CS 4104
APPLIED MACHINE LEARNING

Dr. Hashim Yasin

National University of Computer and Emerging Sciences,

Faisalabad, Pakistan.

PERCEPTRON

Perceptron

- A simplest type of ANN system is based on a unit called a **perceptron**. A perceptron
  - takes a **vector of real-valued inputs**,
  - calculates a linear combination of these inputs,
  - then **outputs** a 1 if the result is greater than some ***threshold*** and -1 otherwise.
- More precisely, given inputs $\boldsymbol{x_1}$ through $\boldsymbol{x_n}$ the output $\boldsymbol{o(x_1, \ldots, x_n)}$ computed by the perceptron is

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if} \quad w_0 + w_1 x_1 +, \ldots, +w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

Perceptron

$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if} \quad w_0 + w_1 x_1 +, \dots, + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

- where each $\boldsymbol{w_i}$ is a real-valued constant, or weight,
  - that *determines the contribution of input* $\boldsymbol{x_i}$ to the perceptron output.
- The quantity $(\boldsymbol{w_0})$ is a threshold
  - the weighted combination of inputs $\boldsymbol{w_1 x_1 + \dots + w_n x_n}$ must exceed in order for the perceptron to output a 1.

Perceptron

- We may imagine an *additional constant input* $\boldsymbol{x_0} = \mathbf{1}$, allowing to write the above inequality as,

$$\sum_{i=0}^{n} w_i x_i > 0$$

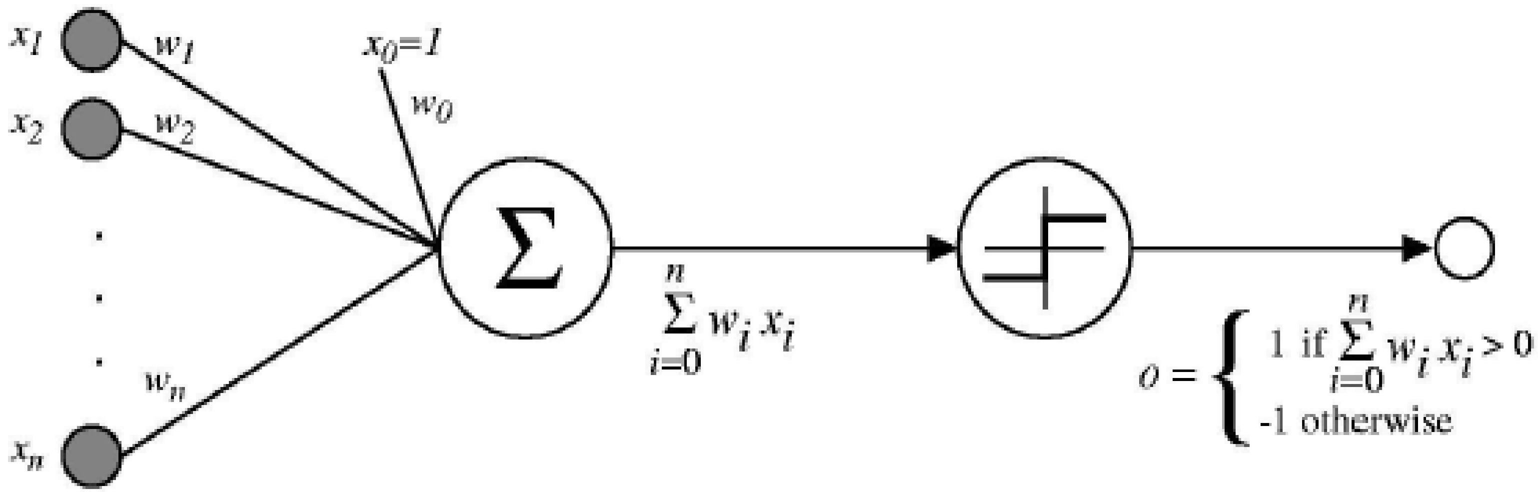
or in **vector form** as

$$o(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad \mathbf{w}.\mathbf{x} > 0 \\ -1 & \quad \text{otherwise} \end{cases}$$

$$\mathbf{x} = \vec{x}$$

$$sgn(y) = \begin{cases} 1 & \text{if} \quad y > 0 \\ -1 & \text{otherwise} \end{cases}$$

Perceptron



$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Perceptron Training Rule

- The ***perceptron training rule,*** which revises the weight w_i associated with input x_i according to the rule:

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- t is target value
- o is perceptron output
- η is small constant (e.g., 0.1) called *learning rate*

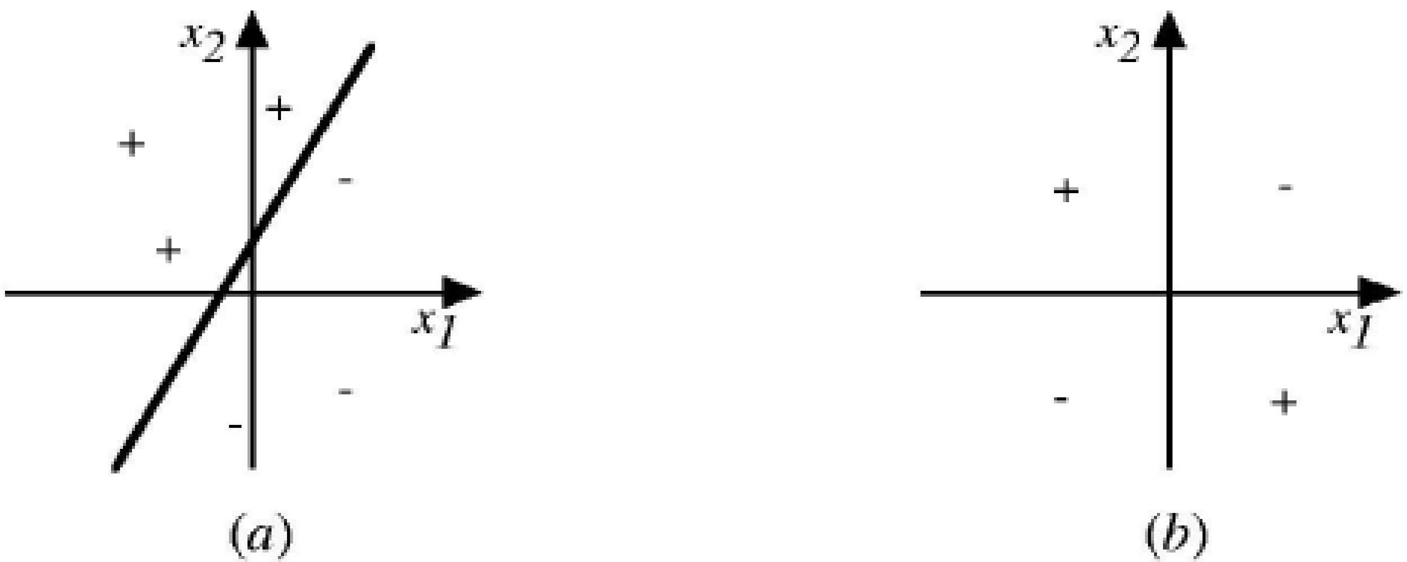
Perceptron Training Rule

- The **perceptron rule** finds a successful weight vector when the training examples are **linearly separable,**
- It fails to converge if the examples are **not linearly separable.**
- The solution is … **Delta Rule** also known as **(Widrow-Hoff Rule)**

Delta Rule

- use ***gradient descent*** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.

Perceptron



The **decision surface** represented by **two-input perceptron** $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$.
(a) A set of training examples and the decision surface of a perceptron that classifies them correctly. ***(b)*** A set of training examples that is not linearly separable.

DELTA RULE

Delta Rule

- In <u>**perceptron training rule,**</u> we employ ***thresholded perceptron***

$$o(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad \mathbf{w}.\mathbf{x} > 0 \\ -1 & \quad \text{otherwise} \end{cases}$$

- The <u>**delta training rule**</u> is the task of training an ***unthresholded perceptron***;
  - a ***linear unit*** for which the output o is given by,

$$o(\mathbf{x}) = \mathbf{w}.\mathbf{x}$$

  - A linear unit corresponds to the *first stage* of a perceptron, ***without*** the threshold.

Delta Rule

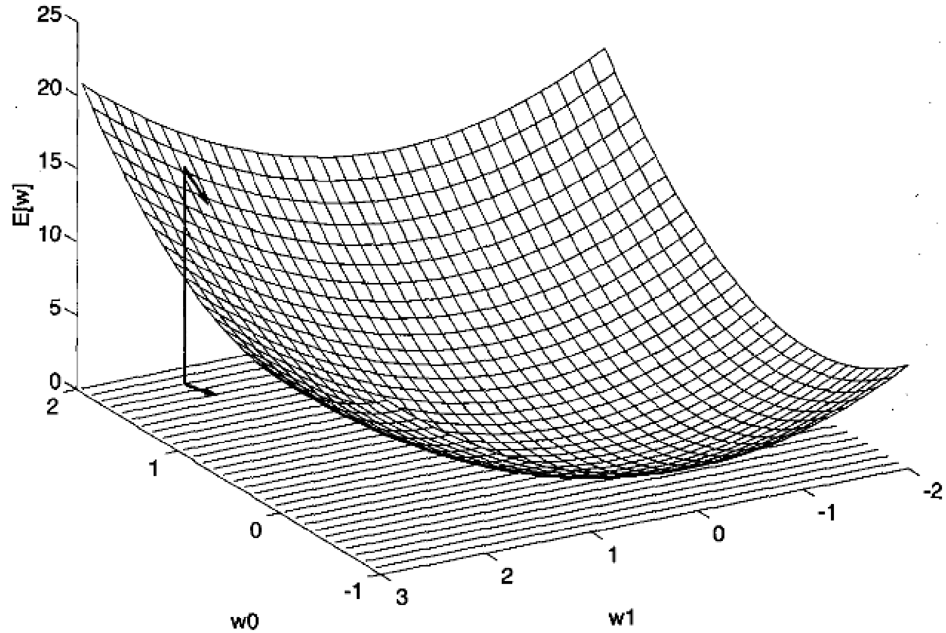
- In order to derive ***a weight learning rule for linear units***,
  - Specify a measure for the ***training error*** of a hypothesis (weight vector), relative to the training examples.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

  - D is the set of training examples,
  - t_d is the target output for training example d,
  - o_d is the output of the linear unit for training example d
  - E is characterized as a function of $\boldsymbol{w}$, because the linear unit output o depends on this weight vector.

Hypothesis Space

- For a linear unit with two weights, the hypothesis space H is the $\boldsymbol{w_0}, \boldsymbol{w_1}$ plane.



Error of different hypotheses.

Gradient Descent

- Gradient descent search determines *a weight vector that minimizes **E*** by
  - Starting with an arbitrary initial weight vector,
  - Repeatedly modifying it in small steps.
  - At each step, the ***weight vector is altered in the direction*** that produces the **steepest descent** along the error surface,
  - This process continues until the **global minimum error** is reached.

Gradient Descent

How can we calculate the direction of steepest descent along the error surface?

- This direction can be found by computing the derivative of ***E*** with respect to each component of the vector $\boldsymbol{w}$.

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n} \right]$$

- Notice $\nabla E(\vec{w})$ is itself a vector, whose components are the partial derivatives of ***E*** with respect to each of the $\boldsymbol{w_i}$.

Gradient Descent

- The **gradient specifies the direction** that produces the steepest increase in E. The training rule for gradient descent is,

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

where,

$$\Delta\vec{w} = -\eta \nabla E(\vec{w})$$

- The negative sign is present because we want to move the weight vector in the direction that ***decreases*** E.

Gradient Descent

- This training rule can also be written in its **component form,**

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

$$w_i \leftarrow w_i + \Delta w_i$$

where,

$$\Delta\vec{w} = -\eta \nabla E(\vec{w})$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

- The steepest descent is achieved by altering each component w_i in proportion to $\frac{\partial E}{\partial w_i}$

Gradient Descent

- The vector of derivatives $\frac{\partial E}{\partial w_i}$ that form the gradient can be obtained by differentiating *E* from delta rule

$$\boxed{E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2}$$

$$\boxed{o(\mathbf{x}) = \mathbf{w}.\mathbf{x}}$$

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_{d\in D}(t_d - o_d)(-x_{id})
\end{aligned}$$

Gradient Descent

- We now have an equation that gives $\frac{\partial E}{\partial w_i}$ in terms of
  - the linear unit inputs x_{id},
  - outputs o_d,
  - target values t_d associated with the training examples
- The weight update rule for gradient descent becomes,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\boxed{\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\, x_{id}}$$

Gradient Descent

GRADIENT-DESCENT(*training_examples*, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i$$

$$\boxed{\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\, x_{id}}$$

Gradient Descent

- Gradient descent is an important general paradigm for learning.
- It is a *strategy for searching through a* ***large or infinite*** *hypothesis space* that can be applied whenever

  1) the hypothesis space contains **continuously parameterized hypotheses** (e.g., the weights in a linear unit),
  2) the **error can be differentiated** with respect to these hypothesis parameters

Gradient Descent

- The **key practical difficulties** in applying gradient descent are:

  a) ***converging to a local minimum*** can sometimes be **quite slow** (i.e., it can require many thousands of gradient descent steps),

  b) ***if there are multiple local minima*** in the error surface, then there is no guarantee that the procedure will find the global minimum.

Stochastic Gradient Descent

- The idea behind stochastic gradient descent is to approximate the gradient descent search by **updating weights incrementally**, following the calculation of the error for ***each* individual example**.

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\, x_{id}$$

$$\Delta w_i = \eta (t - o)\, x_i$$

Stochastic Gradient Descent

- One way to view this stochastic gradient descent is to consider a **distinct error function** defined for ***each individual training example*** ***d*** *as follows*.

$$E_d(\vec{w}) = \frac{1}{2}(t_d - o_d)^2$$

  - where t, and o_d are the target value and the unit output value for training example d.
  - Stochastic gradient descent iterates over the training examples d in D,
  - at each iteration altering weights according to the gradient

Stochastic Gradient Descent

GRADIENT-DESCENT(*training_examples*, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each w_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \eta(t - o)x_i$$

The Key Difference

- In **stochastic gradient descent**, weights are updated upon examining ***each*** training example.
- Whereas in **standard gradient descent**, the error is summed over ***all*** examples before updating weights,
  - **Standard gradient descent** requires **more computation** per weight update step.
  - **Standard gradient descent** is often used with a **larger step size** per weight update than stochastic gradient descent.

The Key Difference

- When there are multiple local minima with respect to $\boldsymbol{E}(\boldsymbol{w})$,

  - The **stochastic gradient descent** can sometimes *avoid falling into these local minima*,
  - It is due to the reason that it uses various $\nabla E_d(\vec{w})$ rather than $\nabla E(\vec{w})$ to guide its search.

Training Rules

- **Perceptron Training Rule:** guarantee to succeed if
  - training examples are linearly separable
  - Sufficiently small learning rate

- **Delta Rule:**
  - use gradient descent
  - converges only asymptotically toward the minimum error hypothesis,
  - converges regardless of whether the training data are linearly separable.

Reading Material

- **Artificial Intelligence,** A Modern Approach

  Stuart J. Russell and Peter Norvig
  - **Chapter 18.**
- **Machine Learning**

  Tom M. Mitchell
  - **Chapter 4.**