# Topic 2

## Artificial neural networks:

### Supervised learning

- Introduction, or applications
- Introduction , or how the brain works
- The neuron as a simple computing element
- The perceptron
- The ADALINE

- Note: lecture notes by M.Negnevitsky (U of Tasmania, Australia) and Bob Keller (Harvey Mudd College, CA) are used

## ANN applications

- **Optical character recognition**

U.S. mail zip-code recognizer

Kanjii: 4000 chars in 15 fonts, 99% accurate, 100k chars/sec (Sharp &Mitsubishi)

- **Communications**

Adaptive noise cancellation

Headphones

Conference telephones

**Process control**

Electric arc furnace control: 30MVA,50kamp transformer, $2M savings

Steel-rolling mill controller

Copier uniformity control (Ricoh)

Anti-lock brakes, etc. (Ford)

Food process control (M&M)

## ANN applications

- **Financial analysis**

Prediction of commodities market (18% vs. 12.3% by traditional methods)

Mortgage risk evaluator (AVCO, Irvine)

Real-estate evalution (Foster Onsley Conley)

Portfolio management (LBS Captial)

Currency trading (Citibank)

- **Crime prevention**

Bomb sniffer (JFK airport)

Credit card fraud detection (Visa, etc.)

## ANN applications

- **Object classification**

Grading grains from video images

Forensics: glass classification

High-energy physics: particle identification

**Warfare**

Missile guidance

- **Optical telescope focusing**
- **Biomedical**

Clinical diagnoses

Patient mortality predictions

Protein structure analysis

Electrode placement

## Approaches to AI

- Reverse Engineering of Biology

Understand real neurons well enough to model

**Simulate** neural behavior

- Simulated Evolution

Provide basic evolutionary mechanism for neurons

**Evolve** intelligent behavior

- Artificial Neural Networks

Develop a parameterized model for a class of problems

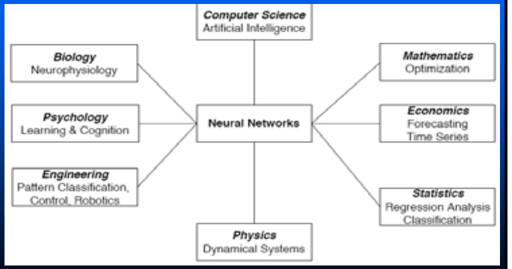
**Learn** the parameters

## Introduction, or how the brain works

Machine learning involves adaptive mechanisms that enable computers to learn from experience, learn by example and learn by analogy. Learning capabilities can improve the performance of an intelligent system over time. The most popular approaches to machine learning are **artificial neural networks** and **genetic algorithms**. This lecture is dedicated to neural networks.

- A **neural network** can be defined as a model of reasoning based on the human brain. The brain consists of a densely interconnected set of nerve cells, or basic information-processing units, called **neurons**.
- The human brain incorporates nearly 10 billion neurons and 60 trillion connections, *synapses*, between them. By using multiple neurons simultaneously, the brain can perform its functions much faster than the fastest computers in existence today.

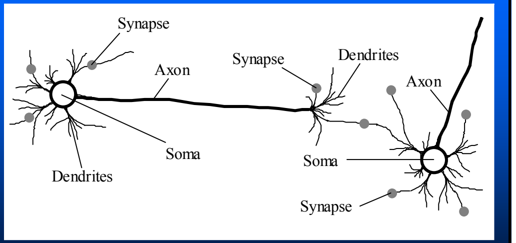
## Neural Networks: an Eclectic Discipline



## Biological Intelligence

- Intelligence, the ability to make decisions based upon input from the environment.
- Intelligence is realized by a **network** of **neurons**, for example the brain and the attendant sensory and motor neurons.
- Each neuron has a very simple structure, but an army of such elements constitutes a tremendous processing power.
- A neuron consists of a cell body, **soma**, a number of fibers called **dendrites**, and a single long fiber called the **axon**.

## Biological neural network



- Our brain can be considered as a highly complex, non-linear and parallel information-processing system.
- Information is stored and processed in a neural network simultaneously throughout the whole network, rather than at specific locations. In other words, in neural networks, both data and its processing are **global** rather than local.
- Learning is a fundamental and essential characteristic of biological neural networks. The ease with which they can learn led to attempts to emulate a biological neural network in a computer.

## Natural NN: characterisitics

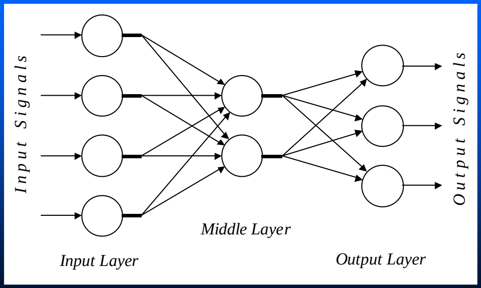
- Human nervous systems accounts for 1-2% of body's weight BUT consumes 25% of body's energy
- Switching speed ~ 1kHZ (1 million times slower than a computer)
- Conduction speed ~ 100 m/s (vs. near light speed in a computer)
- Switching energy ~ 10 $^{-16}$ Joules/op (vs. 10$^{-5}$ joules/op for today's computers)

## Natural NN: characterisitics

- Human estimated to have 1010 -1011 neurons.
- One neuron may connect to 102 –103 others.
- Therefore 1012 -1014 connections are present.

- An artificial neural network consists of a number of very simple processors, also called **neurons**, which are analogous to the biological neurons in the brain.
- The neurons are connected by weighted links passing signals from one neuron to another.
- The output signal is transmitted through the neuron’s outgoing connection. The outgoing connection splits into a number of branches that transmit the same signal. The outgoing branches terminate at the incoming connections of other neurons in the network.

## Architecture of a typical artificial neural network



## Analogy between biological and artificial neural networks

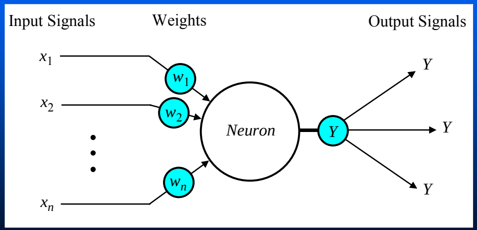
| *Biological Neural Network* | *Artificial Neural Network* |
|---|---|
| Soma | Neuron |
| Dendrite | Input |
| Axon | Output |
| Synapse | Weight |

## Fundamental problems for a given neural model

- How to represent information?
- How to characterize the computational capability of the model?
- How to achieve learning in the model?

## The neuron as a simple computing element

### Diagram of a neuron

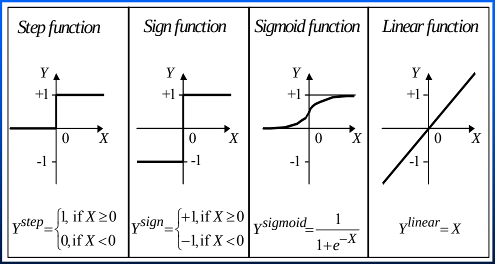


- The neuron computes the weighted sum of the input signals and compares the result with a **threshold value**, θ. If the net input is less than the threshold, the neuron output is –1. But if the net input is greater than or equal to the threshold, the neuron becomes activated and its output attains a value +1.
- The neuron uses the following transfer or **activation function**:

$$X = \sum_{i=1}^{n} x_i w_i \qquad Y = \begin{cases} +1, & \text{if } X \ge \theta \\ -1, & \text{if } X < \theta \end{cases}$$

- This type of activation function is called a **sign function**.

## Activation functions of a neuron

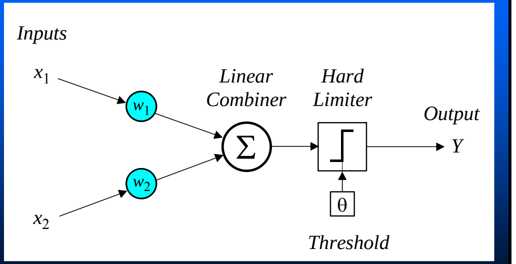


$$Y^{step} = \begin{cases} 1, & \text{if } X \ge 0 \\ 0, & \text{if } X < 0 \end{cases} \qquad Y^{sign} = \begin{cases} +1, & \text{if } X \ge 0 \\ -1, & \text{if } X < 0 \end{cases} \qquad Y^{sigmoid} = \frac{1}{1+e^{-X}} \qquad Y^{linear} = X$$

## Can a single neuron learn a task?

- In 1958, **Frank Rosenblatt** introduced a training algorithm that provided the first procedure for training a simple ANN: a **perceptron**.
- The perceptron is the simplest form of a neural network. It consists of a single neuron with *adjustable* synaptic weights and a *hard limiter*.

## Single-layer two-input perceptron



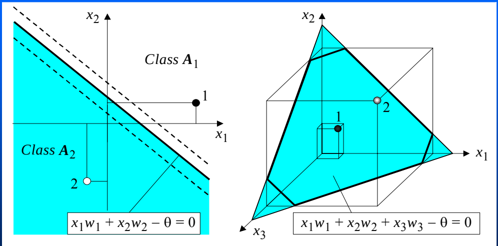
## The Perceptron

- The operation of Rosenblatt's perceptron is based on the **McCulloch and Pitts neuron model**. The model consists of a linear combiner followed by a hard limiter.
- The weighted sum of the inputs is applied to the hard limiter, which produces an output equal to +1 if its input is positive and –1 if it is negative.

- The aim of the perceptron is to classify inputs, $x_1, x_2, \ldots, x_n$, into one of two classes, say $A_1$ and $A_2$.
- In the case of an elementary perceptron, the n-dimensional space is divided by a *hyperplane* into two decision regions. The hyperplane is defined by the ***linearly separable* function**:

$$\sum_{i=1}^{n} x_i w_i - \theta = 0$$

## Linear separability in the perceptrons


$x_2$
Class $A_1$
1
$x_1$
Class $A_2$
2
$x_1w_1 + x_2w_2 - \theta = 0$
$x_2$
1
2
$x_1$
$x_3$
$x_1w_1 + x_2w_2 + x_3w_3 - \theta = 0$

(*a*) Two-input perceptron. (*b*) Three-input perceptron.

## Separating Hyperplane

- The equation $w_1 x_1 + w_2 x_2 + .... + w_n x_n = \theta$ defines a hyperplane in n-space
- If such a hyperplane exists classification problem, the problem called **linearly-separable**.

## Perceptron summary

- A perceptron can solve a classification problem if the problem is linearly separable.
- There are problems a single perceptron cannot solve.
- Perhaps the simplest unsolvable one is the **XOR problem**:

yes: {(0, 1), (1, 0)}, no: {(0, 0), (1, 1)}

## How does the perceptron learn its classification tasks?

This is done by making small adjustments in the weights to reduce the difference between the actual and desired outputs of the perceptron. The initial weights are randomly assigned, usually in the range [−0.5, 0.5], and then updated to obtain the output consistent with the training examples.

- If at iteration *p*, the actual output is *Y*(*p*) and the desired output is $Y_d(p)$, then the error is given by:

$$e(p) = Y_d(p) - Y(p)$$ where $p = 1, 2, 3, \dots$

Iteration *p* here refers to the *p*th training example presented to the perceptron.

- If the error, *e*(*p*), is positive, we need to increase perceptron output *Y*(*p*), but if it is negative, we need to decrease *Y*(*p*).

## The perceptron learning rule

$$w_i(p+1) = w_i(p) + \eta \cdot x_i(p) \cdot e(p)$$

where $p = 1, 2, 3, \dots$

$\eta$ is the **learning rate**, a positive constant less than unity.

The perceptron learning rule was first proposed by **Rosenblatt** in 1960. Using this rule we can derive the perceptron training algorithm for classification tasks.

## Learning rate η

- η governs the rate at which the training rule converges toward the correct solution.
- Typically η < 1.
- Too small an η produces slow convergence.
- Too large of an η can cause oscillations in the process.

## Perceptron’s training algorithm

**Step 1: Initialisation**

Set initial weights $w_1, w_2, \ldots, w_n$ and threshold $\theta$ to random numbers in the range [–0.5, 0.5].

If the error, $e(p)$, is positive, we need to increase perceptron output $Y(p)$, but if it is negative, we need to decrease $Y(p)$.

## Perceptron’s training algorithm (continued)

**Step 2: Activation**

Activate the perceptron by applying inputs $x_1(p)$, $x_2(p), \ldots, x_n(p)$ and desired output $Y_d(p)$. Calculate the actual output at iteration $p = 1$

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

where $n$ is the number of the perceptron inputs, and *step* is a step activation function.

## Perceptron’s training algorithm (continued)

**Step 3: Weight training**

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

where $\Delta w_i(p)$ is the weight correction at iteration $p$.

The weight correction is computed by the **delta rule**:

$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

**Step 4: Iteration**

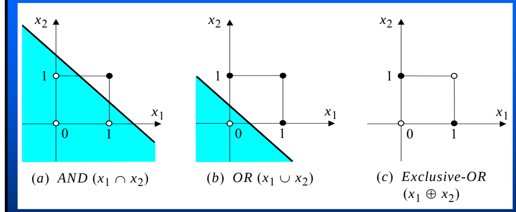
Increase iteration $p$ by one, go back to *Step 2* and repeat the process until convergence.

## Example of perceptron learning: the logical operation *AND*

| Epoch | Inputs | | Desired output | Initial weights | | Actual output | Error | Final weights | |
|---|---|---|---|---|---|---|---|---|---|
| | $x_1$ | $x_2$ | $Y_d$ | $w_1$ | $w_2$ | $Y$ | $e$ | $w_1$ | $w_2$ |
| 1 | 0 | 0 | 0 | 0.3 | –0.1 | 0 | 0 | 0.3 | –0.1 |
| | 0 | 1 | 0 | 0.3 | –0.1 | 0 | 0 | 0.3 | –0.1 |
| | 1 | 0 | 0 | 0.3 | –0.1 | 1 | –1 | 0.2 | –0.1 |
| | 1 | 1 | 1 | 0.2 | –0.1 | 0 | 1 | 0.3 | 0.0 |
| 2 | 0 | 0 | 0 | 0.3 | 0.0 | 0 | 0 | 0.3 | 0.0 |
| | 0 | 1 | 0 | 0.3 | 0.0 | 0 | 0 | 0.3 | 0.0 |
| | 1 | 0 | 0 | 0.3 | 0.0 | 1 | –1 | 0.2 | 0.0 |
| | 1 | 1 | 1 | 0.2 | 0.0 | 1 | 0 | 0.2 | 0.0 |
| 3 | 0 | 0 | 0 | 0.2 | 0.0 | 0 | 0 | 0.2 | 0.0 |
| | 0 | 1 | 0 | 0.2 | 0.0 | 0 | 0 | 0.2 | 0.0 |
| | 1 | 0 | 0 | 0.2 | 0.0 | 1 | –1 | 0.1 | 0.0 |
| | 1 | 1 | 1 | 0.1 | 0.0 | 0 | 1 | 0.2 | 0.1 |
| 4 | 0 | 0 | 0 | 0.2 | 0.1 | 0 | 0 | 0.2 | 0.1 |
| | 0 | 1 | 0 | 0.2 | 0.1 | 0 | 0 | 0.2 | 0.1 |
| | 1 | 0 | 0 | 0.2 | 0.1 | 1 | –1 | 0.1 | 0.1 |
| | 1 | 1 | 1 | 0.1 | 0.1 | 1 | 0 | 0.1 | 0.1 |
| 5 | 0 | 0 | 0 | 0.1 | 0.1 | 0 | 0 | 0.1 | 0.1 |
| | 0 | 1 | 0 | 0.1 | 0.1 | 0 | 0 | 0.1 | 0.1 |
| | 1 | 0 | 0 | 0.1 | 0.1 | 0 | 0 | 0.1 | 0.1 |
| | 1 | 1 | 1 | 0.1 | 0.1 | 1 | 0 | 0.1 | 0.1 |

Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$

## Two-dimensional plots of basic logical operations



A perceptron can learn the operations *AND* and *OR*, but not *Exclusive-OR*.

## Perceptron training algorithm modifications for practical usage

- Put a **limit** on the number of iterations, so that the algorithm will terminate even if the sample set is not linearly separable.
- Include an **error bound** as an extra input. The algorithm can stop as soon as the portion of misclassified samples is less than this bound (as opposed to requiring perfect classification, which would be an error bound of 0).
- Generate the initial weights **randomly**, so that the user does not have to specify them.

## Perceptron training algorithm modifications for practical usage

- Don't require the user to specify the learning rate.
- Instead, vary it so that the chosen sample is always correctly classified after weight modification (although other samples may become incorrectly classified).

We want to change the weight vector so that vector adding $\Delta w = \varepsilon\, \eta$ [ -1, x1, x2 , . . . ., xn ] causes

($\Sigma$ wi xi > 0) to become 0.

Effectively choosing $\eta$ to be just larger than

| w x | / | x x | (where the products are vector inner products) will guarantee this, since

## Some problems about perceptron training

- When a perceptron gives the right answer, no learning takes place.
- Anything below the threshold is interpreted as "no", even if it just below the threshold.
- Might it be better to train the neuron based on how far below the threshold it is?
- This idea is developed in the Adaline training algorithm.

## ADALINE

- The Adaline (Adaptive Linear Neuron or Adaptive Linear Element ) is a model similar to the Perceptron. There are several variations:
- lOne has the threshold function similar to a perceptron.
- l Another uses a **pure linear** function with no threshold.

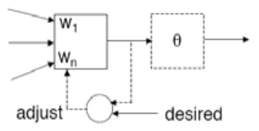
## Adaline inventors

Bernard Widrow,
Professor Emeritus of E.E.,
Stanford University

Marcian Hoff
Co-inventor of Patent 3,821,715
*Microprocessor Concept and Architecture*

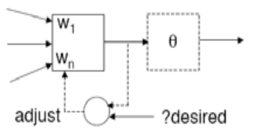
## Adaline Training

- With or without the threshold, the Adaline is **trained** based on the output of the linear function rather than the final output.



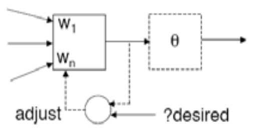
## Adaline Training

- The catch here is that we have to state the **desired** value in terms of the output of the linear part, rather than the output after the threshold.
- What is this for a classifier?



## Adaline Training

- A reasonable approach is to use a *nominal* value such as -0.5 as desired for a “no” classification and a +0.5 for a “yes” classification.



## Adaline Training

- The formula for Adaline weight updating is very similar to the Perceptron:

  Add to the weights $\Delta w$ where

  $$\boxed{\Delta w = \varepsilon\, \eta\, [\,-1, x_1, x_2, \ldots, x_n\,]}$$

  Adaline learning rule

  only now the ***error* is not limited to 1, -1, 0 as before**; it can have a fractional value, since it is based on the output of the linear part of the device.

## Adaline Training

- The weight update formula for the Adaline will be justified eventually.
- One major difference from this vs. the Perceptron is that a learning rate of 1 won’t generally be acceptable. It will need to be smaller, say 0.01. There is a theory that tells us how large we can make it.

## Adaline convergence

- The Adaline admits a more refined stopping criterion:

  The **Mean-Squared Error (MSE)** is the average of the squares of the error taken over all samples. Squaring makes the measure insensitive to the sign of the error. It also provides certain analytic properties.
- This quantity ideally converges toward a specific minimum (which might never be exactly attained). The algorithm can be set to stop when the MSE reaches a

## Alternate rule names

- Because the Adaline rule minimizes MSE, it is sometimes called the “**LMS rule**” [LMS = “least mean square”].
- The term “**Delta rule**” is also sometimes used, although this will be seen to be a rule for a more general class of networks.

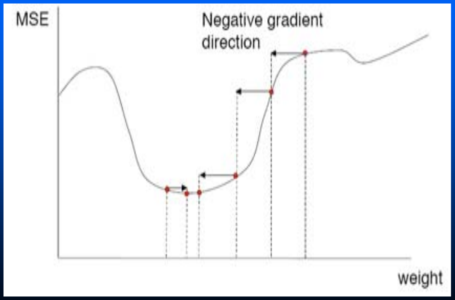
## Minimizing error

- Consider the task of finding the minimum of a function of one variable.
- One standard method for doing this, if the derivative of the function is known, is Newton's method, which entails constructing a tangent line from a current estimate, then using the intersection of the line with the origin for the next.

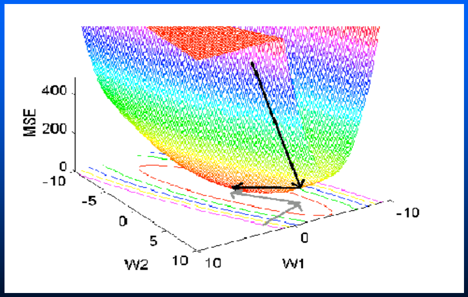
## Gradient descent

- Gradient descent is another method for finding the minimum.
- It consists of computing the gradient of the function, then taking a small step in the direction of negative gradient, which hopefully corresponds to decreased function value, then repeating for the new value of the dependent variable.
- A single dimension for weights (including bias) is atypical.
- For the general case, the gradient is a **vector** of gradient components, one for each weight (including bias).

## Gradient Descent



## Two-dimensioanl descent



## Computing gradients

Note: squared

- MSE = J(w) = $\Sigma(\text{desired-actual})^2/n$ where $\Sigma$ is over n samples.
- desired is a fixed constant for each sample.
- actual = $\Sigma w_j x_j$ ($\Sigma$ over input lines)
- So J(w) = $\Sigma(\text{desired}- \Sigma w_j x_j)^2/n$

## On-line approximation to gradient

- On-line means based on a single sample, rather than batch, which means using all samples
- $J \approx (d- \Sigma w_j x_j)^2 \leftarrow$ (note: no outer sum) (d = desired)
- ith gradient component = $\partial J/\partial w_i$

$$= \partial/\partial w_i (d - \Sigma w_j x_j)^2$$
$$= 2 \underbrace{(d - \Sigma w_j x_j)}_{= \text{error}, \varepsilon} \underbrace{\partial/\partial w_i (d - \Sigma w_j x_j)}_{-x_i} = -2\varepsilon x_i$$

## Computing gradients

- $i$th gradient component = $-2\,\varepsilon\,x_i$
- However we want to move in the *direction* of ***negative*** gradient, tempered by the learning rate $\eta$, so:

  Amount to *add* to weight is

  $\boxed{\Delta w_i = 2\,\varepsilon\,\eta\,x_i}$

  which we recognize as the LMS (Adaline) rule (2 can be folded into $\eta$).

## Vector version

- MSE = $J(w) = E[(d - w^T x)^2]$ (E = expectation or mean averaged over samples)

  $= E[d^2 - 2dw^T x + w^T x\, x^T w]$

  $= E[d^2] - E[2dw^T x] + E[w^T x\, x^T w]$

  $= E[d^2] - 2w^T E[dx] + w^T E[x\, x^T]\, w$

  $= c - 2w^T h + w^T R\, w$, for appropriate const. c, h, R

- J(w) is a quadratic form in w with coefficients derived from the data vectors x
- R is called the auto-correlation matrix

## Analytic gradient

- $\nabla J(w) = \nabla(\, c + w^T b + (1/2) w^T A\, w) = b + A\, w$
- It can be shown that if J has a **minimum**, it will be at a point $w^*$ where $\nabla J(w^*) = 0$,

  i.e. $\underline{b + A\, w^* = 0}$

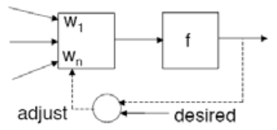
  i.e. $\boxed{w^* = A^{-1}\, b}$

  where $A = 2E[x\, x^T]$, $b = -2E[dx]$

- In general, $w^* = A^{-1} b$ is a stable point
- It may correspond to a minimum, maximum, or saddle

## Generalizing Adaline

- Suppose that we replace the threshold stage with a general function f and revert to expressing desired in terms of its output:

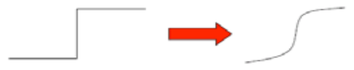


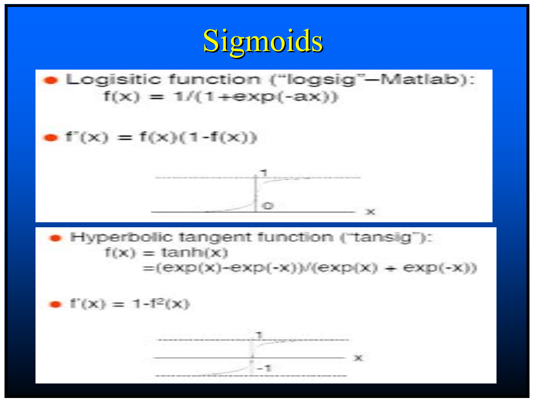
## Generalized LMS rule (or Delta rule)

- $\Delta w = 2\, e\, \eta\, f'(\Sigma w_j x_j)\, x_i$ assuming that f has a derivative f'.
- $\Sigma w_j x_j$ is often called the **"net" value** or **"activation" value**, and f the **activation function**.
- For the special case of f being the identity function, this reduces to the LMS rule we had before.
- Why worry about this generalization?
- It will have a number important uses.

## Use #1

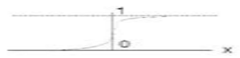
- In the Adaline with threshold, we can't very well treat the model analytically, due to the fact that we have a non-continuous function at the output.
- But we can *approximate* the non-continuous function with a continuous one:





## Sigmoids

- Logisitic function ("logsig"—Matlab):
  f(x) = 1/(1+exp(-ax))
- f'(x) = f(x)(1-f(x))



- Hyperbolic tangent function ("tansig"):
  f(x) = tanh(x)
  =(exp(x)-exp(-x))/(exp(x) + exp(-x))
- f'(x) = 1-f$^2$(x)