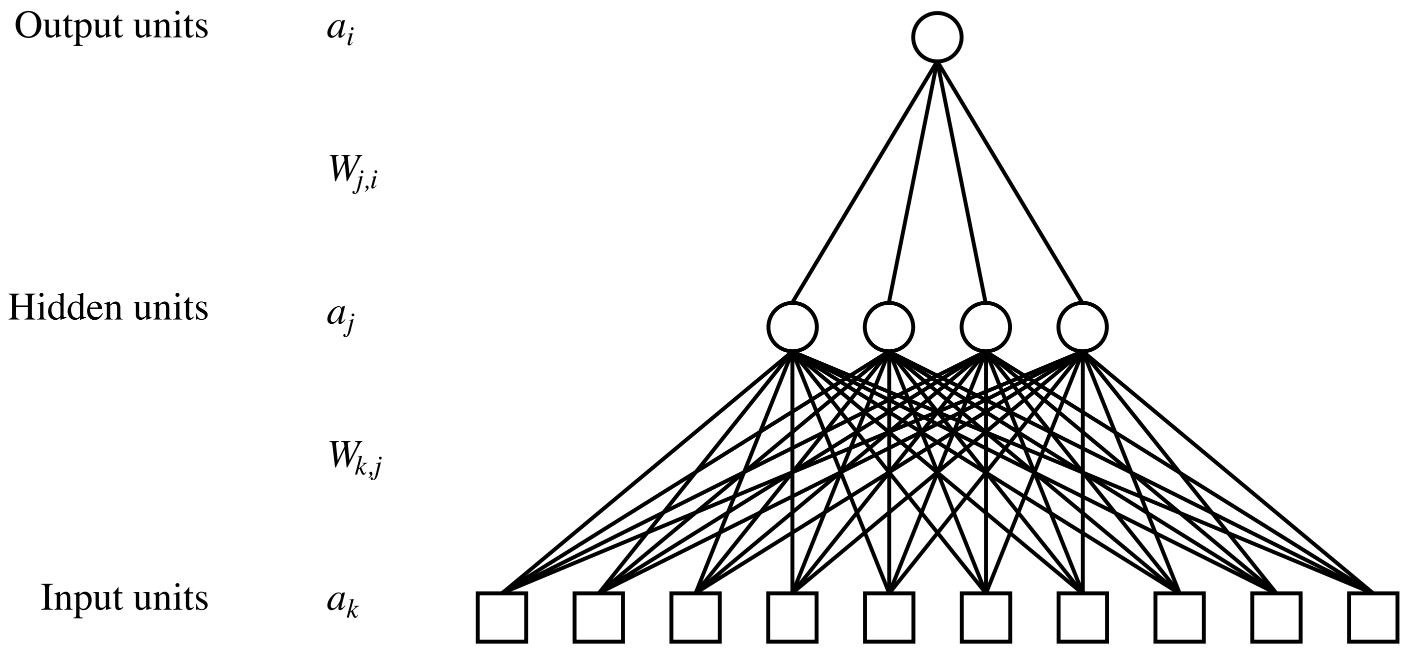
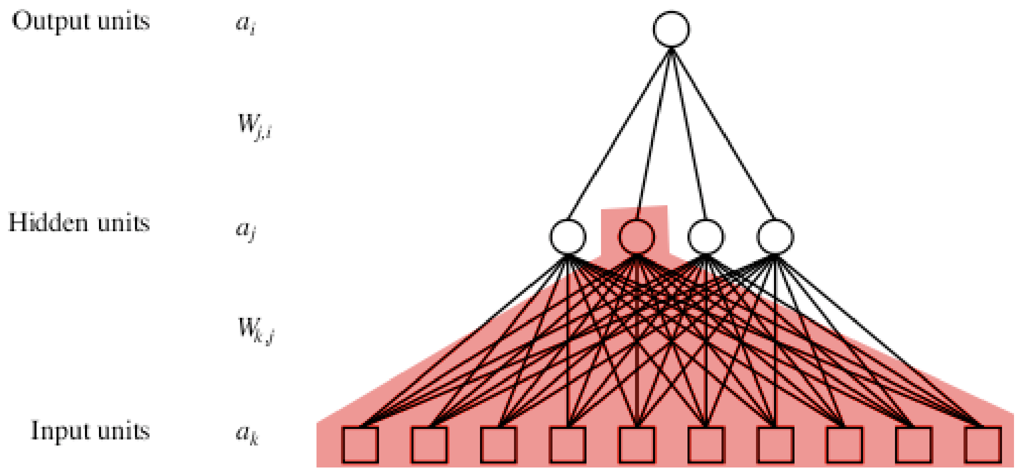
Multi-Layer Feed-Forward Neural Networks

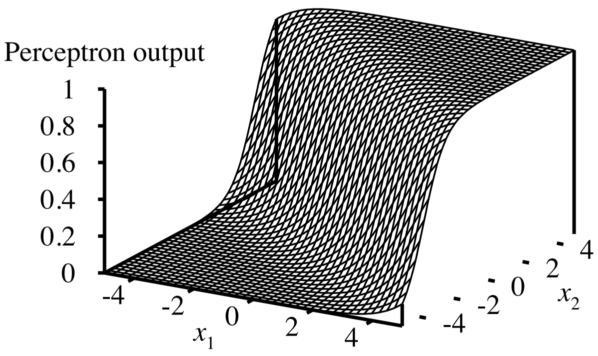
The most common case involves a single hidden layer:



Multi-Layer Feed-Forward Neural Networks

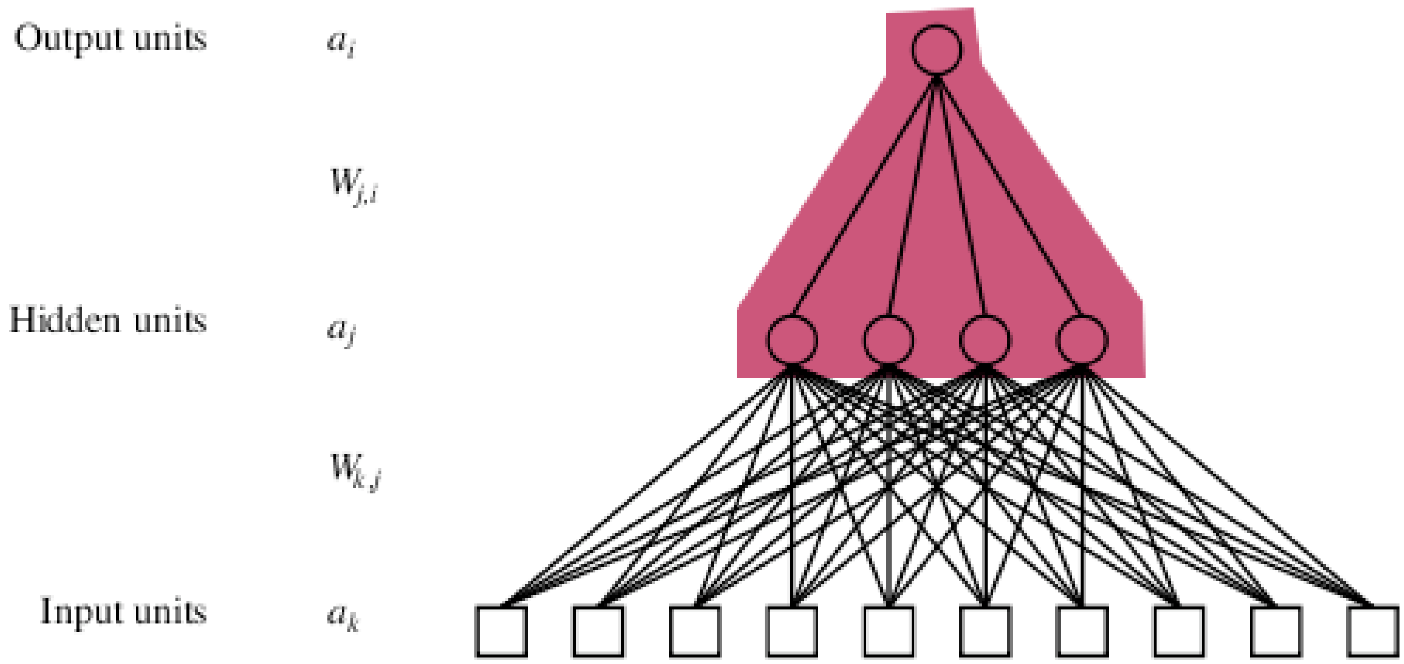
Each *hidden unit* can be considered as single output perceptron network:





Multi-Layer Feed-Forward Neural Networks

The output unit of the multi-layer network can then be considered a soft-thresholded linear combination of the hidden units (which are equivalent to single output unit perceptrons):



Multi-Layer Feed-Forward Neural Networks

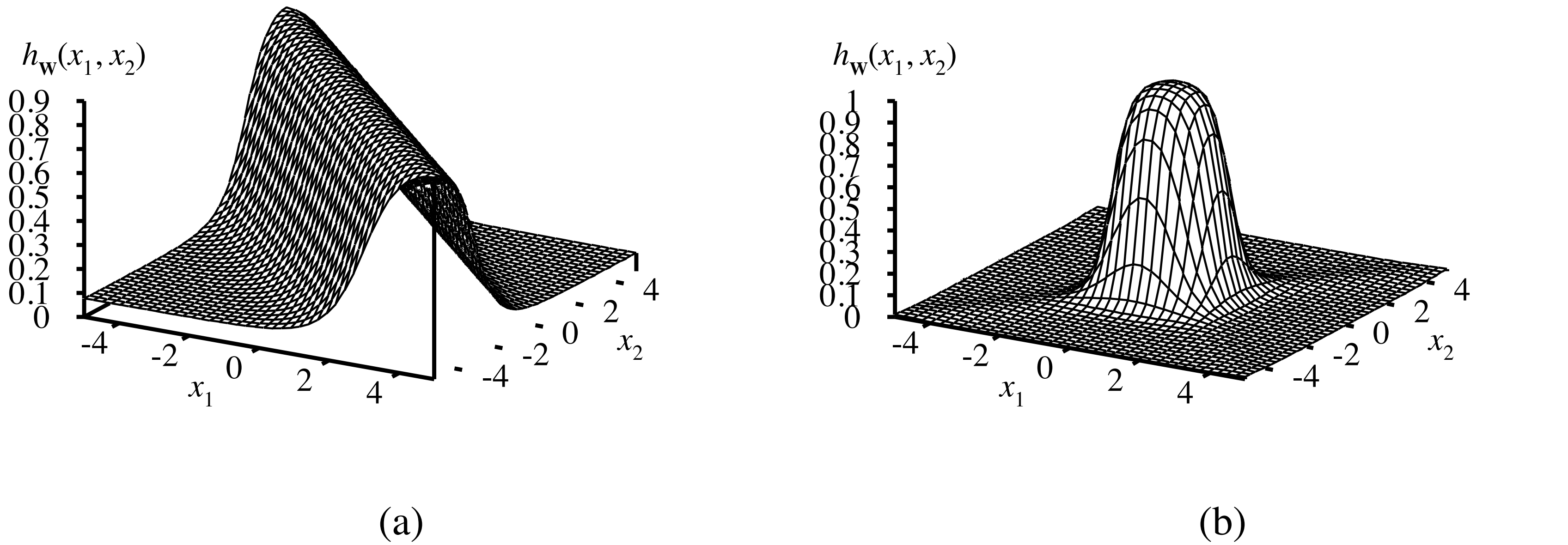


Figure 20.23 (a) The result of combining two opposite-facing soft threshold functions to produce a ridge. (b) The result of combining two ridges to produce a bump.

Expressibility of Multi-Layer Neural Networks

- With a single, sufficiently large hidden layer, it is possible to represent *any continuous* function of the inputs with *arbitrary* accuracy.
- Unfortunately, for any *particular* network, it is harder to characterize exactly which functions can be represented and which ones cannot.
- As a consequence, given a particular learning problem, it is unknown how to choose the *right number of hidden units* in advance.
- One usually resorts to cross validation, but this can be computationally expensive for large networks.

Back-Propagation Learning

- We need to consider multiple output units for multi-layer networks. Let $(\mathbf{x}, \mathbf{y})$ be a single sample with its desired output labels $\mathbf{y} = \{y_1, \ldots, y_i, \ldots, y_M\}$.
- The error at the output units is just $\mathbf{y} - h_{\mathbf{W}}(\mathbf{x})$, and we can use this to adjust the weights between the hidden layer and the output layer.
- The above steps produces a term equivalent to the error at the hidden layer, i.e. the error at the output layer is back-propagated to the hidden later.
- This is subsequently used to update the weights between the input units and the hidden layer.

Back-Propagation Learning in Detail

Step 1: Update the weights between the hidden and output layers.

- Let Err_i be the i-th component of the error vector $\mathbf{y} - h_{\mathbf{W}}(\mathbf{x})$.
- Define $\Delta_i = Err_i \times g'(in_i)$.
- The weight-update rule becomes

$$W_{j,i} \longleftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

This is similar to weight-updates for Perceptrons!

Back-Propagation Learning in Detail

Step 2: Back-propagate the error to the hidden layer.

- The idea is that the hidden node j is "responsible" for some fraction of the error Δ_i in each of the output nodes to which it connects.
- Thus the Δ_i values are divided according to the strength (weight) of the connection between the hidden node and the output node:

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

Back-Propagation Learning in Detail

Step 3: Update the weights between the input units and the hidden layer.

- Again, this is similar to weight-updates in Perceptrons:

$$W_{k,j} \longrightarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

Summary of Back-Propagation Learning

For the general case of *multiple hidden* layers:

1. Compute the Δ values for the output units, using the observed error.
2. Starting with the output layer, repeat the following for each layer in the network, until the earliest hidden later is reached:
    - Propagate the Δ values back to the previous layer.
    - Update the weights between the two layers.
3. Repeat Steps 1 to 2 for all training samples.

Algorithm for Back-Propagation Learning

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 $W_{j,i}$, activation function g

  repeat
    for each e **in** $examples$ **do**
      for each node j in the input layer **do** $a_j \leftarrow x_j[e]$
      for $\ell = 2$ **to** M **do**
        $in_i \leftarrow \sum_j W_{j,i} \, a_j$
        $a_i \leftarrow g(in_i)$
      for each node i in the output layer **do**
        $\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$
      for $\ell = M - 1$ **to** 1 **do**
        for each node j in layer ℓ **do**
          $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i} \, \Delta_i$
          for each node i in layer $\ell + 1$ **do**
            $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$
  until some stopping criterion is satisfied
  return NEURAL-NET-HYPOTHESIS($network$)

Figure 20.25 The back-propagation algorithm for learning in multilayer networks.

Results of Back-Propagation Learning

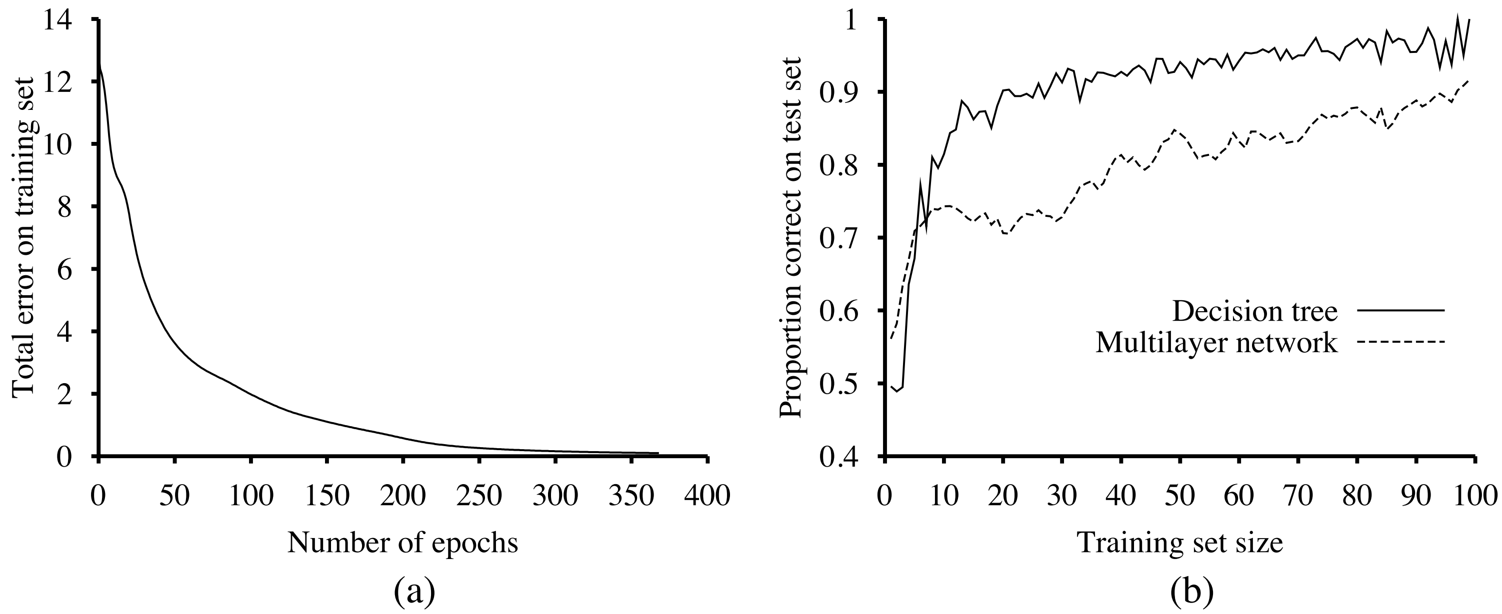


Figure 20.26 (a) Training curve showing the gradual reduction in error as weights are modified over several epochs, for a given set of examples in the restaurant domain. (b) Comparative learning curves showing that decision-tree learning does slightly better than back-propagation in a multilayer network.