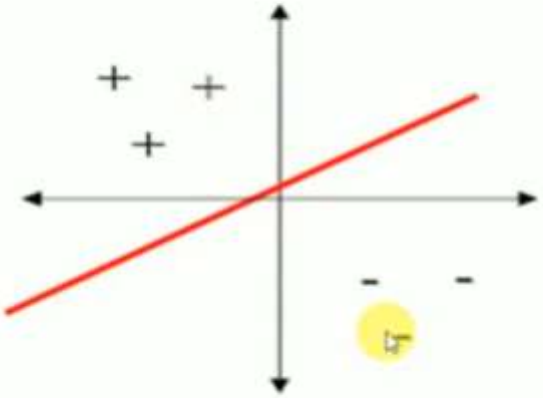
Module 5 : Logistic regression

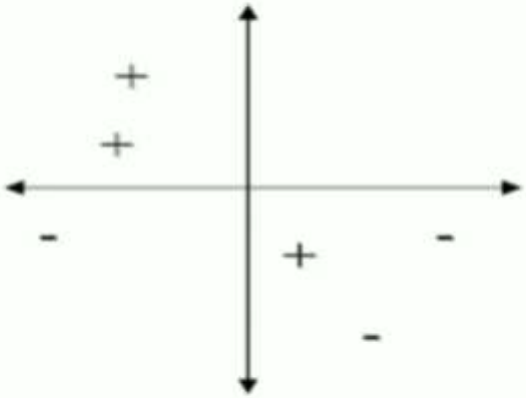
The development of the perceptron was a big step towards the goal of creating useful connectionist networks capable of learning complex relations between inputs and outputs. In the late 1950's, the connectionist community understood that what was needed for further development of connectionist models was a mathematically-derived (and thus potentially more flexible and powerful) rule for learning. By early 1960's, the Delta Rule [also known as the *Widrow & Hoff Learning rule* or the *Least Mean Square* (LMS) rule] was invented by *Widrow and Hoff*. This rule is similar to the perceptron learning rule by *McClelland & Rumelhart, 1988*, but is also characterized by a mathematical utility and elegance missing in the perceptron and other early learning rules.

The **Delta Rule** uses the difference between *target activation* (i.e., target output values) and *obtained activation* to drive learning. For reasons discussed below, the use of a *threshold activation function* (as used in both the *McCulloch-Pitts* network and the perceptron) is dropped & instead a linear sum of products is used to calculate the activation of the output neuron (alternative activation functions can also be applied). Thus, the activation function is called a *Linear Activation function*, in which the output node's activation is simply equal to the sum of the network's respective input/weight products. The strength of network connections (i.e., the values of the weights) are adjusted to reduce the difference between *target* and *actual* output activation (i.e., error). A graphical depiction of a simple two-layer network capable of deploying the Delta Rule is given in the figure below (Such a network is not limited to having only one output node):

Gradient Descent and the Delta Rule

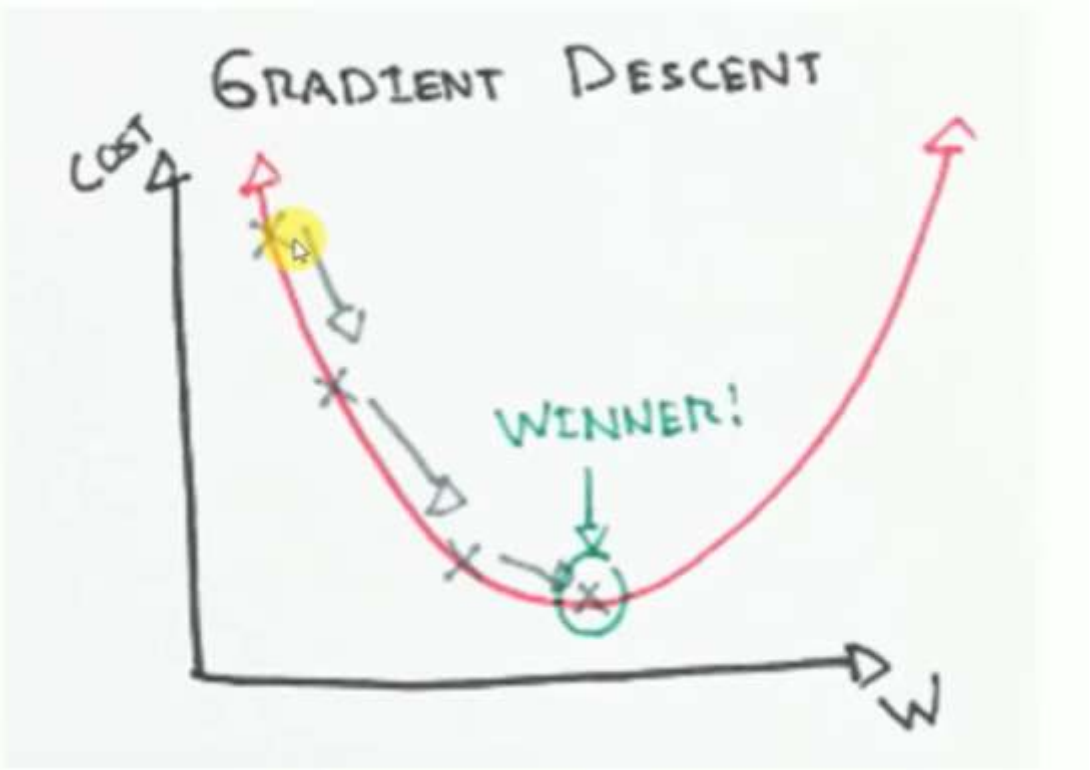


Linearly separable



Non-linearly separable

- Perceptron rule finds a successful weight vector when the training examples are linearly separable, but it can fail to converge if the examples are not linearly separable.
- A second training rule, called the ***delta rule,*** is designed to overcome this difficulty.
- If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept.
- The key idea behind the delta rule is to use ***gradient descent*** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.
- This rule is important because gradient descent provides the basis for the BACKPROPAGATON algorithm, which can learn networks with many interconnected units.
- It is also important because gradient descent can serve as the basis for learning algorithms that must search through hypothesis spaces containing many different types of continuously parameterized hypotheses.



- The delta training rule is best understood by considering the task of training an ***unthresholded*** perceptron; that is, a ***linear unit*** for which the output ***o*** is given by

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

- Thus, a linear unit corresponds to the first stage of a perceptron, without the threshold.
- In order to derive a weight learning rule for linear units, let us begin by specifying a measure for the ***training error*** of a hypothesis (weight vector), relative to the training examples.
- Although there are many ways to define this error, one common measure is

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

- where D is the set of training examples, ***td*** is the target output for training example *d*, and ***od*** is the output of the linear unit for training example d.

- 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 $\vec{w}$.
- This vector derivative is called the ***gradient*** of E with respect to $\vec{w}$, written ***as*** $\nabla E(\vec{w})$.

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

- Since the gradient specifies the direction of steepest increase of E, the training rule for gradient descent is,

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

where

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

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

where

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

- Here n is a positive constant called the learning rate, which determines the step size in the gradient descent search. The negative sign is present because we want to move the weight vector in the direction that ***decreases*** E.
- This training rule can also be written in its component form

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

where

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

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

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_{d\in D}(t_d - o_d)(-x_{id})$$

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

$$o(\vec{x}) = \vec{w}\cdot\vec{x}$$

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

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

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

- Gradient Descent and the Delta Rule is used separate the Non-Linearly Separable data.
- Weights are updated using the following rule,

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

- Where,

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

GRADIENT DESCENT ALGORITHM

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

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

GRADIENT DESCENT ALGORITHM

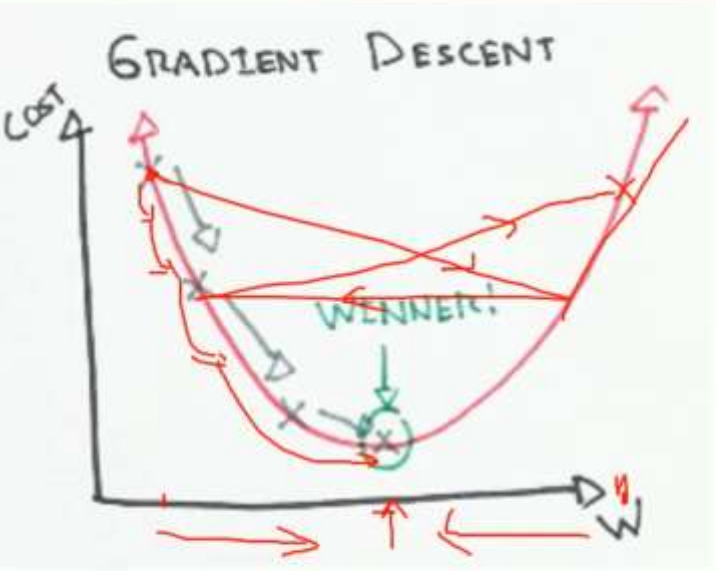
Gradient descent is an important general paradigm for learning.

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), and
2. the error can be differentiated with respect to these hypothesis parameters.

- The key practical difficulties in applying gradient descent are
  1. converging to a local minimum can sometimes be quite slow (i.e., it can require many thousands of gradient descent steps), and
  2. if there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.

Gradient Descent - Delta Rule



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

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

where

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

Gradient Descent - Delta Rule

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_{d\in D}(t_d - o_d)(-x_{id})$$

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

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

$$o(\vec{x}) = \vec{w}\cdot\vec{x}$$

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

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

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