

Neural Networks

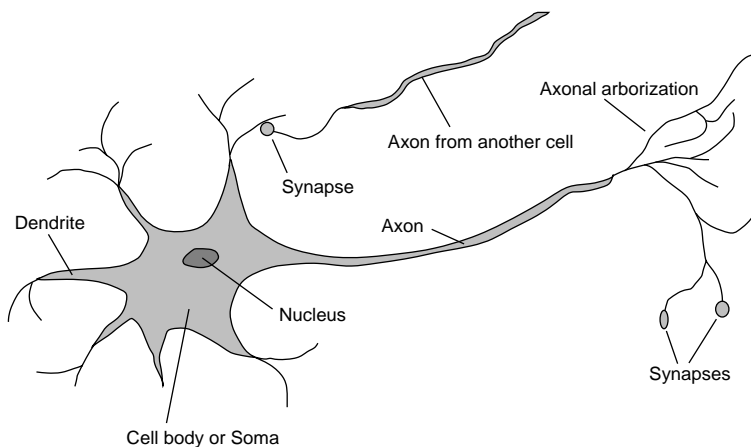
AIMA2e Chapter 19, Sections 1-5

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

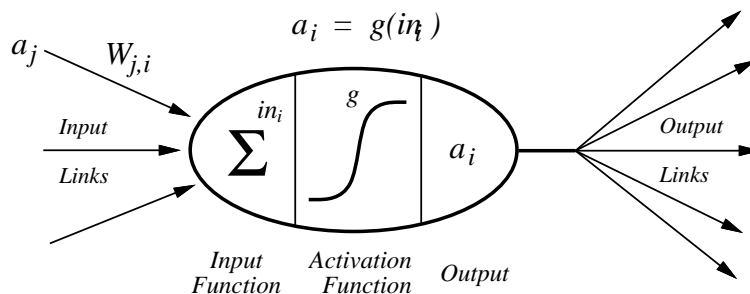


3

McCulloch–Pitts “unit”

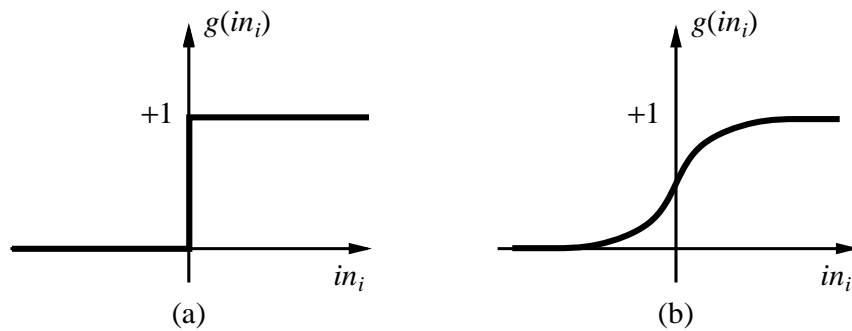
Output is a “squashed” linear function of the inputs:

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



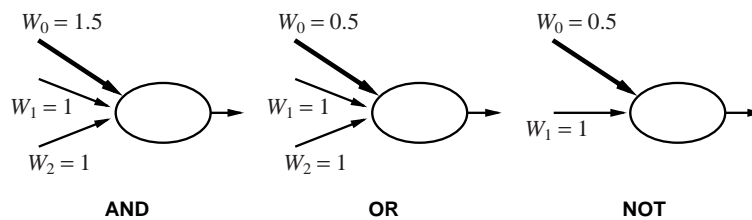
4

Activation functions



5

Implementing logical functions



McCulloch and Pitts: every Boolean function can be implemented

6

Network structures

Feed-forward networks:

- single-layer perceptrons
- multi-layer perceptrons

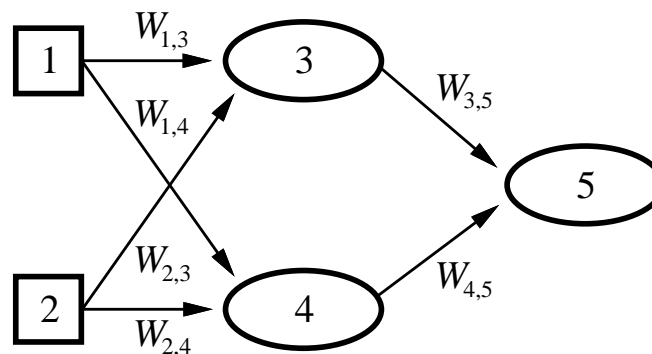
Feed-forward networks implement functions, have no internal state

Recurrent networks:

- Hopfield networks have symmetric weights ($W_{i,j} = W_{j,i}$)
 $g(x) = \text{sign}(x)$, $a_i = \pm 1$; *holographic associative memory*
- Boltzmann machines use stochastic activation functions,
 \approx MCMC in BNs
- recurrent neural nets have directed cycles with delays
 \Rightarrow have internal state (like flip-flops), can oscillate etc.

7

Feed-forward example

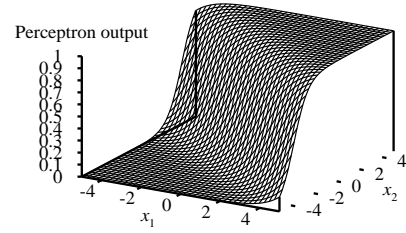
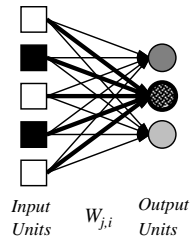


Feed-forward network = a parameterized family of nonlinear functions:

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

8

Perceptrons



9

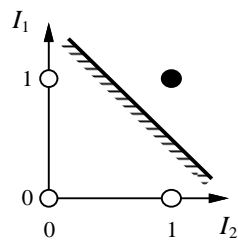
Expressiveness of perceptrons

Consider a perceptron with $g = \text{step function}$ (Rosenblatt, 1957, 1960)

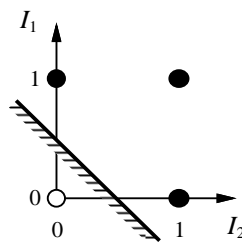
Can represent AND, OR, NOT, majority, etc.

Represents a **linear separator** in input space:

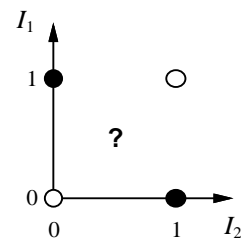
$$\sum_j W_j x_j > 0 \quad \text{or} \quad \mathbf{W} \cdot \mathbf{x} > 0$$



(a) I_1 and I_2



(b) I_1 or I_2



(c) I_1 xor I_2

10

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:

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

Simple weight update rule:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

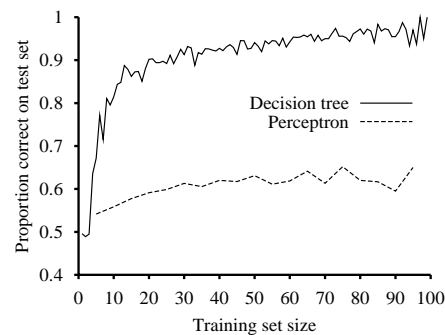
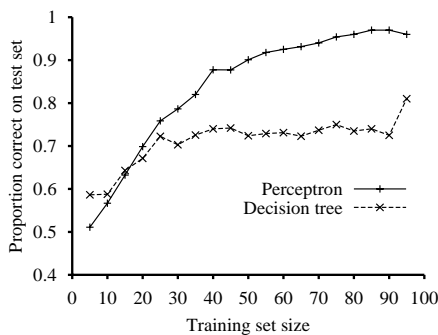
E.g., +ve error \Rightarrow increase network output

\Rightarrow increase weights on +ve inputs, decrease on -ve inputs

Perceptron learning contd.

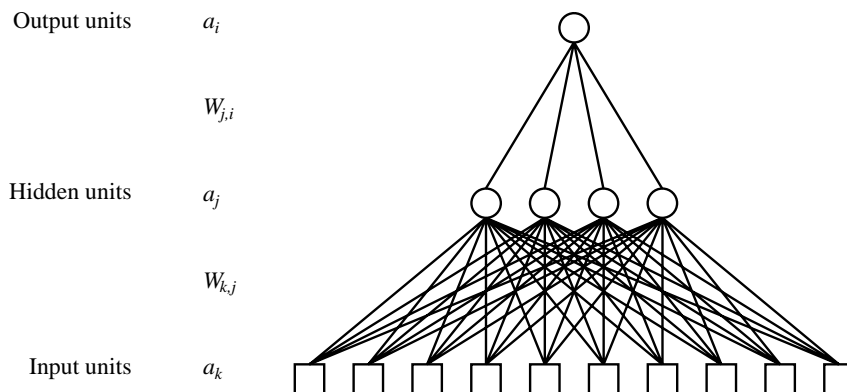
Perceptron learning rule converges to a consistent function

for any linearly separable data set



Multilayer perceptrons

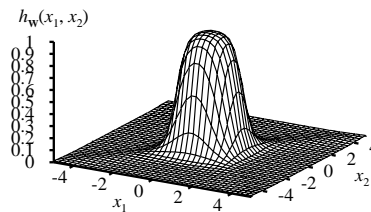
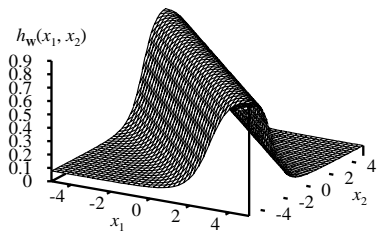
Layers are usually fully connected;
numbers of **hidden units** typically chosen by hand



13

Expressiveness of MLPs

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



14

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)

15

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.

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

16

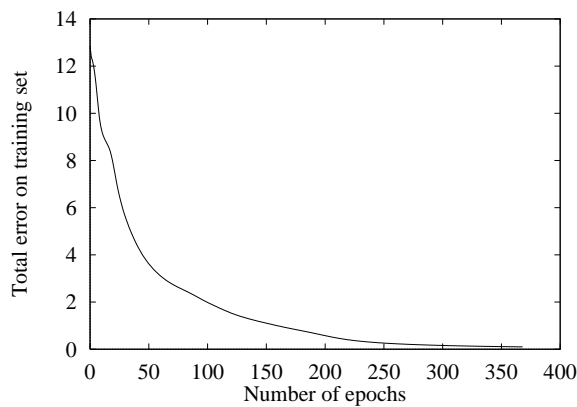
Back-propagation derivation contd.

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

17

Back-propagation learning contd.

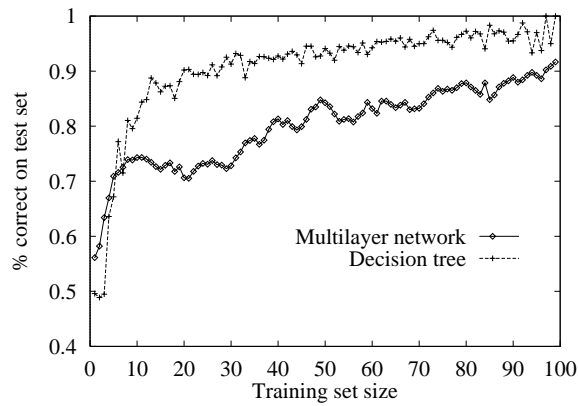
At each **epoch**, sum gradient updates for all examples and apply



Usual problems with slow convergence, local minima

18

Back-propagation learning contd.



19

Handwritten digit recognition



3-nearest-neighbor = 2.4% error

400-300-10 unit MLP = 1.6% error

LeNet: 768-192-30-10 unit MLP = 0.9% error

20

Summary

Most brains have lots of neurons; each neuron \approx linear-threshold unit (?)

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, credit cards, etc.