Artificial neural networks

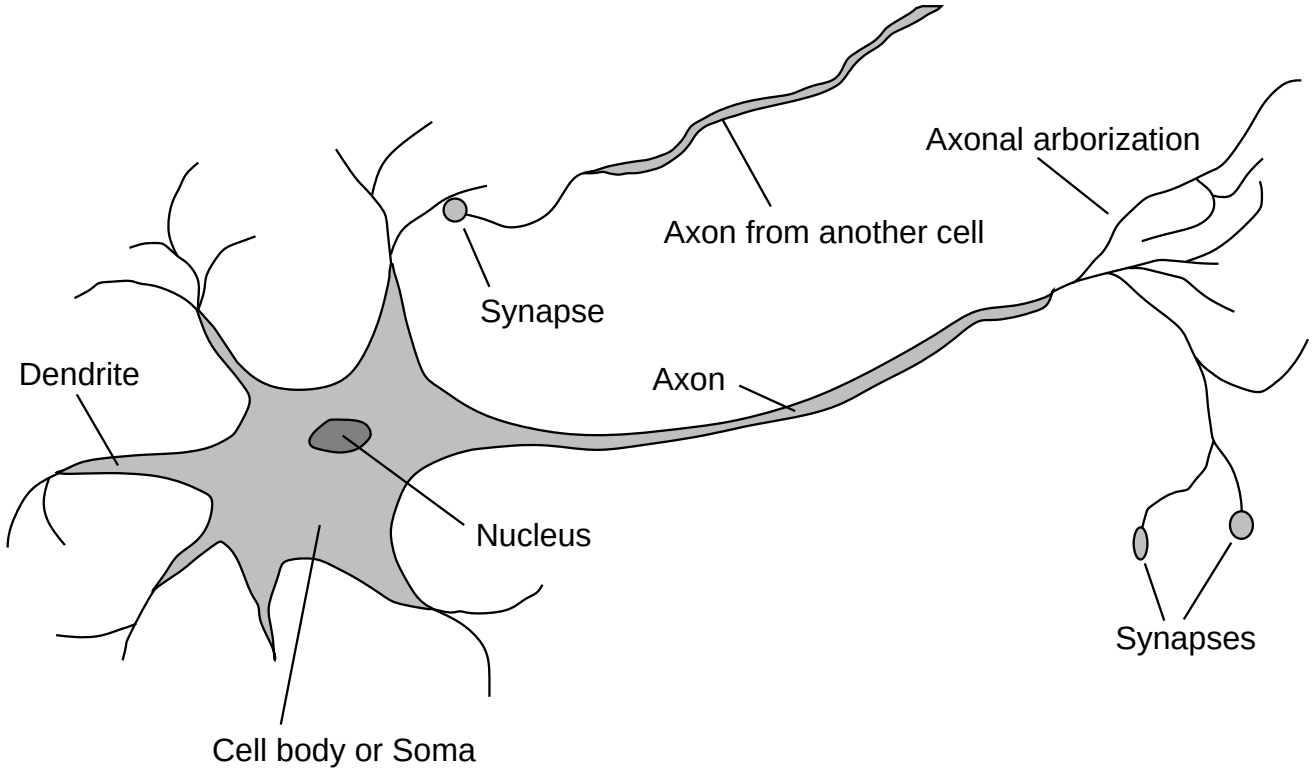
Chapter 18, Section 7

Outline

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

Brains

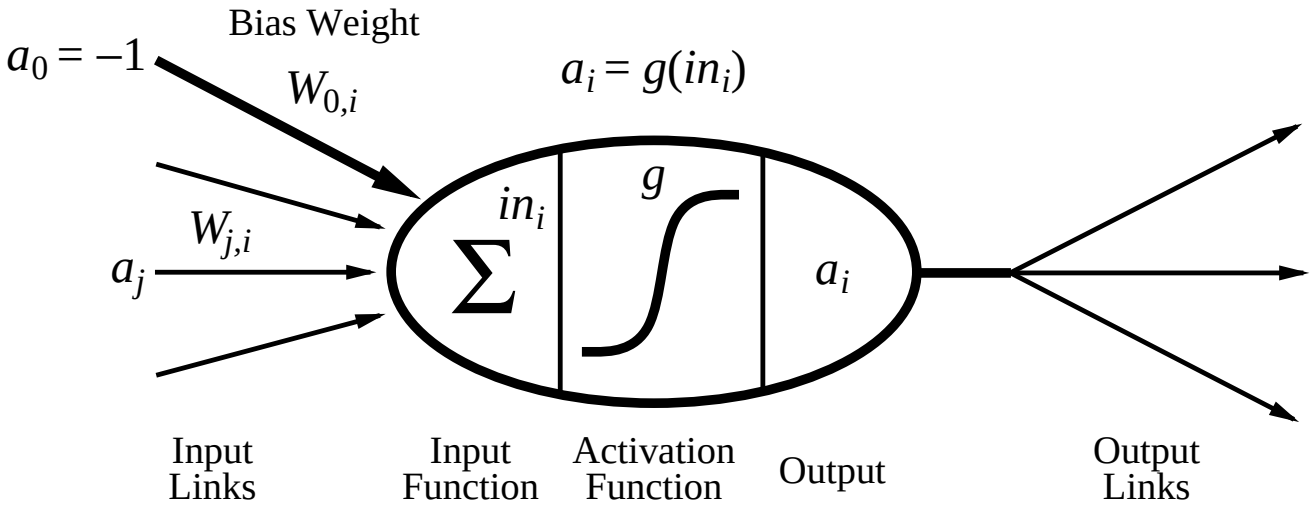
10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time
Signals are noisy "spike trains" of electrical potential



McCulloch–Pitts simplified neuron

Output is a "squashed" linear function of the inputs:

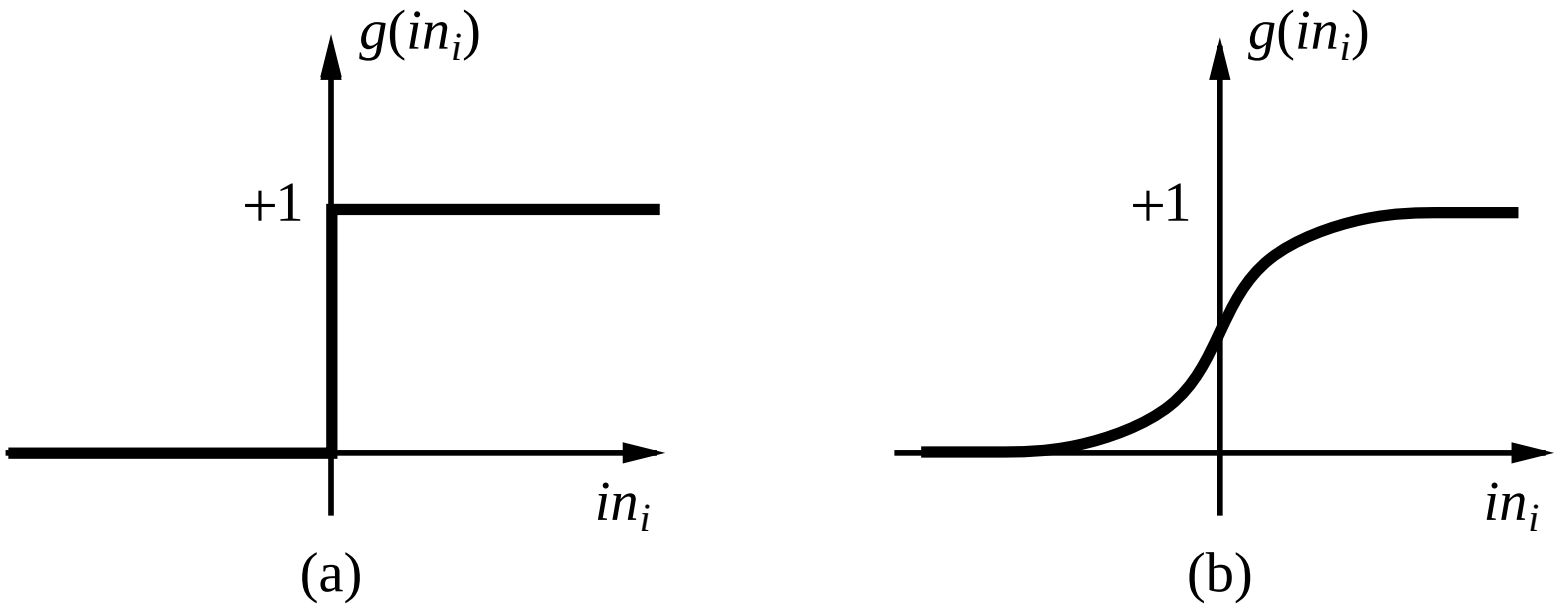
$$a_i = g(in_i) = g(\mathbf{w_i} \cdot \mathbf{a}) = g\left(\sum_j w_{j,i}\, a_j\right)$$



Note that $a_0 = -1$ is a constant input, and $w_{0,i}$ is the bias weight

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

Activation functions



(a) is a step function or threshold function, $g(x) = 1$ if $x \geq 0$, else 0

(b) is a sigmoid function, $g(x) = 1/(1 + e^{-x})$

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

Network structures

Feed-forward networks:

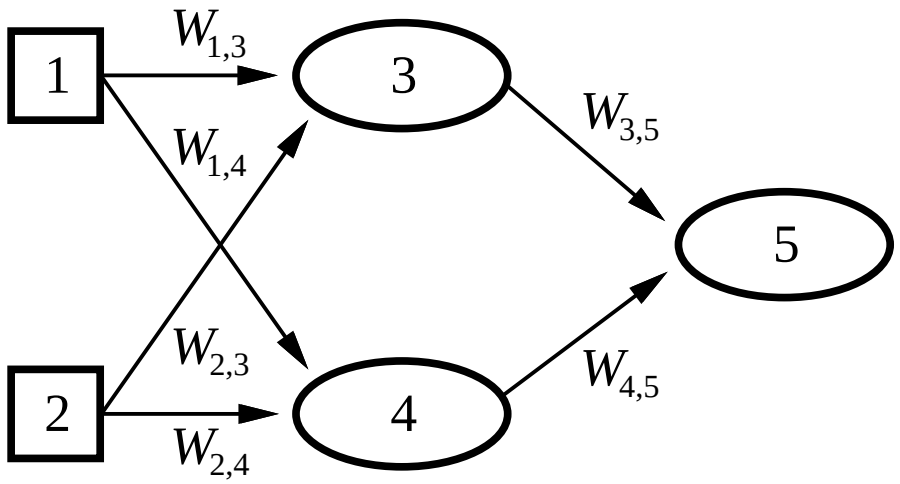
- single-layer perceptrons
- multi-layer networks

Feed-forward networks implement functions, and have no internal state

Recurrent networks have directed cycles with delays

$\Rightarrow$ they have internal state (like flip-flops), can oscillate etc.

Feed-forward example



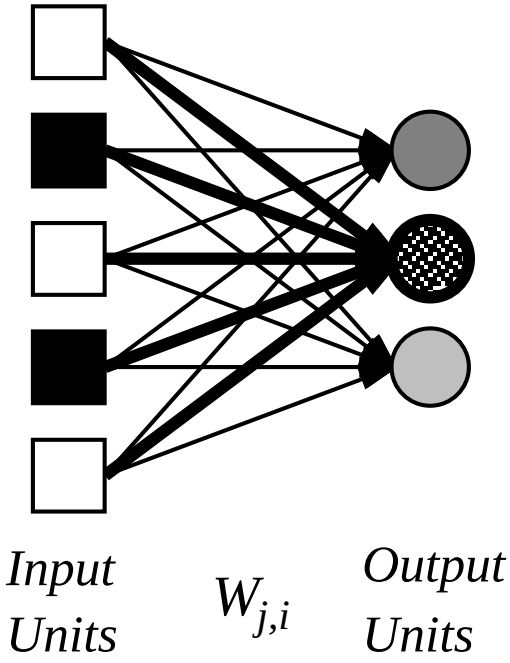
Feed-forward network = a parameterized family of nonlinear functions:

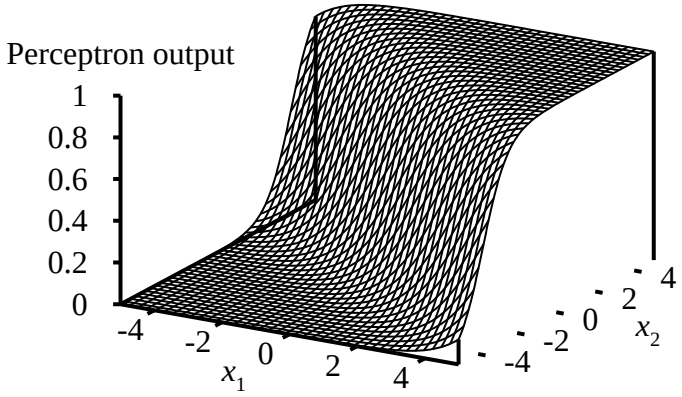
$$\begin{aligned} a_5 &= g(w_{3,5} \cdot a_3 + w_{4,5} \cdot a_4) \\ &= g(w_{3,5} \cdot g(w_{1,3} \cdot a_1 + w_{2,3} \cdot a_2) + w_{4,5} \cdot g(w_{1,4} \cdot a_1 + w_{2,4} \cdot a_2)) \end{aligned}$$

Adjusting the weights changes the function:

$\Rightarrow$ this is how we do learning!

Single-layer perceptrons





Output units all operate separately:

- there are no shared weights
- each output unit corresponds to a separate function

Adjusting weights moves the location, orientation, and steepness of the cliff

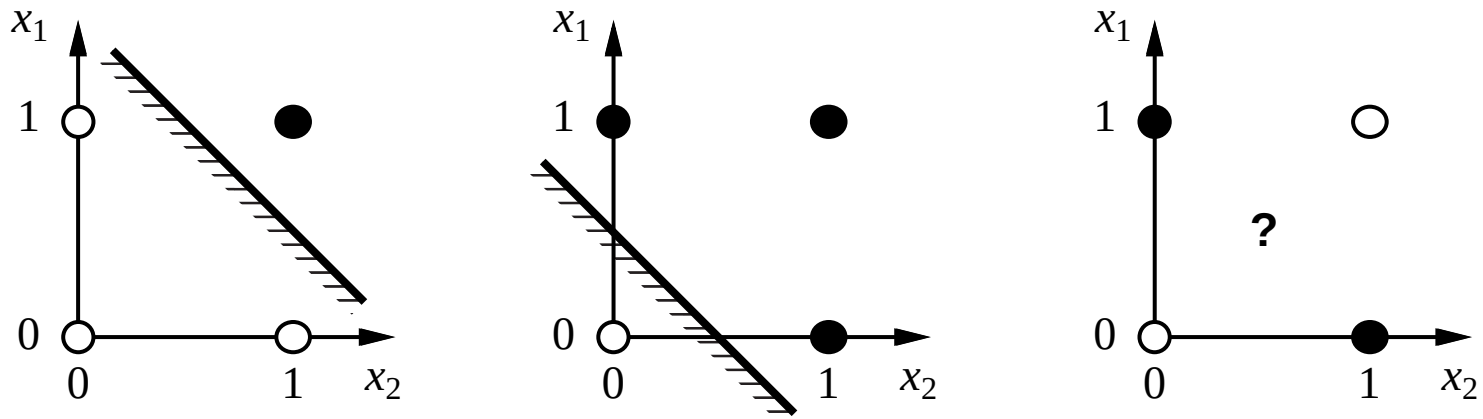
Expressiveness of perceptrons

Consider a perceptron with $g =$ the step function

Can represent AND, OR, NOT, majority, etc., but not XOR

Represents a linear separator in input space:

$$\sum_j w_j x_j > 0 \quad \text{or} \quad \mathbf{w} \cdot \mathbf{x} > 0 \quad \text{or} \quad h_{\mathbf{w}}(\mathbf{x})$$



(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

Perceptron learning

Learn by adjusting weights to reduce the error on the training set

The perceptron learning rule:

$$w_j \leftarrow w_j + \alpha(y - h)x_j$$

where $h = h_{\mathbf{w}}(\mathbf{x}) \in \{0, 1\}$ is the calculated hypothesis,
$y \in \{0, 1\}$ is the desired value, and
$0 < \alpha < 1$ is the learning rate.

Or, in other words:

- if $y = 1, h = 0$, add αx_j to w_j
- if $y = 0, h = 1$, subtract αx_j from w_j
- otherwise $y = h$, do nothing

Perceptrons = linear classifiers

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

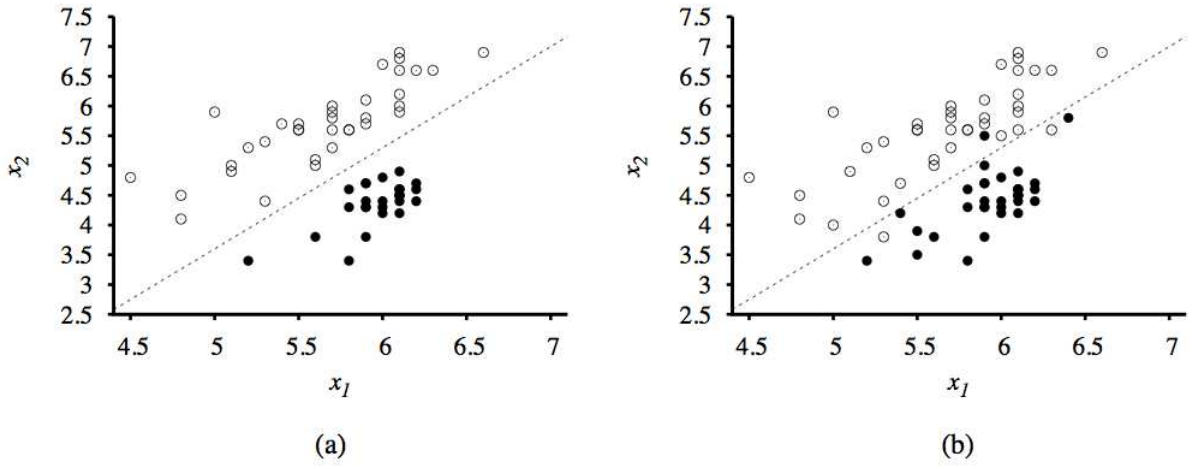


Figure 18.15 **FILES: .** (a) Plot of two seismic data parameters, body wave magnitude x_1 and surface wave magnitude x_2, for earthquakes (white circles) and nuclear explosions (black circles) occurring between 1982 and 1990 in Asia and the Middle East (?). Also shown is a decision boundary between the classes. (b) The same domain with more data points. The earthquakes and explosions are no longer linearly separable.

But what if the data set is not linearly separable?

Data that are not linearly separable

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

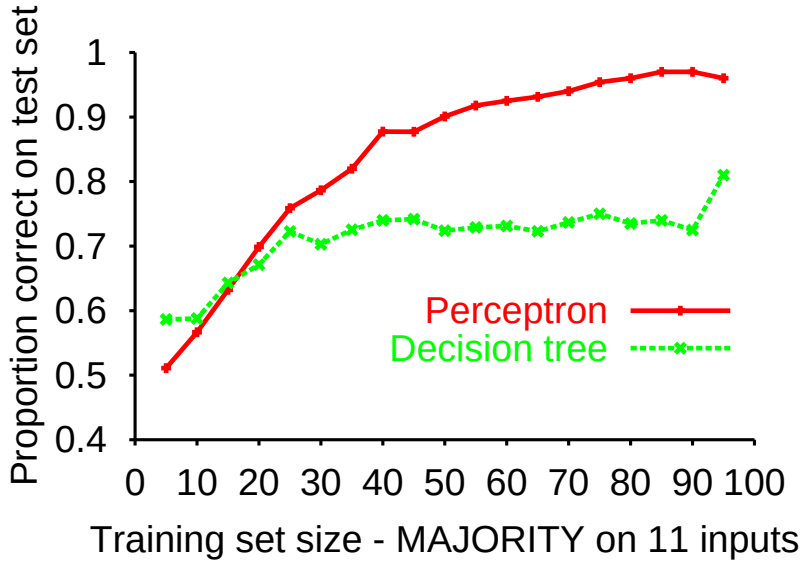
But what can we do if the data set is not linearly separable?

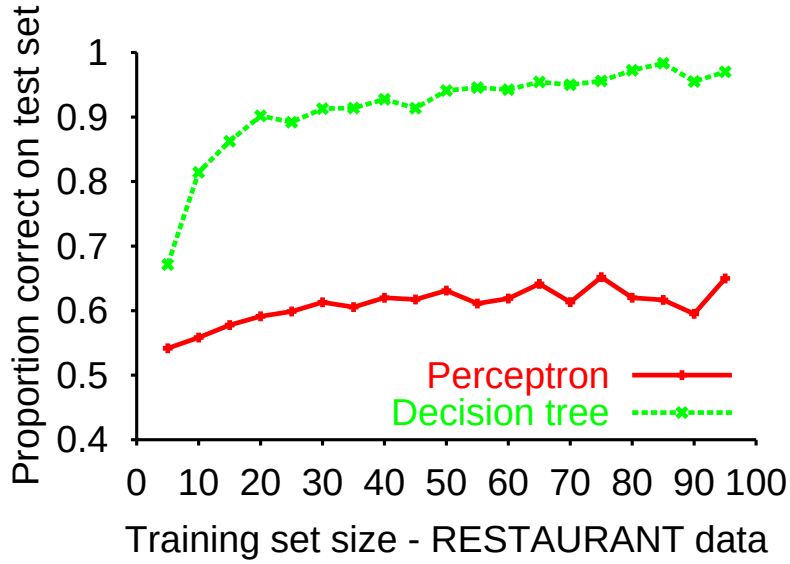
- Stop after a fixed number of iterations
- Stop when the total error does not change between iterations
- Let α decrease between iterations

Perceptrons vs decision trees

Perceptron learns the majority function easily, DTL is hopeless

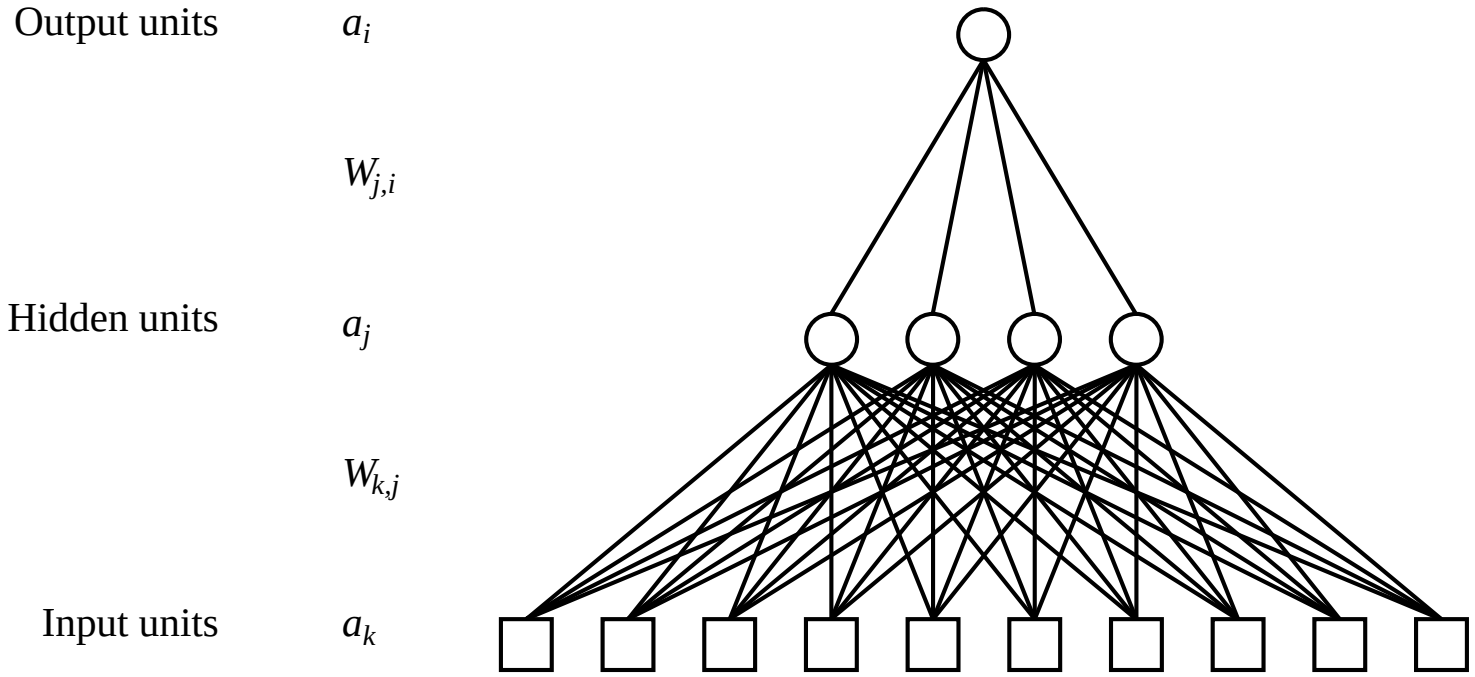
DTL learns the restaurant function easily, perceptron cannot represent it





Multilayer perceptrons

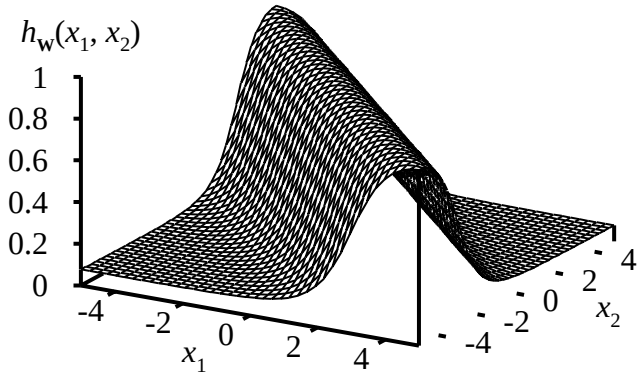
Layers are usually fully connected;
the number of hidden units are typically chosen by hand

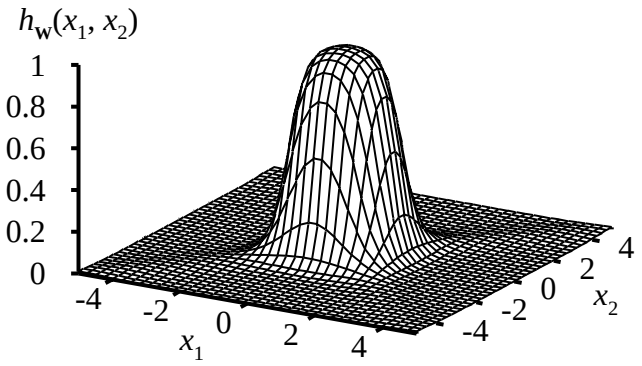


Expressiveness of MLPs

What functions can be described by MLPs?

- with 2 hidden layers: all continuous functions
- with 3 hidden layers: all functions

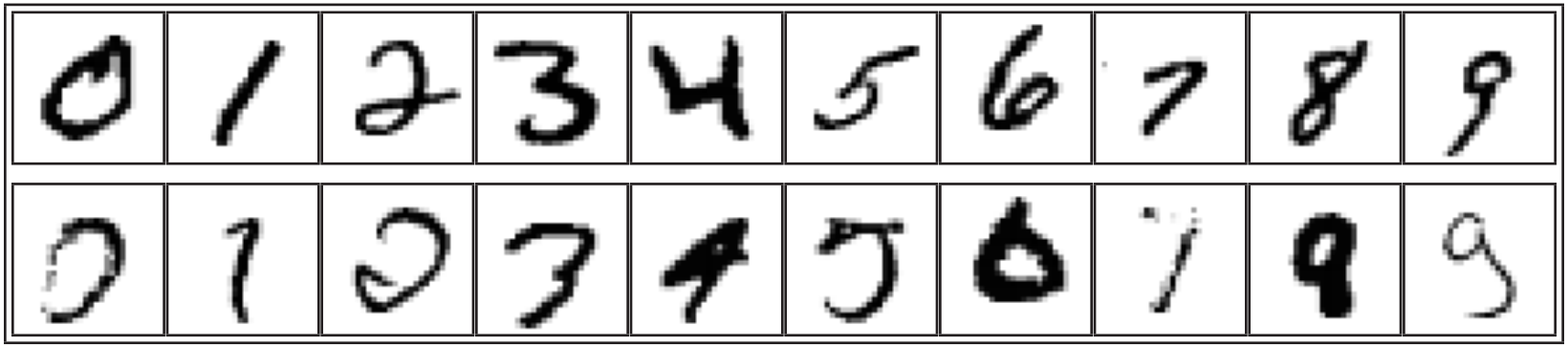




Combine two opposite-facing threshold functions to make a ridge
Combine two perpendicular ridges to make a bump
Add bumps of various sizes and locations to fit any surface
The proof requires exponentially many hidden units

Example: Handwritten digit recognition

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



3-nearest-neighbor classifier = 2.4% error
MLP (400 inputs, 300 hidden, 10 output) = 1.6% error
LeNet, an MLP specialized for image analysis = 0.9% error
SVM, without any domain knowledge = 1.1% error