

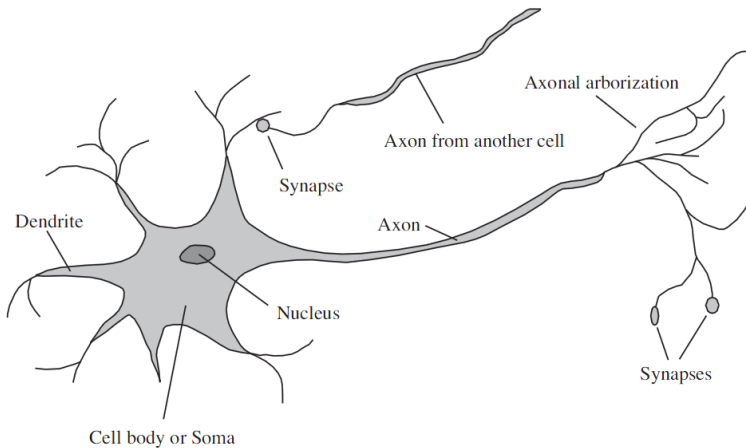
# Machine learning: Part 3

- Linear and logistic regression
- Backpropagation for neural nets

Slides based on those of Pascal Poupart

- Seat of human intelligence
- Where memory/knowledge resides
- Responsible for thoughts and decisions
- Can learn
- Consists of nerve cells called neurons

# Neuron

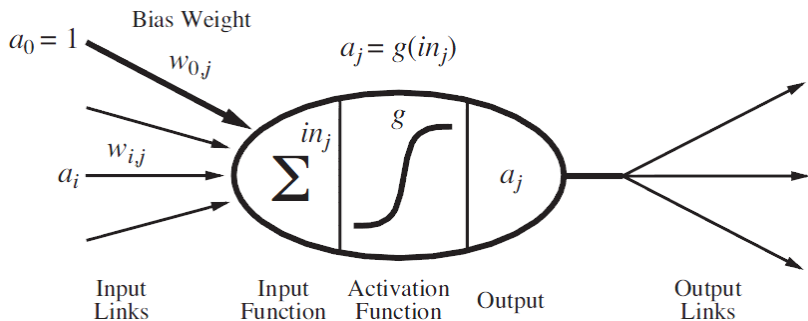


A neuron makes connections with 10 to 100,000 other neurons at junctions called synapse (突触)

# Artificial Neural Networks

- Idea: mimic the brain to do computation
- Artificial neural network:
  - Nodes (a.k.a. units) correspond to neurons
  - Links correspond to synapses
- Computation:
  - Numerical signal transmitted between nodes corresponds to chemical signals between neurons
  - Nodes modifying numerical signal correspond to neurons firing rate

# A simple mathematical model of neuron



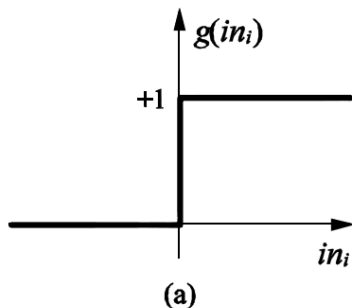
it “fires” when a linear combination of its inputs exceeds some (hard or soft) threshold

# Activation Function

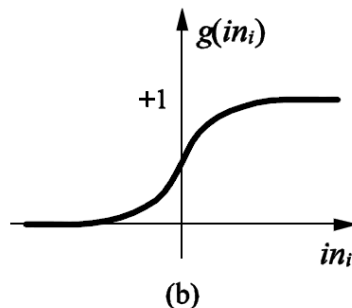
- Should be nonlinear
  - Otherwise network is just a linear function
- Often chosen to mimic firing in neurons
  - Unit should be “active” (output near 1) when fed with the “right” inputs
  - Unit should be “inactive” (output near 0) when fed with the “wrong” inputs

# Common activation functions

## Threshold

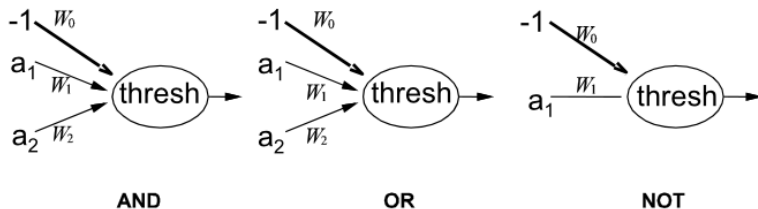


## Sigmoid



$$g(x) = 1/(1+e^{-x})$$

- McCulloch and Pitts (1943)
  - Design ANNs to represent Boolean fns
- What should be the weights of the following units to code AND, OR, NOT ?



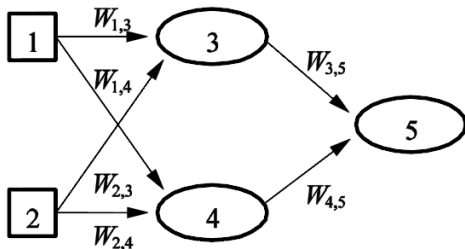


# Network structures

- Feed-forward network (前馈网络)
  - Directed acyclic graph
  - No internal state
  - Simply computes outputs from inputs
- Recurrent network (循环网络)
  - Directed cyclic graph
  - Dynamical system with internal states
  - Can memorize information

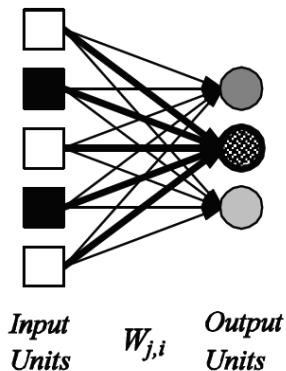
## An example

Simple network with two inputs, one hidden layer of two units, one output unit



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

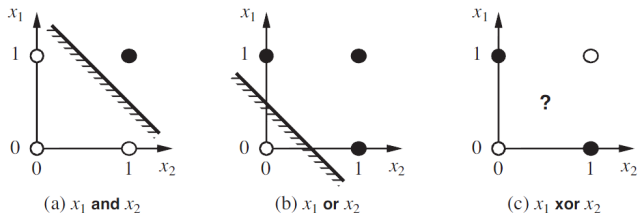
## Single layer feed-forward network



# Threshold Perceptron Hypothesis Space

- Hypothesis space  $h_w$ : All binary classifications with parameters  $w$  s.t.  $w \cdot x \geq 0 \rightarrow 1, w \cdot x < 0 \rightarrow 0$
- Since  $w \cdot x$  is linear in  $x$ , perceptron is called a linear separator (线性分离器)

# Are all Boolean gates linearly separable?

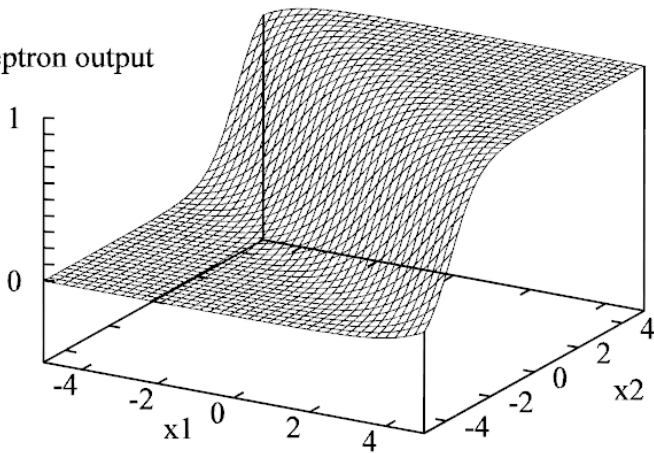


**Figure 18.21** Linear separability in threshold perceptrons. Black dots indicate a point in the input space where the value of the function is 1, and white dots indicate a point where the value is 0. The perceptron returns 1 on the region on the non-shaded side of the line. In (c), no such line exists that correctly classifies the inputs.

# Sigmoid Perceptron

Represent “soft” linear separators

Perceptron output



# Loss function

- The loss function  $L(x, y, y')$  is defined as the amount of utility lost by predicting  $h(x) = y'$  when the correct answer is  $f(x) = y$
- Often a simplified version is used,  $L(y, y')$ , that is independent of  $x$
- Three commonly used loss functions:
  - Absolute value loss:  $L_1(y, y') = |y - y'|$
  - Squared error loss:  $L_2(y, y') = (y - y')^2$
  - 0/1 loss:  $L_{0/1}(y, y') = 0$  if  $y = y'$ , else 1
- Let  $E$  be the set of examples. Total loss  $L(E) = \sum_{e \in E} L(e)$

# Linear regression

- Regression with linear functions
- As an optimization search in weight space
- Use gradient descent (梯度下降)
- An iterative method to find the minimum of a function
- Start with any initial set of weights
- In each step, decrease each weight in proportion to its partial derivative:

$$w_i \leftarrow w_i - \alpha \partial \text{Loss}(w) / \partial w_i$$

- $\alpha$  is called the learning rate



# Linear regression

- $h_w(x) = w \cdot x = \sum_i w_i x_i$
- Squared error loss:  $Loss(w) = (y - h_w(x))^2$
- Chain rule:  $\partial g(f(x))/\partial x = g'(f(x))\partial f(x)/\partial x$
- $\partial Loss(w)/\partial w_i = -2(y - h_w(x))x_i$
- $w_i \leftarrow w_i + \alpha(y - h_w(x))x_i$

# Logistic regression

- A logistic function is the sigmoid of a linear function
- Logistic regression: regression with logistic functions
- $g(x) = 1/(1 + e^{-x})$
- $h_w(x) = g(w \cdot x)$
- $g' = g(1 - g)$
- $Loss(w) = (y - h_w(x))^2$
- $\partial Loss(w)/\partial w_i = -2(y - h_w(x))g'(w \cdot x)x_i$   
 $= -2(y - h_w(x))h_w(x)(1 - h_w(x))x_i$
- $w_i \leftarrow w_i + \alpha(y - h_w(x))h_w(x)(1 - h_w(x))x_i$

# The algorithm

initialize  $w$  arbitrarily

**repeat**

  for each  $e$  in examples do

$$p \leftarrow g(w \cdot x(e))$$

$$\delta \leftarrow y(e) - p$$

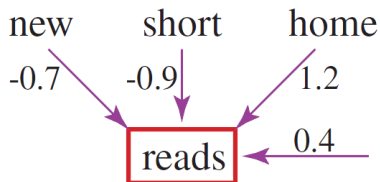
  for each  $i$  do

$$w_i \leftarrow w_i + \alpha \delta p (1 - p) x_i$$

until some stopping criterion is satisfied

return  $w$

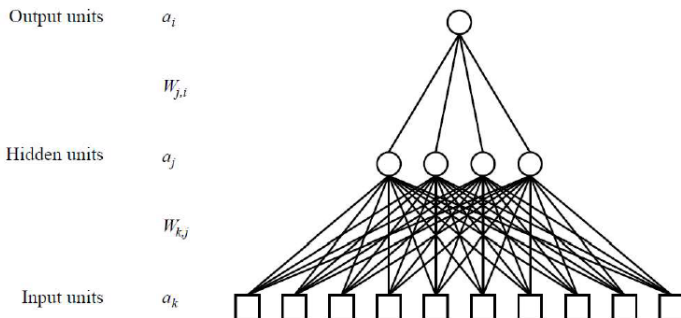
# A simple example



Ex	new	short	home	reads		$\delta$	error
				Predicted	Obs		
e1	0	0	0	$f(0.4) = 0.6$	0	-0.6	0.36
e2	1	1	0	$f(-1.2) = 0.23$	0	-0.23	0.053
e3	1	0	1	$f(0.9) = 0.71$	1	0.29	0.084

# Multilayer Feed-forward Neural Networks

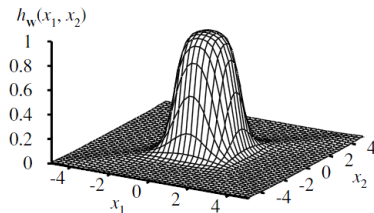
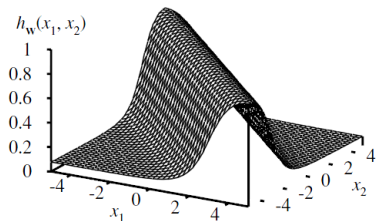
- Perceptron can only represent (soft) linear separators
- With multiple layers, what functions can be represented?  
Virtually any function!



$$a_i = g\left(\sum_j W_{ji} g\left(\sum_k W_{kj} a_k\right)\right)$$

# Multilayer networks

- Adding two sigmoid units with parallel but opposite “cliffs” produces a ridge
- Adding two intersecting ridges (and thresholding) produces a bump



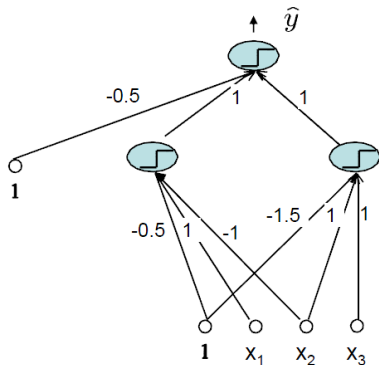
# Multilayer networks

By tiling bumps of various heights together, we can approximate any function

**Theorem:** Neural networks with at least one hidden layer of sufficiently many sigmoid units can approximate any function arbitrarily closely.

# Neural nets for Boolean functions: an example

Boolean function:  $x_1 \wedge \neg x_2 \vee x_2 \wedge x_3$



OR units

AND units

Any Boolean function can be put into CNF or DNF, and hence can be represented with a neural network with one hidden layer



# Weight training

- A set of examples, each with input vector  $x$  and output vector  $y$
- Squared error loss:  $Loss = \sum_k Loss_k$ ,  $Loss_k = (y_k - a_k)^2$ , where  $a_k$  is the  $k$ -th output of the neural net
- The weights are adjusted as follows:  
$$w_{ij} \leftarrow w_{ij} - \alpha \partial Loss / \partial w_{ij}$$
- How can we compute the gradient efficiently given an arbitrary network structure?
- Answer: backpropagation algorithm

# Forward and backward phases

Forward phase:

- Propagate inputs forward to compute the output of each unit
- Output  $a_j$  at unit  $j$ :  $a_j = g(in_j)$  where  $in_j = \sum_i w_{ij}a_i$

Backward phase:

- Propagate errors backward
- For an output unit  $j$ :  $\Delta_j = g'(in_j)(y_j - a_j)$
- For an hidden unit  $i$ :  $\Delta_i = g'(in_i) \sum_j w_{ij} \Delta_j$

```

function BACK-PROP-LEARNING(examples, network) returns a neural network
  inputs: examples, a set of examples, each with input vector  $\mathbf{x}$  and output vector  $\mathbf{y}$ 
           network, a multilayer network with  $L$  layers, weights  $w_{i,j}$ , activation function  $g$ 
  local variables:  $\Delta$ , a vector of errors, indexed by network node

  repeat
    for each weight  $w_{i,j}$  in network do
       $w_{i,j} \leftarrow$  a small random number
    for each example  $(\mathbf{x}, \mathbf{y})$  in examples do
      /* Propagate the inputs forward to compute the outputs */
      for each node  $i$  in the input layer do
         $a_i \leftarrow x_i$ 
      for  $\ell = 2$  to  $L$  do
        for each node  $j$  in layer  $\ell$  do
           $in_j \leftarrow \sum_i w_{i,j} a_i$ 
           $a_j \leftarrow g(in_j)$ 
      /* Propagate deltas backward from output layer to input layer */
      for each node  $j$  in the output layer do
         $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$ 
      for  $\ell = L - 1$  to  $1$  do
        for each node  $i$  in layer  $\ell$  do
           $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$ 
      /* Update every weight in network using deltas */
      for each weight  $w_{i,j}$  in network do
         $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$ 
  until some stopping criterion is satisfied
  return network

```

# Output layer

$$\begin{aligned}\frac{\partial Loss_k}{\partial w_{j,k}} &= -2(y_k - a_k) \frac{\partial a_k}{\partial w_{j,k}} = -2(y_k - a_k) \frac{\partial g(in_k)}{\partial w_{j,k}} \\ &= -2(y_k - a_k) g'(in_k) \frac{\partial in_k}{\partial w_{j,k}} = -2(y_k - a_k) g'(in_k) \frac{\partial}{\partial w_{j,k}} \left( \sum_j w_{j,k} a_j \right) \\ &= -2(y_k - a_k) g'(in_k) a_j = -a_j \Delta_k ,\end{aligned}$$

# Hidden layers

$$\begin{aligned}\frac{\partial Loss_k}{\partial w_{i,j}} &= -2(y_k - a_k) \frac{\partial a_k}{\partial w_{i,j}} = -2(y_k - a_k) \frac{\partial g(in_k)}{\partial w_{i,j}} \\&= -2(y_k - a_k) g'(in_k) \frac{\partial in_k}{\partial w_{i,j}} = -2\Delta_k \frac{\partial}{\partial w_{i,j}} \left( \sum_j w_{j,k} a_j \right) \\&= -2\Delta_k w_{j,k} \frac{\partial a_j}{\partial w_{i,j}} = -2\Delta_k w_{j,k} \frac{\partial g(in_j)}{\partial w_{i,j}} \\&= -2\Delta_k w_{j,k} g'(in_j) \frac{\partial in_j}{\partial w_{i,j}} \\&= -2\Delta_k w_{j,k} g'(in_j) \frac{\partial}{\partial w_{i,j}} \left( \sum_i w_{i,j} a_i \right) \\&= -2\Delta_k w_{j,k} g'(in_j) a_i = -a_i \Delta_j ,\end{aligned}$$

# Forward and backward phases

Forward phase:

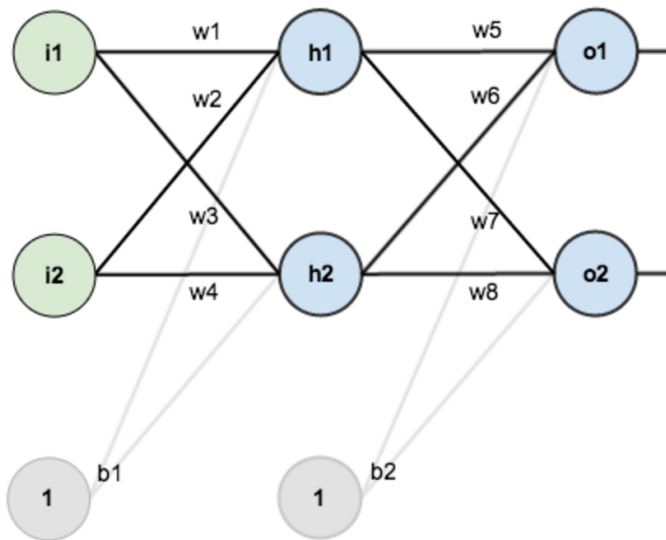
- Propagate inputs forward to compute the output of each unit
- Output  $a_j$  at unit  $j$ :  $a_j = g(in_j)$  where  $in_j = \sum_i w_{ij}a_i$

Backward phase:

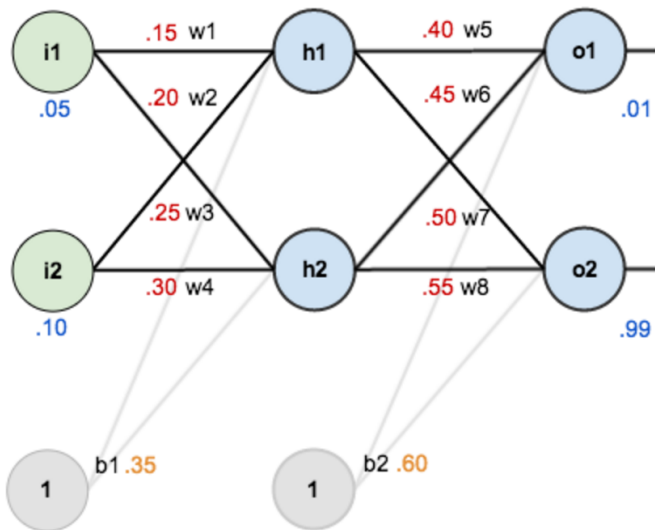
- Propagate errors backward
- For an output unit  $j$ :  
$$\Delta_j = g'(in_j)(y_j - a_j) = a_j(1 - a_j)(y_j - a_j)$$
- For an hidden unit  $i$ :  
$$\Delta_i = g'(in_i) \sum_j w_{ij} \Delta_j = a_i(1 - a_i) \sum_j w_{ij} \Delta_j$$

Weight updating:  $w_{ij} \leftarrow w_{ij} + \alpha a_i \delta_j$

# The network structure



# The numbers





# Forward pass

- $in_{h_1} = w_1 i_1 + w_2 i_2 + b_1 = 0.05 * 0.15 + 0.10 * 0.20 + 0.35 = 0.3775$
- $out_{h_1} = g(in_{h_1}) = \frac{1}{1+e^{-0.3775}} = 0.593269992$
- $out_{h_2} = 0.596884378$
- $in_{o_1} = w_5 out_{h_1} + w_6 out_{h_2} + b_2 = 0.40 * 0.593269992 + 0.45 * 0.596884378 + 0.60 = 1.105905967$
- $out_{o_1} = g(in_{o_1}) = \frac{1}{1+e^{-1.105905967}} = 0.75136507$
- $out_{o_2} = 0.772928465$

# Backward pass

Let  $\alpha = 0.5$

- $\Delta_{o_1} = 0.75136507(1 - 0.75136507)(0.01 - 0.75136507) = -0.138498562$
- $w_5^+ = w_5 + \alpha \cdot out_{h_1} \cdot \Delta_{o_1} = 0.40 - 0.5 * 0.593269992 * 0.138498562 = 0.35891648$
- $w_6^+ = w_6 + \alpha \cdot out_{h_2} \cdot \Delta_{o_1} = 0.45 - 0.5 * 0.596884378 * 0.138498562 = 0.408666186$

# Backward pass

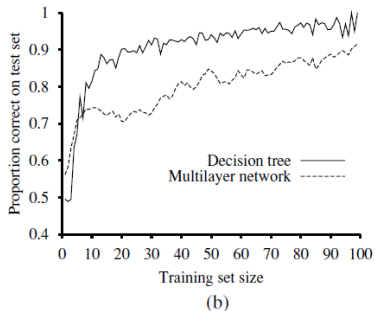
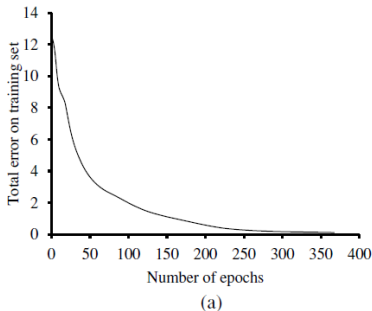
- $\Delta_{o_2} = 0.772928465(1 - 0.772928465)(0.99 - 0.772928465) = 0.0380982366$
- $w_7^+ = w_7 + \alpha \cdot out_{h_1} \cdot \Delta_{o_2} = 0.50 + 0.5 * 0.593269992 * 0.0380982366 = 0.511301270$
- $w_8^+ = w_8 + \alpha \cdot out_{h_2} \cdot \Delta_{o_2} = 0.55 + 0.5 * 0.596884378 * 0.0380982366 = 0.561370121$

# Backward pass

- $\Delta_{h_1} = g'(in_{h_1})(w_5\Delta_{o_1} + w_7\Delta_{o_2}) =$   
 $0.593269992(1 - 0.593269992)(0.40 * (-0.138498562) +$   
 $0.50 * 0.0380982366) = -0.241300709 * 0.036350306$
- $w_1^+ = w_1 + \alpha \cdot i_1 \cdot \Delta_{h_1} =$   
 $0.15 - 0.5 * 0.05 * 0.241300709 * 0.036350306 = 0.149780716$
- $w_2^+ = 0.19956143$
- $w_3^+ = 0.24975114$
- $w_4^+ = 0.29950229$

# The restaurant example

- First, we need to determine the structure of the network.
- We have 10 attributes, so we will need 10 input units.
- Should we have one hidden layer or two? How many nodes in each layer? Should they be fully connected?
- There is no good theory that will tell us the answer.
- We can use cross-validation: try several different structures and see which one works best.
- It turns out that a network with one hidden layer containing four nodes is about right for this problem.



**Figure 18.25** (a) Training curve showing the gradual reduction in error as weights are modified over several epochs, for a given set of examples in the restaurant domain. (b) Comparative learning curves showing that decision-tree learning does slightly better on the restaurant problem than back-propagation in a multilayer network.