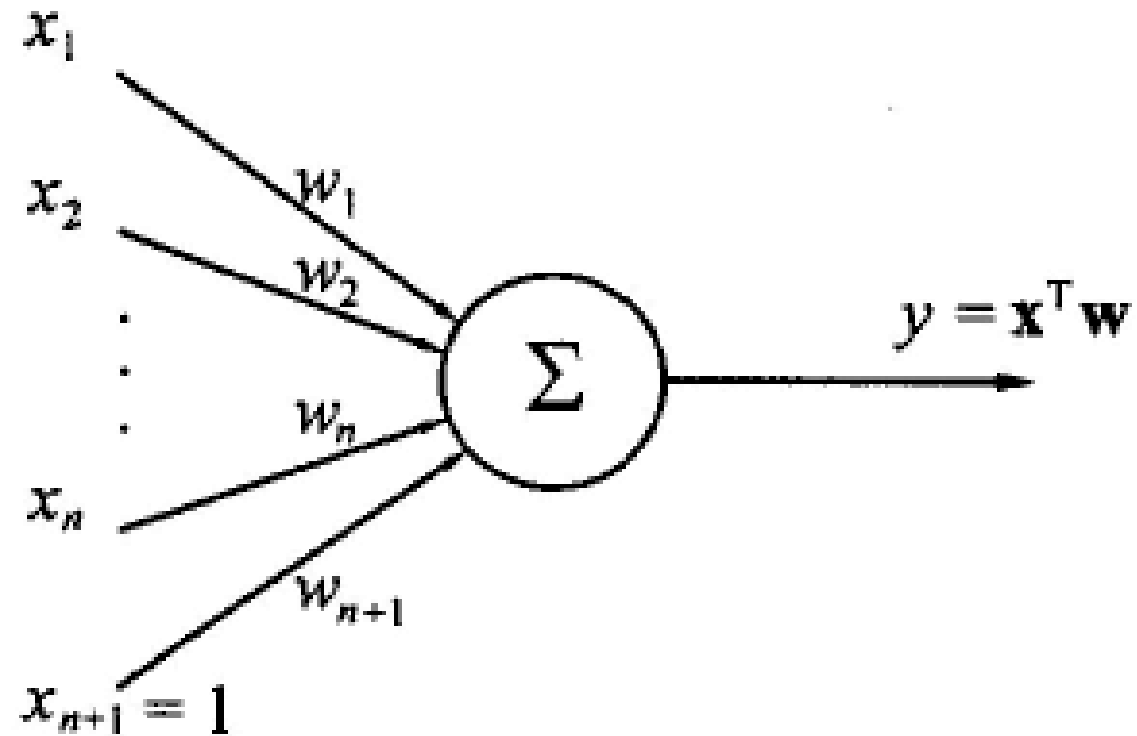
## Perceptron learning rule demonstration



## Widrow-Hoff Learning Rule

The  $\mu$ -LMS learning rule (Widrow and Hoff, 1960) represents the most analyzed and most applied simple learning rule. It is also of special importance due to its possible extension to learning in multiple unit neural nets.

This rule was originally used to train the linear unit, also known as the *adaptive linear combiner element* (ADALINE), shown in Figure 2-3. In this case, the output of the linear unit in response to the input  $\mathbf{x}^k$  is simply  $y^k = (\mathbf{x}^k)^T \mathbf{w}$ . The Widrow-Hoff rule was proposed originally as an ad hoc rule which embodies the so-called minimal disturbance principle. Later, it was discovered (Widrow and Stearns, 1985) that this rule converges in the mean square to the solution  $\mathbf{w}^*$  that corresponds to the least- mean-square (LMS) output error.



**Figure 2-3** Adaptive linear combiner element (ADALINE).

Let

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^m (d^i - y^i)^2 \quad (2.32)$$

be the sum of squared error (SSE) loss (criterion / objective)) function, where

$$y^i = (\mathbf{x}^i)^T \mathbf{w} \quad (2.33)$$

Now, using steepest gradient-descent search to minimize  $J(\mathbf{w})$  in Equation (2.32) gives

$$\begin{aligned} \mathbf{w}^{k+1} &= \mathbf{w}^k - \mu \nabla J(\mathbf{w}) \\ &= \mathbf{w}^k + \mu \sum_{i=1}^m (d^i - y^i) \mathbf{x}^i \end{aligned} \quad (2.34)$$

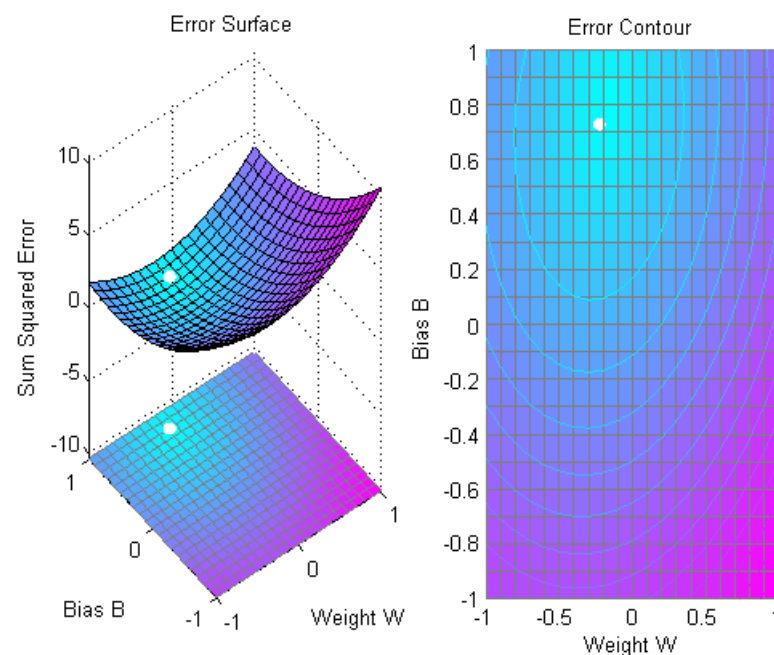


Given this objective function  $J(\mathbf{w})$ , the search point  $\mathbf{w}^k$  can be incrementally improved at each iteration by sliding downhill on the surface defined by  $J(\mathbf{w})$  in  $\mathbf{w}$  space. Specifically, we may use  $J$  to perform a discrete gradient-descent search that updates  $\mathbf{w}^k$  so that a step is taken downhill in the "steepest" direction along the search surface  $J(\mathbf{w})$  at  $\mathbf{w}^k$ . This can be achieved by making  $\Delta\mathbf{w}^k$  proportional to the gradient of  $J$  at the present location  $\mathbf{w}^k$ ; formally, we may write

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \rho \nabla J(\mathbf{w})|_{\mathbf{w}=\mathbf{w}^k} = \mathbf{w}^k - \rho \left[ \frac{\partial J}{\partial w_1} \frac{\partial J}{\partial w_2} \dots \frac{\partial J}{\partial w_{n+1}} \right]^T |_{\mathbf{w}=\mathbf{w}^k} \quad (2.21)$$

Here, the initial search point  $\mathbf{w}^1$  and the learning rate (step size)  $\rho$  are to be specified by the user. Equation (2.21) can be called the *steepest gradient descent search rule* or, simply, *gradient descent*. Next, substituting the gradient

The criterion function  $J(\mathbf{w})$  in Equation (2.32) is quadratic in the weights because of the linear relation between  $y^i$  and  $\mathbf{w}$ . In fact,  $J(\mathbf{w})$  defines a convex<sup>1</sup> hyperparaboloidal surface with a single minimum  $\mathbf{w}^*$  (the global minimum).




---

<sup>1</sup> A function of the form  $f: R^n \rightarrow R$  is said to be convex if the following condition is satisfied:

$$(1-\lambda)f(\mathbf{u}) + \lambda f(\mathbf{v}) \geq f[(1-\lambda)\mathbf{u} + \lambda\mathbf{v}]$$

for any pair of vectors  $\mathbf{u}$  and  $\mathbf{v}$  in  $R^n$  and any real number  $\lambda$  in the closed interval  $[0,1]$ .

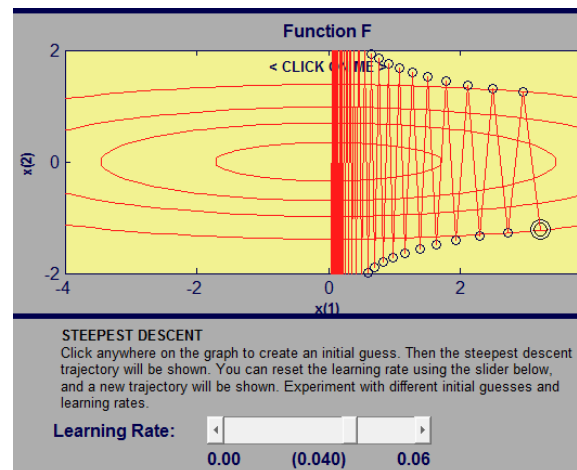
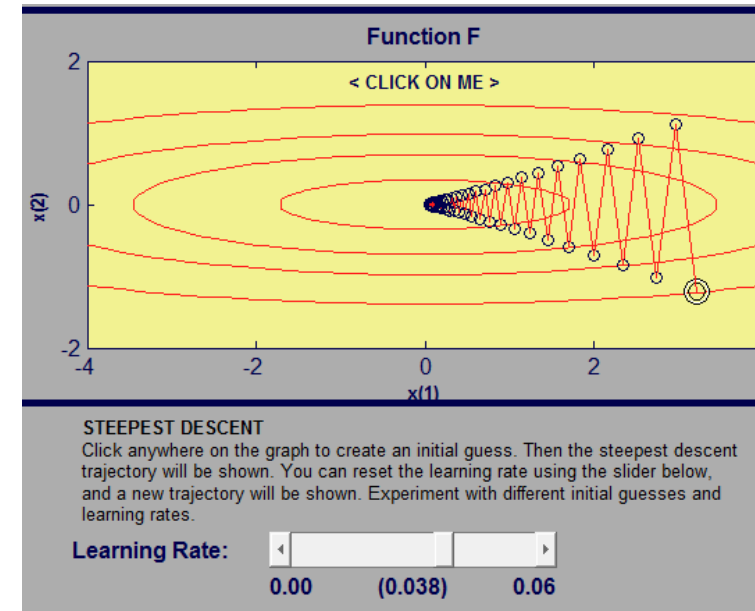
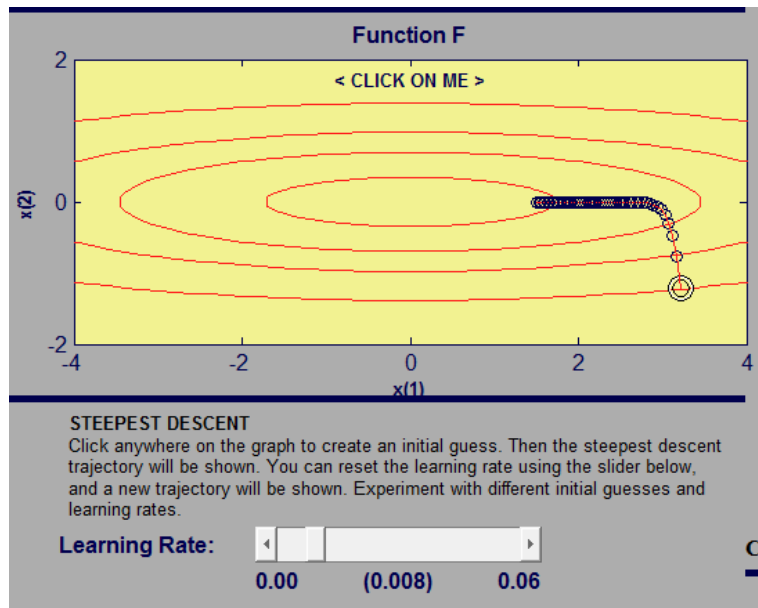
Therefore, if the positive constant  $\mu$  is chosen sufficiently small, the gradient-descent search implemented by Equation (2.34) will asymptotically converge toward the solution  $\mathbf{w}^*$  regardless of the setting of the initial search point  $\mathbf{w}^1$ . The learning rule in Equation (2.34) is sometimes referred to as the *batch LMS rule*.

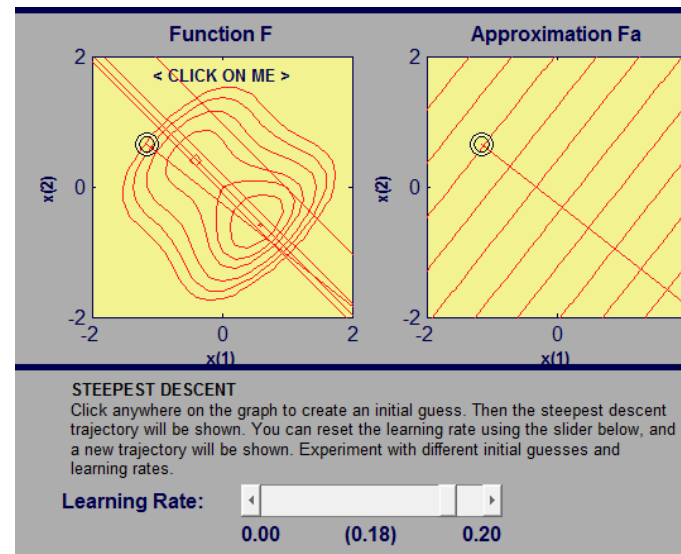
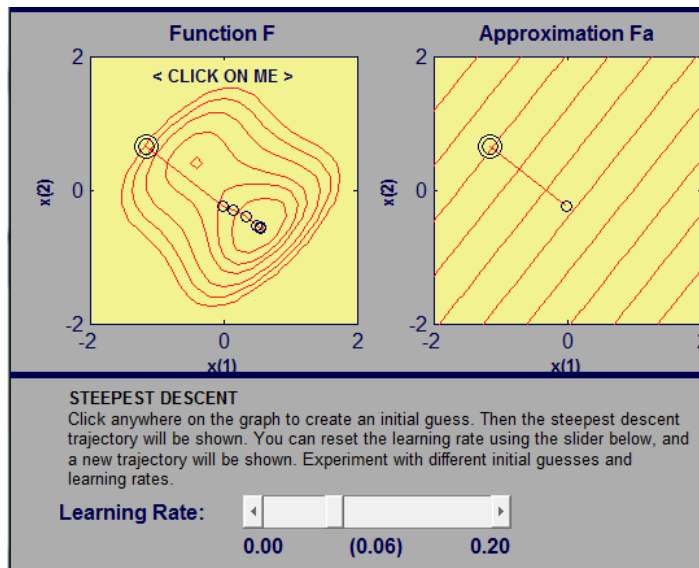
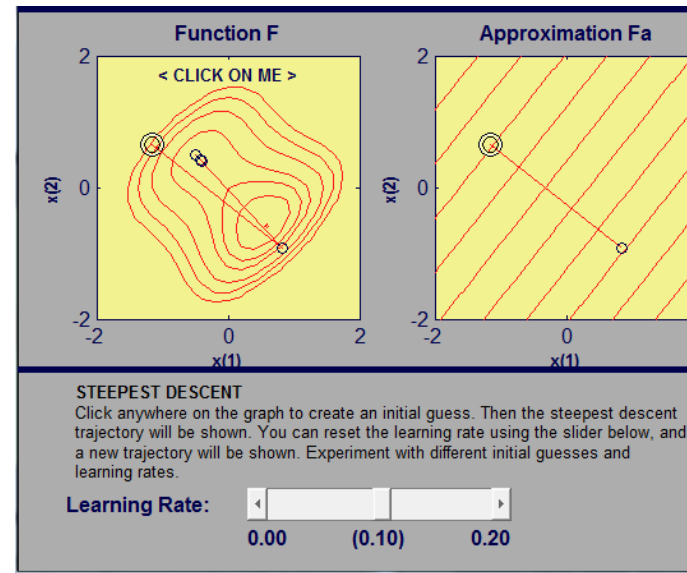
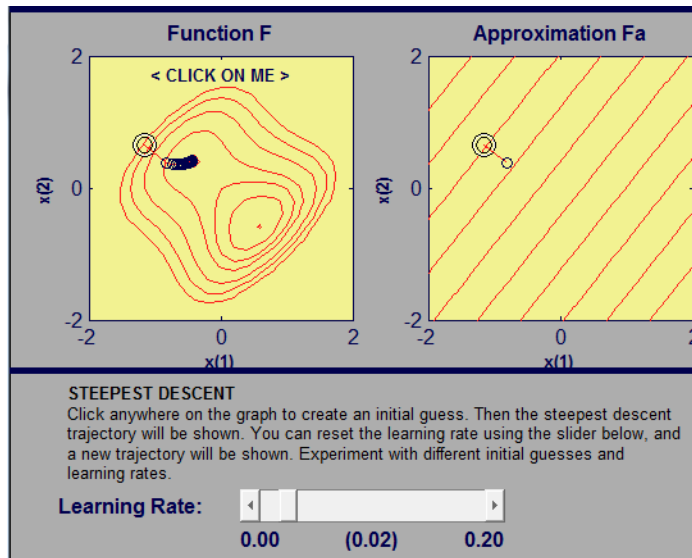
The incremental version of Equation (2.34), known as the  $\mu$ -LMS or LMS *rule*, is given by

$$\begin{cases} \mathbf{w}^1 = 0 \text{ or arbitrary} \\ \mathbf{w}^{k+1} = \mathbf{w}^k + \mu(d^k - y^k)\mathbf{x}^k \end{cases} \quad (2.35)$$

For ensuring the convergence of the  $\mu$ -LMS rule for "most practical purposes":

$$0 < \mu < \frac{2}{\max_i \|\mathbf{x}^i\|^2} \quad (2.37)$$





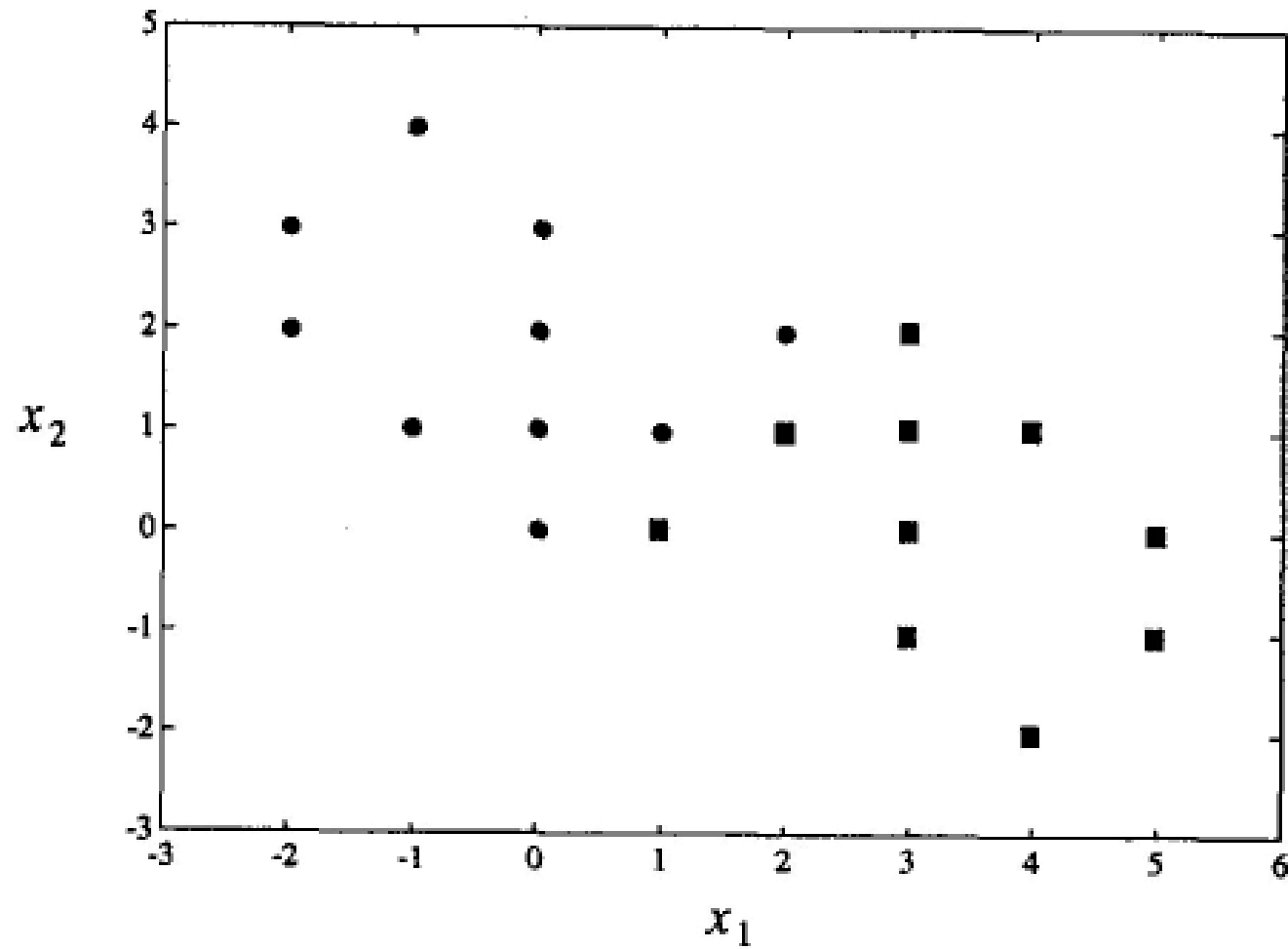
It can be shown that  $\mathbf{w}^*$  is given by

$$\mathbf{w}^* = \mathbf{X}^\dagger \mathbf{d} \quad (2.38)$$

where  $\mathbf{X} = [\mathbf{x}^1 \ \mathbf{x}^2 \ \dots \ \mathbf{x}^m]$ ,  $\mathbf{d} = [d^1 \ d^2 \ \dots \ d^m]^\mathrm{T}$ , and  $\mathbf{X}^\dagger = (\mathbf{X}\mathbf{X}^\mathrm{T})^{-1} \mathbf{X}$  is the generalized inverse

The LMS rule also may be applied to synthesize the weight vector  $\mathbf{w}$  of a perceptron for solving two-class classification problems.

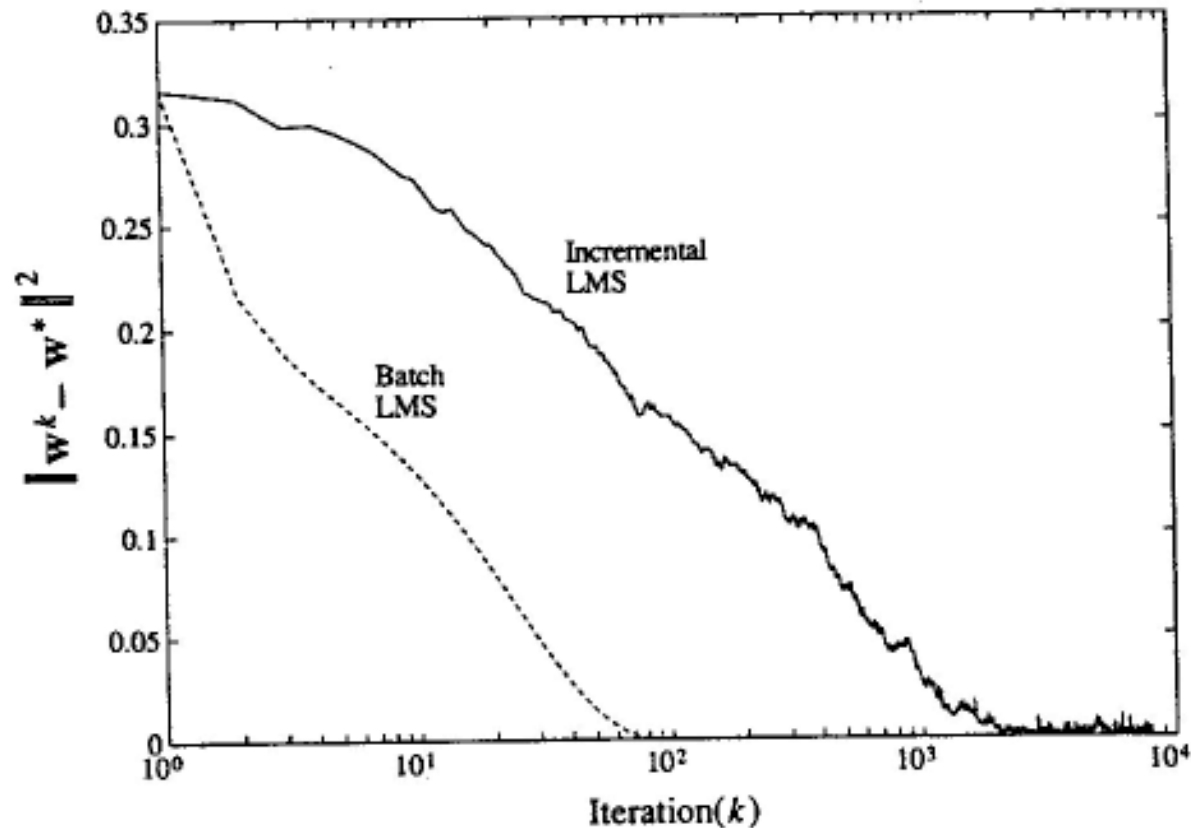
*Example 2.1* This example presents the results of a set of simulations that should help give some insight into the dynamics of the batch and incremental LMS learning rules. Specifically, we are interested in comparing the convergence behavior of the discrete-time dynamical systems in Equations (2.34) and (2.35). Consider the training set depicted in Figure 2-4 for a simple mapping problem. The 10 squares and 10 filled circles in this figure are positioned at the points whose coordinates  $(x_1, x_2)$  specify the two components of the input vectors. The squares and circles are to be mapped to the targets  $+1$  and  $-1$ , respectively. For example, the left-most square in the figure represents the training pair  $\{[1, 0]^T, 1\}$ . Similarly, the right most circle represents the training pair  $\{[2, 2]^T, -1\}$ .



**Figure 2-4** A 20-sample training set used in the simulations associated with Example 2.1. Points signified by a square and a filled circle should map into +1 and  $-1$ , respectively.

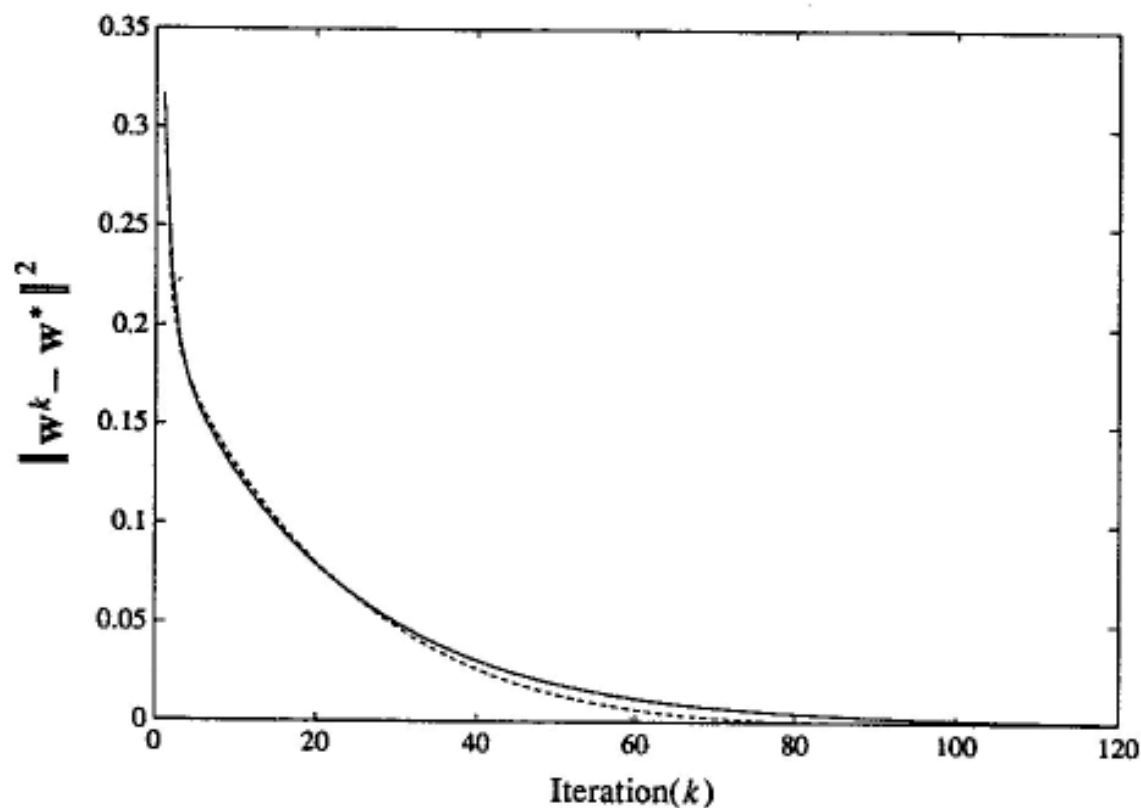


Figure 2-5 shows plots for the evolution of the square of the distance between the vector  $\mathbf{w}^k$  and the (computed) minimum SSE solution  $\mathbf{w}^*$  for batch LMS (dashed line) and incremental LMS (solid line). In both simulations, the learning rate (step size)  $\mu$  was set to 0.005. The initial search point  $\mathbf{w}^1$  was set to  $[0,0]^T$ . For the incremental LMS rule, the training examples are selected randomly from the training set. The batch LMS rule converges to the optimal solution  $\mathbf{w}^*$  in less than 100 steps. Incremental LMS requires more learning steps, on the order of 2000 steps, to converge to a small neighborhood of  $\mathbf{w}^*$ .



**Figure 2-5** Plots (learning curves) for the square of the distance between the search point  $\mathbf{w}^k$  and the minimum SSE solution  $\mathbf{w}^*$  generated using two versions of the LMS learning rule. The dashed line corresponds to the batch LMS rule in Equation (2.34). The solid line corresponds to the incremental LMS rule in Equation (2.35) with a random order of presentation of the training patterns. In both cases,  $w^1 = 0$  and  $\mu = 0.005$  are used. Note the logarithmic scale for the iteration number  $k$ .

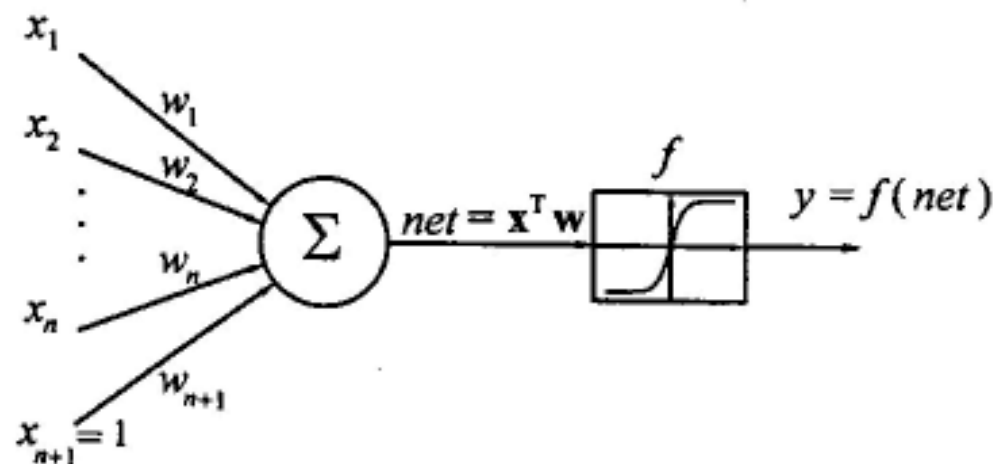
The effect of a deterministic order of presentation of the training examples on the incremental LMS rule is shown by the solid line in Figure 2-6. Here, the training examples are presented in a predefined order, which did not change during training. The same initialization and step size are used as before. In order to allow for a more meaningful comparison between the two LMS rule versions, one learning step of incremental LMS is taken to mean a full cycle through the 20 samples. For comparison, the simulation result with batch LMS learning is plotted in the figure (see dashed line). These results indicate a very similar behavior in the convergence characteristics of incremental and batch LMS learning. This is so because of the small step size used. Both cases show asymptotic convergence toward the optimal solution  $\mathbf{w}^*$ , but with a relatively faster convergence of the batch LMS rule near  $\mathbf{w}^*$ . This is attributed to the use of more accurate gradient information.



**Figure 2-6** Learning curves for the batch LMS (dashed line) and incremental LMS (solid line) learning rules for the data in Figure 2-5. The result for the batch LMS rule shown here is identical to the one shown in Figure 2-5 (this result looks different only because of the present use of a linear scale for the horizontal axis). The incremental LMS rule results shown assume a deterministic, fixed order of presentation of the training patterns. Also, for the incremental LMS case,  $\mathbf{w}^k$  represents the weight vector after the completion of the  $k$ th learning "cycle." Here, one cycle corresponds to 20 consecutive learning iterations.

## The Delta Rule

The following rule is similar to the  $\mu$ -LMS rule except that it allows for units with a differentiable nonlinear activation function  $f$ . Figure 2-7 illustrates a unit with a sigmoidal activation function. Here, the unit's output is  $y = f(\text{net})$ , with  $\text{net}$  defined as the vector inner product  $\mathbf{x}^T \mathbf{w}$ .



**Figure 2-7** A perceptron with a differentiable sigmoidal activation function.

Again, consider the training pairs  $\{\mathbf{x}^i, d^i\}$ ,  $i=1, 2, \dots, m$ , with  $\mathbf{x}^i \in R^{n+1}$  ( $x_{n+1}^i = 1$  for all  $i$ ) and  $d^i \in [-1, +1]$ .

Performing gradient descent on the instantaneous SSE criterion function  $J(\mathbf{w}) = \frac{1}{2}(d - y)^2$ , whose gradient is given by

$$\nabla J(\mathbf{w}) = -(d - y) f'(net) \mathbf{x} \quad (2.52)$$

leads to the delta rule:

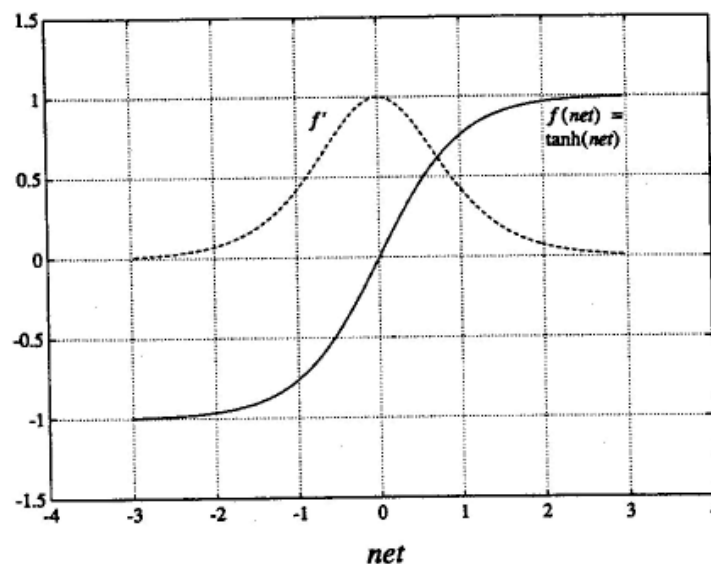
$$\begin{cases} \mathbf{w}^1 \text{ arbitrary} \\ \mathbf{w}^{k+1} = \mathbf{w}^k + \rho [d^k - f(net^k)] f'(net^k) \mathbf{x}^k = \mathbf{w}^k + \rho \delta^k \mathbf{x}^k \end{cases} \quad (2.53)$$

where  $net^k = (\mathbf{x}^k)^T \mathbf{w}^k$  and  $f' = \frac{df}{d net}$ .

If  $f$  is defined by  $f(\text{net}) = \tanh(\beta \text{net})$ , then its derivative is given by  $f'(\text{net}) = \beta[1 - f^2(\text{net})]$ .

If  $f$  is defined by  $f(\text{net}) = 1/(1 + e^{-\beta \text{net}})$ , the derivative is  $f'(\text{net}) = \beta f(\text{net})[1 - f(\text{net})]$ .

Figure 2-8 plots  $f$  and  $f'$  for the hyperbolic tangent activation function with  $\beta = 1$ . Note how  $f$  asymptotically approaches  $+1$  and  $-1$  in the limit as  $\text{net}$  approaches  $+\infty$  and  $-\infty$ , respectively.



**Figure 2-8** Hyperbolic tangent activation function  $f$  and its derivative  $f'$ , plotted for  $-3 \leq \text{net} \leq +3$ .

One disadvantage of the delta learning rule is immediately apparent upon inspection of the graph of  $f'(net)$  in Figure 2-8. In particular, notice how  $f'(net) \approx 0$  when  $net$  has large magnitude (i.e.,  $|net| > 3$ ); these regions are called *flat spots* of  $f'$ . In these flat spots, we expect the delta learning rule to progress very slowly (i.e., very small weight changes even when the error  $(d - y)$  is large), because the magnitude of the weight change in Equation (2.53) directly depends on the magnitude of  $f'(net)$ . Since slow convergence results in excessive computation time, it would be advantageous to try to eliminate the flat spot phenomenon when using the delta learning rule. One common flat spot elimination technique involves replacing  $f'$  by  $f'$  plus a small positive bias  $\varepsilon$ . In this case, the weight update equation reads as

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \rho \left[ d^k - f(net^k) \right] \left[ f'(net^k) + \varepsilon \right] \mathbf{x}^k \quad (2.54)$$

One of the primary advantages of the delta rule is that it has a natural extension that may be used to train multilayered neural nets. This extension, known as *error back propagation*, will be discussed in Unit 4.



## Computation Capabilities of Perceptron

- a. Can a given training set be learned without error by a perceptron?
- b. How many of the  $2^n$  possible  $\pm 1$ -valued assignments of values  $t_1, \dots, t_n$  can be made by a perceptron to the set  $S = \{\mathbf{x}_i\}$  of input variables ?

The implementation of a binary-valued ( $\pm 1$ ) function  $y$  is equivalent to partitioning the possible input vectors into two families,

$$\{\mathbf{x} : y(\mathbf{x}) = 1\}, \quad \{\mathbf{x} : y(\mathbf{x}) = -1\}.$$

For a Rosenblatt perceptron these two sets are half-spaces

$$H^+ = \{\mathbf{x} : y = 1\} = \{\mathbf{x} : \mathbf{w}^T \mathbf{x} - \tau \geq 0\},$$

$$H^- = \{\mathbf{x} : y = -1\} = \{\mathbf{x} : \mathbf{w}^T \mathbf{x} - \tau < 0\},$$

with hyperplane boundary

$$H = \{ \mathbf{x} : \mathbf{w}^T \mathbf{x} - \tau = 0 \};$$

in two dimensions ( $d = 2$ ) the hyperplane boundary is a straight line that divides the plane  $\mathbb{R}^2$  into two halves, above and below the line.

Given a training set  $T = \{(\mathbf{x}_i, t_i), i = 1 : n\}$ , we identify the finite sets of inputs

$$S^+ = \{ \mathbf{x} : (\exists i) t_i = 1, \mathbf{x} = \mathbf{x}_i \}, \quad S^- = \{ \mathbf{x} : (\exists i) t_i = -1, \mathbf{x} = \mathbf{x}_i \}.$$

**Definition (Linear Separability)** We say that  $S^+$ ,  $S^-$  are **linearly separable** if there exists  $H$  specified by  $\mathbf{w}$ ,  $\tau$  such that

$$S^+ \subset H^+, S^- \subset H^-$$

or, equivalently,

$$(\forall \mathbf{x} \in S^+) \mathbf{w}^T \mathbf{x} - \tau \geq 0, (\forall \mathbf{x} \in S^-) \mathbf{w}^T \mathbf{x} - \tau < 0.$$

Hence, the answer to the Question (a) is positive if the training set is linearly separable.

To respond to Question (b) we must find out how many of the  $2^n$  possible training sets  $T$  are linearly separable.

Given distinct inputs  $S = \{x_1, \dots, x_n\}$ , there are  $2^n$  possible assignments of the corresponding classes  $t_1, t_2, \dots, t_n$ . Each assignment provides a **dichotomy** of  $S$  into two disjoint subsets.

Let  $L(S)$  denote the number of linearly separable dichotomies of  $S$  implementable by a perceptron.

## Introduce

$$\hat{\mathcal{L}}(n, d) = \max_{\{S: ||S||=n, S \subset \mathbb{R}^d\}} \mathcal{L}(S)$$

as the maximum number of dichotomies that can be formed by a perceptron when we consider all sets  $S$  of size  $n$  with elements drawn from  $\mathbb{R}^d$ .

Introduce the finite difference equation system

$$D(n, d) = D(n - 1, d) + D(n - 1, d - 1), \quad (2.3.2)$$

with  $D(n, d)$  satisfying

$$D(1, d) = \hat{\mathcal{L}}(1, d) = 2, \quad D(n, 1) = \hat{\mathcal{L}}(n, 1) = 2(n + 1) - 2 = 2n, \quad (2.3.3)$$

**Theorem (Upper Bound to  $L$ )** *The unique solution to Eqs. (2.3.2) and (2.3.3) and upper bound to  $L(S)$  is*

$$D(n, d) = \begin{cases} 2 \sum_{i=0}^d \binom{n-1}{i}, & \text{if } n > d + 1; \\ 2^n, & \text{if } n \leq d + 1 \end{cases} \quad (2.3.4)$$

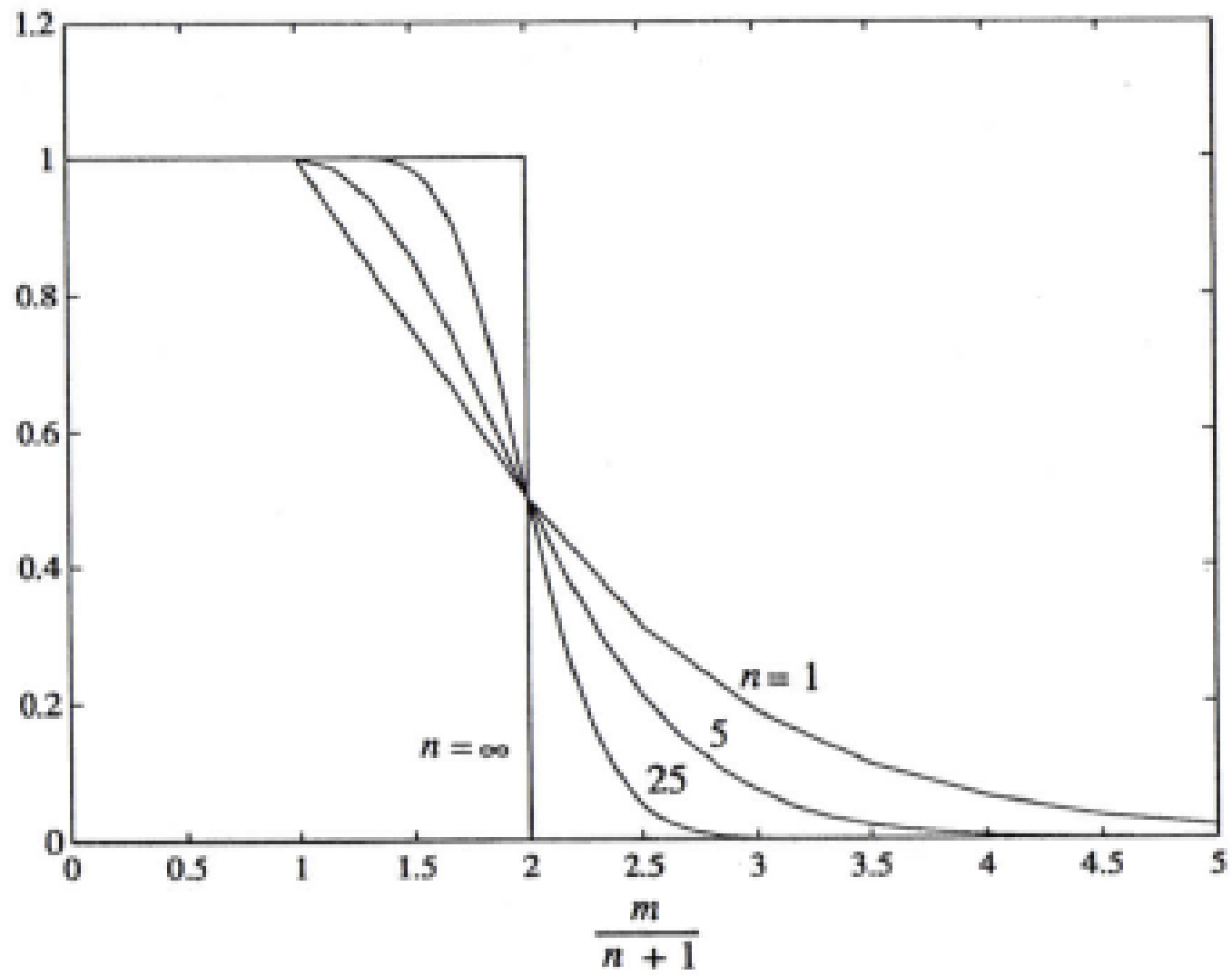
A rough sense of our results is provided by the following table for  $d = 2, 3, 4$ .

$n$ :	2	3	4	5	6	7	8
$2^n$ :	4	8	16	32	64	128	256
$D(n, 2)$ :	4	8	14	22	32	44	58
$D(n, 3)$ :	4	8	16	30	52	84	128
$D(n, 4)$ :	4	8	16	32	62	114	198

So long as  $D(n, d) < 2^n$ , we cannot train a perceptron to learn all training sets. Observe that even for  $n = 4(5)(6)$  a perceptron can no longer learn all training sets in  $\mathbb{R}^2(\mathbb{R}^3)(\mathbb{R}^4)$ .

Hence, the probability that a randomly selected  $T$  will be classifiable by a Rosenblatt perceptron is given by

$$P(n, d) = D(n, d)/2^n = \left(\frac{1}{2}\right)^{n-1} \sum_{k=0}^d \binom{n-1}{k}. \quad (2.4.1)$$



## *Binary-Valued Inputs*

If the inputs to the perceptron are binary-valued, then Question (b) concerns the implementation of Boolean functions, the synthesis of switching circuits.

The above results can be adapted to the implementation of a Boolean function of  $d$  variables by choosing the training set size  $n$  to be the total number  $2^d$  of inputs to a Boolean function of  $d$  variables.

Because the  $2^d$  binary-valued sequences that now comprise the input set  $S$  are not in general position as points in  $\mathbb{R}^d$ , our evaluation of  $D(2^d, d)$  is only an upper bound to the exact number  $B(d)$  of Boolean functions that are implementable by a perceptron.

A comparison of the number  $D(2^d, d)$ , assisted by the bound

$$B(d) \leq D(2^d, d) < 2^{d^2} / (d - 1)!,$$



with the exact count of  $2^{2^d}$  possible Boolean functions reveals that, even we use the exhaustive training set, asymptotically very few Boolean functions can be implemented by a single perceptron.

$n:$	1	2	3	4	5	6
$B(d):$	4	14	104	1882	94,572	15,028,134