Machine Learning

Lecture 5

Artificial Neural Networks.

Multi-layer perceptrons. Error back propagation

Outline

- Perceptrons
- Gradient descent
- Multi-layer networks
- Backpropagation

Biological Neural Systems

- Neuron switching time : > 10^{-3} secs
- Number of neurons in the human brain: ~10^{10}
- Connections (synapses) per neuron : ~10^{4}–10^{5}
- Face recognition : 0.1 secs
- High degree of parallel computation
- Distributed representations
- Associative processing of images
- Flexibility and robustness based on learning

Properties of Artificial Neural Nets (ANNs)

- Many simple neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed processing
- Learning by tuning the connection weights
- Some models provide learning by creation of new neurons

Kinds of NN

- Supervised
- Feedforward
  - Linear
    - Hebbian - Hebb (1949), Fausett (1994)
    - Perceptron - Rosenblatt (1958), Minsky and Papert (1969/1988), Fausett (1994)
    - Adaline - Widrow and Hoff (1960), Fausett (1994)
    - Higher Order - Bishop (1995)
    - Functional Link - Pao (1989)
  - MLP: Multilayer perceptron - Bishop (1995), Reed and Marks (1999), Fausett (1994)
    - Backprop - Rumelhart, Hinton, and Williams (1986)
    - Cascade Correlation - Fahlman and Lebiere (1990), Fausett (1994)
    - Quickprop - Fahlman (1989)
    - RPROP - Riedmiller and Braun (1993)
  - RBF networks - Bishop (1995), Moody and Darken (1989), Orr (1996)
    - OLS: Orthogonal Least Squares - Chen, Cowan and Grant (1991)
  - CMAC: Cerebellar Model Articulation Controller - Albus (1975), Brown and Harris (1994)
  - Classification only
    - LVQ: Learning Vector Quantization - Kohonen (1988), Fausett (1994)
    - PNN: Probabilistic Neural Network - Specht (1990), Masters (1993), Hand (1982), Fausett (1994)
  - Regression only
    - GNN: General Regression Neural Network - Specht (1991), Nadaraya (1964), Watson (1964)

Kinds of NN (2)

- Feedback - Hertz, Krogh, and Palmer (1991), Medsker and Jain (2000)
  - BAM: Bidirectional Associative Memory - Kosko (1992), Fausett (1994)
  - Boltzman Machine - Ackley et al. (1985), Fausett (1994)
  - Recurrent time series
    - Backpropagation through time - Werbos (1990)
    - Elman - Elman (1990)
    - FIR: Finite Impulse Response - Wan (1990)
    - Jordan - Jordan (1986)
    - Real-time recurrent network - Williams and Zipser (1989)
    - Recurrent backpropagation - Pineda (1989), Fausett (1994)
    - TDNN: Time Delay NN - Lang, Waibel and Hinton (1990)

Kinds of NN (3)

- Unsupervised - Hertz, Krogh, and Palmer (1991)
- Competitive
  - Vector Quantization
    - Grossberg - Grossberg (1976)
    - Kohonen - Kohonen (1984)
    - Conscience - Desieno (1988)
  - Self-Organizing Map
    - Kohonen - Kohonen (1995), Fausett (1994)
    - GTM: - Bishop, Svensén and Williams (1997)
    - Local Linear - Mulier and Cherkassky (1995)
  - Adaptive resonance theory
    - ART 1 - Carpenter and Grossberg (1987a), Moore (1988), Fausett (1994)
    - ART 2 - Carpenter and Grossberg (1987b), Fausett (1994)
    - ART 2-A - Carpenter, Grossberg and Rosen (1991a)
    - ART 3 - Carpenter and Grossberg (1990)
    - Fuzzy ART - Carpenter, Grossberg and Rosen (1991b)
  - DCL: Differential Competitive Learning - Kosko (1992)
- Dimension Reduction - Diamantaras and Kung (1996)
  - Hebbian - Hebb (1949), Fausett (1994)
  - Oja - Oja (1989)
  - Sanger - Sanger (1989)
  - Differential Hebbian - Kosko (1992)
- Autoassociation
  - Linear autoassociator - Anderson et al. (1977), Fausett (1994)
  - BSB: Brain State in a Box - Anderson et al. (1977), Fausett (1994)
  - Hopfield - Hopfield (1982), Fausett (1994)
- Nonlearning
- Hopfield - Hertz, Krogh, and Palmer (1991)
- various networks for optimization - Cichocki and Unbehauen (1993)

Appropriate Problem Domains for Neural Network Learning

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Form of target function is unknown
- Humans do not need to interpret the results (black box model)

ALVINN

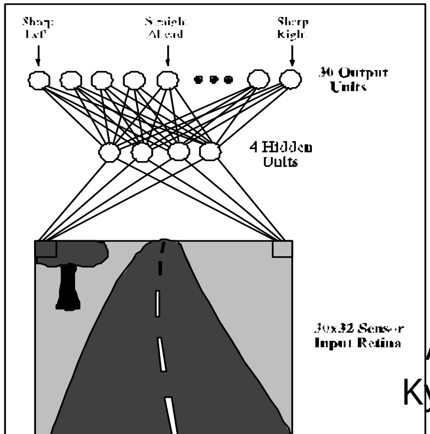
Drives 70 mph on a public highway

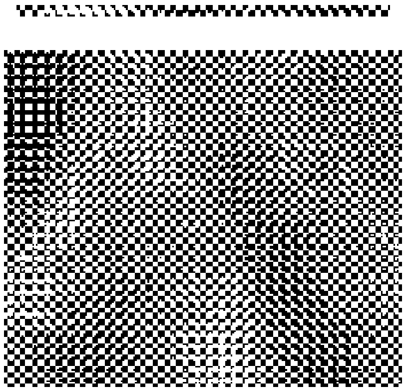
Camera image

30 outputs for steering

4 hidden units

30x32 pixels as inputs

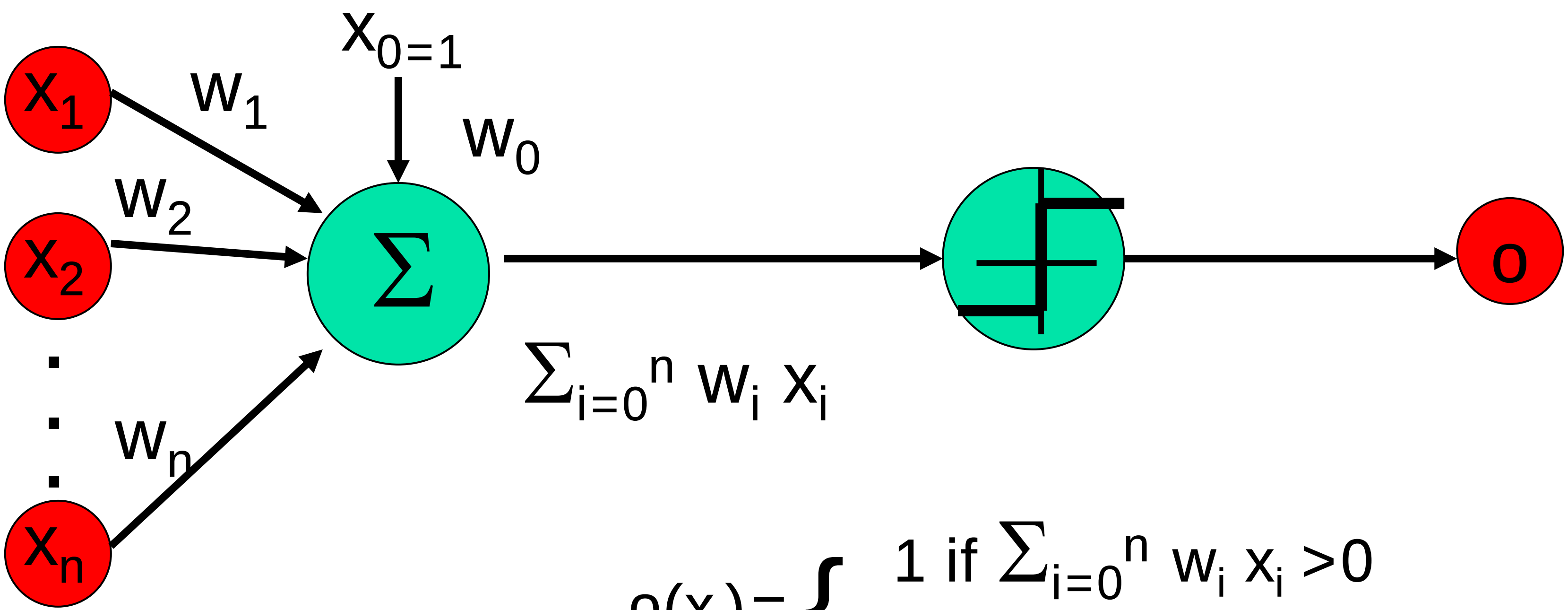




30x32 weights into one out of four hidden unit

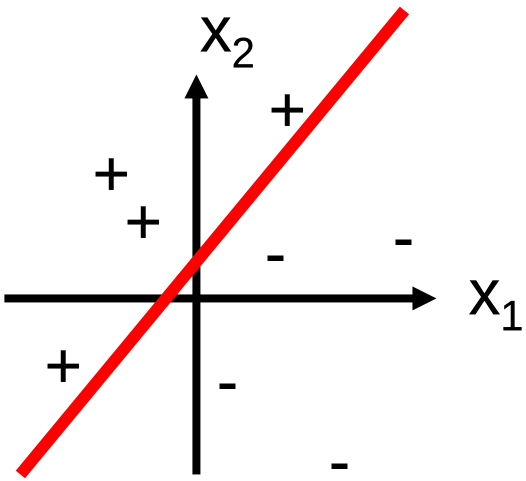
Perceptron

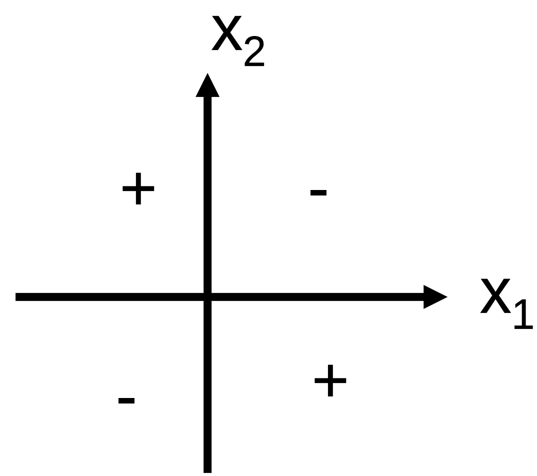
- Linear treshold unit (LTU)



$$o(x_i) = \begin{cases} 1 \text{ if } \Sigma_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$$

Decision Surface of a Perceptron





- Perceptron is able to represent some useful functions
- And(x_1,x_2) choose weights w_0=-1.5, w_1=1, w_2=1
- But functions that are not linearly separable (e.g. Xor) are not representable

Perceptron Learning Rule

$w_i = w_i + \Delta w_i$

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

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

o is the perceptron output

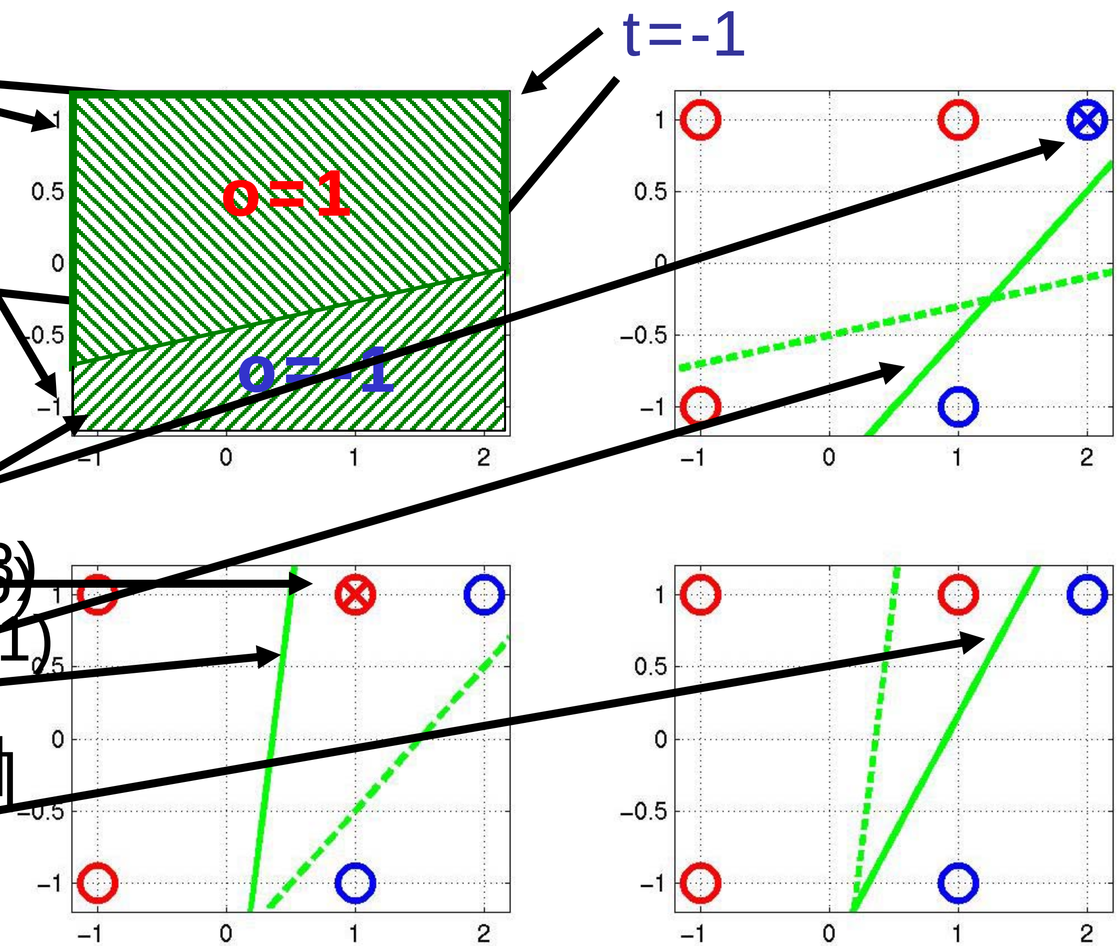
η Is a small constant (e.g. 0.1) called *learning rate*

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

Perceptron Learning Rule

w=[0.25 –0.1 0.5]

$x_2 = 0.2\, x_1 - 0.5$



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

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

o=sgn(0.45-0.6+0.3)

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

o=sgn(0.25+0.1-0.5)

=1

o=sgn(0.25-0.7+0.1)

=-1

=-1

Δw=[0.2 –0.2 –0.2]

Δw=[-0.2 –0.4 –0.2]

Δw=[0.2 0.2 0.2]

Gradient Descent Learning Rule

- Consider linear unit without threshold and continuous output o (not just –1,1)
  - $o=w_0 + w_1 x_1 + \ldots + w_n x_n$
- Train the w_i's such that they minimize the squared error
  - $E[w_1,\ldots,w_n] = \frac{1}{2} \sum_{d\in D} (t_d-o_d)^2$

  where D is the set of training examples

Gradient Descent

$D=\{\langle(1,1),1\rangle,\langle(-1,-1),1\rangle,$
$\langle(1,-1),-1\rangle,\langle(-1,1),-1\rangle\}$

Gradient:

$\nabla E[w]=[\partial E/\partial w_0,\ldots\ \partial E/\partial w_n]$

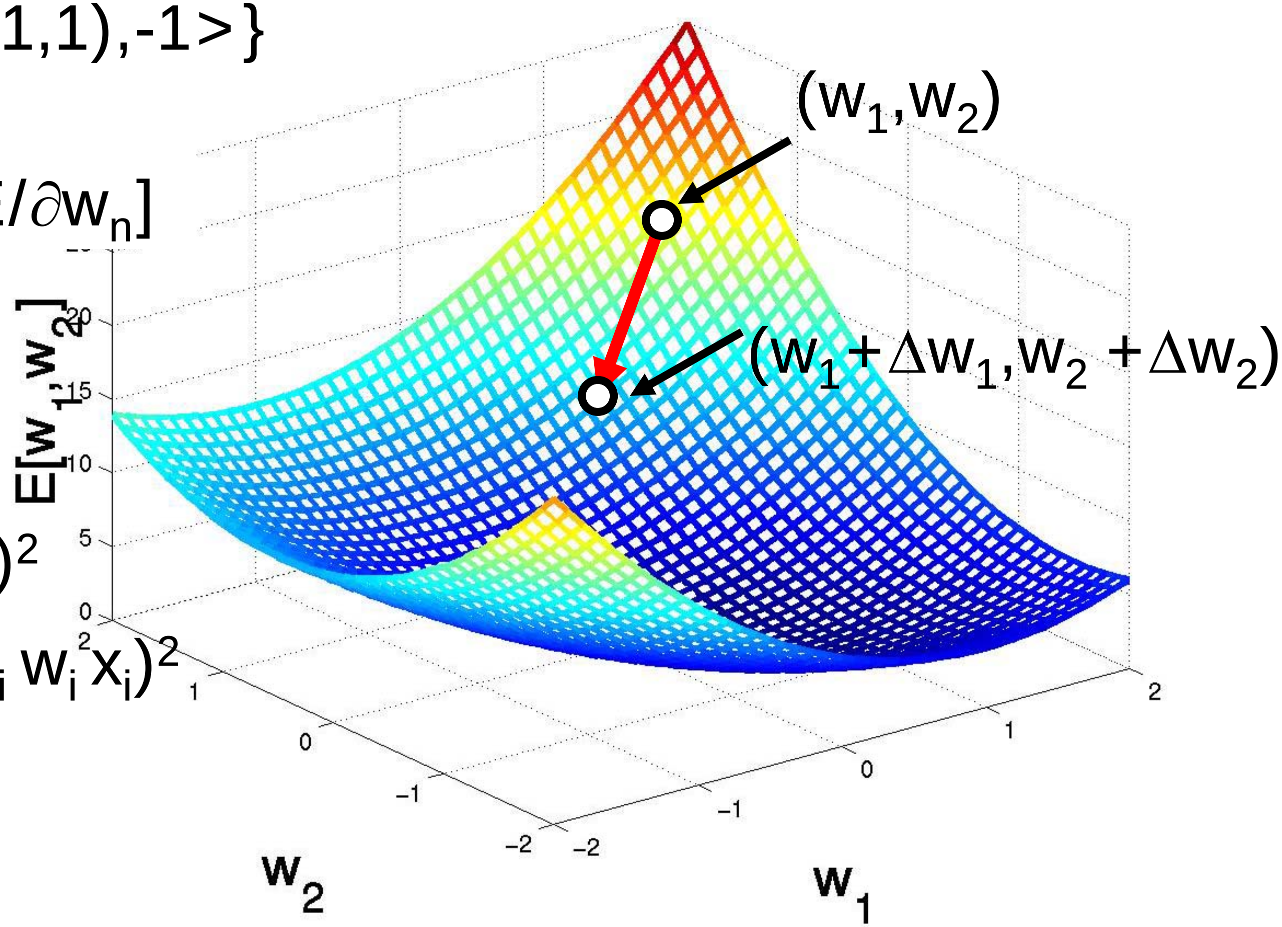
$\Delta w=-\eta\ \nabla E[w]$

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

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

$=\ \partial/\partial w_i\ 1/2\Sigma_d(t_d-\Sigma_i\ w_i x_i)^2$

$=\ \Sigma_d(t_d-\ o_d)(-x_i)$



Gradient Descent

Gradient-Descent(*training_examples*, η)

Each training example is a pair of the form <(x_1,...x_n),t> where (x_1,...,x_n) is the vector of input values, and t is the target output value, η is the learning rate (e.g. 0.1)

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero
  - For each <(x_1,...x_n),t> in *training_examples* Do
    - Input the instance (x_1,...,x_n) to the linear unit and compute the output o
    - For each linear unit weight w_i Do
      - $\Delta w_i = \Delta w_i + \eta\ (t-o)\ x_i$
  - For each linear unit weight wi Do
    - $w_i = w_i + \Delta w_i$

Incremental Stochastic Gradient Descent

- Batch mode : gradient descent

  $w=w - \eta \nabla E_D[w]$ over the entire data D

  $E_D[w]=1/2\Sigma_d(t_d-o_d)^2$

- Incremental mode: gradient descent

  $w=w - \eta \nabla E_d[w]$ over individual training examples d

  $E_d[w]=1/2 (t_d-o_d)^2$

Incremental Gradient Descent can approximate Batch Gradient Descent arbitrarily closely if η is small enough

Comparison Perceptron and Gradient Descent Rule

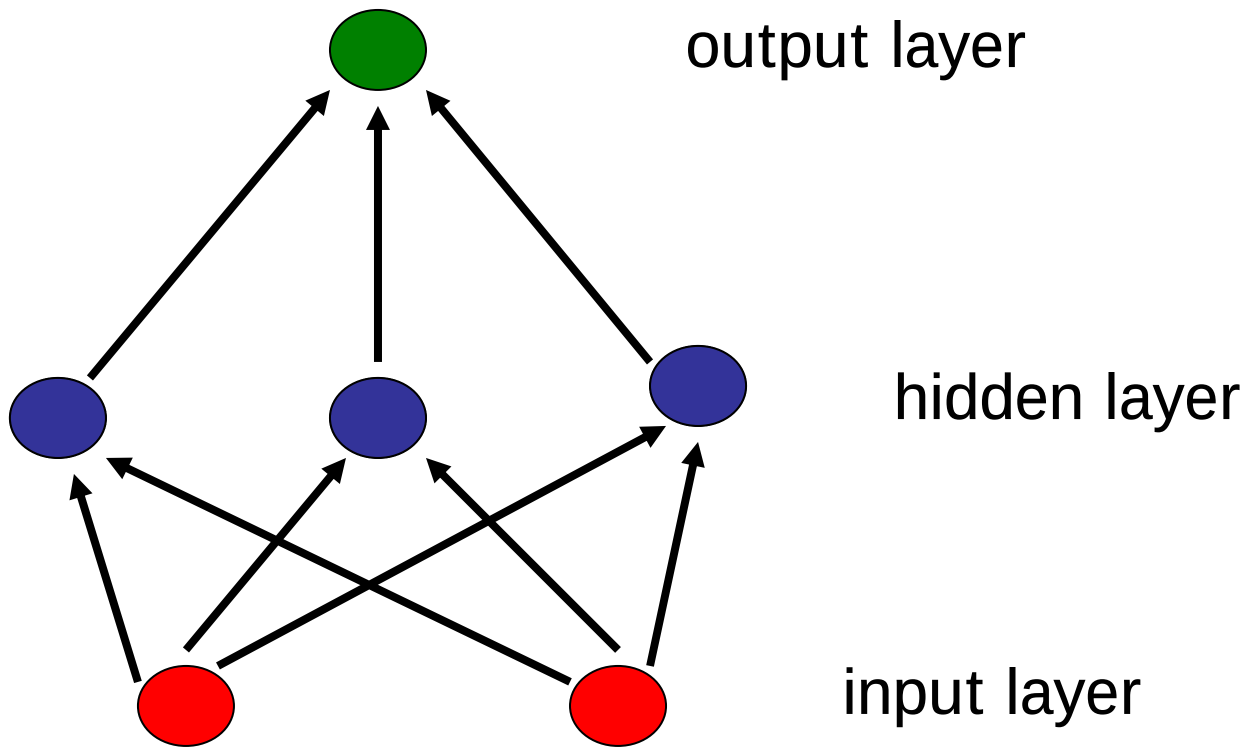
Perceptron learning rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate η

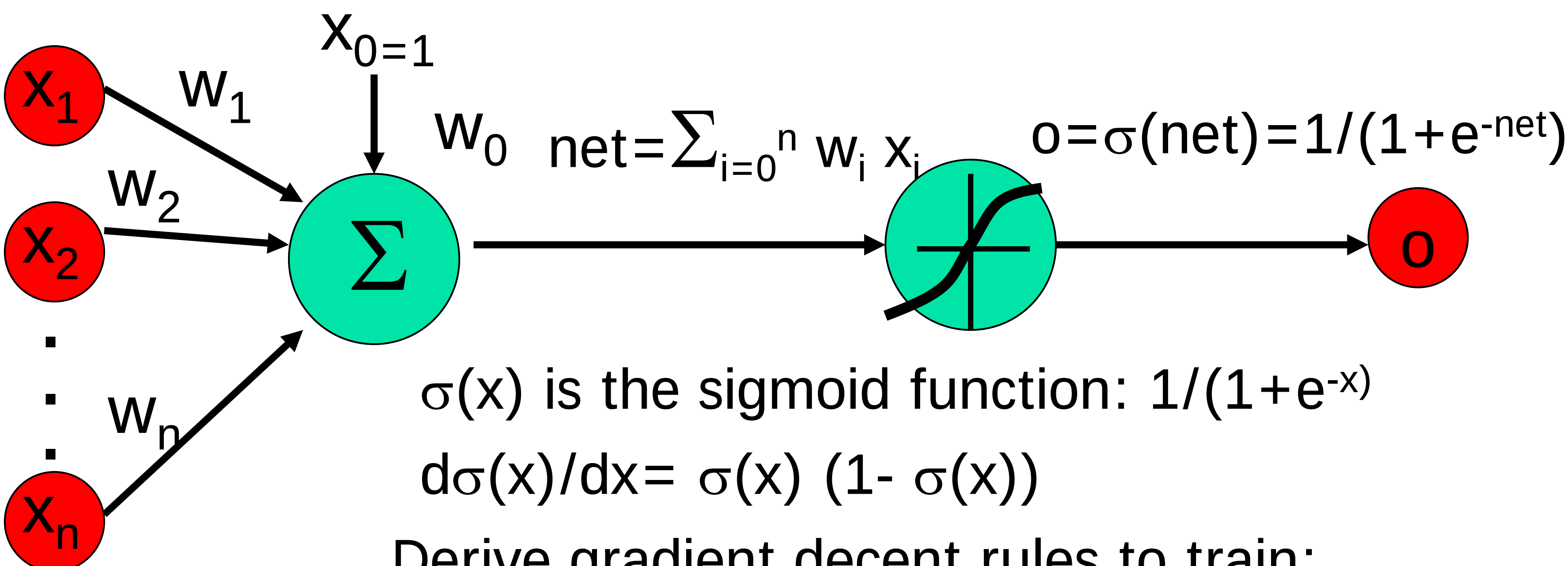
Linear unit training rules uses gradient descent

- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not separable by H

Multi-Layer Networks



Sigmoid Unit



$\sigma(x)$ is the sigmoid function: $1/(1+e^{-x})$

$d\sigma(x)/dx= \sigma(x)\ (1- \sigma(x))$

Derive gradient decent rules to train:

- one sigmoid function

  $\partial E/\partial w_i = -\sum_d (t_d-o_d)\ o_d\ (1-o_d)\ x_i$

- Multilayer networks of sigmoid units backpropagation:

Kinds of sigmoid used in perceptrons

Exponential

$$f(s) = \frac{1}{1 + e^{-2\alpha s}}$$

Rational

$$f(s) = \frac{s}{|s| + \alpha}$$

Hyperbolic tangent

$$f(s) = th\frac{s}{\alpha} = \frac{e^{\frac{s}{\alpha}} - e^{-\frac{s}{\alpha}}}{e^{\frac{s}{\alpha}} + e^{-\frac{s}{\alpha}}}$$

Backpropagation Algorithm

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - For each training example <(x_1,...x_n),t> Do
    - Input the instance (x_1,...,x_n) to the network and compute the network outputs o_k
    - For each output unit k
      - $\delta_k = o_k(1-o_k)(t_k-o_k)$
    - For each hidden unit h
      - $\delta_h = o_h(1-o_h) \sum_k w_{h,k} \delta_k$
    - For each network weight $w_{,j}$ Do
    - $w_{i,j} = w_{i,j} + \Delta w_{i,j}$ where

      $\Delta w_{i,j} = \eta \delta_j x_{i,j}$

Backpropagation

- Gradient descent over entire *network* weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum

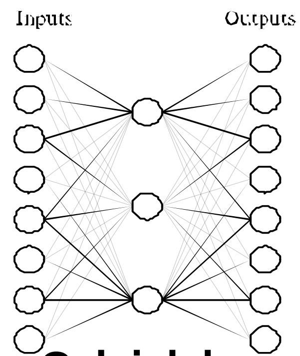
  -in practice often works well (can be invoked multiple times with different initial weights)
- Often include weight *momentum* term

$$\Delta w_{i,j}(n) = \eta\, \delta_j\, x_{i,j} + \alpha\, \Delta w_{i,j}(n-1)$$

- Minimizes error training examples
  - Will it generalize well to unseen instances (over-fitting)?
- Training can be slow typical 1000-10000 iterations

  (use Levenberg-Marquardt instead of gradient descent)
- Using network after training is fast

8-3-8 Binary Encoder -Decoder



8 inputs   3 hidden   8 outputs

A target function:

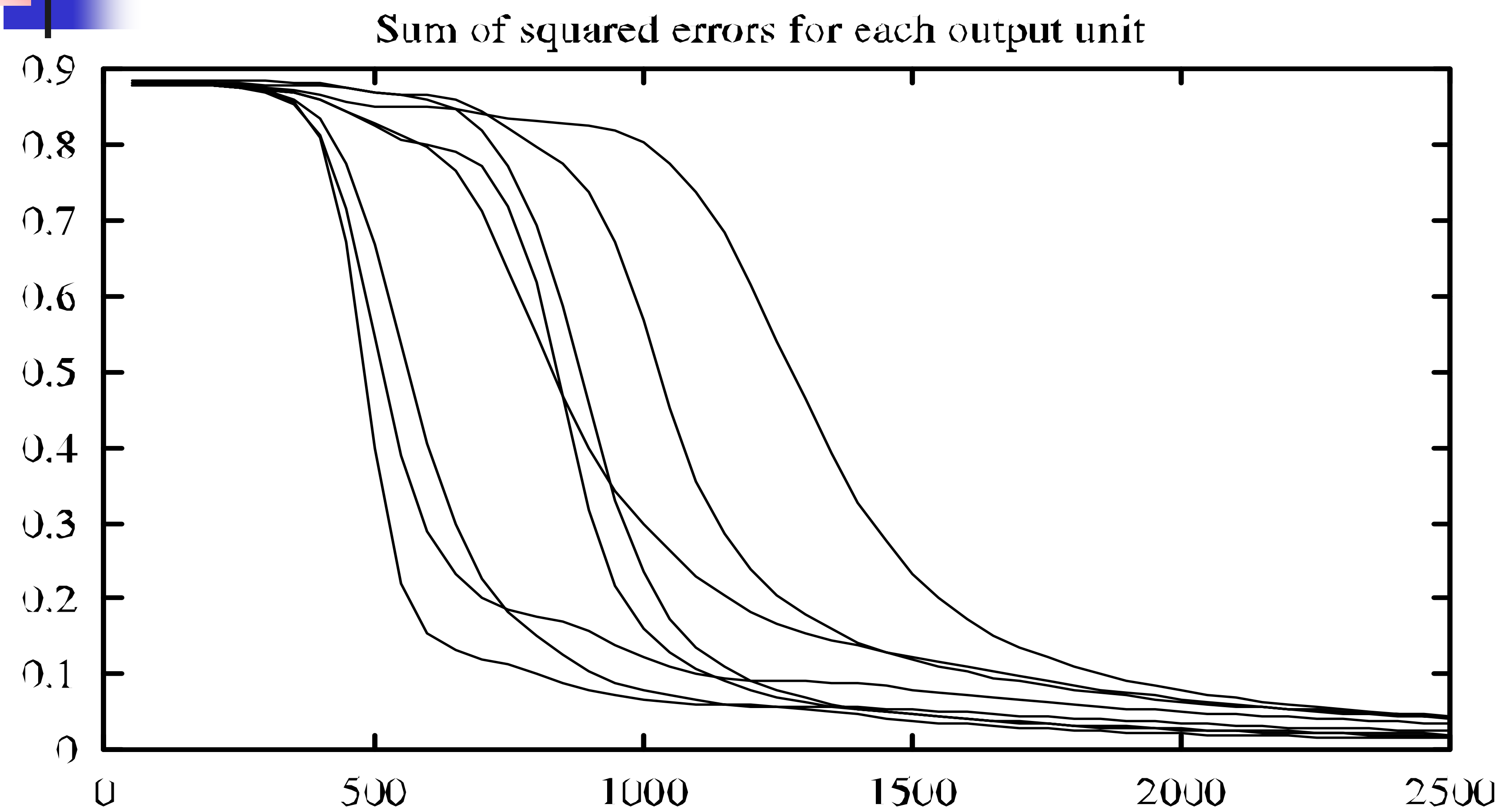
Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

Hidden values

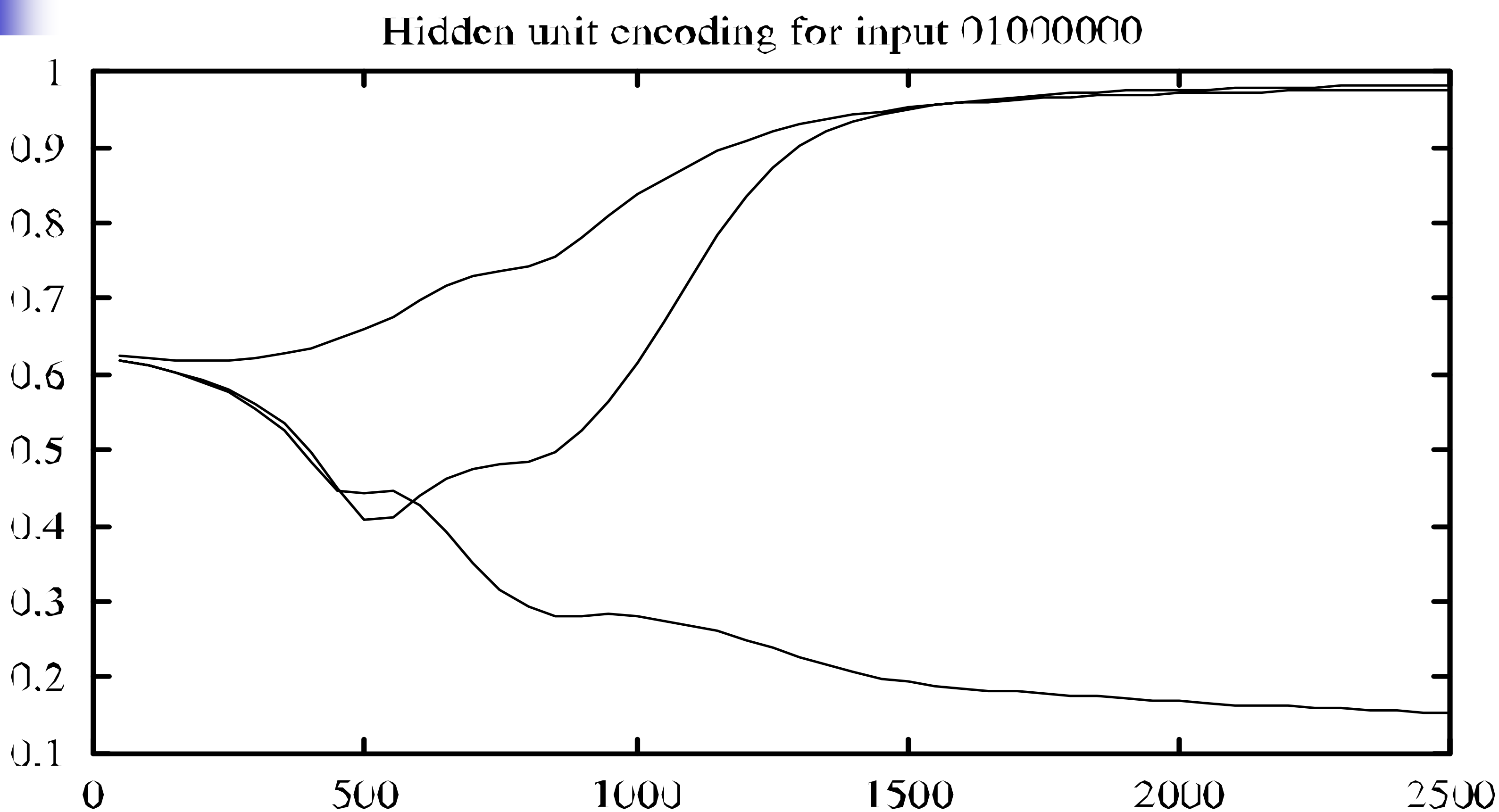
.89 .04 .08
.01 .11 .88
.01 .97 .27
.99 .97 .71
.03 .05 .02
.22 .99 .99
.80 .01 .98
.60 .94 .01

Can this be learned??

Sum of Squared Errors for the Output Units



Hidden Unit Encoding for Input 0100000



Convergence of Backprop

Gradient descent to some local minimum

- Perhaps not global minimum
  - Add momentum
  - Stochastic gradient descent
  - Train multiple nets with different initial weights

Nature of convergence

- Initialize weights near zero
- Therefore, initial networks are near-linear
- Increasingly non-linear functions possible as training progresses

Expressive Capabilities of ANN

Boolean functions

- Every boolean function can be represented by network with single hidden layer
- But might require exponential (in number of inputs) hidden units

Continuous functions

- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers

Two tasks solved by MLP

- Classification (recognition)
  - Usually binary outputs
- Regression (approximation)
  - Analog outputs

Advantages and disadvantages of MLP with back propagation

- Advantages:
  - Guarantee of possibility of solving of tasks
- Disadvantages:
  - Low speed of learning
  - Possibility of overfitting
  - Impossible to relearning
  - It is needed to select of structure for solving of concrete task (usually it is problem)

Increase of speed of learning

- Preprocessing of features before getting to inputs of percepton
- Dynamical step of learning (in begin one is large, than one is decreasing)
- Using of second derivative in formulas for modification of weights
- Using hardware implementation

Fight against of overfitting

- Don’t select too small error for learning or too large number of iteration

Choice of structure

- Using of constructive learning algorithms
  - Deleting of nodes (neurons) and links corresponding to one (prunning networks)
  - Appending new neurons if it is needed (growth networks)
- Using of genetic algorithms for selection of suboptimal structure

Impossible to relearning

- Using of constructive learning algorithms
  - Deleting of nodes (neurons) and links corresponding to one
  - Appending new neurons if it is needed
  - This is incremental learning