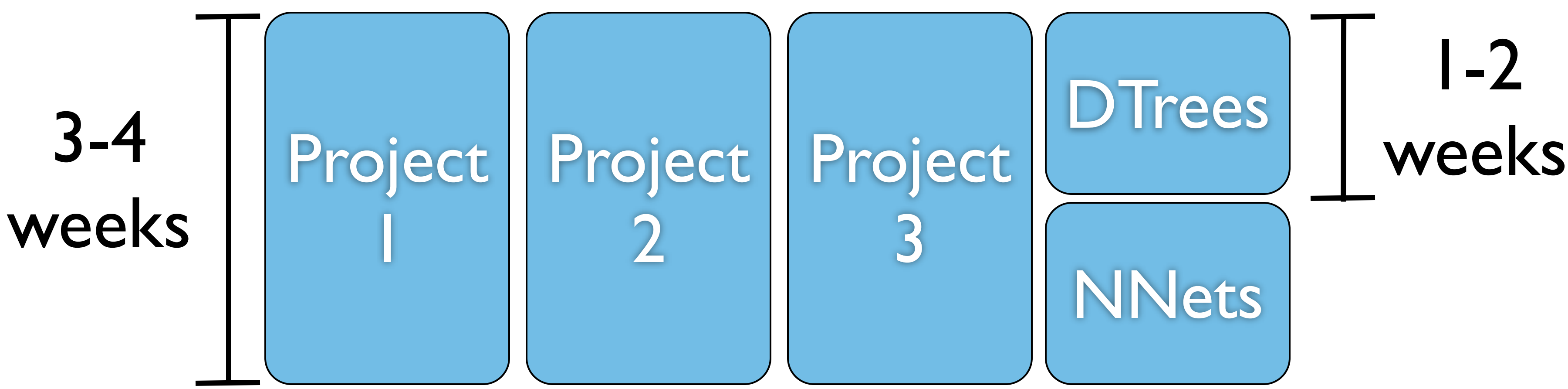
More Supervised Learning Methods

- Neural Networks
- Support Vector Machines (SVMs)
- k-Nearest Neighbors (KNN)
- Ensemble methods

Project 4: Learning

- Part A: Decision Tree Learner (due: 11/14)
- Part B: Neural Net Learner (due: 11/24)



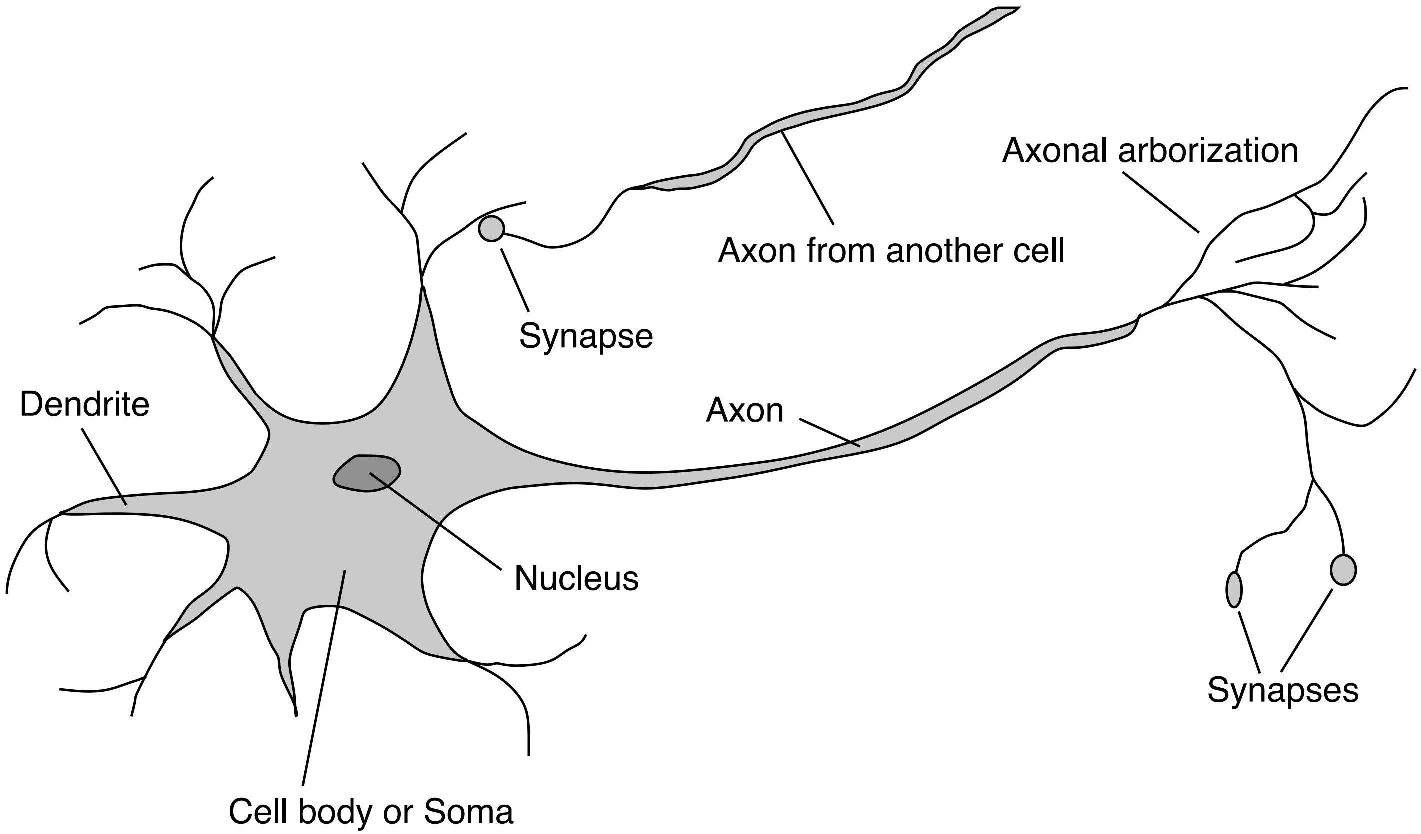
Neural Nets

Outline

- Brains
- Neural Networks
- Perceptrons
- Multilayer Perceptrons
- Applications of NNets

Brains

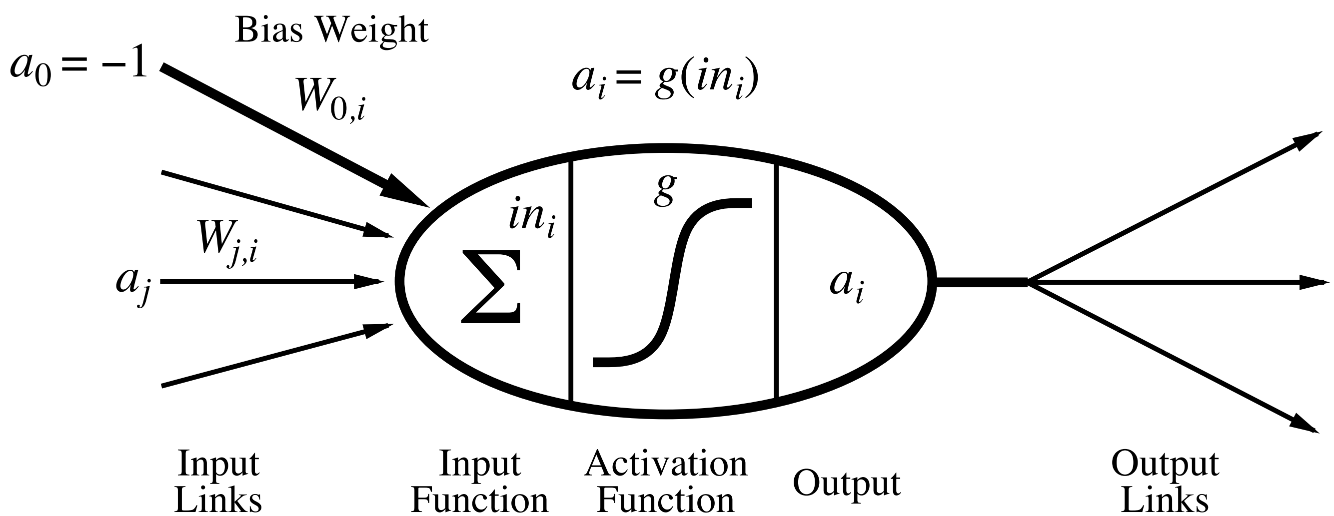
10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time
Signals are noisy "spike trains" of electrical potential



McCulloch-Pitts "unit"

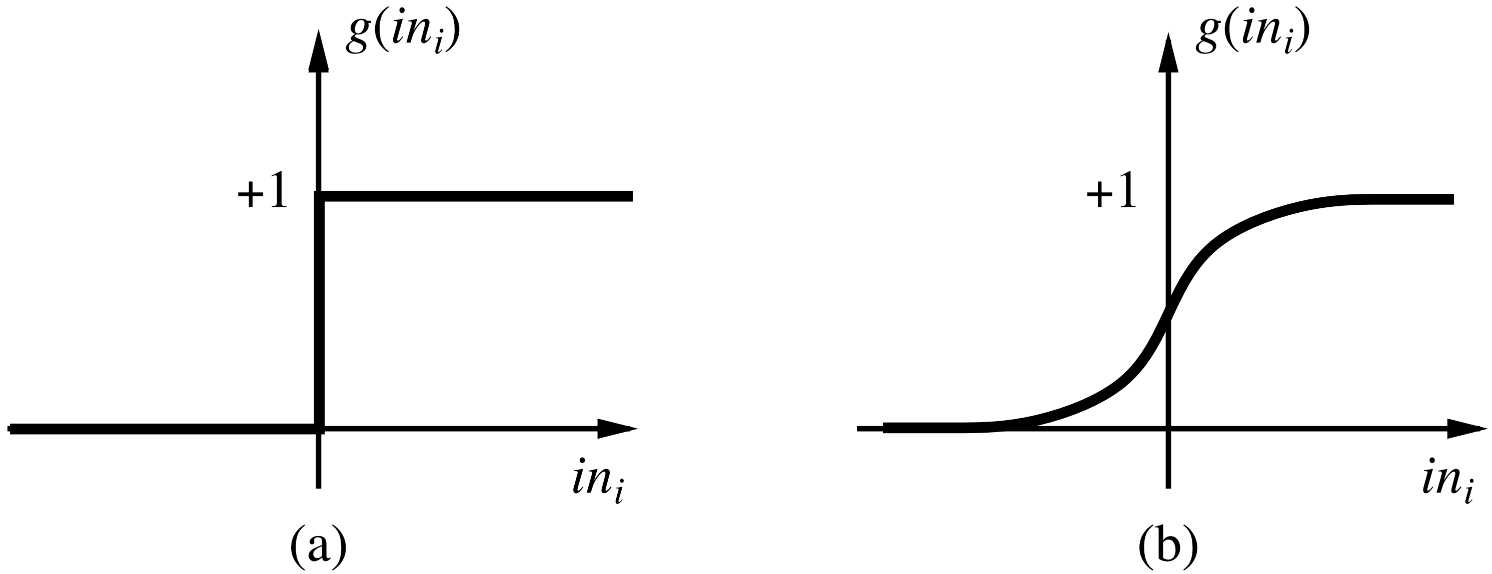
Output is a "squashed" linear function of the inputs:

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



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

Activation Functions

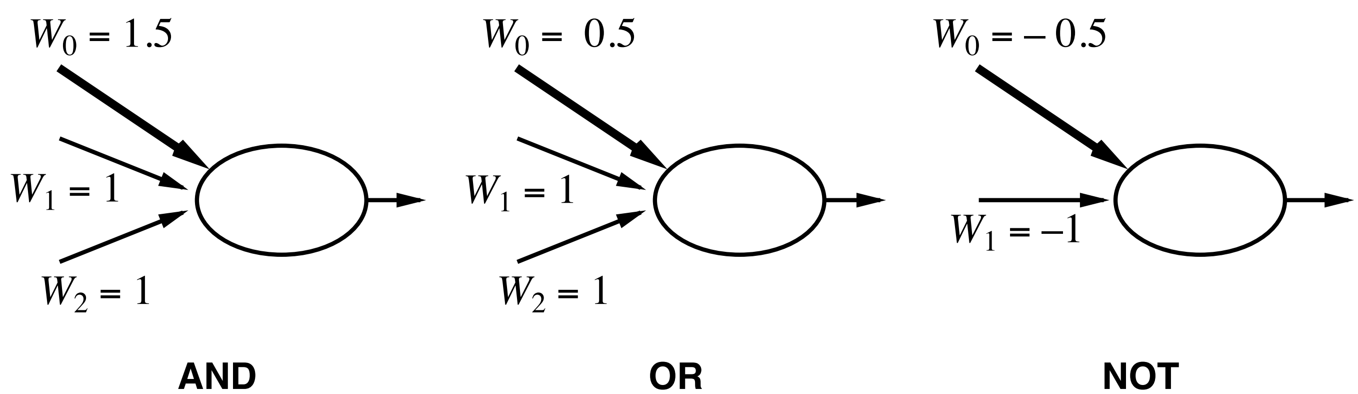


(a) is a step function or threshold function

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

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

Implementing Logic Functions



McCulloch and Pitts: every Boolean function can be implemented

Network Structures

Feed-forward networks:

– single-layer perceptrons
– multi-layer perceptrons

Feed-forward networks implement functions, have no internal state

Network Structures

Feed-forward networks:

– single-layer perceptrons

– multi-layer perceptrons

Feed-forward networks implement functions, have no internal state

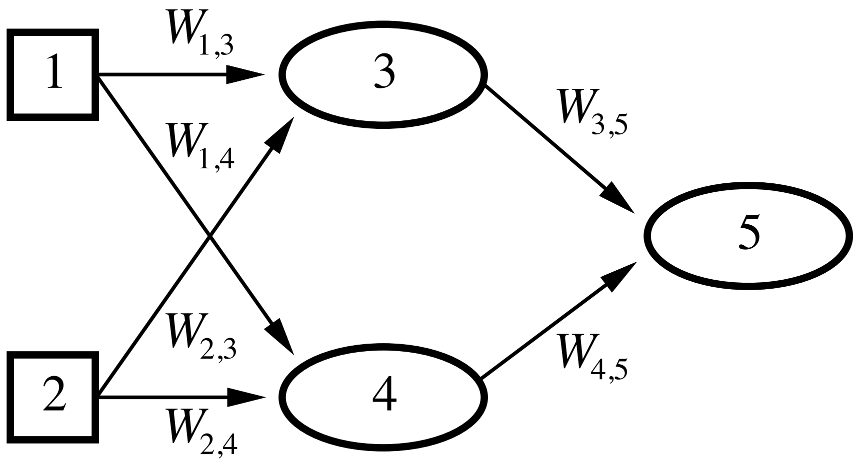
Recurrent networks:

– Hopfield networks have symmetric weights ($W_{i,j} = W_{j,i}$)
  $g(x) = \mathsf{sign}(x)$, $a_i = \pm 1$; **holographic associative memory**

– Boltzmann machines use stochastic activation functions,
  $\approx$ MCMC in Bayes nets

– recurrent neural nets have directed cycles with delays
  $\Rightarrow$ have internal state (like flip-flops), can oscillate etc.

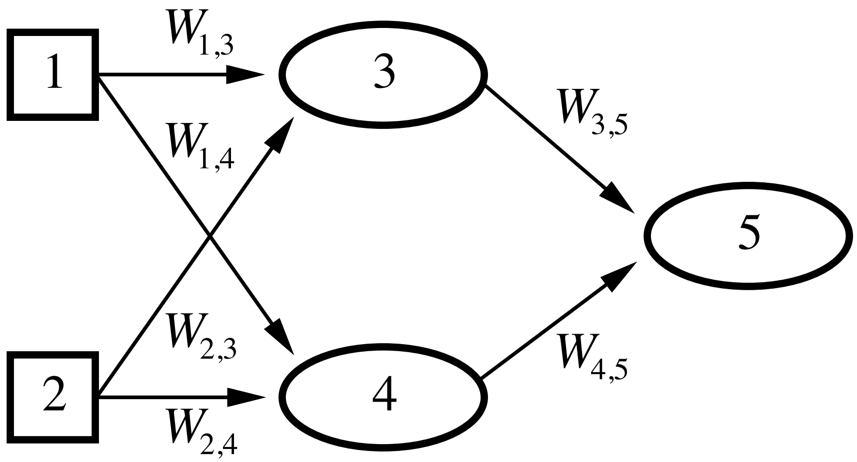
Feed-forward Example



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

$$a_5 = g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4)$$

Feed-forward Example

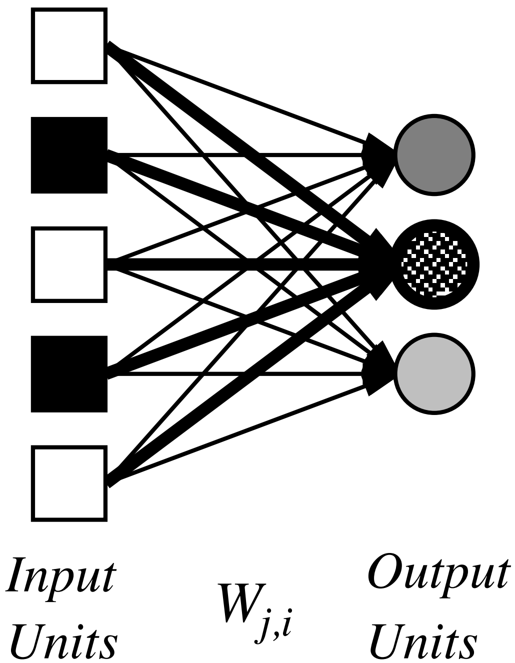


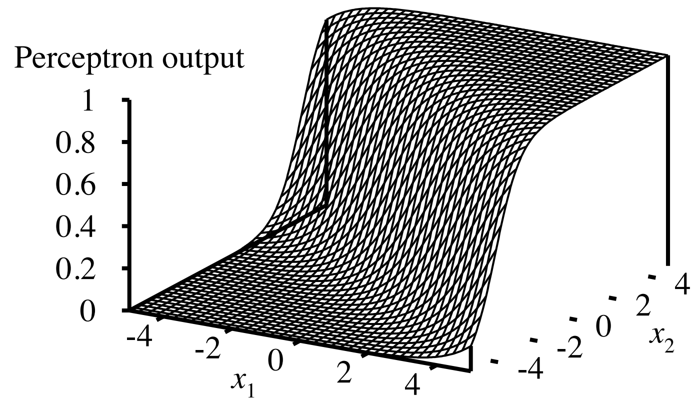
Feed-forward network = a parameterized family of nonlinear functions:

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

Adjusting weights changes the function: do learning this way!

Single-layer Perceptrons





Output units all operate separately—no shared weights

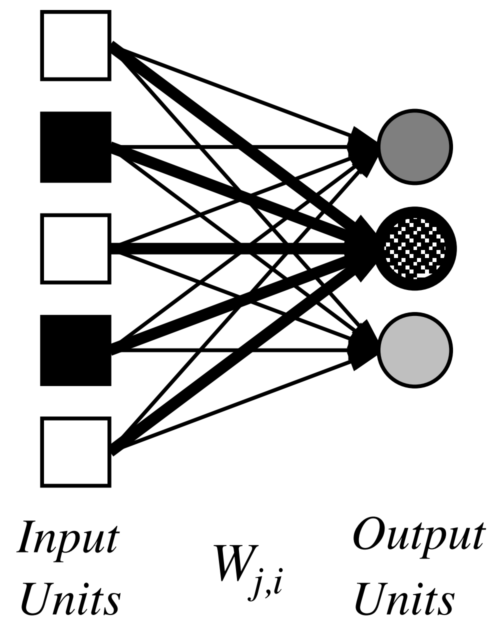
Adjusting weights moves the location, orientation, and steepness of cliff

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

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2 \ ,$$



Why is it squared?

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:

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

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

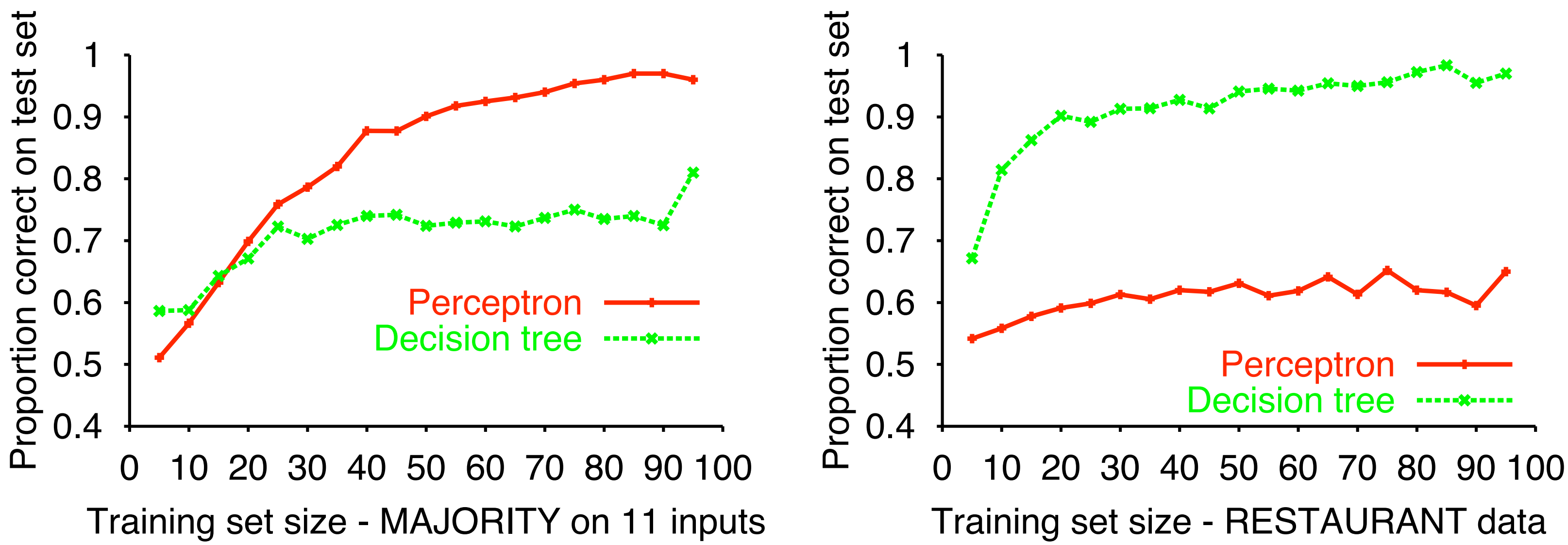
$\Rightarrow$ increase weights on +ve inputs, decrease on -ve inputs

Perceptron Learning

- Algorithm:
  - Initialize network weights to random values
  - Consider each training example 1 at a time
  - Adjust weights after each example
  - One pass through the examples is an epoch,
  - Repeat for multiple epochs until a stopping condition is met: e.g, when changes to weights become small then a local minima in the search has been reached.

Perceptron Learning

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



Perceptron learns majority function easily, DTL is hopeless

DTL learns restaurant function easily, perceptron cannot represent it

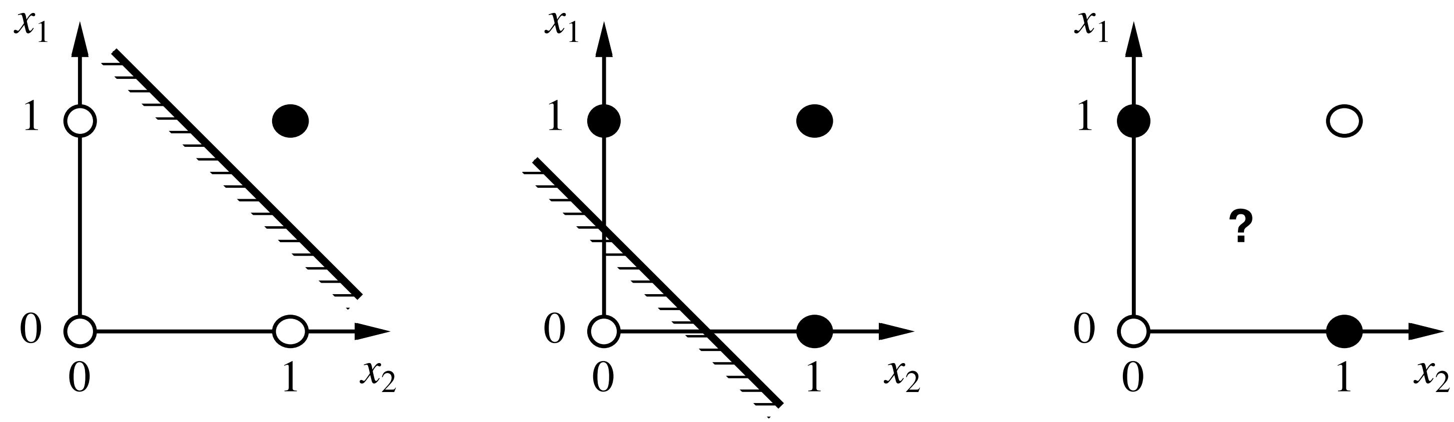
Single-layer Perceptrons

Consider a perceptron with g = step function (Rosenblatt, 1957, 1960)

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

Represents a linear separator in input space:

$\Sigma_j W_j x_j > 0$ or $\mathbf{W} \cdot \mathbf{x} > 0$

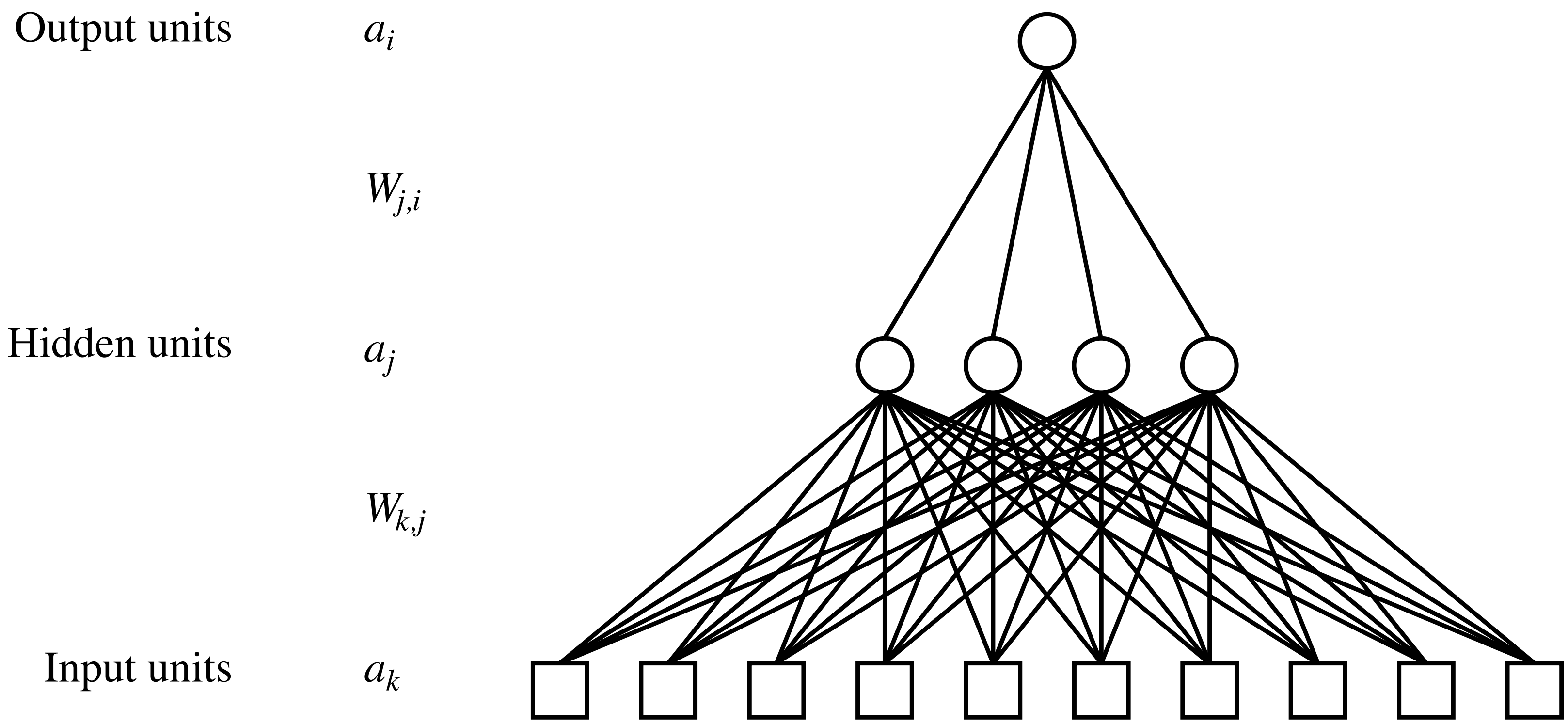


(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

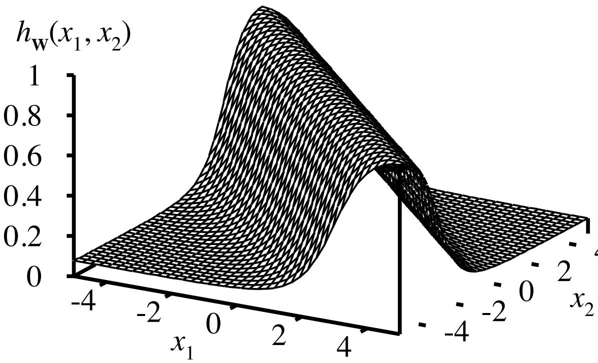
Minsky & Papert (1969) pricked the neural network balloon

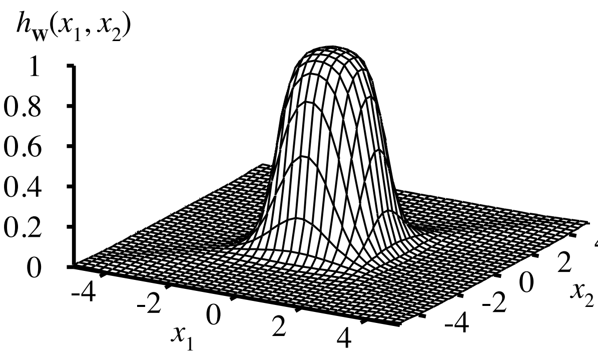
Multi-layer Perceptrons

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



Multi-layer Perceptrons





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

Proof requires exponentially many hidden units (cf DTL proof)

Learning Multi-layer NNets

- Before we just updated a single layer of weights based on the error
- Now we need to propagate the error from the 2nd layer back to the 1st layer
- Algorithm: Back-propagation

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:

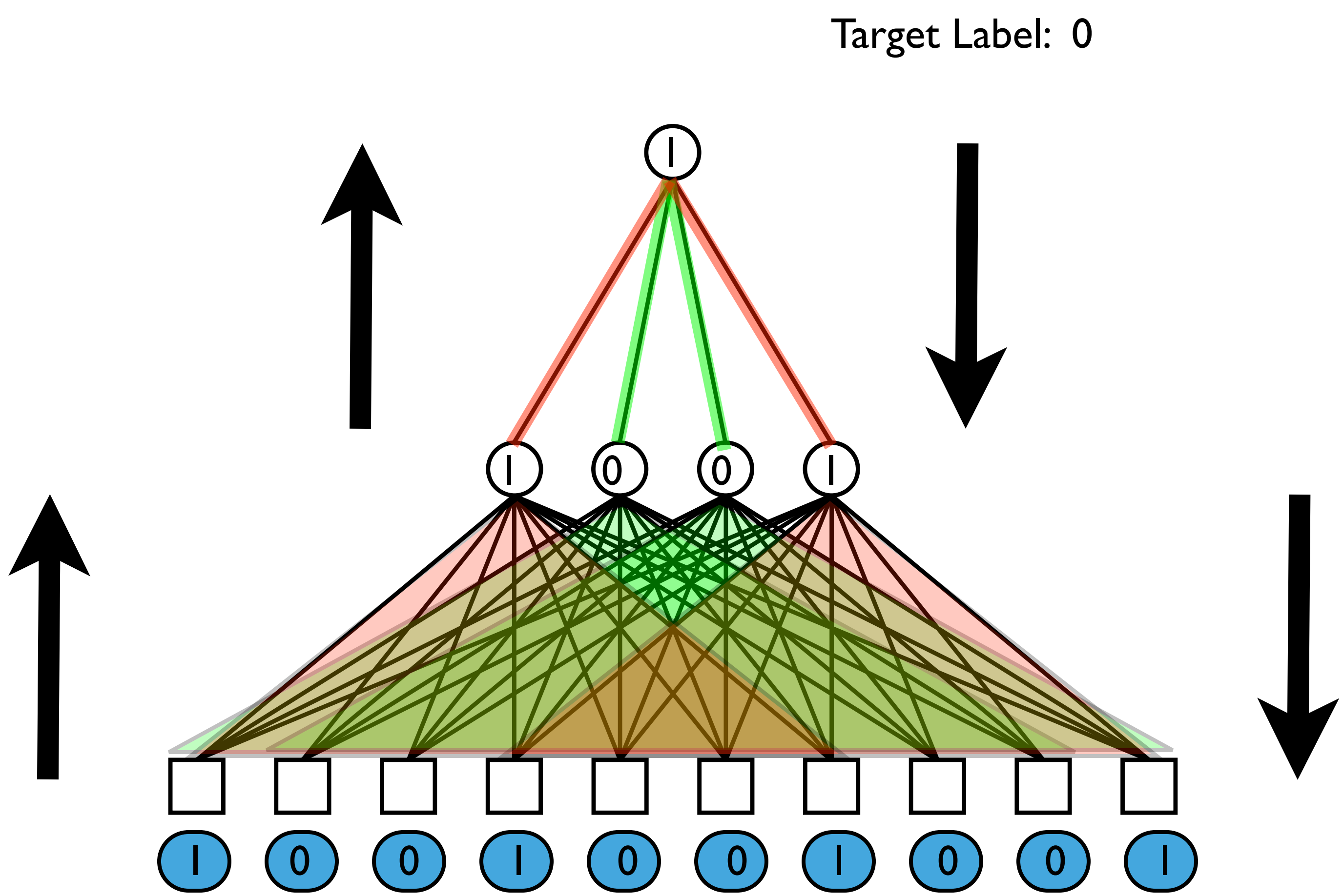
$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \ .$$

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)

Ex: Back Propagation



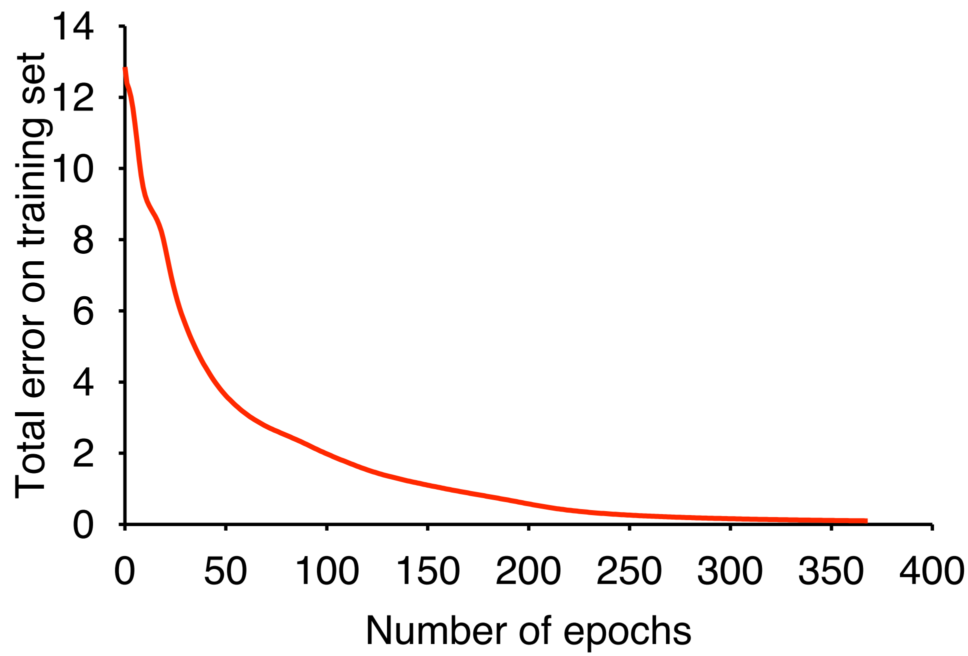
BackProp Learning

- Algorithm -- similar to Perceptron
  - Initialize network weights to random values
  - Consider each training example 1 at a time
  - Computer error at each output node, update weights
  - Propagate error back to previous layer, update weights
  - One pass through the examples is an epoch, repeat for multiple epochs until a stopping condition

Back-propagation Learning

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

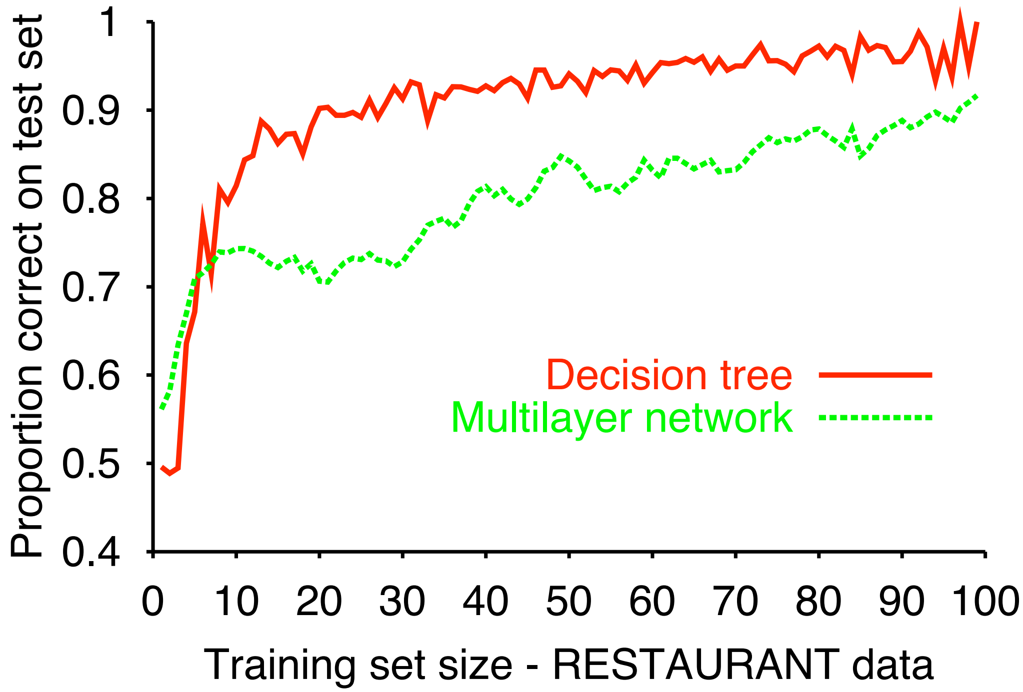
Training curve for 100 restaurant examples: finds exact fit



Typical problems: slow convergence, local minima

Back-propagation Learning

Learning curve for MLP with 4 hidden units:



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