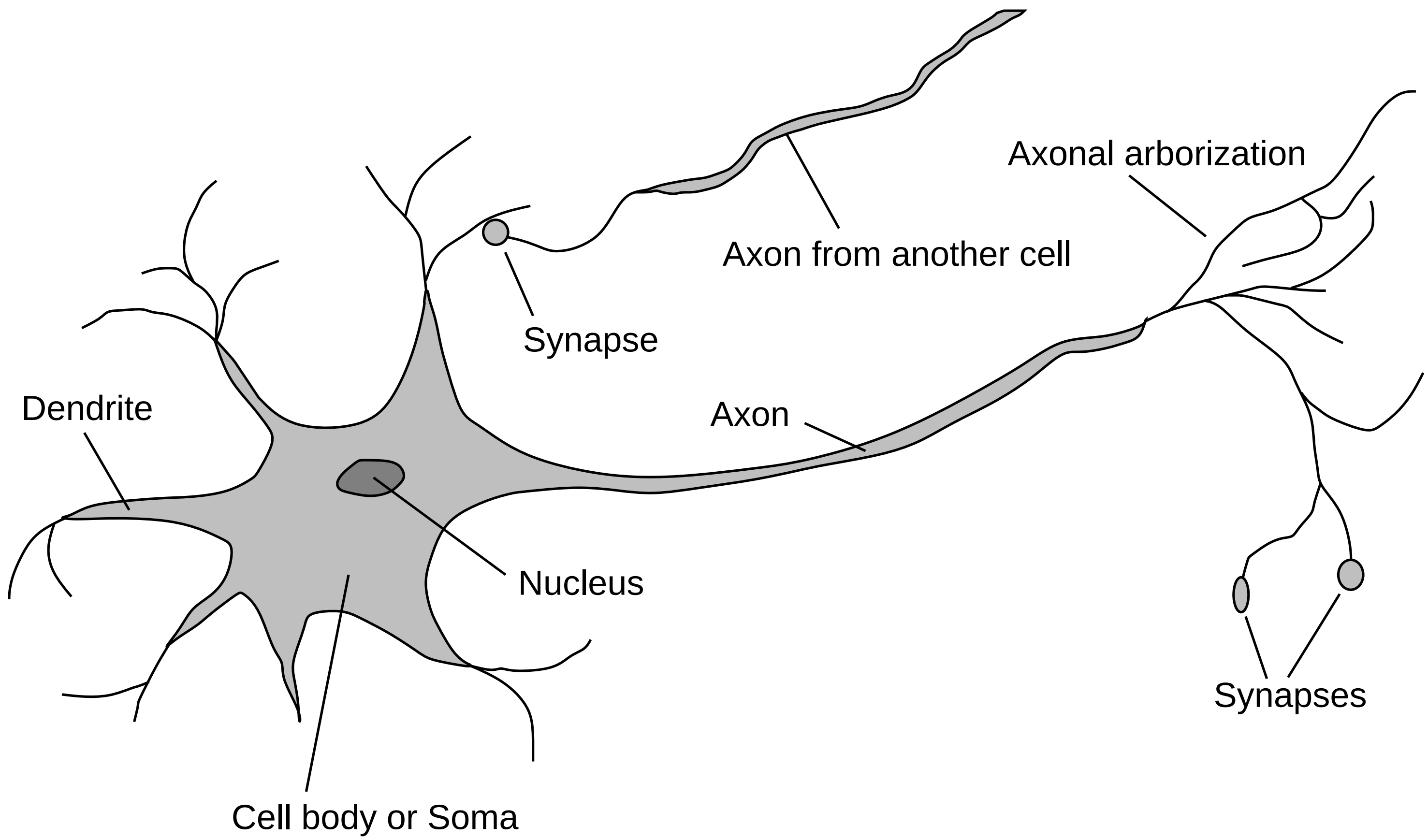
Class Notes CIS 675
Neural Networks

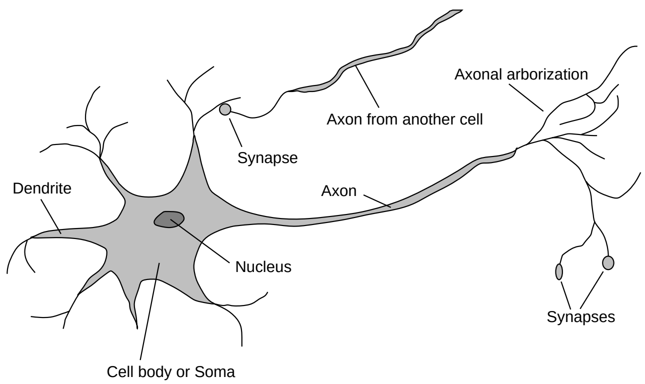
Neural Networks

- How the brain works
- Neural Networks
- Mathematical Representation
- Structure of Neural Networks
- Examples: Boolean functions
- Perceptrons

How the Brain Works



How the Brain Works (2)



- **neuron**: fundamental functional unit of all nervous system tissue,
- **soma**: cell body, contains cell nucleus
- **dendrites**: a number of fibers, inputs
- **axon**: single long fiber, many branches, output
- **synapse**: junction of axon and dendrites, each neuron forms synapses with 10 to 100,000 other neurons.

How the Brain Works (3)

Signals are propagated from neuron to neuron be a electrochemical reaction:

1. chemical substances are released from the synapses and enter the dendrites, raising or lowering the electrical potential of the cell body;
2. when a the potential reaches a threshold, an electrical pulse or **action potential** is sent down the axon;
3. the pulse spreads out along the branches of the axon, eventually reaching synapes and releasing transmitters into the bodies of other cells;

- **excitory** synapses: increase potential,
- **inhibitory** synapses: decrease potential.

How the Brain Works (4)

- Synaptic connections exhibit **plasticity** – long term changes in the strength of connections in response to the pattern of stimulation.
- Neuron also form new connections with other neurons, and sometimes entire collections of neurons migrate.
- These mechanisms are thought to form the basis of learning.

How the Brain Works (5)

Comparing brains with digital computers

	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}

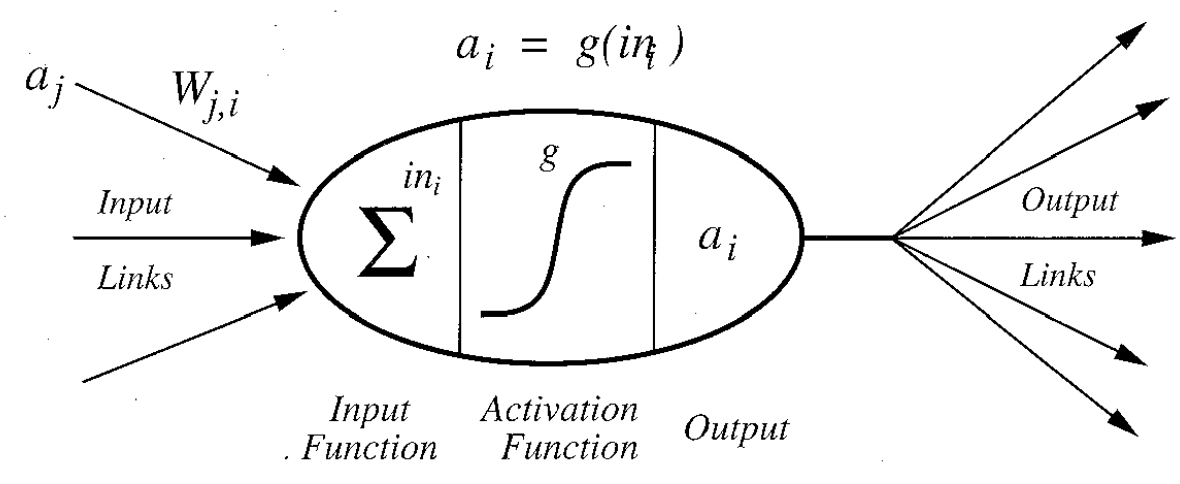
(Computer resources at the state of 1994)

- Brain is highly parallel: a huge number of neurons can be updated simultaneously (as opposed to a single CPU computer).
- Brains are more fault tolerant than computers.

Neural Networks

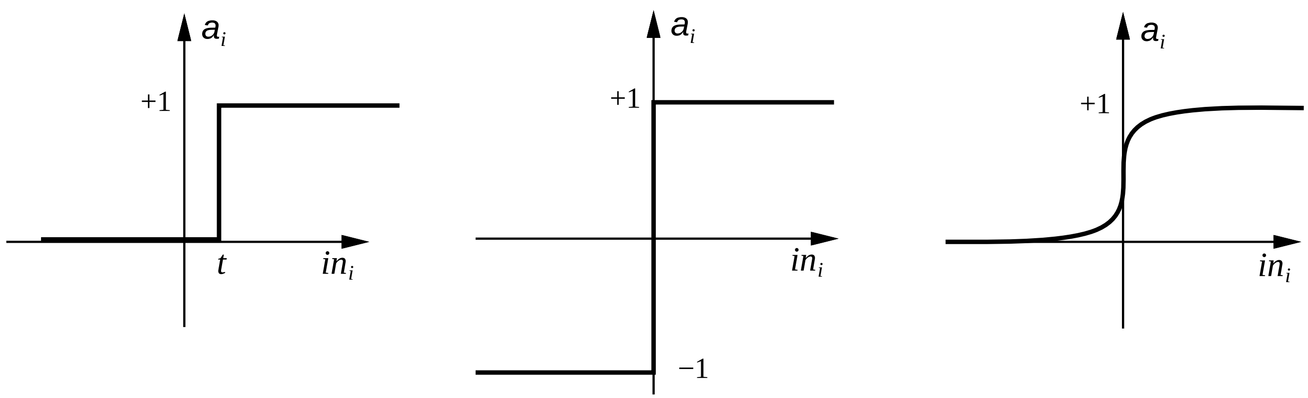
- A neural network (NN) is composed of a number of nodes, or **units**, connected by **links**. Each link has a numeric **weight** associated with it.
- Weights are the primary means of long-term storage in neural networks, and learning usually takes place by updating the weights.
- Some of the units are connected to the external environment → input/output units.
- Each unit has:
  1. a set of input links from other units,
  2. a set of output links to other units,
  3. a current activation level,
  4. and a means of computing the activation level at the next time step, given its input weights.

Mathematical Representation



- **input function**: linear, $in_i = \sum_j W_{j,i} a_j = \mathbf{W}_i \cdot \mathbf{a}_i$
- **activation function**: nonlinear, $a_i \leftarrow g(in_i)$, g can be a step, sign, or sigmoid function.

Mathematical Representation (2)



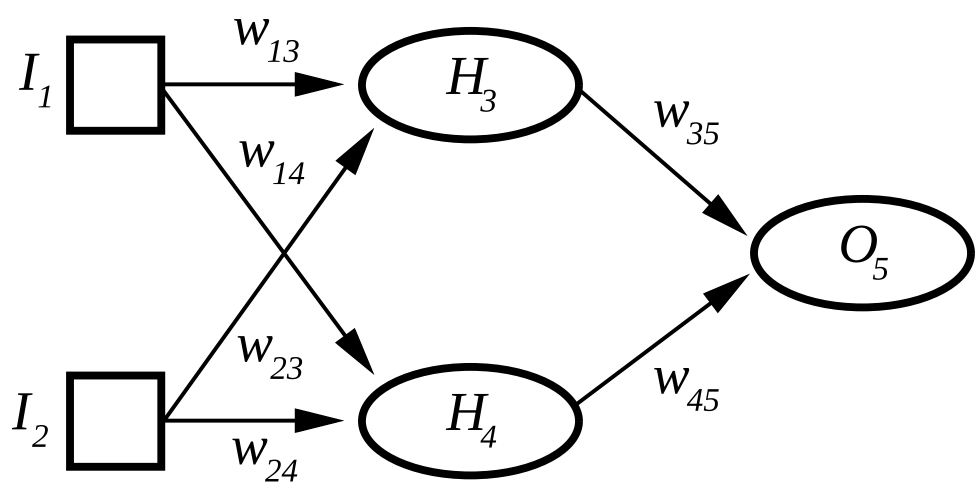
(a) Step function **(b) Sign function** **(c) Sigmoid function**

- usually, all units use the same function,
- threshold parameter t can be replaced by static input neuron with a certain weight.
- in most cases the weights change during the learning process, thresholds and functions remain the same.

Mathematical Notation

Notation	Meaning
a_i $\mathbf{a}_i$	Activation value of unit i (also the output of the unit) Vector of activation values for the inputs to unit i
g g'	Activation function Derivative of the activation function
Err_i Err^e	Error (difference between output and target) for unit i Error for example e
I_i $\mathbf{I}$ $\mathbf{I}^e$	Activation of a unit i in the input layer Vector of activations of all input units Vector of inputs for example e
in_i	Weighted sum of inputs to unit i
N	Total number of units in the network
O O_i $\mathbf{O}$	Activation of the single output unit of a perceptron Activation of a unit i in the output layer Vector of activations of all units in the output layer
t	Threshold for a step function
T $\mathbf{T}$ $\mathbf{T}^e$	Target (desired) output for a perceptron Target vector when there are several output units Target vector for example e
$W_{j,i}$ W_i $\mathbf{W}_i$ $\mathbf{W}$	Weight on the link from unit j to unit i Weight from unit i to the output in a perceptron Vector of weights leading into unit i Vector of all weights in the network

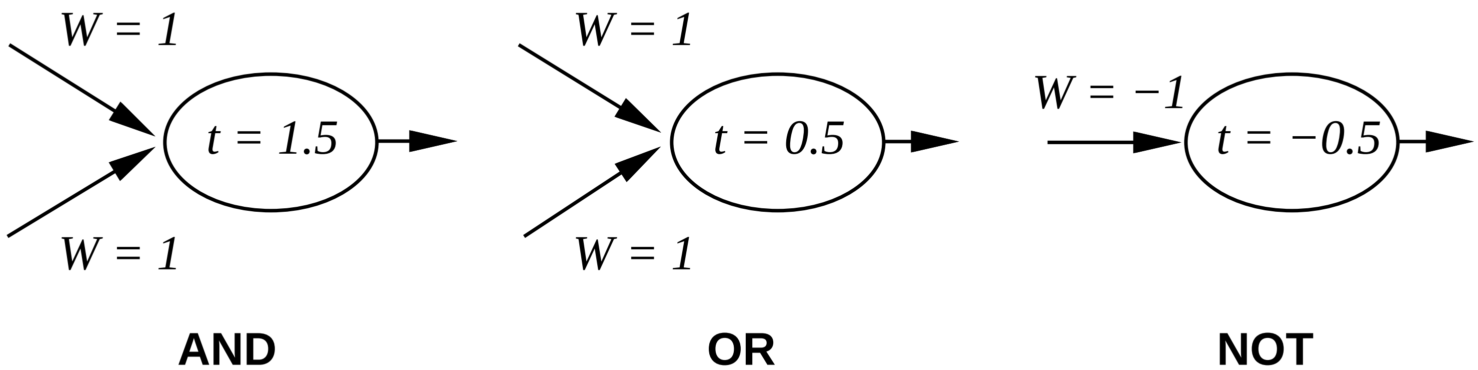
Structure of Neural Networks

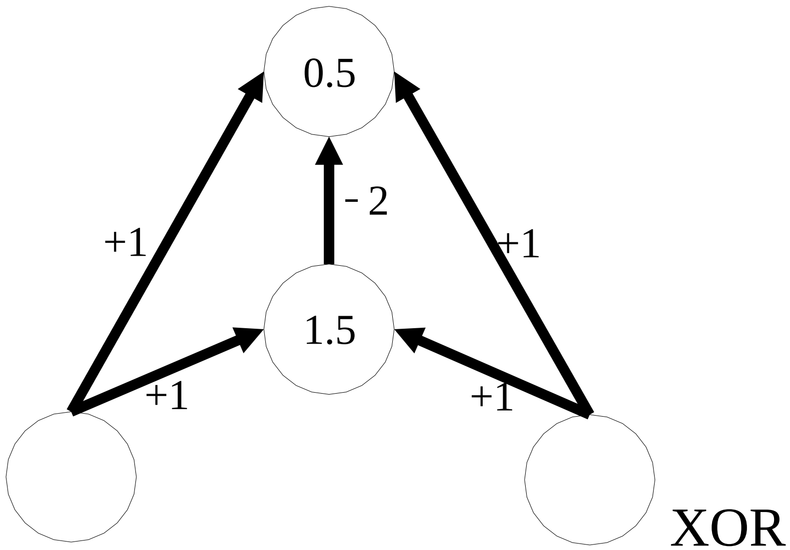


- **multilayer networks**: input, hidden, output
- **feed forward**: directed links from one layer to the next.

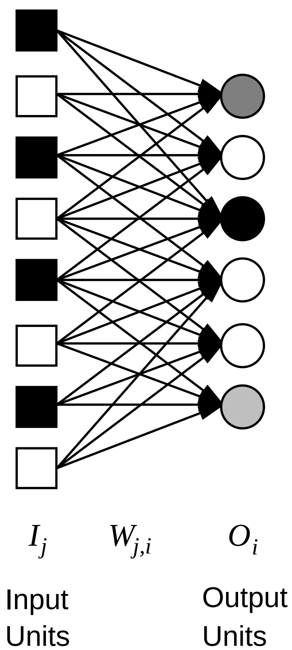
$$
\begin{aligned}
a_5 &= g(W_{3,5}\, a_3 + W_{4,5}\, a_4) \\
&= g(W_{3,5}\, g(W_{1,3}\, a_1 + W_{2,3}\, a_2) + W_{4,5}\, g(W_{1,4}\, a_1 + W_{2,4}\, a_2))
\end{aligned}
$$

Examples: Boolean functions

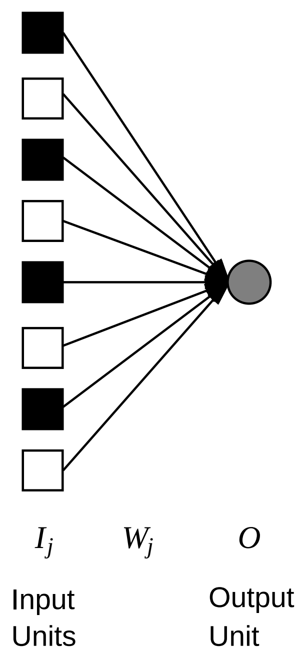




Perceptrons



Perceptron Network



Single Perceptron

- Term originally used for multi-layered feed-forward networks of any topology(1950), today synonym for a single-layer, feed-forward network.
- Output units are independent from each other.

$$O = Step_0\left(\sum_j W_j I_j\right)$$

Perceptron (2)

- What can perceptrons represent? Linear separable functions. E.g. boolean majority, AND, OR, NOT. But not XOR.
- There is a perceptron algorithm that will learn any linearly separable function, given enough training
- Training $\rightarrow$ update weights

$$\begin{aligned} Err &= T - O \\ W_j &= W_j + \alpha \times I_j \times Err \end{aligned}$$

with **learning rate** α.

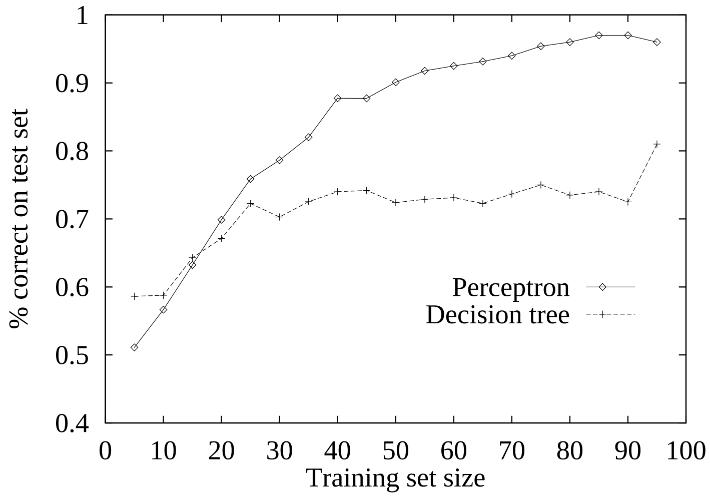
- Algorithm: repeat update of weights for every example until error is minimized $\rightarrow$ **gradient descent**.

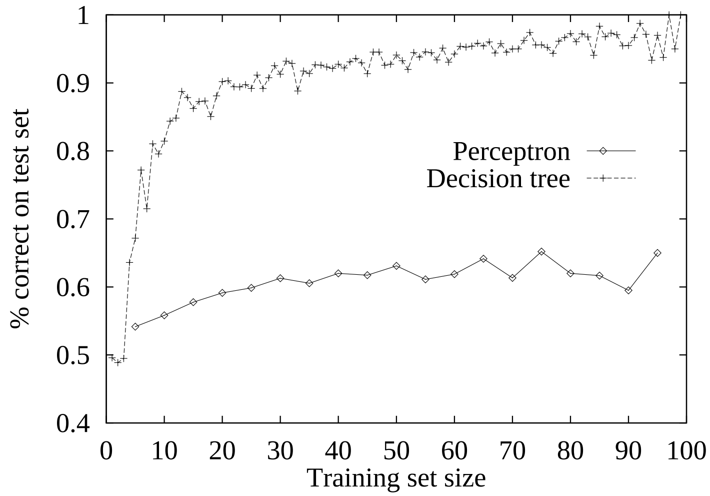
Perceptron: Learning Method

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

network ← a network with randomly assigned weights
repeat
  for each *e* **in** *examples* **do**
    O ← NEURAL-NETWORK-OUTPUT(*network*, *e*)
    T ← the observed output values from *e*
    update the weights in *network* based on *e*, **O**, and **T**
  end
until all examples correctly predicted or stopping criterion is reached
return *network*

Limitations of Perceptrons



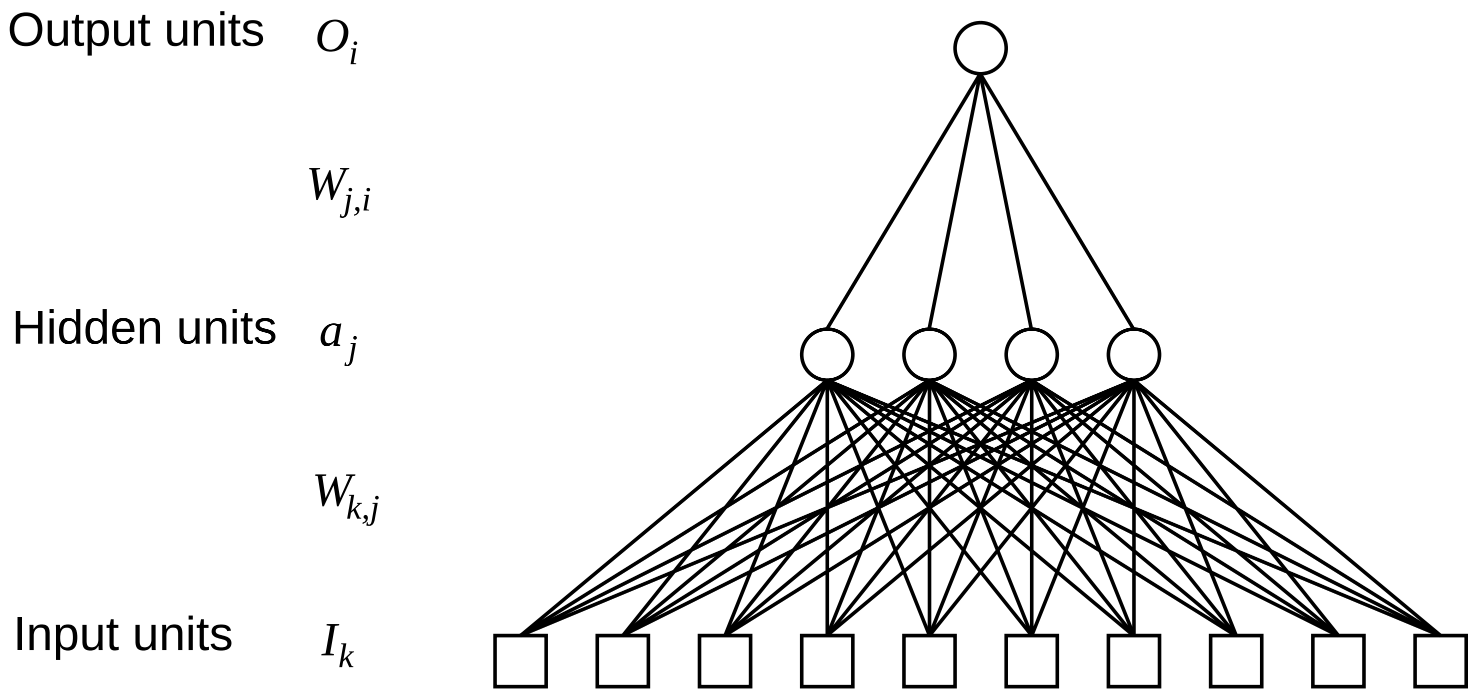


Comparing the performance of perceptron and decision trees

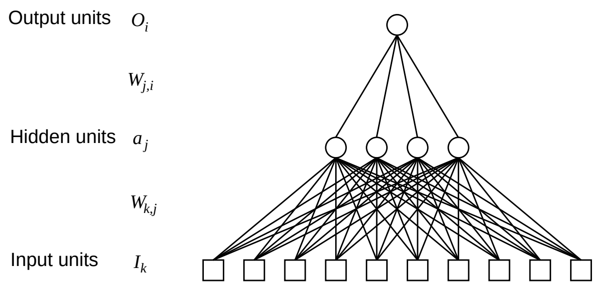
Left: Perceptrons are better in learning the majority function (of 11 inputs).

Right: Decision trees are better at learning the *WillWait* predicate for the restaurant example.

Multilayer Feed-Forward Networks



Multilayer Feed-Forward Networks (2)



- Multilayer networks are able to learn functions that are not linear separable.
- In contrast to singlelayer networks, multilayer learning algorithms are neither efficient nor guaranteed to converge to a global optimum.
- Most popular: **Back-Propagation Learning**

Back-Propagation Learning

- Learning: modify the weights of the network in order to minimize the difference between output and training examples $\rightarrow$ change weights to what amount?
- Update of weights by layers: weight changes depend on the activation of the units in the layer below.

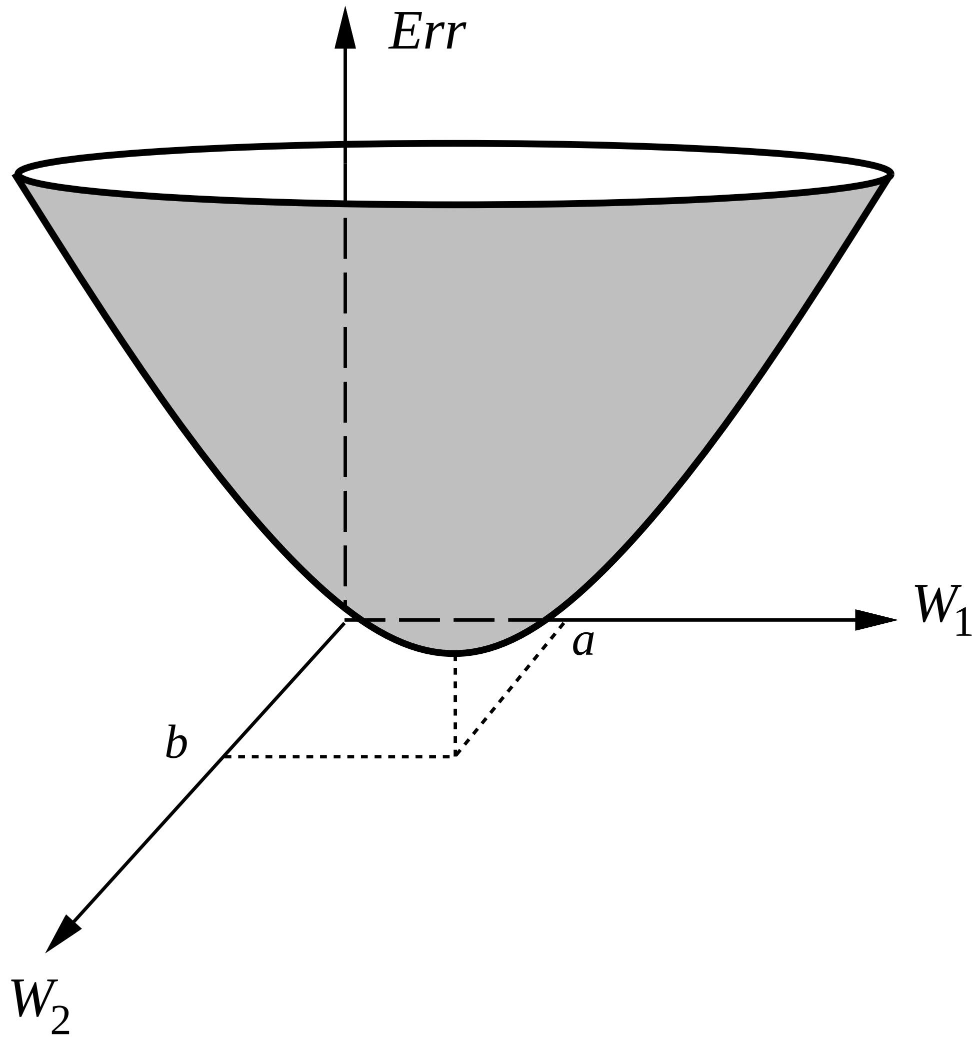
$$
\begin{aligned}
Err_i &= (T_i - O_i) \\
W_{j,i} &\leftarrow W_{j,i} + \alpha \times a_j \times Err_i \times g'(in_i)
\end{aligned}
$$

Back-Propagation Algorithm

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* */
        $\mathbf{O} \leftarrow$ RUN-NETWORK(*network*, $\mathbf{I}^e$)
      /* *Compute the error and* Δ *for units in the output layer* */
        $\mathbf{Err}^e \leftarrow \mathbf{T}^e - \mathbf{O}$
      /* *Update the weights leading to the output layer* */
        $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times Err_i^e \times g'(in_i)$
      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*

Gradient Descent in the Weights Space



Discussion

- **Expressiveness**: attribute-base representation, no expressive power of general logical representation. Well-suited for continuous inputs and outputs.
- **Computational efficiency**: depends on the amount of computation time required to train the network. Various methods can be used to reach convergence (Simulated Annealing).
- **Generalization**: NN can generalize well, in particular if there is a smooth dependency between input and output. Though, it is difficult to predict the success of a NN, or systematically optimize the design.
- **Sensitivity to noise**: one of the strongest feature.
- **Transparency**: NN are practically black boxes; they provide no insight at all.
- **Prior Knowledge**: can be provided in the design of the network topology → rule of thumb, many years of experience, and in form of data pre-processing.