Machine Learning

Chapter 4 Artificial Neural Network （ANN）

4.1 Introduction

- ANN provide a robust approach to approximating real-valued, discrete-valued, and vector-valued functions
  - For certain types of problem, such as learning to interpret complex real-world sensor data, ANN are very effective

- BP Algorithm has proven successful in many practical problems

4.1.1 Biological Motivation

- The biological learning system is built of very complex webs of interconnected neurons.
  - The human brain is a densely interconnected network (10^{11} neurons, each connected to 10^4 others)
  - Neuron activity is typically excited or inhibited through connections to other neurons.
- The information-processing abilities of biological neural systems
  - The neuron switching times is slow compared to computer speed. Yet humans are able to quickly make complex decisions (why?)
  - highly parallel computation based on distributed representations
- ANN system is designed to capture such abilities

- In fact, ANN are loosely motivated by biological neural systems

- Two groups of researchers in ANN
  - Using ANNs to study and model biological learning processes
  - Obtaining highly effective machine learning algorithms, independent of whether these algorithms mirror biological processes

- Within this book our interest fits the latter group

4.2 Neural Network Representations

- ALVINN (Pomerleau 1993)
  - Automatically driving
  - The input : 30x32 image
    - the lines entering the node from below are its inputs.
  - Hidden units
    - their output is available only within the network and is not available as part of the network output.
  - Output units
    - Each output unit corresponds to a particular steering direction, and the output values determine the driving action

Drives 70 mph on a public highway

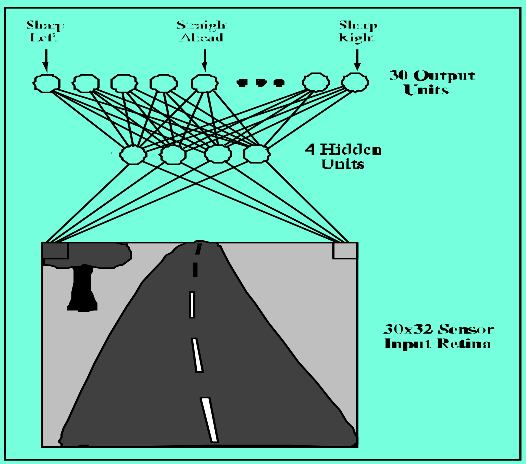
Camera image

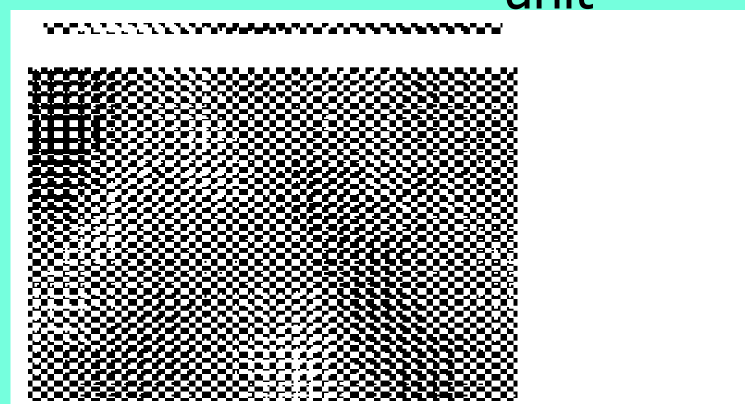
30x32 weights into one out of four hidden unit

30 outputs for steering

4 hidden units

30x32 pixels as inputs





- The network structure of ALVINN
  - The individual units are interconnected in layers that form a directed acyclic graph.

- Types of structures of ANN
  - Acyclic or cyclic
  - Directed or undirected

- We focus on the BP Network
  - structure : a directed graph, possible containing cycles.
  - Learning: choosing a weight value for each edge in the network

4.3 Appropriate Problems For Neural Network Learning

- Problem with the following characteristics
  - Instances are represented by many attribute-value pairs
    - more symbolic representations, high dimensions
  - The target function : discrete-valued, real-valued, or a vector
  - The training examples may contain errors
    - noisy, complex sensor data.(Cameras, microphones)

- Fast evaluation of the learned target function may be required
- Long training times are acceptable
- The ability of humans to understand the learned target function is not important

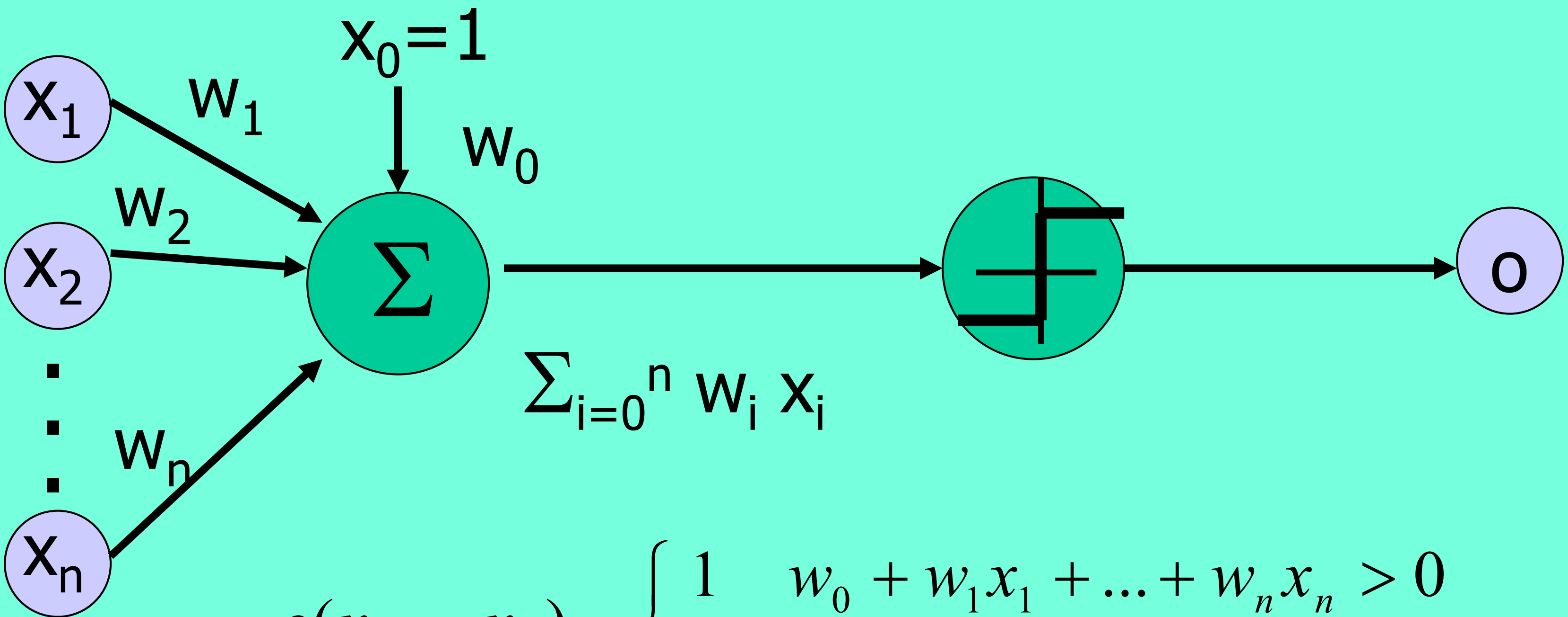
- BP is the most commonly used ANN learning technique

The Organization of the rest of this chapter

- Learning algorithms for training single units
- Several designs for the primitive units
  - perceptron
  - linear unit
  - sigmoid unit
- BP algorithm
- Several general issues
  - The representational capabilities of ANNs
  - Nature of the hypothesis space search
  - Overfitting problems
  - Alternatives to the BP algorithm
- A detailed example

4.4 Perceptron

- Linear threshold unit (LTU)



$$o(x_1,\ldots,x_n)=\begin{cases}1 & w_0+w_1x_1+\ldots+w_nx_n>0\\ -1 & otherwise\end{cases}$$

- Perceptron function

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

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

- Hypotheses space in perceptron learning

$$H = \{\vec{w} \mid \vec{w} \in R^{n+1}\}$$

4.4.1 Representational Power of Perceptrons

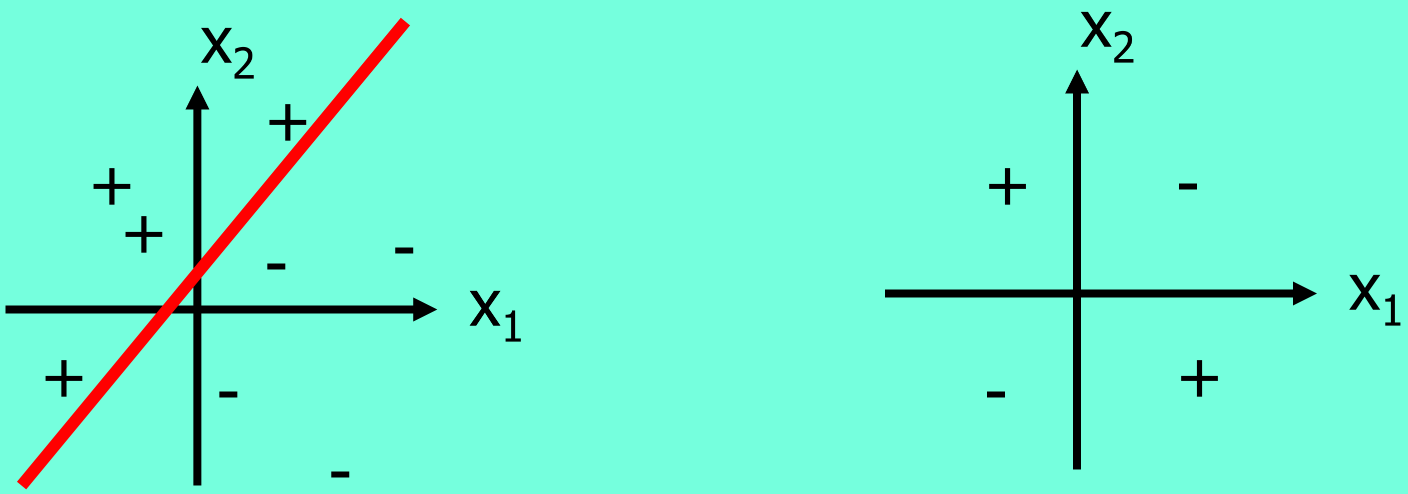
- Perceptron can be viewed as a hyperplane decision surface in n-dimensional space of instances
  - Positive instances lying on one side of the hyperplane
  - Negative instances lying on the other side
- The equation for this decision hyperplane

$$\vec{w} \cdot \vec{x} = 0$$

- Linearly separable sets

- Representational Power of a single perceptron .
  - Boolean functions : m-of-n functions (and or)
    - AND   OR NAND NOR
  - Functions that are not linearly separable (e.g. Xor)   are not representable

Decision Surface of a Perceptron



- And(x_1,x_2) choose weights w_0=-1.5, w_1=1, w_2=1
- Xor are not representable

- Representational Power of Networks of threshold units
  - representing a rich variety of functions
  - Every boolean function can be represented by some network of perceptrons only two level deep
  - One way
    - Representing the Boolean function in disjunctive normal form
    - $(x_1 \wedge x_2 \wedge x_3) \vee (x_3 \wedge x_4 \wedge x_5)$

4.4.2 The Perceptron Training Rule

- Understanding how to learn the weights for a single perceptron
- The learning problem
  - determining a weight vector that causes the perceptron to produce the correct 1 or -1 for training examples
- Learning algorithms
    - Perceptron training rule
    - The delta rule
  - Both converge to an acceptable weight vector .
    - the base for multi-layer network learning algorithm.

Perceptron Learning Rule

$w_i = w_i + \Delta w_i$

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

$t=c(x)$: the target value

o : the perceptron output

η : a small constant (e.g. 0.1), called *learning rate*

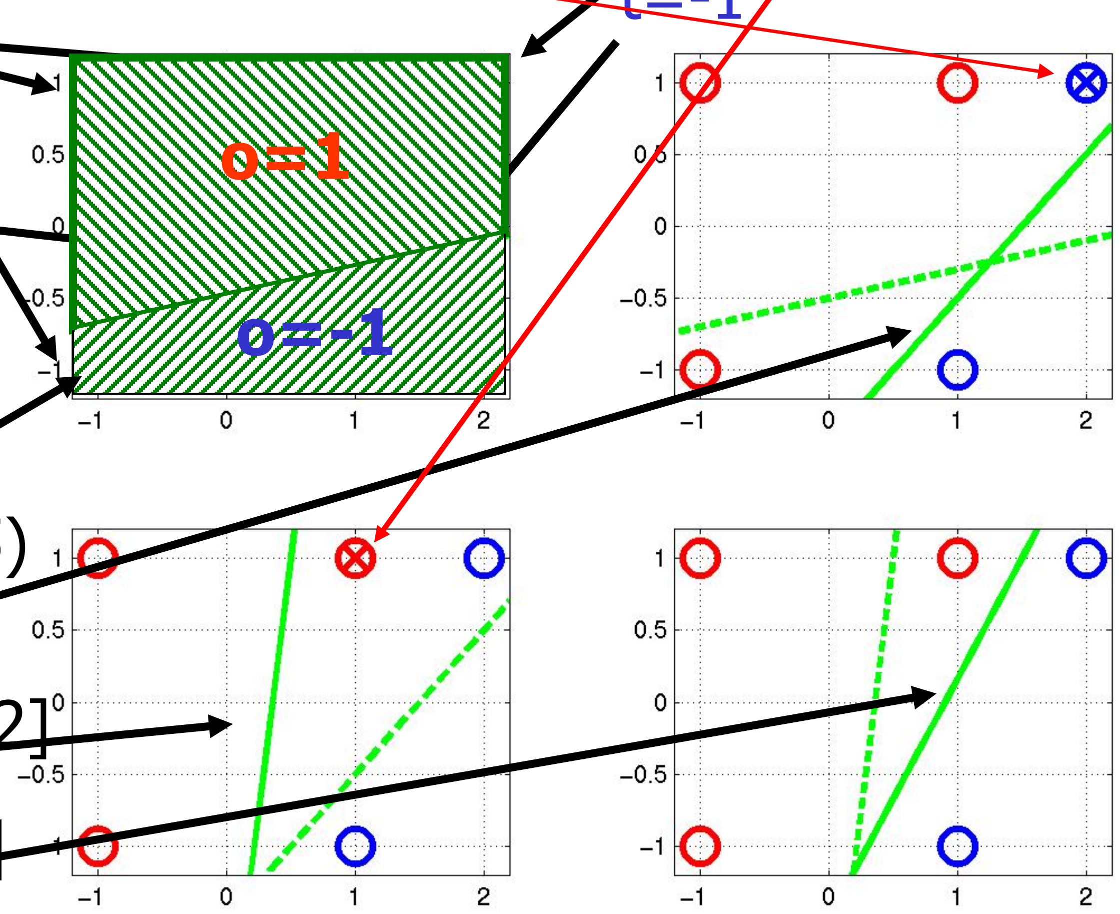
- If the output is correct ($t=o$), w_i are not changed
- If the output is incorrect ($t \neq o$), w_i are changed such that the output of the perceptron for the new weights is *closer* to t.
- Converge condition of the learning algorithm
  - the training data is linearly separable
  - η is sufficiently small

(x,t)=([2,1],-1)

o=sgn(0.45-0.6+0.3) =1

(x,t)=([1,1],1)

o=sgn(0.25-0.7+0.1) =-1



w0=[0.25 –0.1 0.5]

$0.25-0.1\ x_1+0.5\ x_2=0$

(x,t)=([-1,-1],1)

o=sgn(0.25+0.1-0.5)

=-1

Δw1=[0.2 –0.2 –0.2]

Δw2=[-0.2 –0.4 –0.2]

Δw3=[0.2 0.2 0.2]

- Convergence is assured（Minskey & Papert 1969）
  - Provided the training examples are linearly separable and provided a sufficiently small η
  - If the data are not linearly separable， Convergence is not assured

4.4.3 Gradient Descent and the Delta Rule

- Delta Rule overcome the difficulty of the perceptron rule
  - if the training example are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept

- The key idea behind the delta rule
  - using gradient descent to search H to find the weights that best fit the training examples

- Understanding Delta rule
  - Task of Delta rule: training an unthresholded perceptron

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

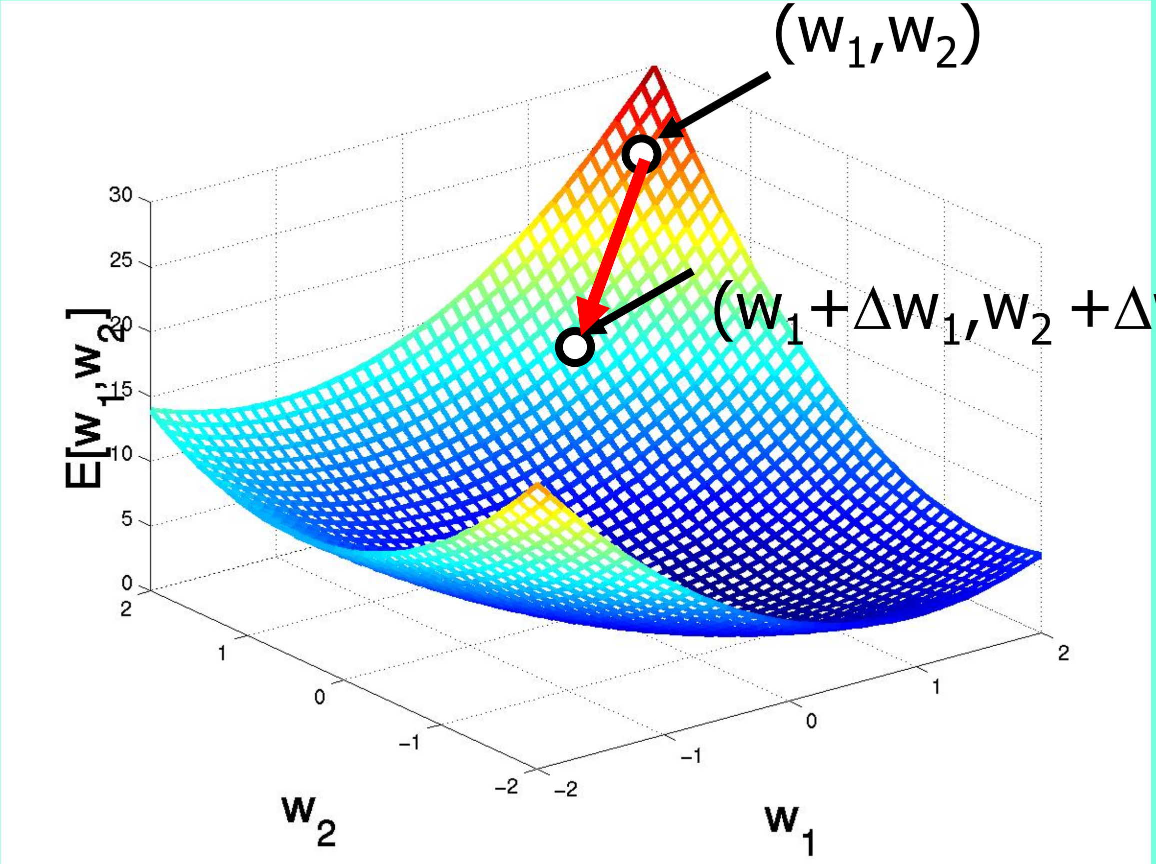
  - Specify a measure for the training error of a hypothesis (weight vector)

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

  - Finding hypothesis $\vec{w}$ that minimizes E
    - that Chapter 6 provides a Bayesian justification for choosing this particular definition of E, under certain conditions the hypothesis that minimizes E is also the most probable hypothesis in H given the training data

4.4.3.1 Visualizing The Hypothesis Space

- Figure 4-4
  - Hypothesis space H is w1,w2 plane. For linear units this error surface has a single global minimum



- Gradient descent search
  - determines a weight vector that minimizes E
    - starting with an arbitrary initial weight vector w
    - w is altered in the direction that produces the steepest decent along the error surface
    - repeatedly modifying w at each step until the global minimum error is reached

4.4.3.2 Derivation of The Gradient Descent Rule

- How can we find the direction of steepest descent?
  - computing the derivative of E with respect to each component of the vector $\vec{w}$, written $\nabla E(\vec{w})$
  - The gradient specifies the direction that produces the steepest increase in E.
    - the negative of this vector therefore gives the direction of steepest decrease
- Training rule for gradient descent

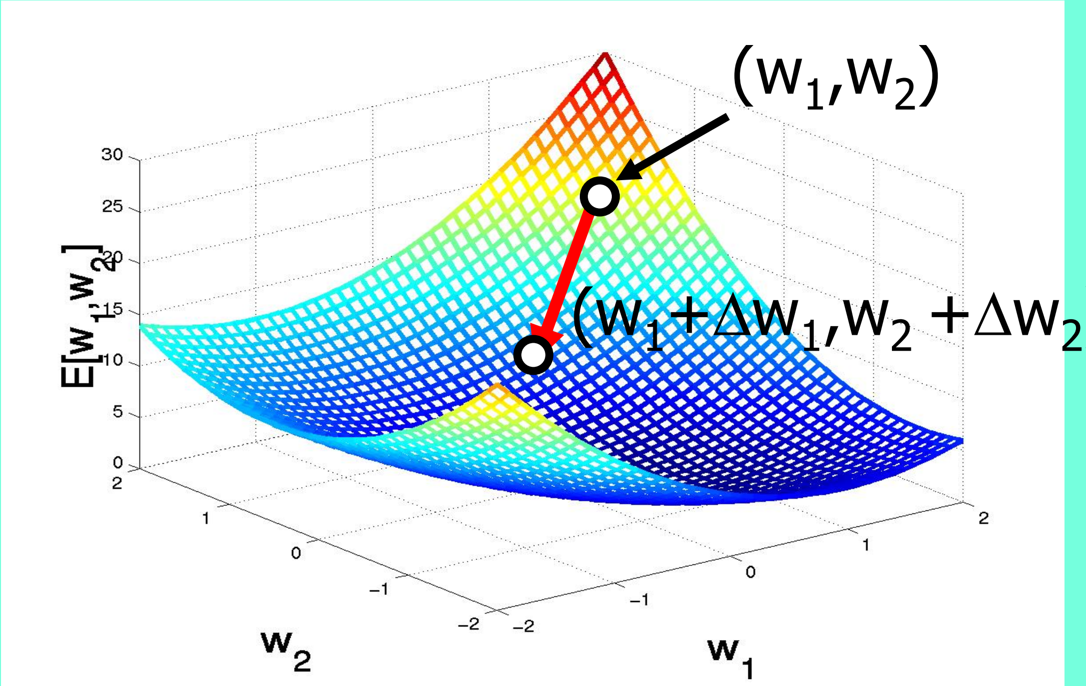
$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

where

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

Gradient Descent

D={<(1,1),1>,<(-1,-1),1>, <(1,-1),-1>,<(-1,1),-1>}



Gradient:
$\nabla E[w]=[\partial E/\partial w_1, \ldots \partial E/\partial w_n]$

$\Delta w=-\eta \nabla E[w]$

$\Delta w_i=-\eta \partial E/\partial w_i$

$= -\eta \partial (1/2\Sigma_d(t_d-o_d)^2) /\partial w_i$

$= -\eta 1/2\Sigma_d \partial ((t_d-o_d)^2) /\partial w_i$

$= -\eta 1/2\Sigma_d \partial ((t_d-o_d)^2) / \partial o_d * (\partial o_d / \partial w_i)$

$= \eta \Sigma_d(t_d- o_d)(x_{id})$

$o_d=\Sigma w_i x_{id}$

Gradient Descent

Gradient-Descent(*training_examples*, η)

- Each training example is a pair of the form <(x1,…xn),t> where (x1,…,xn) is the vector of input values, and t is the target output value, η is the learning rate (e.g. 0.1)
- Initialize each wi to some small random value
- Until the termination condition is met, Do
  - Initialize each Δwi to zero
  - For each <(x1,…xn),t> in *training_examples* Do
    - Input the instance (x1,…,xn) to the linear unit and compute the output o
    - For each linear unit weight wi Do
      - Δwi= Δwi + η (t-o) xi
  - For each linear unit weight wi Do
    - wi=wi+Δwi

- Convergence is assured
  - Because the error surface contain only a single global minimum, this algorithm will converge to a weight vector $\vec{w}$ with minimum error, regardless of whether the training examples are linearly separable, given a sufficiently small learning rate

- One common modification to the algorithm
  - gradually reducing the value of learning rate as the number of gradient descent steps grows

4.4.3.3 Stochastic Approximation to Gradient Descent

- Gradient descent is an important general paradigm for learning.
  - Used for searching through a larger or infinite H
  - H should satisfy the following conditions
    - H contains continuously parameterized hypotheses
    - The error can be differentiated with respect to these hypothesis parameters
- The key practical difficulties in applying gradient descent
  - Converging speed is slow
  - If there are multiple local minimum in the error surface, then there is no guarantee that the procedure will find the global minimum

- Stochastic gradient(incremental gradient descent)
  - Approximate this gradient descent search by updating weights incrementally, following the calculation of the error for each individual example
  - The modified training rule on table 4.1

Stochastic gradient(*training_examples*, η)

- Each training example is a pair of the form <(x1,…xn),t> where (x1,…,xn) is the vector of input values, and t is the target output value, η is the learning rate (e.g. 0.1)
- Initialize each wi to some small random value
- Until the termination condition is met, Do
  - Initialize each Δwi to zero
  - For each <(x1,…xn),t> in *training_examples* Do
    - Input the instance (x1,…,xn) to the linear unit and compute the output o
    - $\Delta wi = \Delta wi + \eta\ (t-o)\ xi$
    - $wi = wi + \Delta wi$

- Understanding Stochastic gradient
  - Create a distinct error function for each individual training example d

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

  - The sequence of these weight updates, when iterated over all training examples, provides a reasonable approximation to the gradient of the original error function
    - $\Delta wi = \Delta wi + \eta\ (t-o)\ xi$
  - SGD approximates true GD arbitrarily closely
    - When the learning rate sufficiently small,

- The key difference between standard gradient descent and stochastic gradient descent
  - See page94 (three points)
    - Number of training samples
    - Step size per weight update
    - Chance for escaping local minima
- Both stochastic and standard gradient descent methods are commonly used in practice

- Delta rule are also called LMS rule, Adaline rule, Windrow-Hoff rule
- The difference between Delta rule equation(4.10) and the perceptron training rule in equation (4.4.2) (Same?)
  - $\Delta wi = \eta$ (t-o) xi
- Delta can also be used to train thresholded perceptron units
  - If the unthresholded output o can be trained to fit these values perfectly, then the threshold output o' will fit them as well
  - Even when the target values cannot be fit perfectly, the thresholded o' value will correctly fit the target value whenever the linear unit output o has the correct sign
- Delta rule will learn weights that minimize the error in the linear unit output o, these weights will not necessarily minimize the error in the thresholded output o'

4.4.4 Remarks

- Key difference between the perceptron rule and delta rule
  - Perceptron rule update weights based on the error in the thresholded perceptron output
  - Delta rule updates weights based on the error in the unthresholded linear combination inputs
- Difference in convergence properties
  - Perceptron rule converges after a finite numberof iterations to a h, provided the training examples are linearly separable
  - Delta rule converges only toward the minimum error h, possible requiring unbounded time, regardless of whether the training data are linearly separable

- A third algorithm for learning the weight vector is linear programming
  - Linear programming is a general, efficient method for solving sets of linear inequalities
  - This approach is valid only when the training examples are linearly separable
  - This approach can't scale to training multilayer networks. In contrast, the gradient descent approach can be easily extended to multilayer network