

Neural Networks

Introduction to Artificial Intelligence

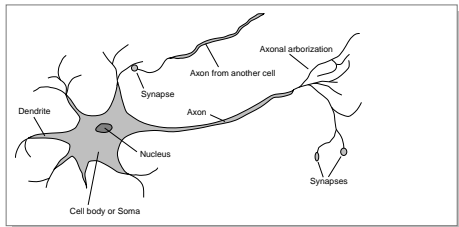
G. Lakemeyer

Winter Term 2018/19

Brain vs. Computer

Analogy to the human brain:

many processors (**neurons**)
and connections (**synapses**),
which process information
locally and in parallel.



von Neumann computer vs. the brain:

	Computer	Human Brain
Computational units	1 CPU, 10^5 gates	10^{11} neurons
Storage units	10^9 bits RAM, 10^{10} bits disk	10^{11} neurons, 10^{14} synapses
Cycle time	10^{-8} sec	10^{-3} sec
Bandwidth	10^9 bits/sec	10^{14} bits/sec
Neuron updates/sec	10^5	10^{14}

Note: The brain cycle is slow (10^{-3} s), but updates work in parallel. A serial simulation on a computer needs several hundred cycles.

Advantages of Neural Networks

- High processing speed through massive parallelism;
- still works if parts of the network are damaged (robustness);
- graceful degradation;
- designed for inductive learning.

It seems reasonable (obvious?) to try and recreate these advantages by designing artificial neural networks.

Research in this area started already in the 40-ies.
(McCulloch und Pitts 1943)

Some Terminology of Neural Networks

Units: represent the nodes (neurons) in the network.

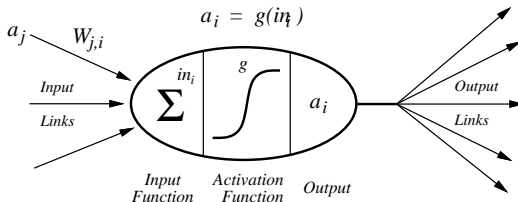
Input/output units: Special nodes connected to the external environment.

Links: Edges between nodes in the network. Each node has input and output edges.

Weights: Each edge has a weight, usually a real number.

Activation level: The value computed by a unit given its weighted inputs. It is passed to the neighbor nodes via the output edges.

How a Unit Works

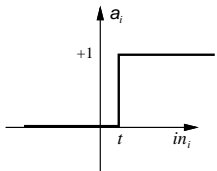


$$a_i := g(in_i) = g \left(\sum_j w_{j,i} \times a_j \right)$$

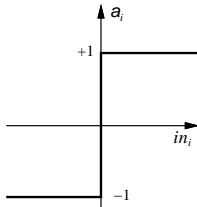
g is usually a **nonlinear** function.

Activation Functions

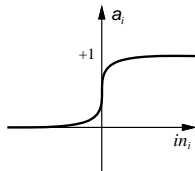
Some important activation functions are:



(a) Step function



(b) Sign function



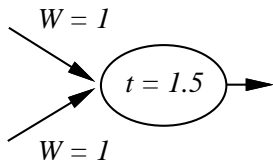
(c) Sigmoid function

Note: t in *step* represents a **threshold**, that is, the minimum total weighted input necessary for the neuron to fire (similar to actual neurons in the brain).

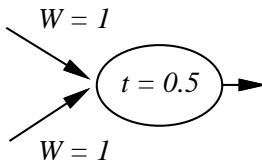
Mathematically, one can always use a threshold of 0 by having an additional input with activation level $a_0 = -1$ and $W_{0,i} = t$. Then

$$a_i = \text{step}_t \left(\sum_{j=1}^n W_{j,i} \times a_j \right) = \text{step}_0 \left(\sum_{j=0}^n W_{j,i} \times a_j \right)$$

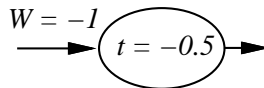
Representing Boolean Functions



AND



OR



NOT

Thus neural nets can at least represent any arbitrary Boolean function.

[We will see below that they can represent much more.]

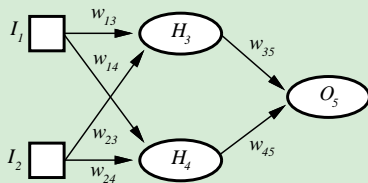
Network Topologies

Feed-forward: Units and links represent a directed acyclic graph.

Recurrent: Units and links represent arbitrary directed graphs.

Nets are often arranged in **layers**, that is, nodes of one layer are linked only to nodes of the next layer. There are no links between nodes of a layer.
(When counting layers, the input layer is ignored.)

Example of a feed-forward 2-layer network:



Of all networks feed-forward nets are the best understood.
(The output is solely a function of the inputs and the weights.)
Recurrent networks are often unstable, oscillate, or have chaotic behavior.

Note: The brain is massively recurrent.

Recurrent Network Types – Hopfield Nets

- Recurrent nets with symmetric bi-directional links ($W_{i,j} = W_{j,i}$).
- All units are both input and output units.
- Nets function as **associative memory**.
- After training a new input is matched against the “closest” example seen during training.

Example:

Training with n photographs.

New input = part of a photograph used during training.

Then the network reconstructs the whole image.

Note: Each weight is a partial representation of every image!

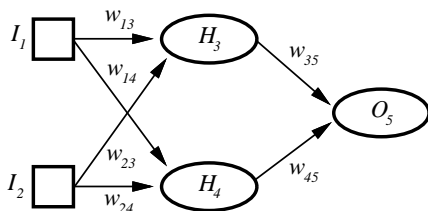
Theorem

A Hopfield Net can reliably store $0, 138 \times N$ examples, where N is the number of units.

Recurrent Network Types – Boltzmann Machines

- Also use symmetric weights.
- There are units which are neither input nor output units.
- Stochastic Activation function.
- State transitions are like simulated annealing search for the configuration that best approximates the training set.
(Formally identical to a certain kind of belief nets.)

Feed-Forward (FF) Nets



Hidden Units:

Units without direct connection to the external environment. Hidden units are organized in one or more hidden layers.

Perceptrons:

FF Networks without hidden units.

FF Nets represent complex nonlinear functions.

- With 1 hidden layer every **continuous function** is representable.
- With 2 hidden layers **every function** is representable.

When the topology and activation function g are fixed, the representable functions have a specific parametrized form (parameters = weights).

Example:

$$a_5 = g(W_{3,5}a_3 + W_{4,5}a_4) = g(W_{3,5}g(W_{1,3}a_1 + W_{2,3}a_2) + W_{4,5}g(W_{1,4}a_1 + W_{2,4}a_2))$$

\Rightarrow **Learning** = Search for the correct parameters = **nonlinear regression**.

Optimal Network structure?

Choosing the right network is a difficult problem!

Also the optimal net may be exponentially large (relative to the input).

- Net is too small: the desired function is not representable.
- Net is too big: Net memorizes examples without generalization (analogous to memorizing in decision trees) **Overfitting**

There is no good theory of how to choose the right network, only some heuristics.

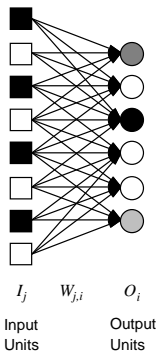
Heuristics of optimal brain damage:

Start with a network with a maximal number of connection. After the 1st training reduce the number of connections using information theory. Iterate.

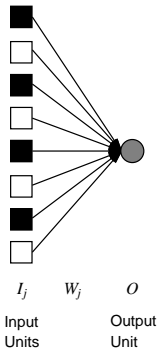
Example:

Network to recognize zip codes. 3/4 of the initial connections could be removed. (There are also methods to move from a network with few nodes to a network with more nodes.)

Perceptrons



Perceptron Network



Single Perceptron

$$O = \text{Step}_0 \left(\sum_j W_j I_j \right) = \text{Step}_0(\mathbf{W} \cdot \mathbf{I})$$

How Do Perceptrons Learn?

function NEURAL-NETWORK-LEARNING(*examples*) **returns** *network*

network \leftarrow a network with randomly assigned weights

repeat

for each *e* **in** *examples* **do**

$\mathbf{O} \leftarrow$ NEURAL-NETWORK-OUTPUT(*network*, *e*)

$\mathbf{T} \leftarrow$ the observed output values from *e*

 update the weights in *network* based on *e*, \mathbf{O} , and \mathbf{T}

end

until all examples correctly predicted or stopping criterion is reached

return *network*

Error = $T - O$ with

O = predicted (actual) output

T = correct output

Update-Rule: $W_j := W_j + \alpha \times I_j \times \text{Error}$

Theorem: [Rosenblatt 1960]

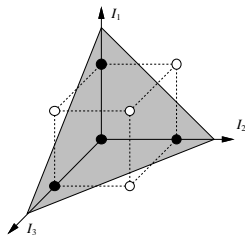
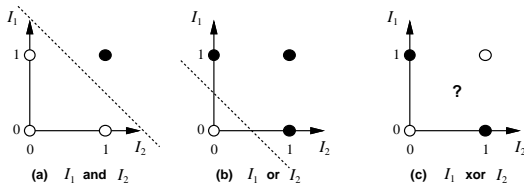
Learning using the update rule always converges to the correct output for representable functions.

What can Perceptrons represent?

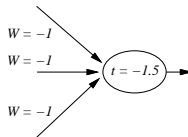
Answer: Not much!

Perceptrons can represent exactly the class of **linearly separable functions**.

Examples:

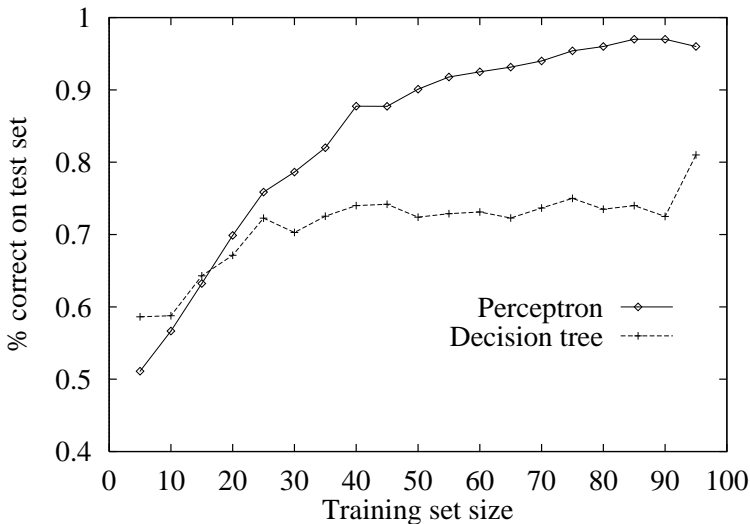


(a) Separating plane

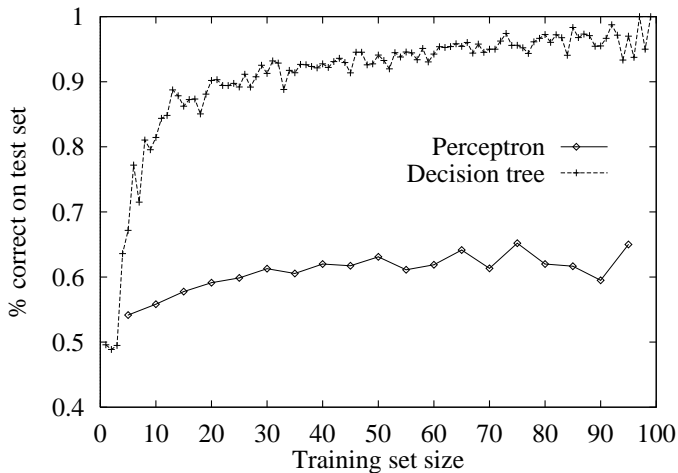


(b) Weights and threshold

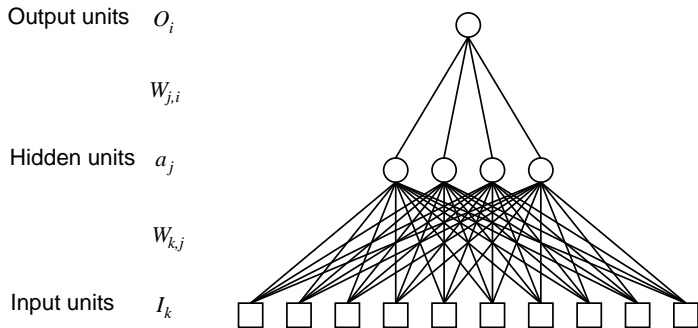
Learning Behavior I: Majority function



Learning Behavior II: Restaurant Example



Multi-layer Feed-Forward Networks



(This topology is suitable for the restaurant example.)

Back-Propagation

function BACK-PROP-UPDATE(*network*, *examples*, α) **returns** a network with modified weights

inputs: *network*, a multilayer network
examples, a set of input/output pairs
 α , the learning rate

repeat

for each *e* **in** *examples* **do**

/* Compute the output for this example */

$\mathbf{O} \leftarrow \text{RUN-NETWORK}(\text{network}, \mathbf{I}^e)$

/* Compute the error and Δ for units in the output layer */

$\text{Err}^e \leftarrow \mathbf{T}^e - \mathbf{O}$

/* Update the weights leading to the output layer */

$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \text{Err}_i^e \times g'(in_i)$

for each subsequent layer **in** *network* **do**

/* Compute the error at each node */

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

/* Update the weights leading into the layer */

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

end

end

until *network* has converged

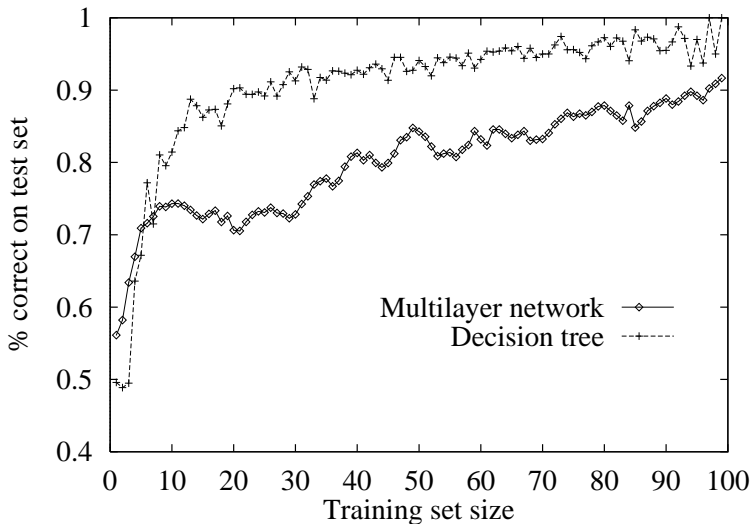
return *network*

$\text{Err}_i^e = \text{Error } T_i^e - O_i^e$ of the *i*-th output unit.

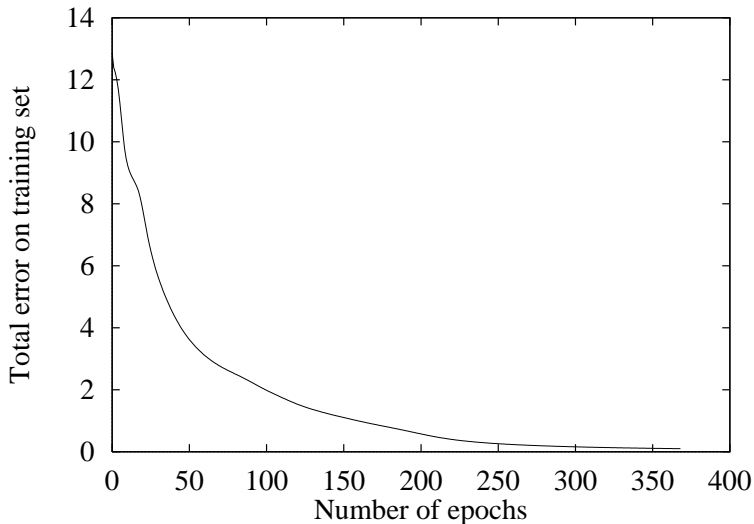
Err^e = error vector.

$\Delta_i = \text{Error}_i \times g'(in_i)$

Restaurant Example



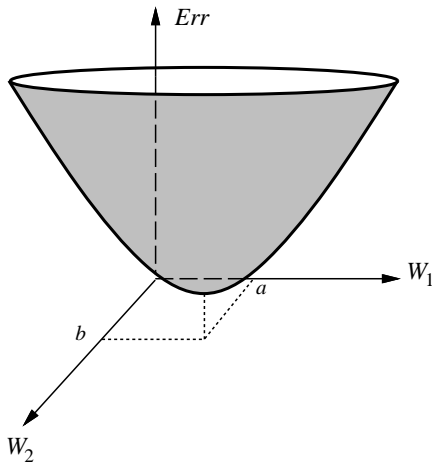
Restaurant Example



Backprop = Gradient Descent

Gradient Descent

is like Hill Climbing except that we are looking for minima instead of maxima.



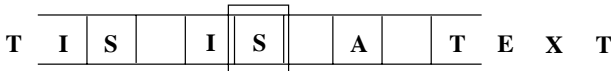
Summary

- What are they good for?
Neural nets are useful for **attribute-based** representations (like decision trees), in particular also for attributes with **continuous** values.
- Size of a net:
 $2^n/n$ units to represent any Boolean function.
($\Rightarrow 2^n$ weights). In practice one often needs much less.
- Efficiency of Learning:
no useful theoretical results.
Also a problem with local minima. **Simulated annealing** sometimes helpful (see Boltzmann Machines)
- Generalization: Good if the right network is used :-)
- Noise: no problem.
- Transparency:
bad! Often one has no idea why a network yields a certain output.
Decision trees are much better in this regard!
- Using additional knowledge: bad. No theoretical results.

Application: Pronunciation

NETtalk synthesizes speech from written text (in English) [Sejnowski and Rosenberg 1987]

Input: seven-character window sliding over the written text.:



uses **29 input units** per character (for the 26 letters of the alphabet, blank, comma, and 1 unit for other characters)

Hidden layer: 80 units

Output: Units to encode phonemes.

Training set: text with 1024 words. After 50 epochs 95% correctness.

Problems: words with identical spelling but different pronunciation (e.g. lead).

Result on test set: 78% correctness.

(Today there are better systems not based on neural nets, but NETtalk paved the way for the commercial success of neural nets.)

Application: Recognizing Zip Codes

Neural net to recognize handwritten numbers (zip codes) [Le Cun 89]

Input: 16×16 pixels per digit.

3 hidden layers: 768; 192; 30 units.

Output: 10 units for digits 0–9.

Limitation of connections was crucial:

- each unit of the 1st layer was connected with a 5×5 array of the input (25 connections).
- First layer was organized in 12 groups with 64 units each. All units of the same group use identical weights.
- The whole network used only 9760 different weights instead of 200,000.

Training set: 7300 examples.

Test set: 2000 examples; 99% correctness!

Is used by US Postal Service, realized on VLSI.