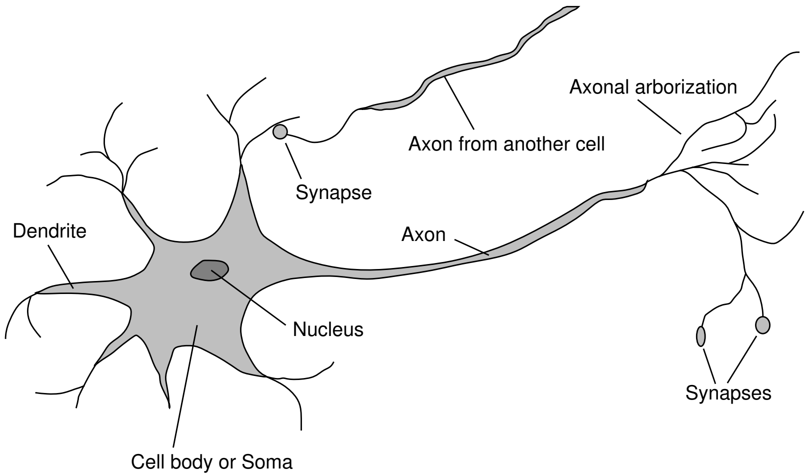
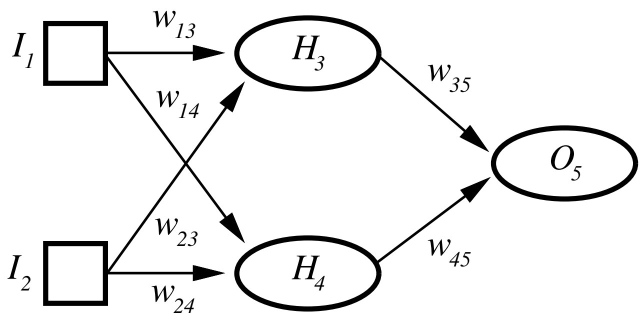
Biological Neurons



Artificial Neural Networks

- Using complex networks of simple computing elements as mathematical models to mimic the functions of the brain.
- An example:



- Unit types:
  - Inputs (often not considered as units because no processing is involved)
  - Hidden units
  - Output units
- Two major issues:
  - Representation – what class of functions does a network with some chosen structure represent.
  - Learning – how to find appropriate network structure and/or weights to solve a given problem.

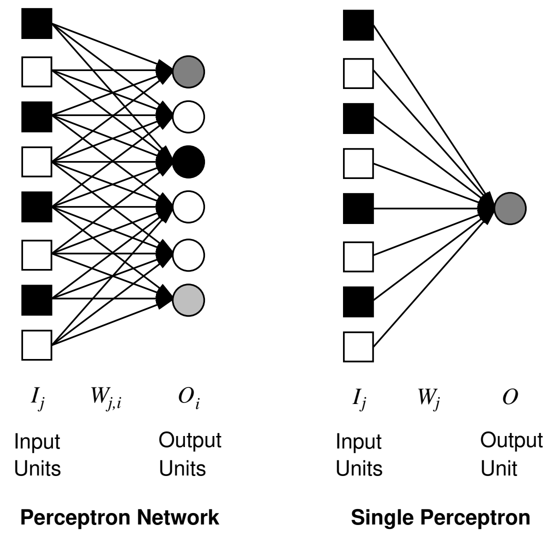
Neural Network Learning

```
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
```


Simple Perceptrons

- A simple perceptron (or simply called perceptron) is a feed-forward network with only one layer of adjustable (learnable) weights connected to one or more threshold units (as output units).



- Step function:

$$\mathrm{step}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

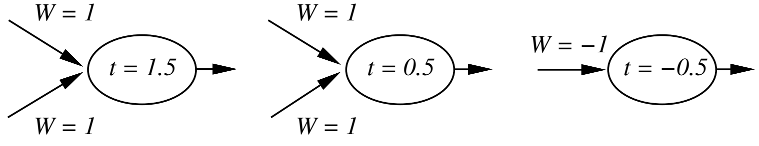
- Activation of threshold output unit with step function relating the input and output:

$$O = \mathrm{step}\left(\sum_{j=1}^{n} w_j I_j - \theta\right) \quad \text{or} \quad O = \mathrm{step}\left(\sum_{j=0}^{n} w_j I_j\right)$$

where $w_0 = -\theta$ and $I_0 = 1$.

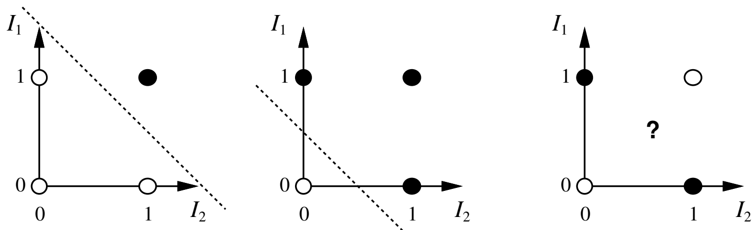
Representing Boolean Functions

- Single perceptrons for AND, OR, NOT:



AND **OR** **NOT**

- With at most *two* layers of sufficiently many perceptrons, any Boolean function can be represented. (This is obvious since any Boolean function can be represented in either disjunctive normal form (DNF) or conjunctive normal form (CNF).)
- A function can be represented by a single perceptron if and only if the function is *linearly separable*:



(a) I_1 **and** I_2 **(b)** I_1 **or** I_2 **(c)** I_1 **xor** I_2

Perceptron Learning Algorithm

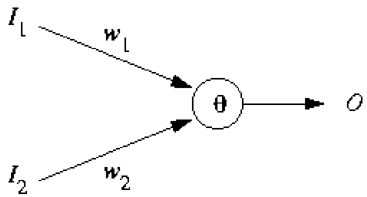
- It is an iterative algorithm.
- Basic idea:
  - If the observed output is different from the predicted one, then small adjustments in the weights are made to reduce the difference.
- Weight updating rule:

$$w_j \leftarrow w_j + \alpha I_j (T - O), \quad \forall j$$

where α is a constant *learning rate* and T is the correct (target) output from the training data.

Example

- Perceptron:

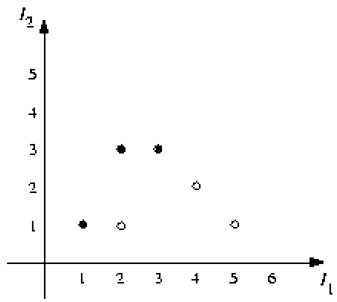


- Input-output relationship:

$$
\begin{aligned}
O &= \mathrm{step}(w_1 I_1 + w_2 I_2 - \theta) \\
&= \mathrm{step}(w_0 + w_1 I_1 + w_2 I_2)
\end{aligned}
$$

where $w_0 = -\theta$.

- Training set:



	I_1	I_2	T
e_1:	5	1	0
e_2:	2	1	0
e_3:	1	1	1
e_4:	3	3	1
e_5:	4	2	0
e_6:	2	3	1

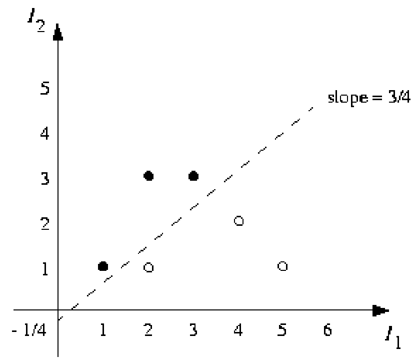
Example (cont'd)

- Trace of learning process:

  Take $\alpha = 1$ and the initial weight values to be 0.

Iteration	$\mathbf{w}^{\mathrm{old}}$	$\mathbf{I}$	T	O	$T = O$?	$\mathbf{w}^{\mathrm{new}}$
1	(0 0 0)	(1 5 1)	0	1	no	(-1 -5 -1)
2	(-1 -5 -1)	(1 2 1)	0	0	yes	(-1 -5 -1)
3	(-1 -5 -1)	(1 1 1)	1	0	no	(0 -4 0)
4	(0 -4 0)	(1 3 3)	1	0	no	(1 -1 3)
5	(1 -1 3)	(1 4 2)	0	1	no	(0 -5 1)
6	(0 -5 1)	(1 2 3)	1	0	no	(1 -3 4)
7	(1 -3 4)	(1 5 1)	0	0	yes	(1 -3 4)
8	(1 -3 4)	(1 2 1)	0	0	yes	(1 -3 4)
9	(1 -3 4)	(1 1 1)	1	1	yes	(1 -3 4)
10	(1 -3 4)	(1 3 3)	1	1	yes	(1 -3 4)
11	(1 -3 4)	(1 4 2)	0	0	yes	(1 -3 4)
12	(1 -3 4)	(1 2 3)	1	1	yes	(1 -3 4)

- Geometric interpretation of solution:

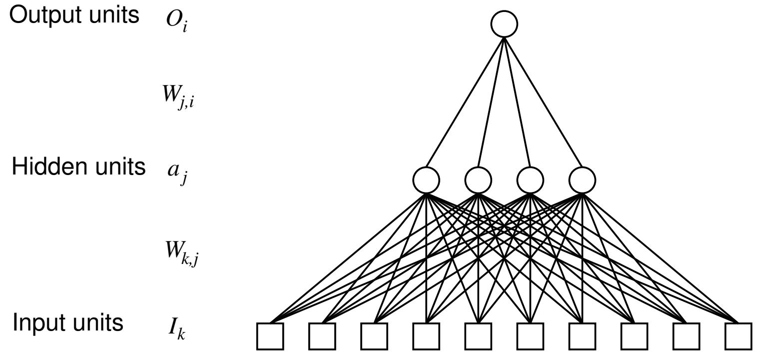


Convergence of Learning Algorithm

- Perceptron convergence theorem:
  - If a training set of examples is linearly separable, then applying the perceptron weight updating rule can always converge to some solution (i.e., a set of weights) in a *finite* number of steps for *any* initial choice of weights.
- The exact number of steps needed depends on:
  - Initial weight values
  - Learning rate in weight updating rule
  - Order of presentation of training examples

Multi-layer Feedforward Networks

- They can be seen as generalizations of simple perceptrons.
- An example:



- They are more powerful in representation power.
- They require more sophisticated learning algorithms.