SysEng 5212 /EE 5370
Introduction to Neural Networks and Applications

Week 3 : Learning rules and learning paradigms, Adaptive filtering and Least mean square algorithm

Cihan H Dagli, PhD

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

dagli@mst..edu

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

Volunteers Tally

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

Volunteers Tally

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

Volunteers Tally

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

Volunteers Tally

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

Lecture outline

- The perceptron model
- Example
- Limitations of the perceptron
- Learning paradigms
- Perceptron learning algorithm
- Error surfaces
- Programming experiment

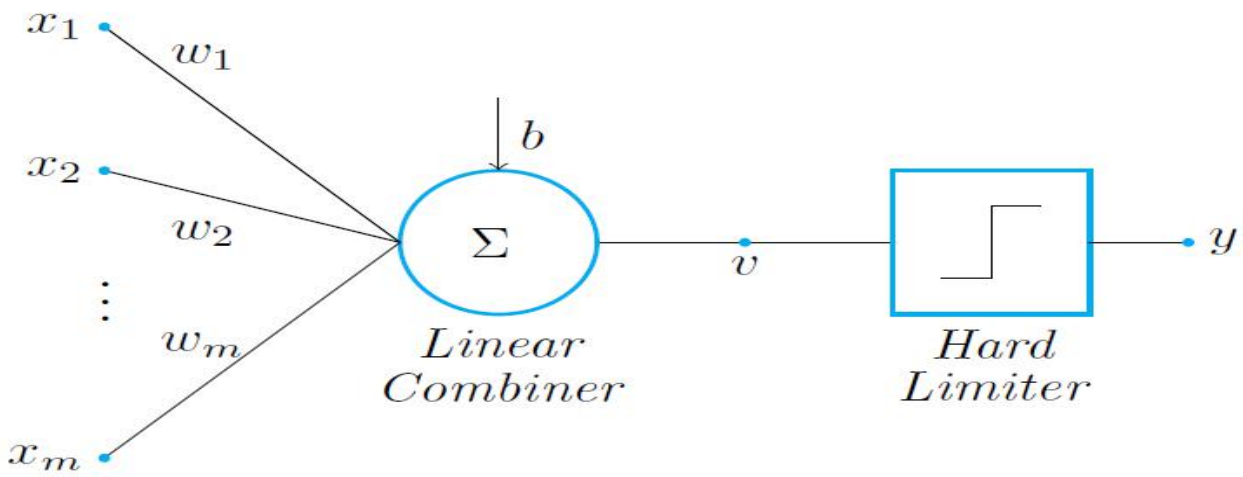
Rosenblatt's Perceptron

Proposed in 1958 by American Psychologist, Frank Rosenblatt.

- First model that was capable of supervised learning
- It was a more general form of the McCulloch Pitts neuron
  - weights can take numerical values, instead of the {+1,-1} of the McCulloch Pitts
- The computing units are threshold elements.
- Learning takes place by a numerical algorithm

Simplest form of NN for the classification of patterns as long as they are linearly separable.

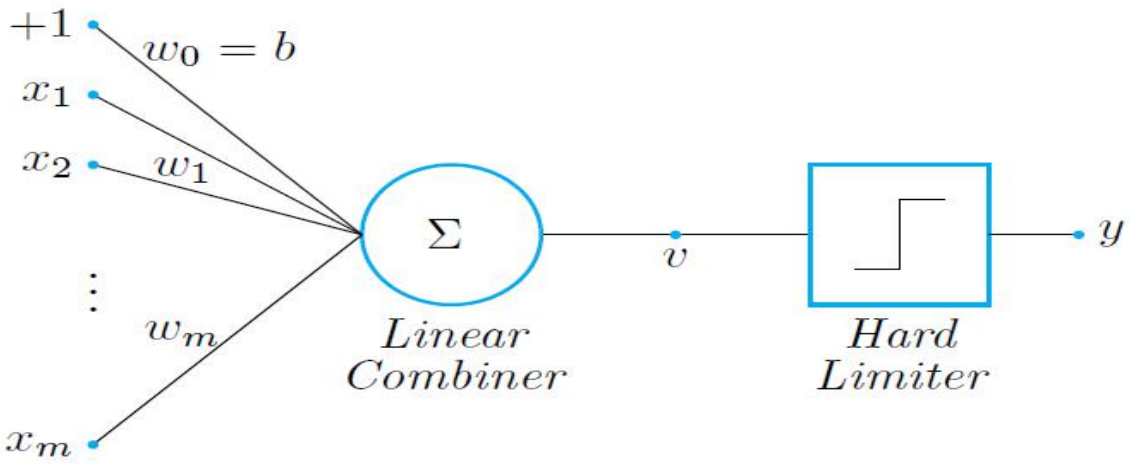
The Perceptron Model



$$v = \sum_{i=1}^{m} (w_i x_i + b)$$

$$y = hardlim(v)$$

Equivalent Model with Bias as a Weight



$$v = \sum_{i=0}^{m} (w_i x_i)$$

$$y = hardlim(v)$$

Extended Notation

Input Vector:

$$\{x_1, x_2, \ldots, x_m\}$$

Extended Input Vector:

$$\{+1, x_1, x_2, \ldots, x_m\}$$

Weight Vector:

$$\{w_1, w_2, \ldots, w_m\}$$

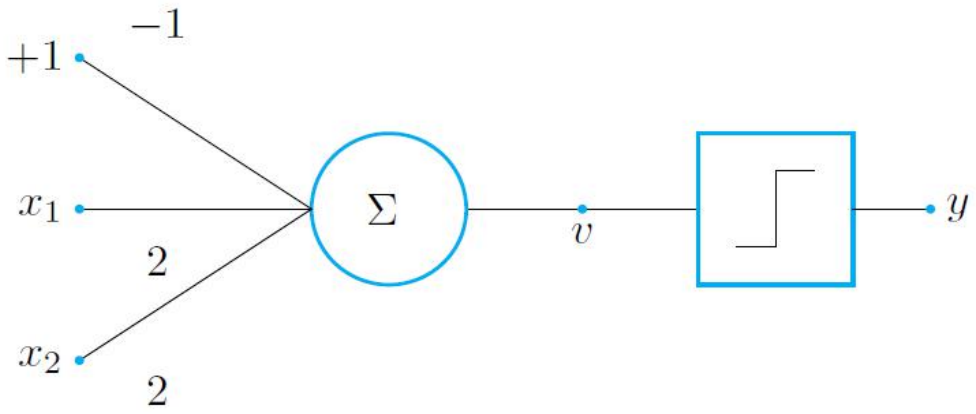
Extended Weight Vector:

$$\{w_0, w_1, w_2, \ldots, w_m\}$$

where $w_0 = b$.

Input and weight vectors will refer to the extended forms unless specified.

Perceptron Example: OR Function



Extended input v

$$X=\begin{bmatrix}1 & 0 & 0\\ 1 & 0 & 1\\ 1 & 1 & 0\\ 1 & 1 & 1\end{bmatrix}$$

Extended weight vector:

$w=[-1\ 2\ 2]$

Perceptron Example: Implementing the OR function

Input to the hard limiter:

$$\mathbf{v}=\mathbf{w}*\mathbf{x}_T=[-1\ 2\ 2]*\begin{bmatrix}1&0&0\\1&0&1\\1&1&0\\1&1&1\end{bmatrix}=[-1\ 1\ 1\ 3]$$

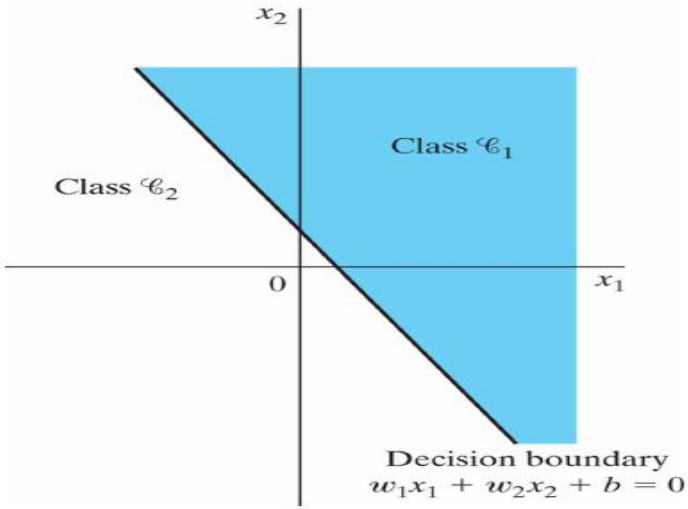
Network output:

$$y=\textit{hardlim}([-1\ 1\ 1\ 3])=[0\ 1\ 1\ 1]$$

MATLAB code:

```
x = [1 0 0; 1 0 1; 1 1 0; 1 1 1]; %Extended neuron input
w = [-1 2 2]; %Extended weight vector
v = (w*x'); %Output of the summing junction
y = hardlim(v) %Output of the neuron
```

The Hyperplane as a Decision Boundary



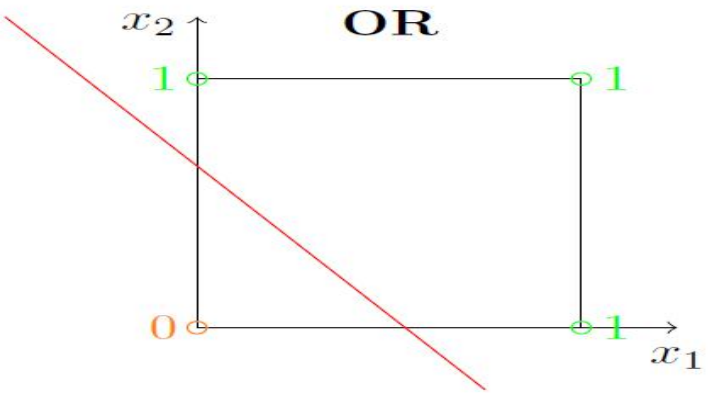
For a two-dimensional, two-class pattern-classification problem, the hyperplane is a straight line. Any point above the line is placed in class C_1, and any point below the line is placed in class C_2.

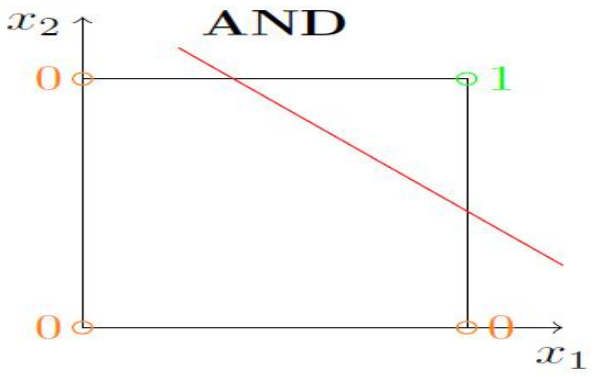
Decision Boundary Examples

x_1	x_1	OR	AND	XOR
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

XOR is true whenever an odd number of inputs is true.

Separation of the input space for the OR and AND functions,

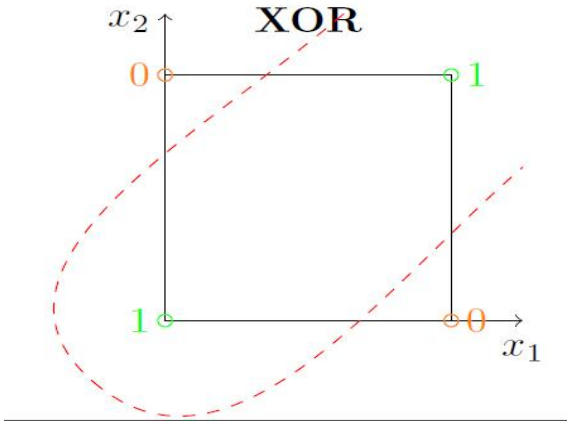




The XOR Input Space

x_1	x_1	XOR
0	0	0
0	1	1
1	0	1
1	1	0

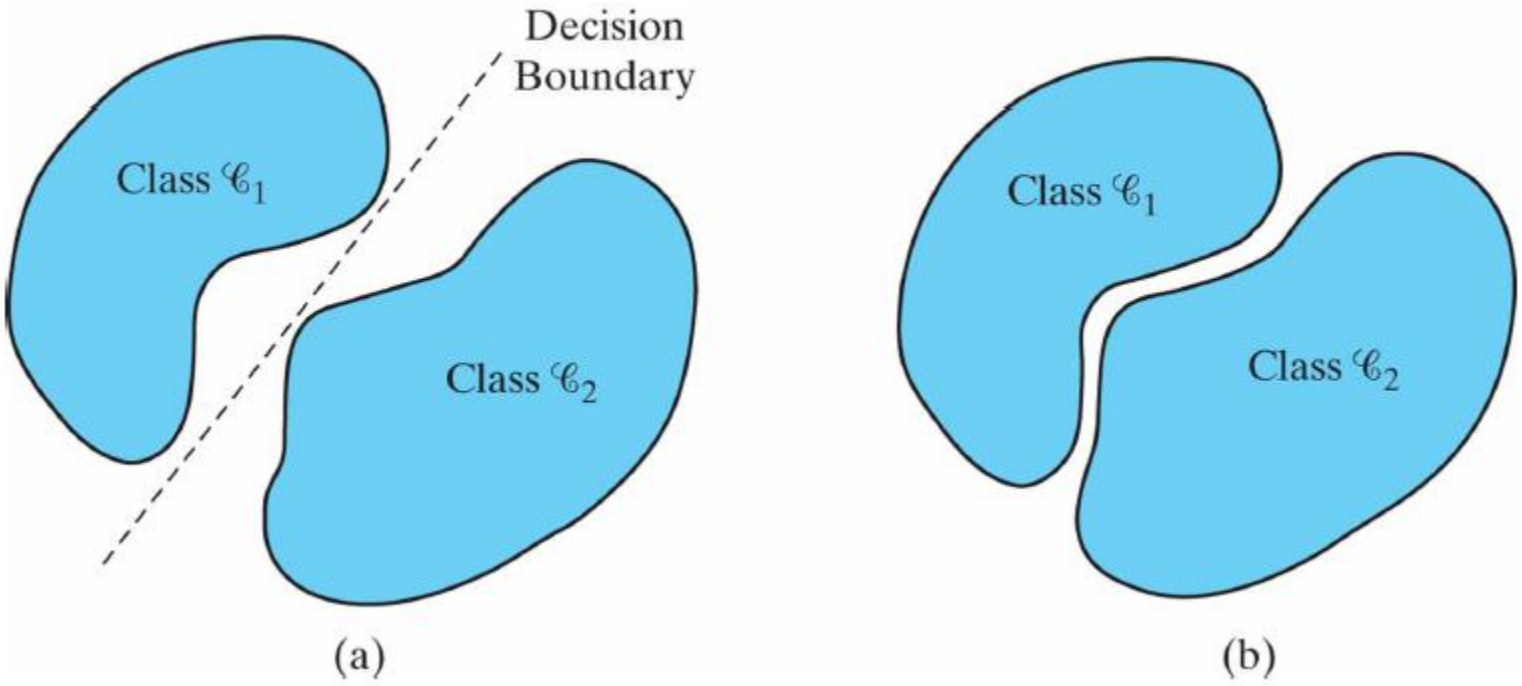
Separation of the input space for the XOR function,



Computational Limits of the Perceptron Model

- No perceptron capable of computing the XOR function exists
- The perceptron can classify the input into separate classes as long as they are linearly separable.
- The perceptron can be extended to solve multiclass pattern recognition problems but the linear separability condition must hold.
- Minsky and Papert used this model to study the capabilities of such weighted networks and found them lacking.

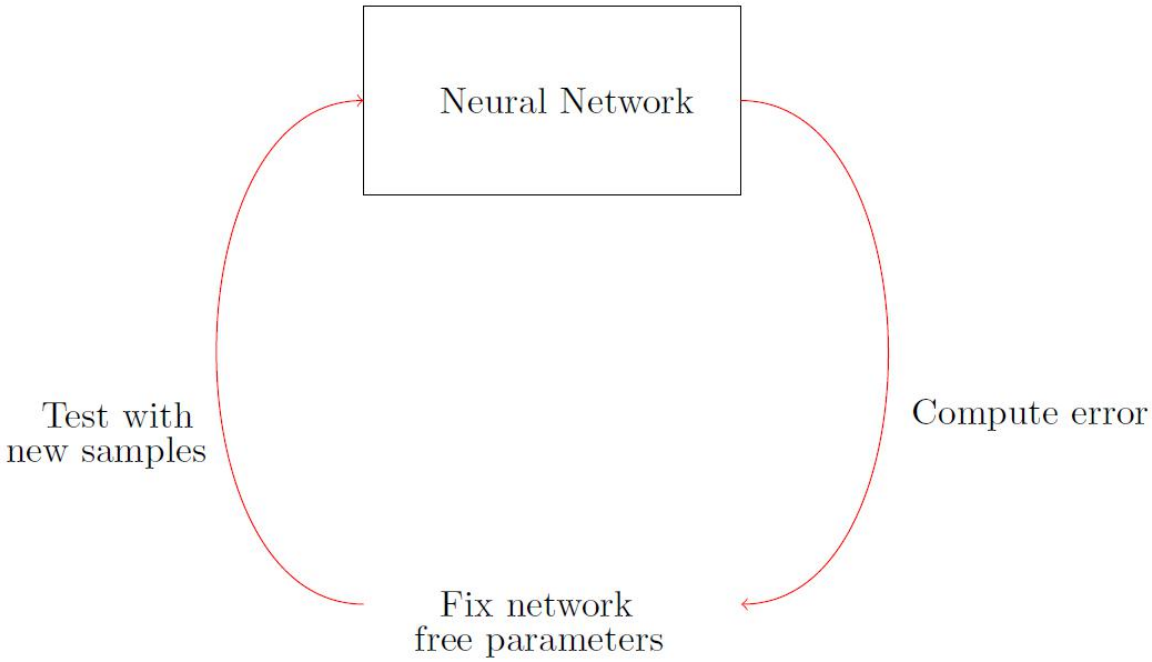
Nonlinearly Separable Patterns



Tuning Network Parameters

- We have used an ad hoc approach for finding weights and biases
- Manually tuning network parameters is not feasible for inputs of more than two or three dimensions.
- To automatically tune weights and biases we use a *learning process*
- A *learning algorithm* is a process that makes use of past experience to iteratively adapt network parameters until a solution is found.

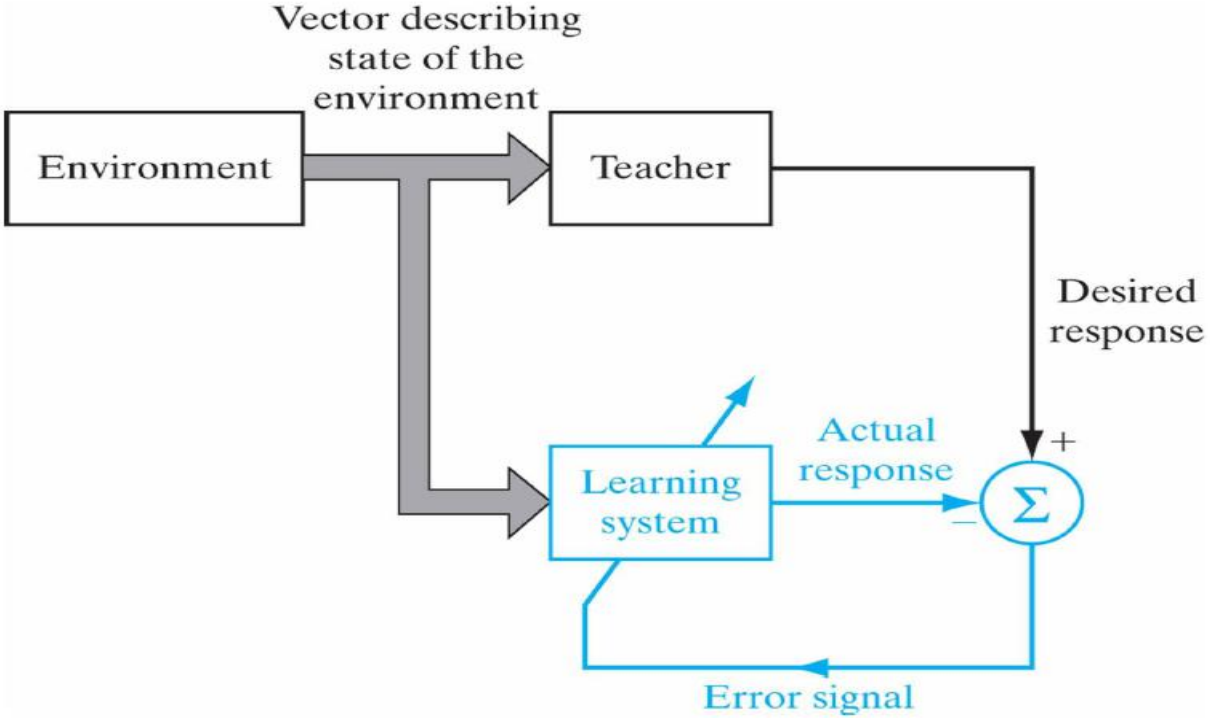
Learning Process



Classes of Learning Algorithms

- Supervised learning: Learning with a teacher
  - error correction learning
- Unsupervised learning: Learning without a teacher
- Reinforcement learning

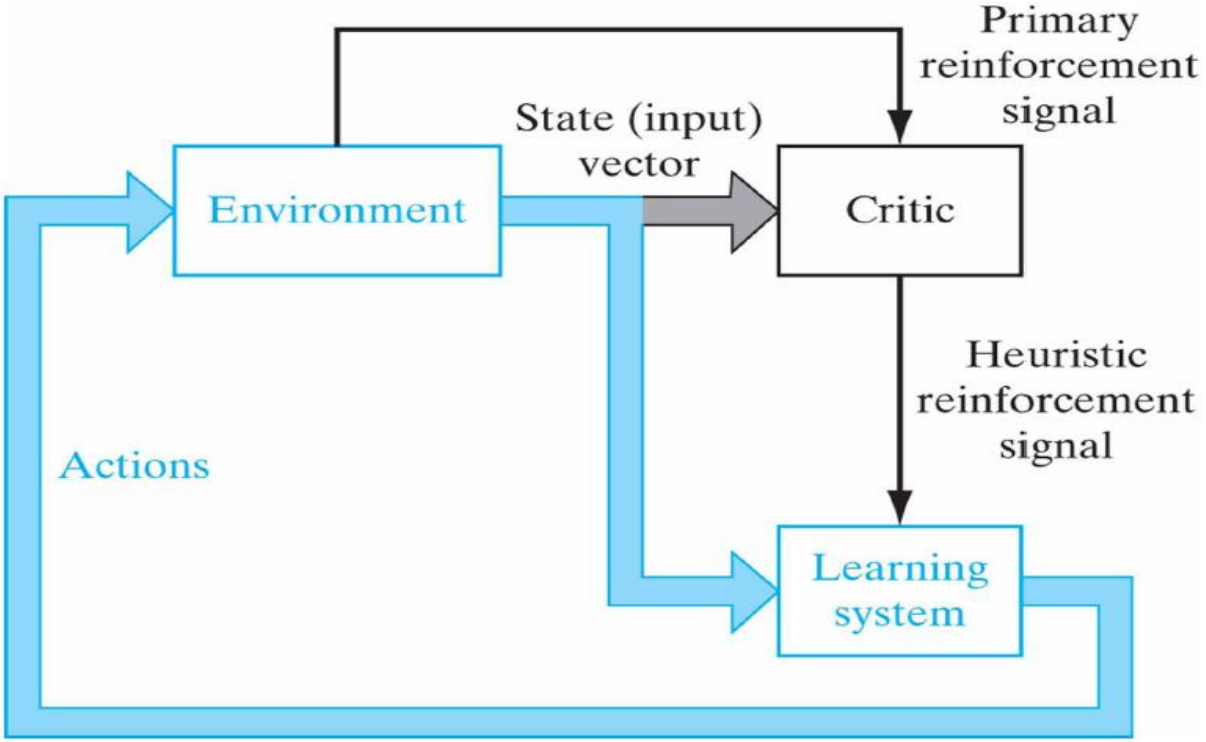
Supervised Learning



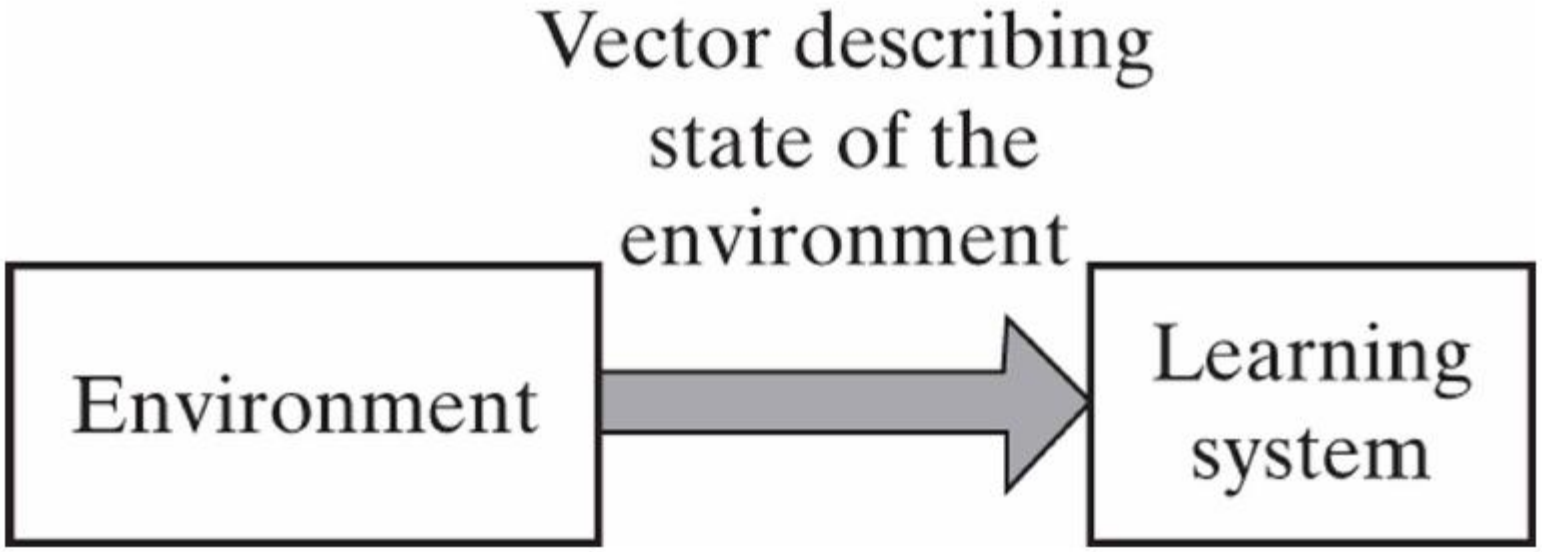
Error Corrective Learning

- Inputs are presented to the network along with the corresponding desired output.
- Network output is compared to the target output and the deviation is measured.
- Weights are adjusted based on the magnitude of the error.
- Each learning algorithm uses its own specific set of rules to update the weights.

Reinforcement Learning



Unsupervised Learning



The Perceptron Learning Algorithm

Consider the extended input and weight vectors where the inputs belong to two linearly separable classes C_1 and C_2.

$\mathbf{x}(n) = [+1, x_1(n), x_2(n), \ldots\ldots, x_m(n)]^T$

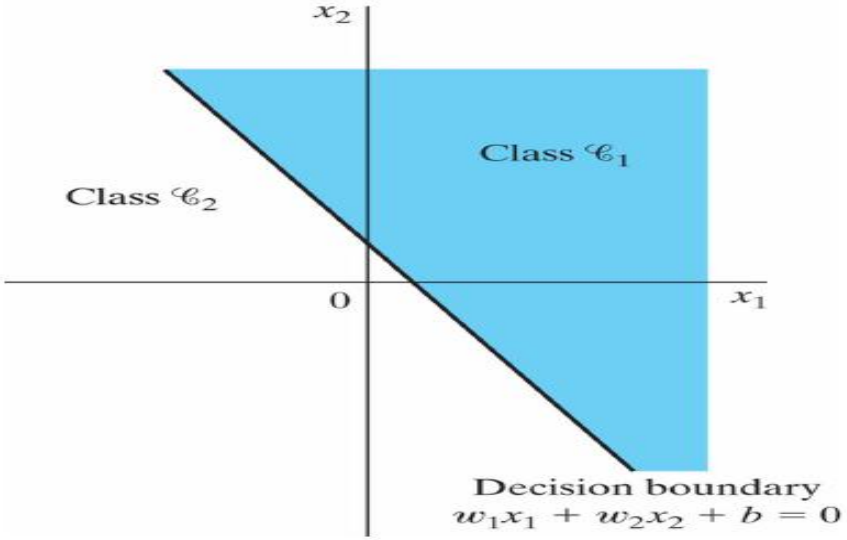
$\mathbf{w}(n) = [b, w_1(n), w_2(n), \ldots\ldots, w_m(n)]^T$

The linear combiner output is written as,

$$v(n) = \sum_{i=0}^{m} w_i(n), x_i(n)$$
$$= \boldsymbol{w}^T(n)\boldsymbol{x}(n)$$

Then, $\boldsymbol{w}^T(n)\boldsymbol{x}(n) = 0$ denes a hyperplane in the input space, which is a decision boundary between the different classes of inputs, such that,

- $\boldsymbol{w}^T(n)\boldsymbol{x}(n) > 0$ for every input vector x belonging to class C1
- $\boldsymbol{w}^T(n)\boldsymbol{x}(n) <= 0$ for every input vector x belonging to class C2



Classical Rules for Adapting the Perceptron Weights

Start: Generate a random weight vector, $\mathbf{w}_0$; set n = 0.

Evaluate: Pick an input sample and evaluate

- If $x \in C_1$ and $\boldsymbol{w}^T x(n) > 0$, select next input
- If $x \in C_2$ and $\boldsymbol{w}^T x(n) \leq 0$, select next input
- If $x \in C_1$ and $\boldsymbol{w}^T x(n) < 0$, go to add
- If $x \in C_2$ and $\boldsymbol{w}^T x(n) \geq 0$, go to subtract

add: Set $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \eta x(n)$ and n=n+1, go to test

subtract: Set $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta x(n)$ and n=n+1, go to test

where, η is the learning rate parameter which controls the adjustment applied to the weight vector.

Error Correction Perceptron Learning Algorithm

Modified learning rule using the error signal, where the network response is,

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

Let d(n) be the desired response, then the error in the response can be calculated as d(n)-y(n). The error correction learning rule is give by,

$$w(n+1) = w(n) + \eta[d(n) - y(n)]x(n)$$

The learning rate parameter is a positive constant in the range $0 < \eta \leq 1$

Summary of the Perceptron Learning Algorithm

TABLE 1.1 Summary of the Perceptron Convergence Algorithm

Variables and Parameters:

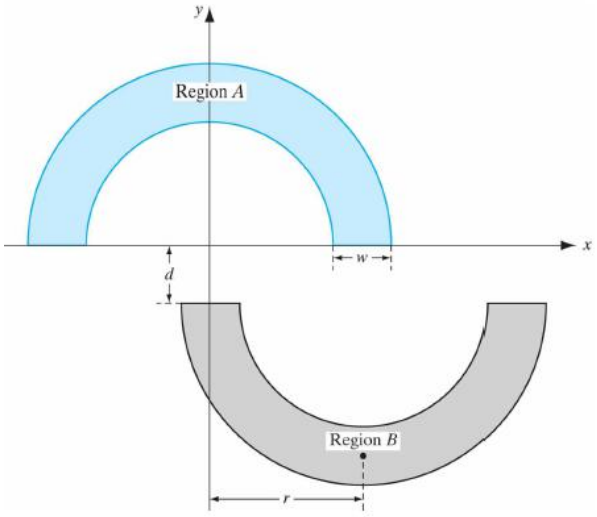
$\mathbf{x}(n)$ = $(m + 1)$-by-1 input vector
$= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$
$\mathbf{w}(n)$ = $(m + 1)$-by-1 weight vector
$= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T$
b = bias
$y(n)$ = actual response (quantized)
$d(n)$ = desired response
η = learning-rate parameter, a positive constant less than unity

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \ldots$.
2. *Activation.* At time-step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response.* Compute the actual response of the perceptron as
$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$
where sgn(·) is the signum function.
4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain
$$\mathbf{w}(n + 1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$
where
$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$
5. *Continuation.* Increment time step n by one and go back to step 2.

Pattern Classification Experiment

Objective: to demonstrate the use of the perceptron learning algorithm for pattern classification under linear separability of input patterns.

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



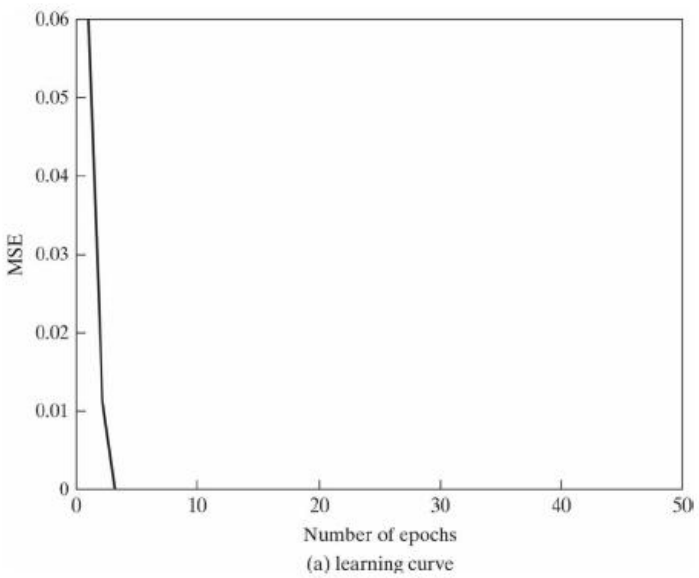
Experimental Setup

Training Data: 2000 training samples, 1000 from each region A and B.

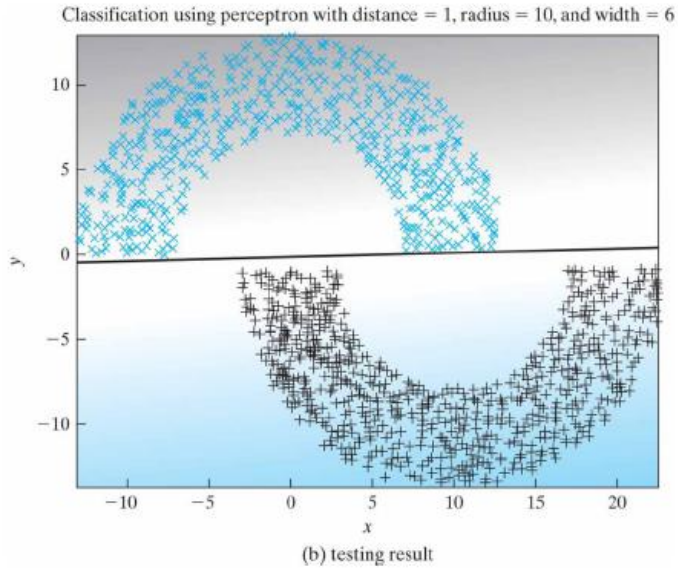
Testing Data: 2000 training samples

- Procedure:
- Generate training and testing data
- Train the perceptron using the training set
- Test performance using the testing dataset
- Generate performance metrics and error plot

Classification Results with Distance d =1



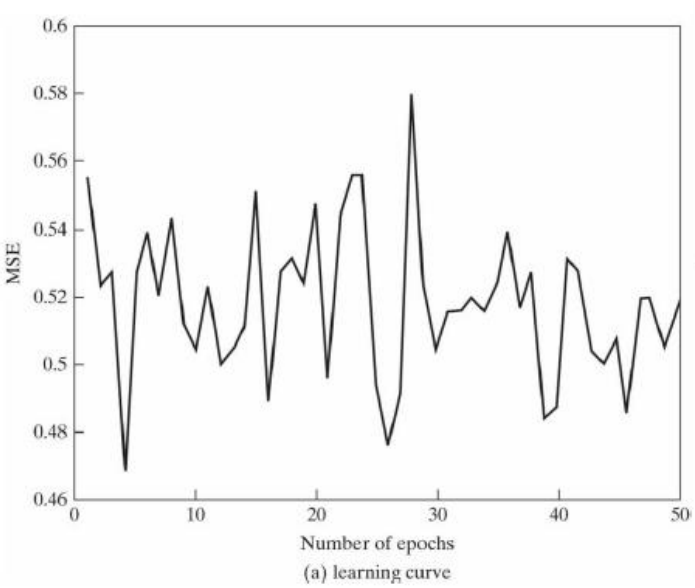
(a) learning curve

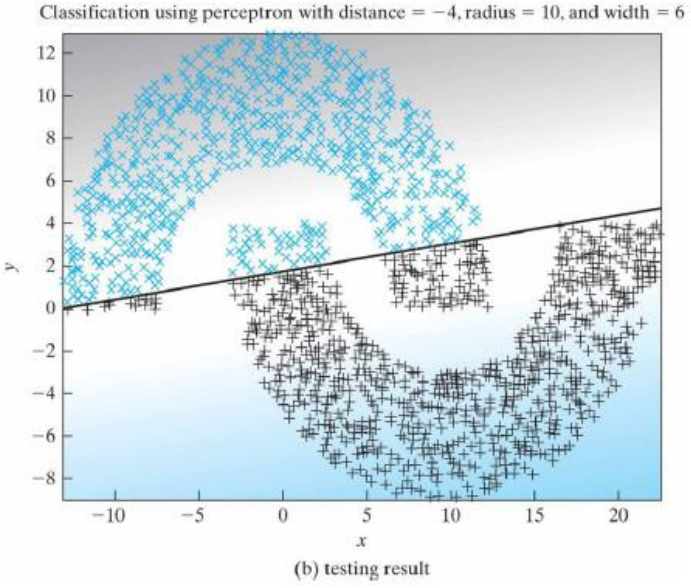


(b) testing result

Perfect linear separability exists allowing perfect classification accuracy. Algorithm converges in 3 iterations.

Classification Results with Distance d = -4





The separation distance violates linear separability of classes. A classification accuracy of 90:7% is achieved.

The Batch Perceptron Algorithm

Instead of adjusting the weights after each input sample is presented, we update the weights after a batch of samples has been presented. Define a general perceptron cost function,

$$J(w) = \sum_{x \in H} (-\boldsymbol{w}^T x)$$

where **H** is the set of samples incorrectly classified by the perceptron. Differentiating w.r.t **w** yields the gradient vector,

$$\nabla J(w) = \sum_{x \in H} (-x)$$

Thus, the weight update rule for the batch process takes the form,

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta \nabla J(w)$$
$$= \mathbf{w}(n) + \eta \sum_{x \in H} (x)$$

The Error Surface and Search Method

Let's reconsider our two-class pattern recognition problem: The AND logic gate.

We have two classes A and B, such that,

$$y = 1 \text{ if } x \in A$$

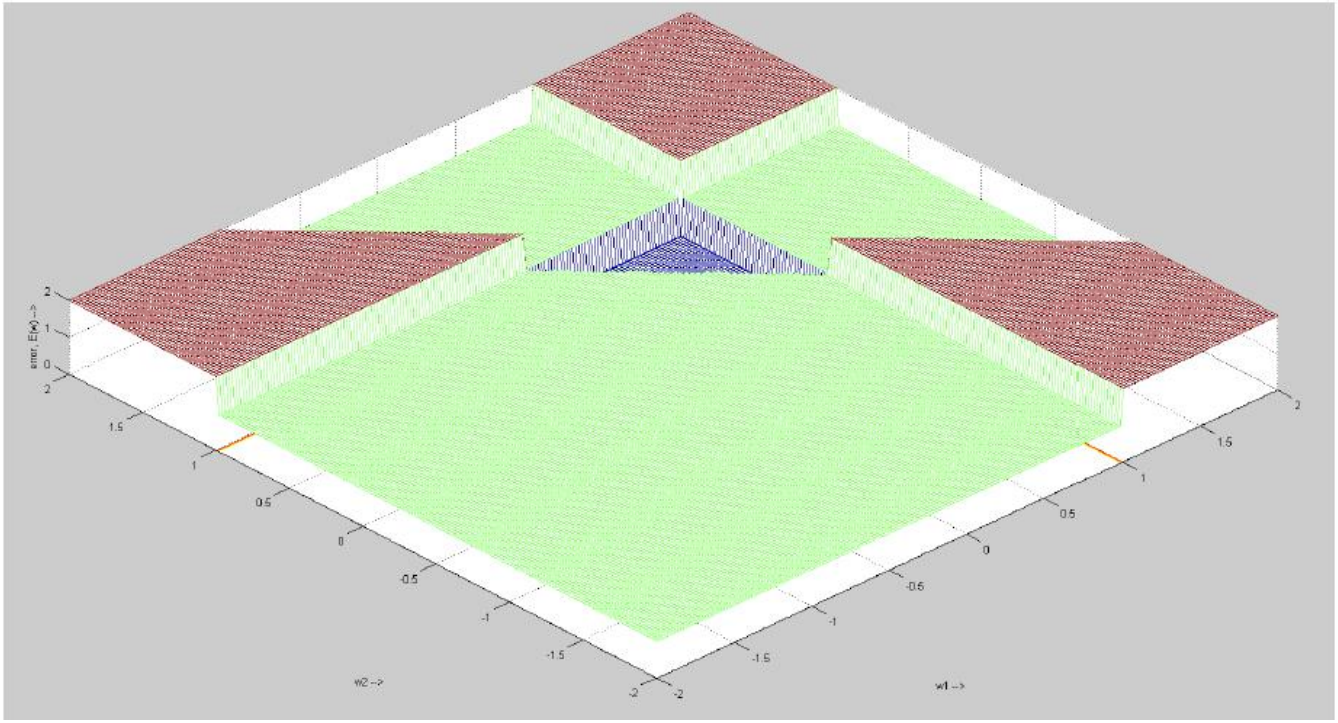
$$y = 0 \text{ if } x \in B$$

Let the error function for this problem be,

$$E(w) = (\sum_{x \in A} (1 - y_x) + \sum_{x \in A} y_x)$$

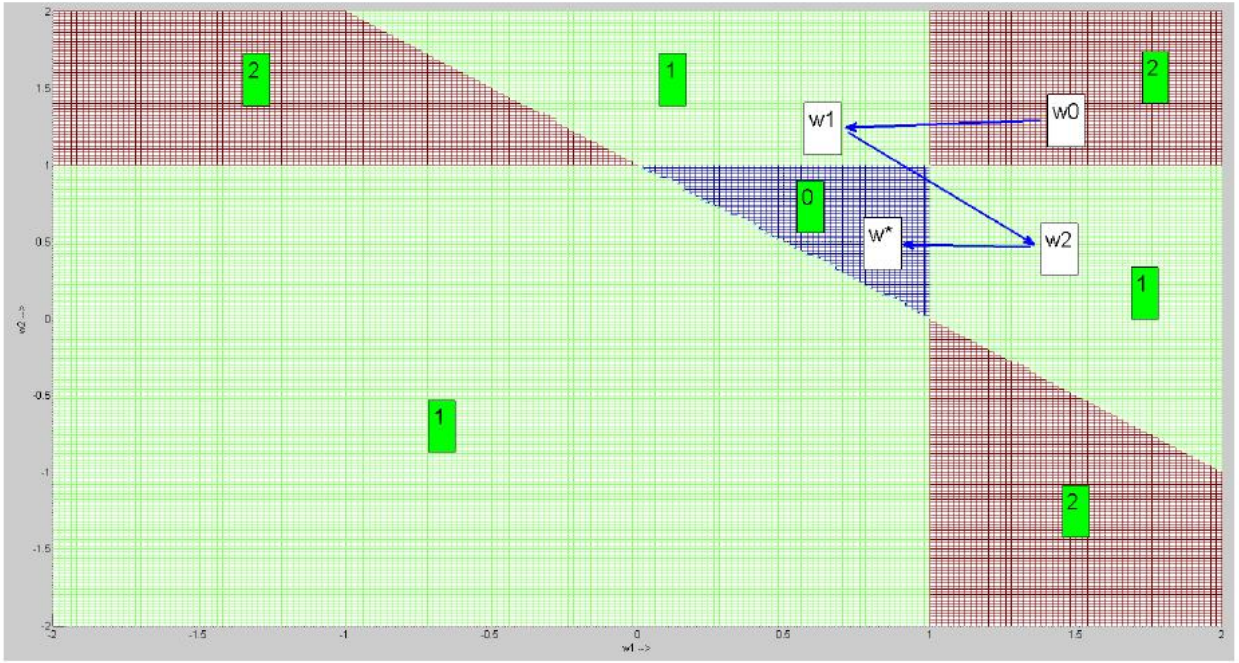
The error is the number of incorrectly classified samples. The maximum possible error is equal to the number of input samples tested. The objective of learning is to find the weight vector w that minimizes E(w).

Error Surface for the AND Operation



The blue triangular region is the region of zero error.

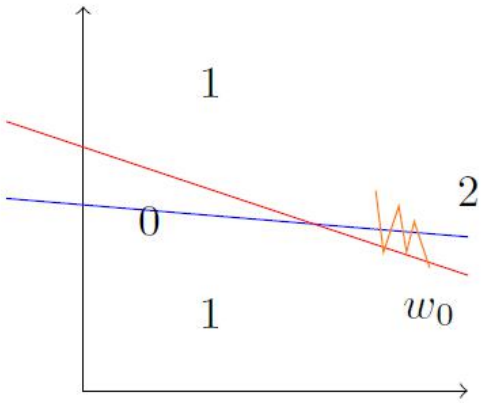
Iterative Search for the Region of Minimal Error



w* is the optimal weight vector

Problem with Perceptron Learning

The search direction is determined by the local error. The global shape of the error surface is not take into account.



Greedy local algorithm makes many unnecessary weight adjustments before converging.