Lecture 14
Artificial Neural Networks

Marco Chiarandini

Deptartment of Mathematics & Computer Science
University of Southern Denmark

Slides by Stuart Russell and Peter Norvig

Course Overview

- ✔ Introduction
  - ✔ Artificial Intelligence
  - ✔ Intelligent Agents
- ✔ Search
  - ✔ Uninformed Search
  - ✔ Heuristic Search
- ✔ Adversarial Search
  - ✔ Minimax search
  - ✔ Alpha-beta pruning
- ✔ Knowledge representation and Reasoning
  - ✔ Propositional logic
  - ✔ First order logic
  - ✔ Inference
- ✔ Uncertain knowledge and Reasoning
  - ✔ Probability and Bayesian approach
  - ✔ Bayesian Networks
  - ✔ Hidden Markov Chains
  - ✔ Kalman Filters
- ✔ Learning
  - ✔ Decision Trees
  - ✔ Maximum Likelihood
  - ✔ EM Algorithm
  - ✔ Learning Bayesian Networks
  - k Nearest Neighbor
  - Neural Networks
  - ✘ Support vector machines

Outline

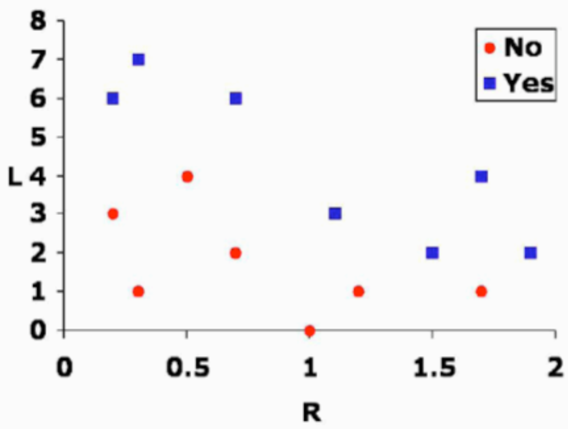
1. K Nearest Neighbor
2. Neural Networks

Non-parametric learning

- When little data available ⇝parametric learning (restricted from the model selected)
- When massive data we can let hypothesis grow from data ⇝non parametric learning
  instance based: construct from training instances

Predicting Bankruptcy

L	R	B
3	0.2	No
1	0.3	No
4	0.5	No
2	0.7	No
0	1.0	No
1	1.2	No
1	1.7	No
6	0.2	Yes
7	0.3	Yes
6	0.7	Yes
3	1.1	Yes
2	1.5	Yes
4	1.7	Yes
2	1.9	Yes

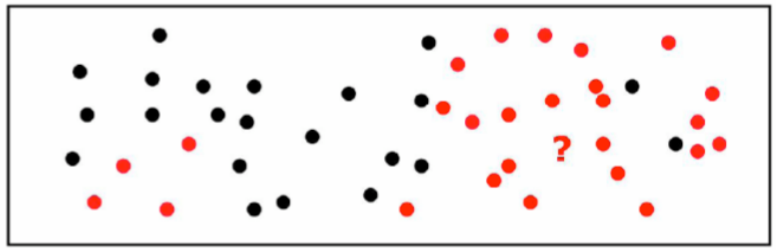


L: #late payments / year
R: expenses / income

Nearest Neighbor

Basic idea:

- Remember all your data
- When someone asks a question
  - find nearest old data point
  - return answer associated with it

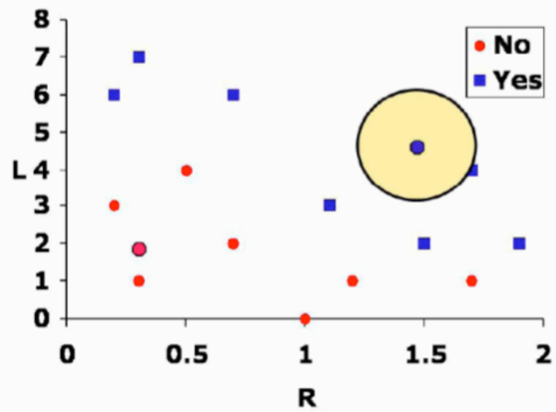


- Find k observations closest to x and average the response

$$\hat{Y} = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

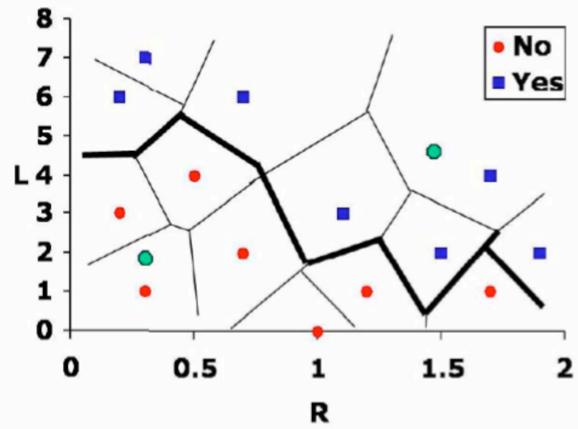
- For qualitative use majority rule
- Needed a distance measure:
  - Euclidean
  - Standardization $x' = \frac{x - \bar{x}}{\sigma_x}$ (Mahalanobis, scale invariant)
  - Hamming

Predicting Bankruptcy



$$D(x^i, x^k) = \sqrt{\sum_j (L^i - L^k)^2 + (5R^i - 5R^k)^2}$$

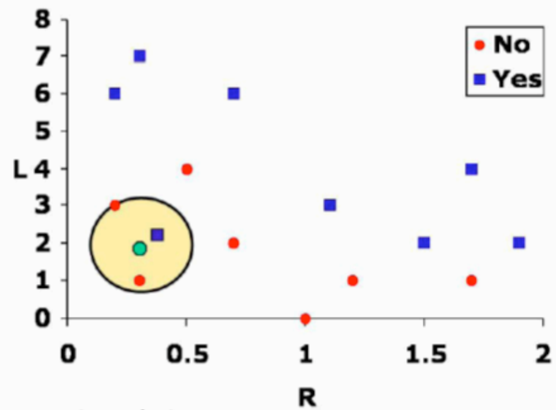
Predicting Bankruptcy



$$D(x^i, x^k) = \sqrt{\sum (L^i - L^k)^2 + (5R^i - 5R^k)^2}$$

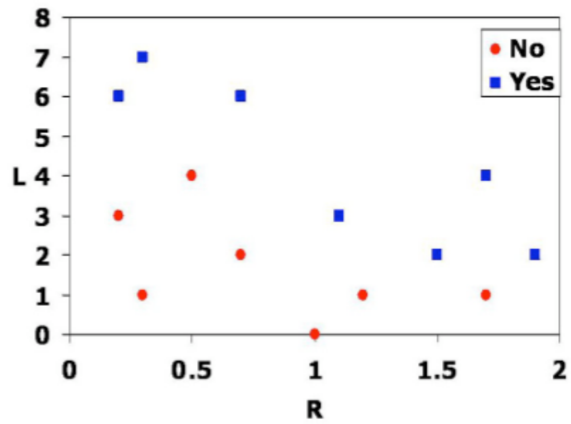
- Learning is fast
- Lookup takes about n computations
  with k-d trees can be faster
- Memory can fill up with all that data
- Problem: Course of dimensionality $b^d = \frac{k}{N}1 \implies b = \frac{k}{N}^{\frac{1}{d}}$

k-Nearest Neighbor

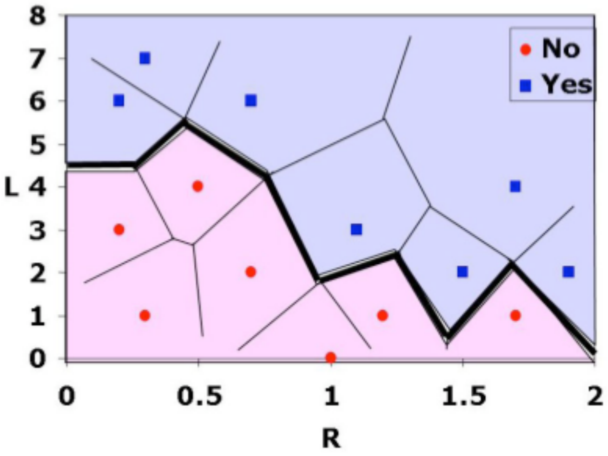


- Find the k nearest points
- Predict output according to the majority
- Choose k using cross-validation

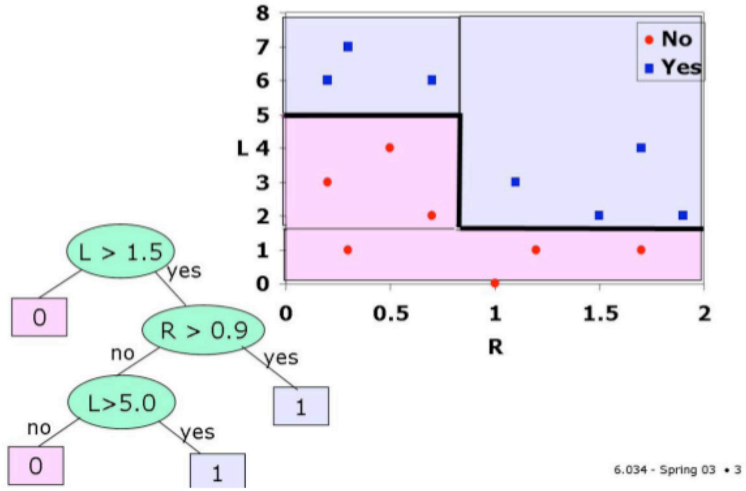
Backruptcy Example



1-Nearest Neighbor



Decision Trees



Outline

1. K Nearest Neighbor

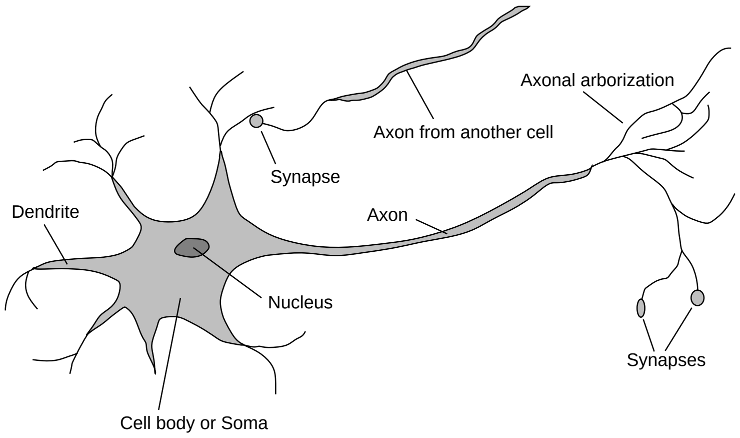
2. Neural Networks

Outline

- Brains
- Neural networks
- Perceptrons
- Multilayer perceptrons
- Applications of neural networks

Brains

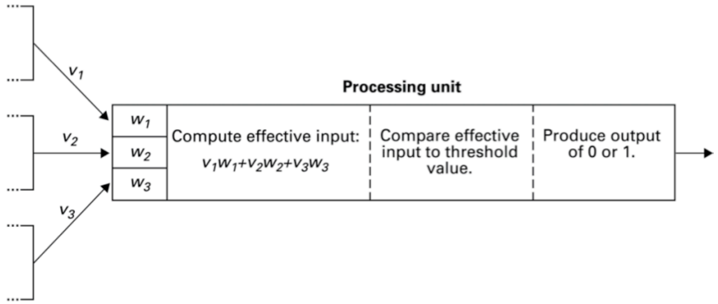
10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time
Signals are noisy "spike trains" of electrical potential

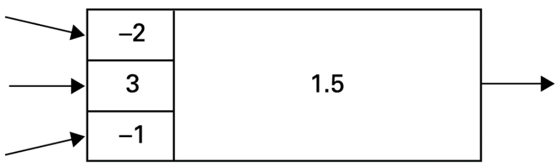


Artificial Neural Networks

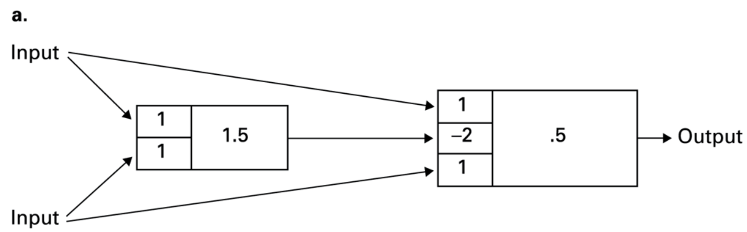
- Artificial Neuron
  - Each input is multiplied by a weighting factor.
  - Output is 1 if sum of weighted inputs exceeds the threshold value; 0 otherwise.
- Network is programmed by adjusting weights using feedback from examples.

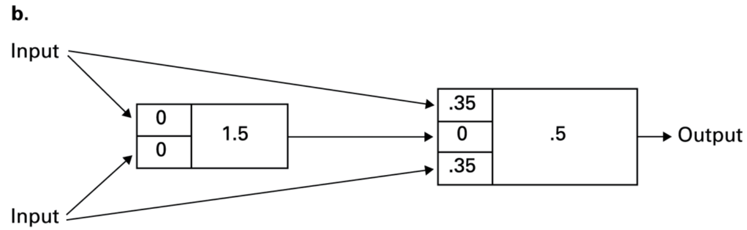
Activities within a processing unit





Neural Network with two layers

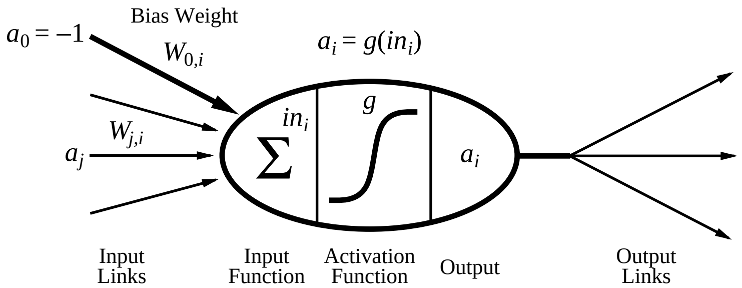




McCulloch–Pitts "unit" (1943)

Output is a function of weighted inputs:

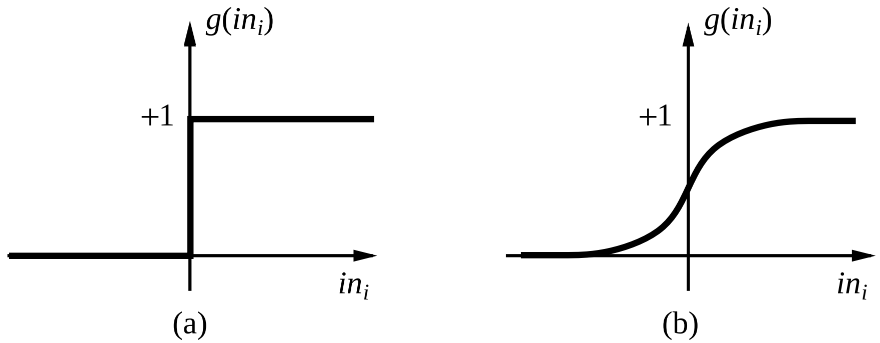
$$a_i = g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$



A gross oversimplification of real neurons, but its purpose is to develop understanding of what networks of simple units can do

Activation functions

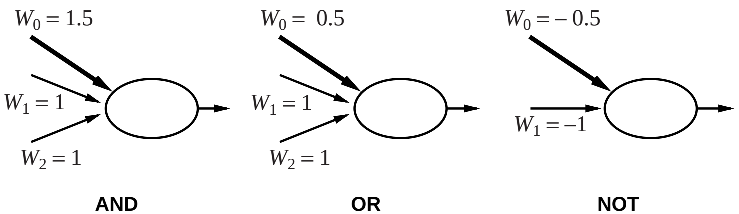
Non linear activation functions



(a) is a step function or threshold function
(b) is a sigmoid function $1/(1+e^{-x})$

Changing the bias weight $W_{0,i}$ moves the threshold location

Implementing logical functions



McCulloch and Pitts: every Boolean function can be implemented

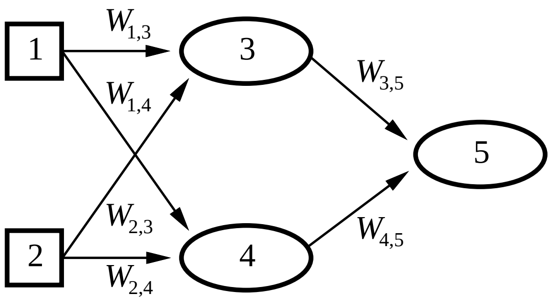
Network structures

- Feed-forward networks:
  - single-layer perceptrons
  - multi-layer perceptrons

  Feed-forward networks implement functions, have no internal state (acyclic)

- Recurrent networks:
  - Hopfield networks have symmetric weights ($W_{i,j} = W_{j,i}$)
    $g(x) = \text{sign}(x)$, $a_i = \{1, 0\}$; **associative memory**
  - recurrent neural nets have directed cycles with delays
    $\Longrightarrow$ have internal state (like flip-flops), can oscillate etc.

Feed-forward example



Feed-forward network = a parameterized family of nonlinear functions:

$$\begin{aligned} a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\ &= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)) \end{aligned}$$

Adjusting weights changes the function: do learning this way!

Use

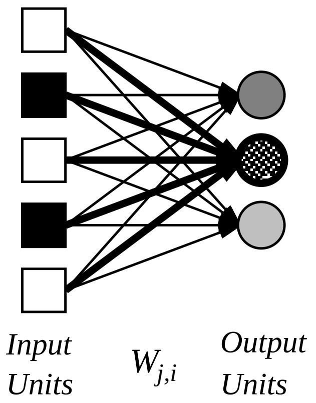
Neural Networks are used in classification and regression

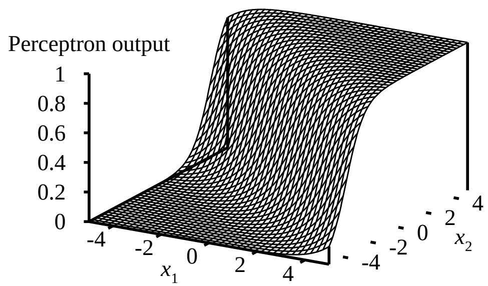
- Boolean classification:
  - value over 0.5 one class
  - value below 0.5 other class
- k-way classification
  - divide single output into k portions
  - k separate output unit

Layer arrangement: units receive inputs from preceding layer)

- single-layer networks (no hidden layer)
- multilayer networks (one or more hidden layers)

Single-layer NN (perceptrons)





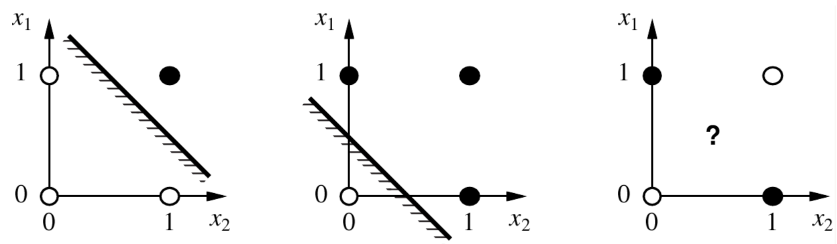
Output units all operate separately—no shared weights
Adjusting weights moves the location, orientation, and steepness of cliff

Expressiveness of perceptrons

Consider a perceptron with g = step function (Rosenblatt, 1957, 1960)
The output is 1 when:

$$\sum_j W_j x_j > 0 \quad \text{or} \quad \mathbf{W} \cdot \mathbf{x} > 0$$

Hence, it represents a linear separator in input space:
- hyperplane in multidimensional space
- line in 2 dimensions



Minsky & Papert (1969) pricked the neural network balloon

Perceptron learning

Learn by adjusting weights to reduce error on training set
The squared error for an example with input $\mathbf{x}$ and true output y is

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2 \ ,$$

Perform optimization search by gradient descent:

$$\frac{\partial E}{\partial W_j} = Err \cdot \frac{\partial Err}{\partial W_j} = Err \cdot \frac{\partial}{\partial W_j}\left(y - g(\sum_{j=0}^{n} W_j x_j)\right)$$
$$= -Err \cdot g'(in) \cdot x_j$$

Simple weight update rule (perceptron learning rule):

$$W^(t+1)_j = W_j^t + \alpha \cdot Err \cdot g'(in) \cdot x_j$$

For threshold perceptron, $g'(in)$ is undefined.
Original perceptron learning rule (Rosenblatt, 1957) simply omits $g'(in)$

Perceptron learning contd.

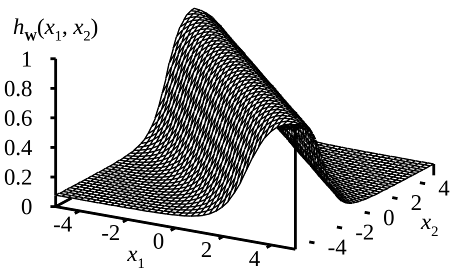
function Perceptron-Learning(*examples*,*network*) **returns** *perceptron weights*
inputs: *examples*, a set of examples, each with input $\mathbf{x} = x_1, x_2, \ldots, x_n$ and output y
inputs: *network*, a perceptron with weights W_j, $j = 0, \ldots, n$ and activation function g

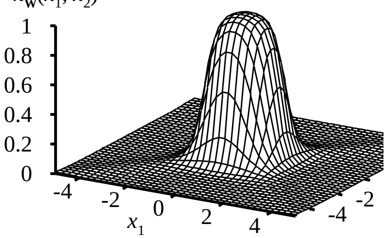
repeat
  for each e **in** *examples* **do**
    $in \leftarrow \sum_{j=0}^{n} W_j x_j[e]$
    $Err \leftarrow y[e] - g(in)$
    $W_j \leftarrow W_j + \alpha \cdot Err \cdot g'(in) \cdot x_j[e]$
  end
until all examples correctly predicted or stopping criterion is reached
return *network*

Perceptron learning rule converges to a consistent function
for any linearly separable data set

Expressiveness of MLPs

All continuous functions with 2 layers, all functions with 3 layers

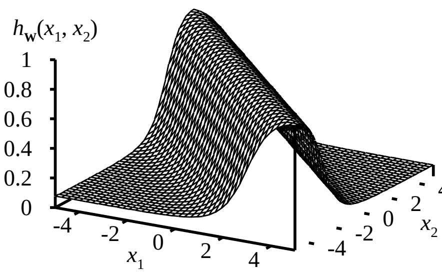


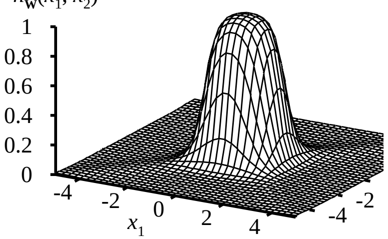


Combine two opposite-facing threshold functions to make a ridge
Combine two perpendicular ridges to make a bump
Add bumps of various sizes and locations to fit any surface
Proof requires exponentially many hidden units (cf DTL proof)

Expressiveness of MLPs

All continuous functions w/ 2 layers, all functions w/ 3 layers





Combine two opposite-facing threshold functions to make a ridge
Combine two perpendicular ridges to make a bump
Add bumps of various sizes and locations to fit any surface
Proof requires exponentially many hidden units (cf DTL proof)

Back-propagation learning

Output layer: same as for single-layer perceptron,

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

where $\Delta_i = Err_i \times g'(in_i)$

Hidden layer: **back-propagate** the error from the output layer:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \ .$$

Update rule for weights in hidden layer:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \ .$$

(Most neuroscientists deny that back-propagation occurs in the brain)

Back-propagation derivation

The squared error on a single example is defined as

$$E = \frac{1}{2} \sum_i (y_i - a_i)^2 \ ,$$

where the sum is over the nodes in the output layer.

$$\begin{aligned}
\frac{\partial E}{\partial W_{j,i}} &= -(y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial g(in_i)}{\partial W_{j,i}} \\
&= -(y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i)g'(in_i)\frac{\partial}{\partial W_{j,i}}\left(\sum_j W_{j,i}a_j\right) \\
&= -(y_i - a_i)g'(in_i)a_j = -a_j\Delta_i
\end{aligned}$$

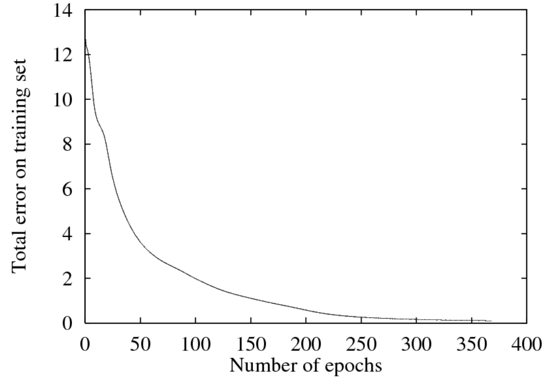
Back-propagation derivation contd.

$$\begin{aligned}
\frac{\partial E}{\partial W_{k,j}} &= -\sum_i (y_i - a_i)\frac{\partial a_i}{\partial W_{k,j}} = -\sum_i (y_i - a_i)\frac{\partial g(in_i)}{\partial W_{k,j}} \\
&= -\sum_i (y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{k,j}} = -\sum_i \Delta_i \frac{\partial}{\partial W_{k,j}}\left(\sum_j W_{j,i}a_j\right) \\
&= -\sum_i \Delta_i W_{j,i}\frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i}\frac{\partial g(in_j)}{\partial W_{k,j}} \\
&= -\sum_i \Delta_i W_{j,i} g'(in_j)\frac{\partial in_j}{\partial W_{k,j}} \\
&= -\sum_i \Delta_i W_{j,i} g'(in_j)\frac{\partial}{\partial W_{k,j}}\left(\sum_k W_{k,j}a_k\right) \\
&= -\sum_i \Delta_i W_{j,i} g'(in_j)a_k = -a_k\Delta_j
\end{aligned}$$

Back-propagation learning contd.

At each epoch, sum gradient updates for all examples and apply

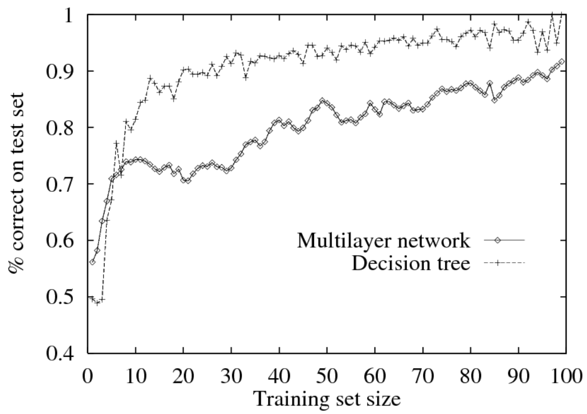
Training curve for 100 restaurant examples: finds exact fit



Typical problems: slow convergence, local minima

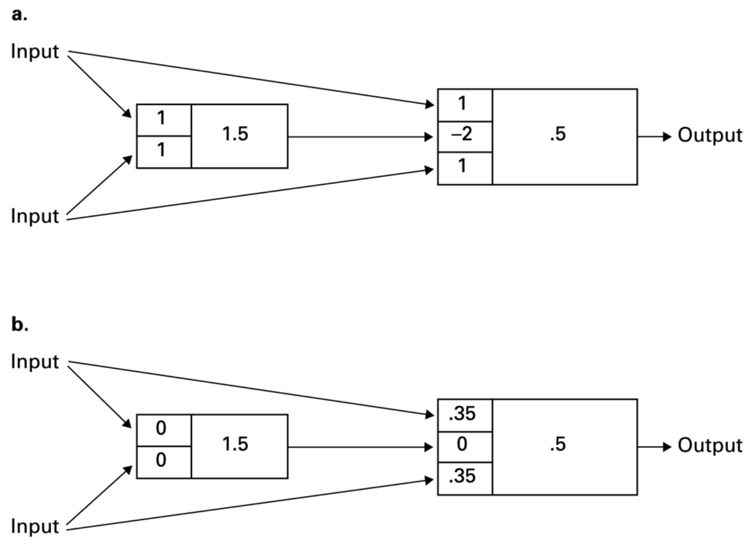
Back-propagation learning contd.

Learning curve for MLP with 4 hidden units:

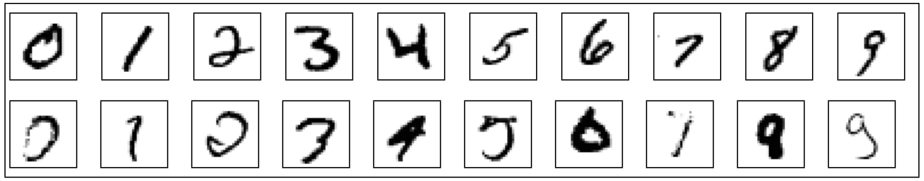


MLPs are quite good for complex pattern recognition tasks, but resulting hypotheses cannot be understood easily

Neural Network with two layers



Handwritten digit recognition



- 400–300–10 unit MLP = 1.6% error
- LeNet: 768–192–30–10 unit MLP = 0.9% error
- Current best (kernel machines, vision algorithms) ≈ 0.6% error
- Humans are at 0.2% – 2.5 % error

Summary

- Perceptrons (one-layer networks) insufficiently expressive
- Multi-layer networks are sufficiently expressive; can be trained by gradient descent, i.e., error back-propagation
- Many applications: speech, driving, handwriting, fraud detection, etc.

Summary on Learning

- Decision Trees and k-NN have problems with high dimensions
- Decision Tree are easily understandable to humans
- Naive Baysian Network is fast to train and update incrementally but often less accurate than k-NN
- For regression problems neural nets with linear output functions, regression trees or locally weighted nearest neighbors are all appropriate choices.

AI State of the Art

- http://www.aaai.org
  once "American Association for Artificial Intelligence", now "Association for the Advancement of Artificial Intelligence"

  http://www.aaai.org/Conferences/IAAI/iaai.php
  Innovative Applications of Artificial Intelligence Conference (IAAI)
- http://www.ijcai.org/
  International Joint Conferences on Artificial Intelligence
- http://www.eccai.org/
  European Coordinating Committee for Artificial Intelligence

  http://www.eccai.org/ecai.shtml
  European Conference on AI
- http://www.daimi.au.dk/~bmayoh/dais.html
  http://www.cs.au.dk/~bmayoh/dais.html
  Danish Artificial Intelligence Society