Artificial Intelligence

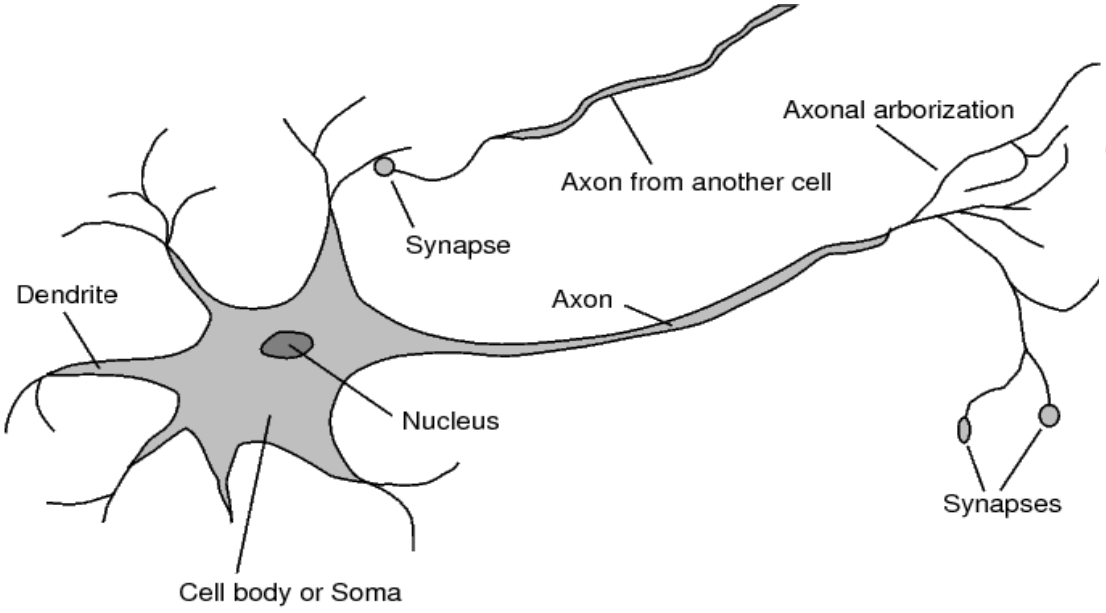
Lecture 15 – Neural Networks

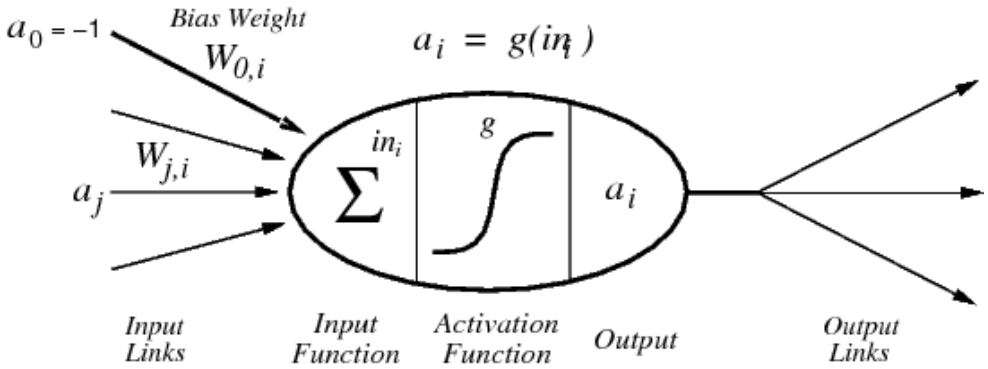
Outline

- Units and links
- Activation functions
- Computing boolean functions
- Network structures
  - perceptrons
  - networks with hidden units
- Learning weights and backpropogation
- Choosing an appropriate network structure

Neural Networks

- A *neuron* is a brain cell that collects and processes electrical signals
- Information processing capabilities of the brain emerge from interactions of networks of neurons
- Early work in AI, e.g., McCulloch & Pitts 1943) investigated artificial neural networks
- Nodes in an (artificial) neural network are much simpler than real neurons





Units & Links

- (Artificial) neural networks are composed of nodes or *units*
- Each unit has an *activation level*, a_i which ranges from -1 (or 0) to +1
- Units are connected by directed *links* – a link from unit *j* to unit *i* serves to propogate the activation a_j of unit *j* to *i*
- Each link also has an associated *weight*, $w_{j,i}$, which determines the strength and sign of the connection

Computing Activation

- To compute its activation, a_i, each unit *i* first computes a weighted sum of its inputs given by

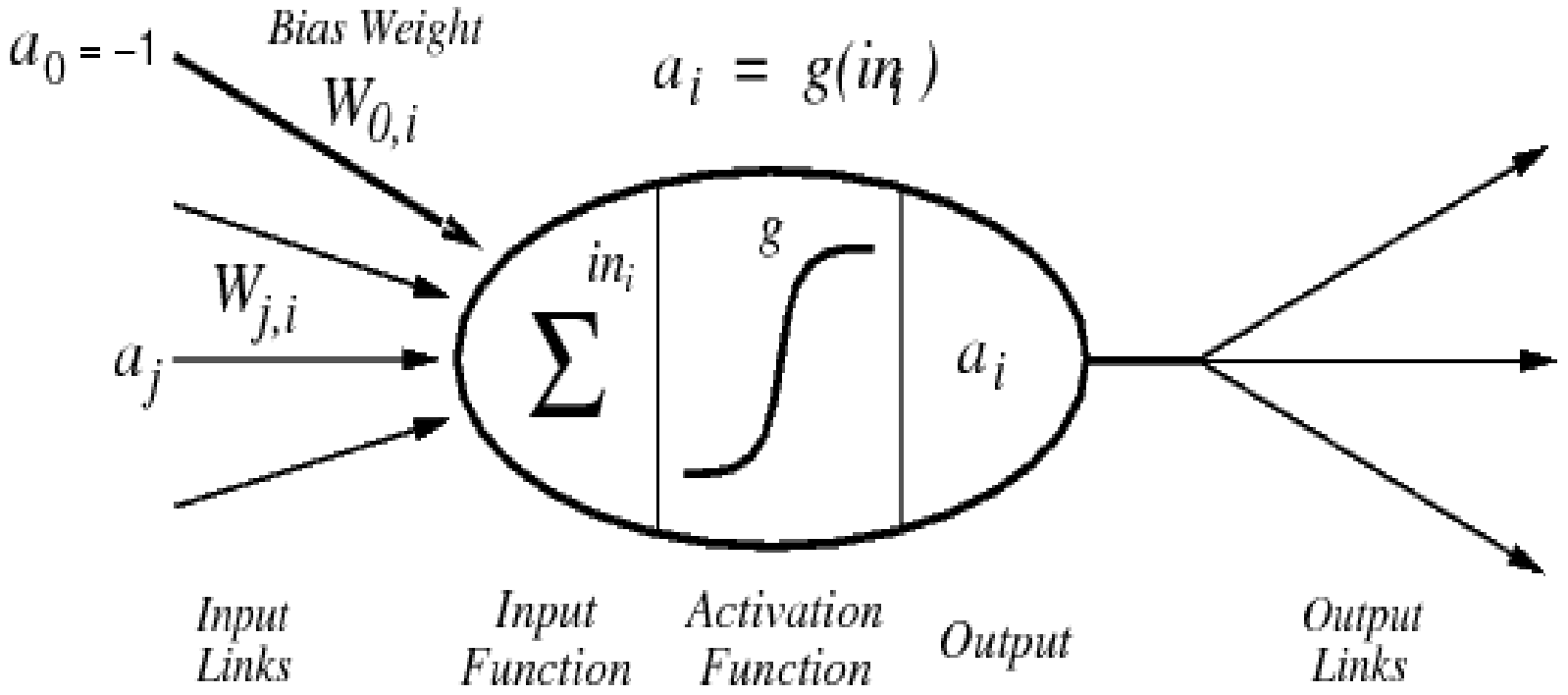
$$in_i = \sum_{j=0}^{n} w_{j,i}\, a_j$$

where *n* is the number of other units which have links to *i* and a_j is the activation of the *j*-th such unit, $0 < j \leq n$

- It then applies an *activation function*, *g*, to this sum to derive its activation

$$a_i = g(in_i) = g\left(\sum_{j=0}^{n} w_{j,i}\, a_j\right)$$

Computing Activation

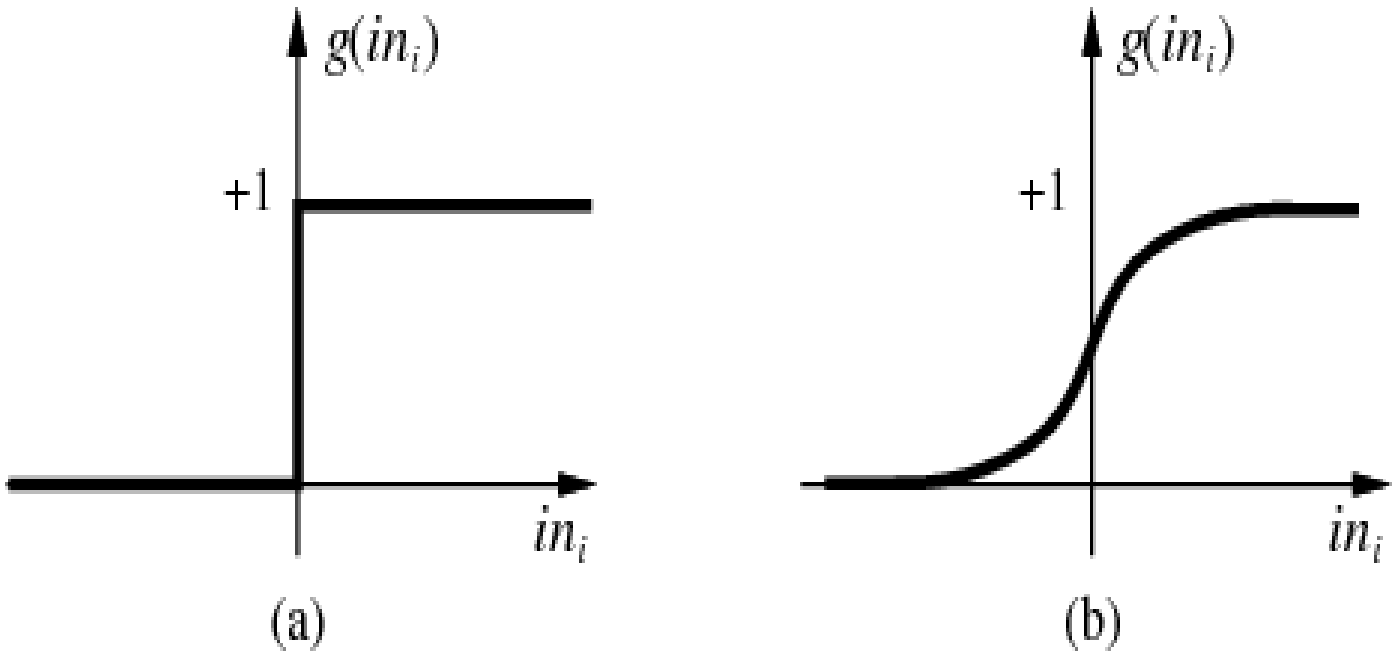


Activation Function

- Activation function must satisfy two properties
- The unit should be active (activation near +1) when the 'right' inputs are given and inactive (activation near -1 or 0) when the 'wrong' inputs are given
- Activation function must be *nonlinear*, otherwise the whole network collapses to a simple linear function

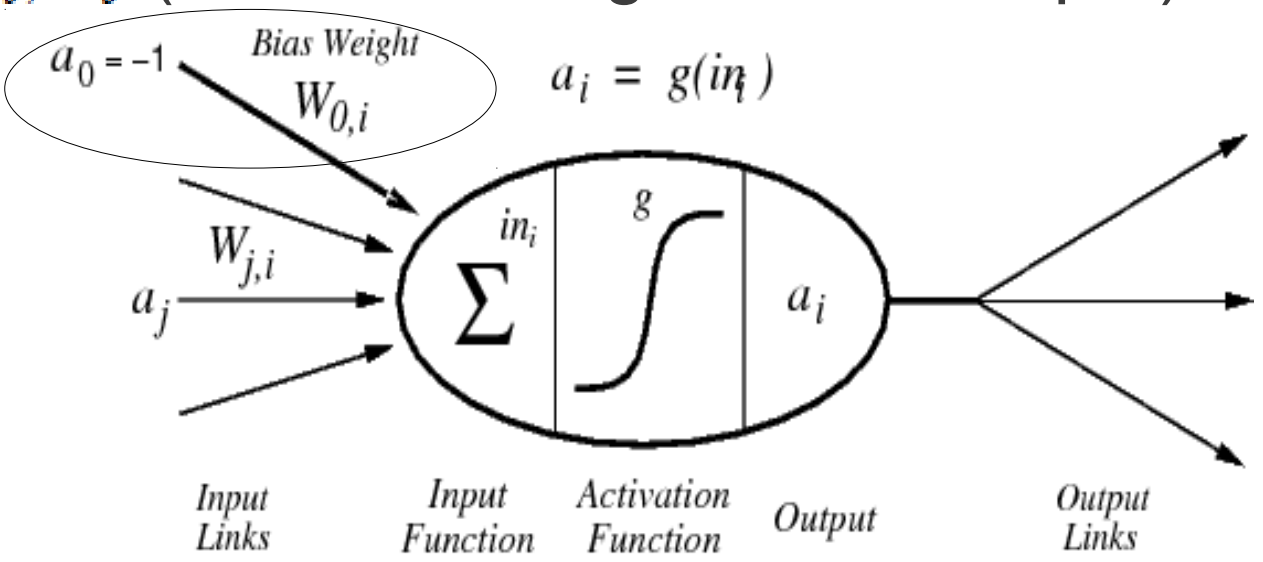
Threshold & Sigmoid Functions

- Two commonly used activation functions are the *threshold* function (a) and the *sigmoid* function (b)
- Threshold function outputs 1 when the input is positive and 0 otherwise
- The sigmoid function is $1/(1 + e^{-x})$



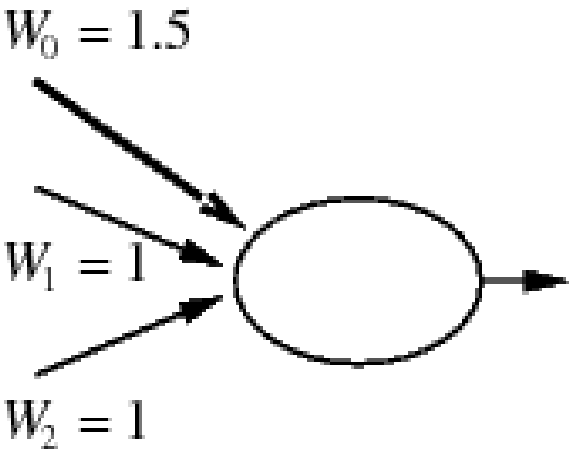
Bias Weight

- Both the threshold and sigmoid functions have a “threshold” at zero
- The bias weight, $w_{0,j}$, sets the actual threshold for the unit
- The unit is activated when the weighted sum of the 'real' inputs $\sum_{j=1}^{n} w_{j,i}\, a_j$ (i.e., excluding the bias input) exceeds $w_{0,j}$

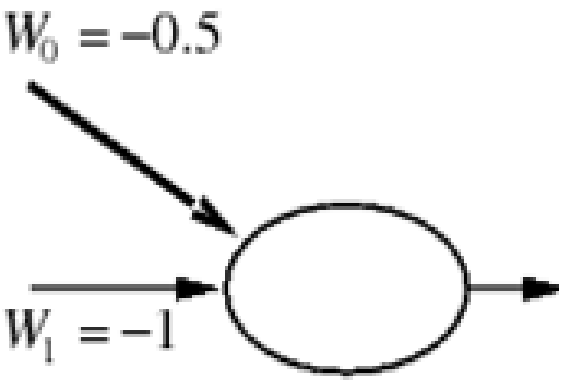


Computing Boolean Functions

- With suitable bias weights, *threshold units* (i.e., units with a threshold activatin function) compute *boolean functions* of their inputs
- We can use such units to build a network to compute any boolean function of its inputs



AND



NOT

Example: Computing AND

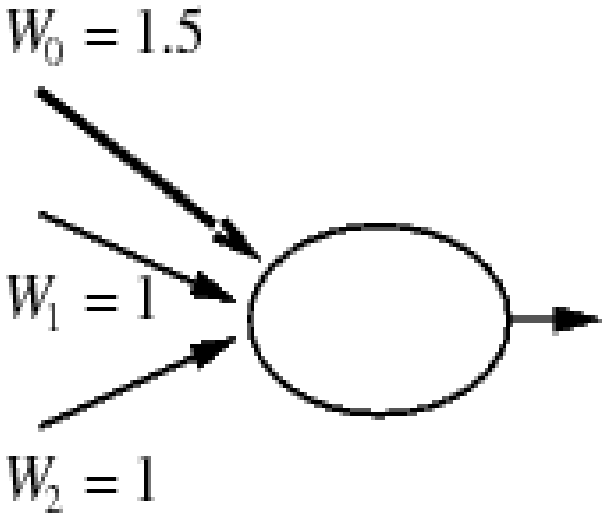
- Assume the input values for links 1 and 2 are both 1, i.e., $a_1 = a_2 = 1$, and $w_{1,i} = w_{2,i} = 1$
- Input value for link 0 is -1, i.e., $a_0 = -1$, and $w_{0,i} = 1.5$

$$in_i = \sum_{j=0}^{n} w_{j,i}\, a_j$$

$$= (1.5 \times -1) + (1 \times 1) + (1 \times 1)$$

- Threshold function outputs 1 when the input is positive and 0 otherwise

$$a_i = g(0.5) = 1$$



AND

Exercise: Computing OR

- Devise a set of weights which compute the boolean function '*or*'
- Assume that the inputs are either 0 or 1, i.e., a_1, a_2 = 0 or 1, and that $a_{0,i}$ = -1
- Find values for $w_{0,i}$, $w_{1,i}$, $w_{2,i}$ such that the unit computes a_1 a_2

Exercise: Computing OR

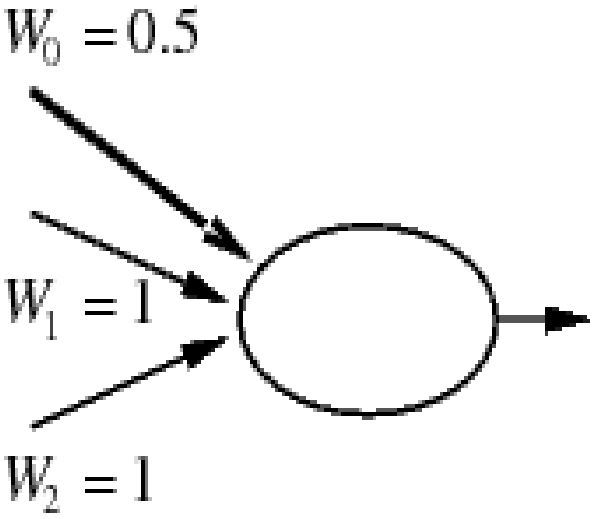
- Assume the input values for links 1 and 2 are both 1, i.e., $a_1 = a_2 = 1$, and $w_{1,i} = w_{2,i} = 1$
- Input value for link 0 is -1, i.e., $a_0 = -1$, and $w_{0,i} = 0.5$

$$in_i = \sum_{j=0}^{n} w_{j,i}\, a_j$$

$$= (0.5 \times -1) + (1 \times 1) + (1 \times 1)$$

- Threshold function outputs 1 when the input is positive and 0 otherwise

$$a_i = g(0.5) = 1.5$$



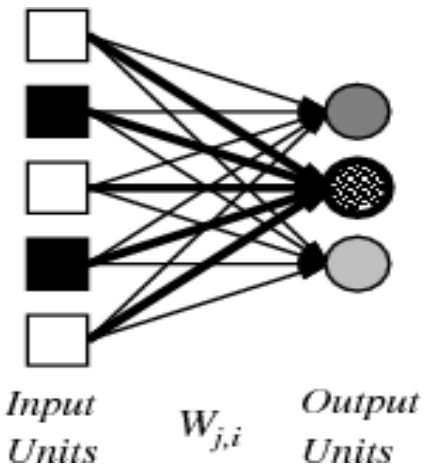
OR

Network Structures

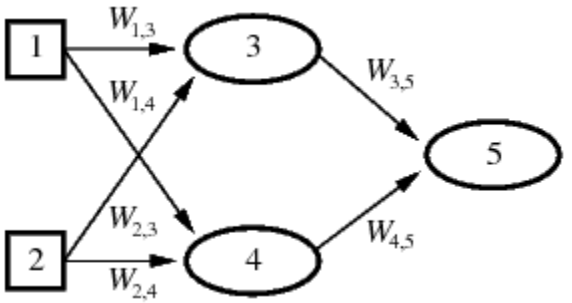
- Acyclic or *feed-forward* networks
  - represents a function of its current inputs
  - has no internal state otther than the weights themselves
- Cyclic or *recurrent* networks
  - feeds its output back into its own inputs
  - response to inputs depends on the initial state of the network, which may depend on previous inputs
  - supports short-term memory
  - activations of units form a dynamical system that may reach a stable state, or exhibit oscillations or even chaotic behaviour

Feed-forward Networks

- In a single layer feed-forward network, or *perceptron*, there are two layers – all input units are connected directly to the output units
- A multi-layer feed-forward network has *hidden units* – one or more layers of units between the inputs and the outputs



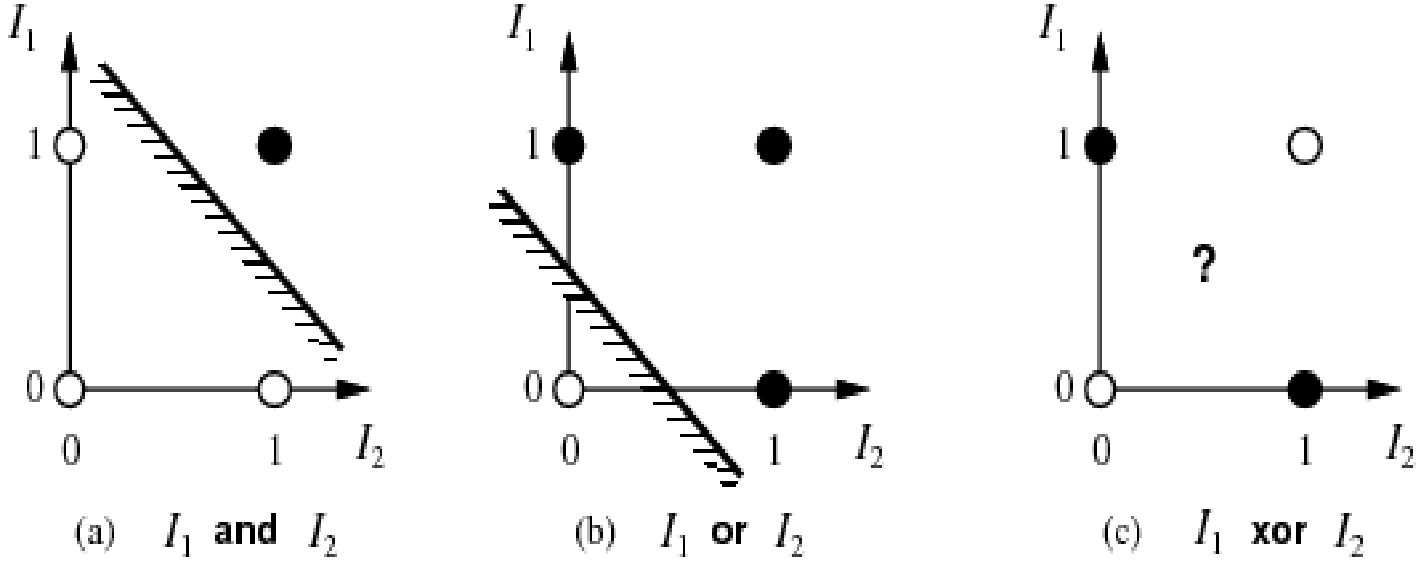
Perceptron



Multi-layer Network

Perceptrons

- With a thresshold activation function, perceptrons can be viewed as representing boolean functions
- However perceptrons can only represent *linearly seperable* functions – can't represent functions such as XOR (Minsky & Papert, 1969)



(a) I_1 and I_2 (b) I_1 or I_2 (c) I_1 xor I_2

Networks with Hidden Units

- Adding hidden layers enlarges the set of hypotheses the network can represent
- With a single hidden layer of sufficient size, it is possible to represent any continous function of the inputs with arbitrary accuracy
- With two hidden layers, discontinous functions can be represented
- However the number of hidden units required grows exponentially with the number of inputs – $2^n/2$ hidden units are required to encode all boolean functions of n inputs
- The most common case is a *single* hidden layer

Learning in Neural Networks

- Neural network learning algorithms work by adjusting the weights to minimise some measure of error on the training set
- The training set examples are run through the network one at a time, adjusting the weights slightly after each example to reduce the error
- The amount by which the weights are adjusted is determined by the *learning rate*
- Each cycle through the examples is called an *epoch*
- Epochs are repeated until some stopping criterion is reached – e.g., the weight changes have become very small

Learning in Perceptrons

- The error measure is usually taken to be the sum of squared errors
- The squared error for a single training example with input *x* and true output *y* is given by

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

  where $h_w(x)$ is the output of the network on the training example
- To reduce the squared error *E* each weight is updated using

$$w_j = w_j + \alpha \times Err \times g'(in) \times x_j$$

  where *α* is the learning rate and *g*' is the derivative of the activation function – i.e., error is apportioned proportional to the link weight

Learning in Multi-layer Networks

- In a perceptron, each output unit is independent of the others – each weight affects only one of the outputs
- If there are hidden units, we can't consider each output in isolation
- We need to consider a vector of output values $h_w(x)$ and each example has a classification vector y
- We also need to backpropogate the error from the output layet to the hidden layer(s)

Backpropogation

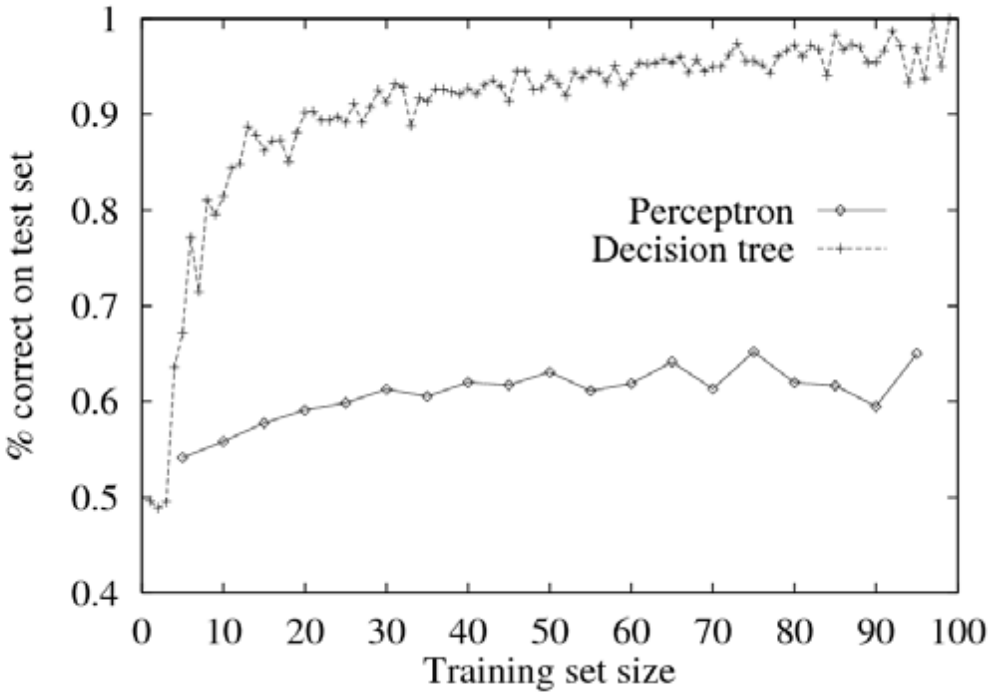
- Compute error values for the output units, using the observed error
- Starting with the output layer repeat for each hidden layer back to the input layer:
  - apportion the error values for nodes in this layer to nodes in the previous layer (on the basis of the current weights)
  - update the weights between the two layers to reduce the error

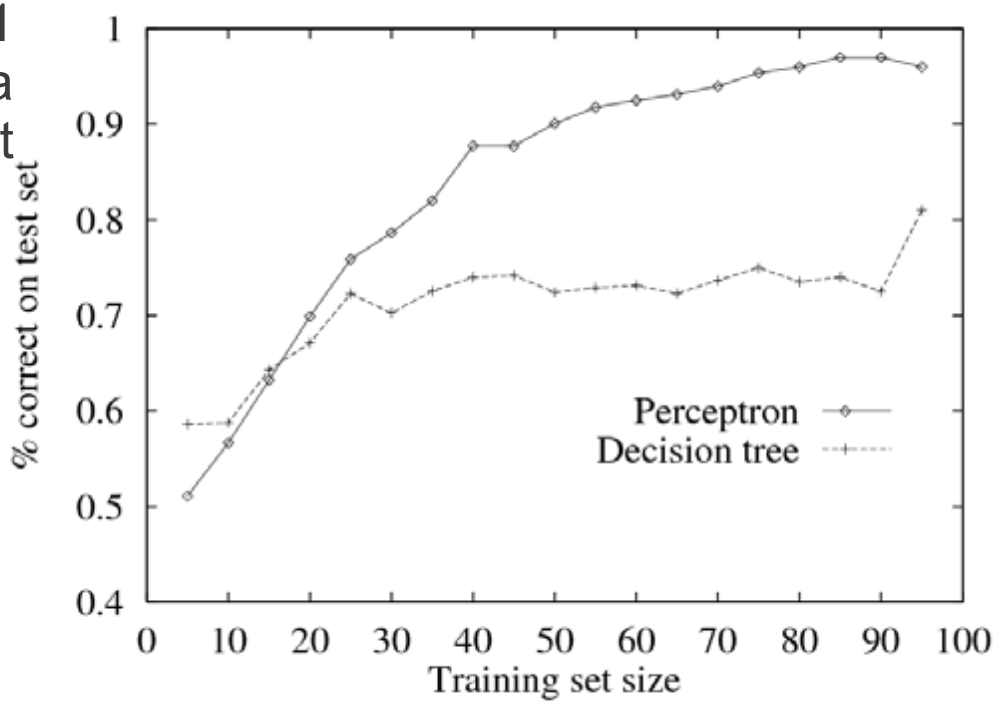
Uses of Neural Networks

- Neural networks can be used for classification or regression
- Activation of the *input units* represents features of the problem or situation
- Activation of the *output unit(s)* represent the output value or classification
- For boolean classification with continuous outputs (e.g., sigmoid units) there is typically a single output with a value > 0.5 interpreted as one class and < 0.5 the other
- With *k* classes there are *k* output units, with the value of each output unit representing the relative likelihood of that class given the input

Examples

- For some problems neural networks are better, while for others decision trees are better
- For example, the restaurant problem is not linearly seperable and a perceptron (with 11 inputs and 1 output) can only learn to classify 65% of the data
- In contrast, a *majority function* with 11 inputs can be learnt quite easily by a perceptron as it's linearly seperable, but is hard for a decision tree

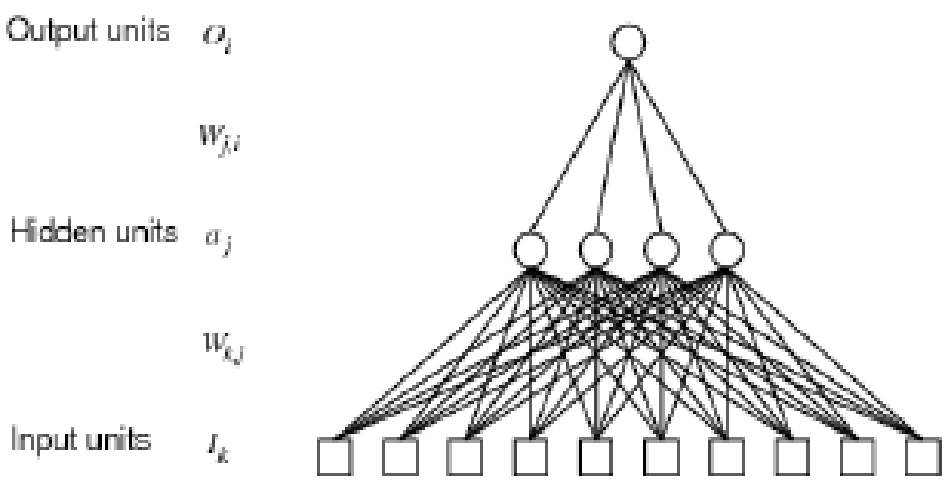


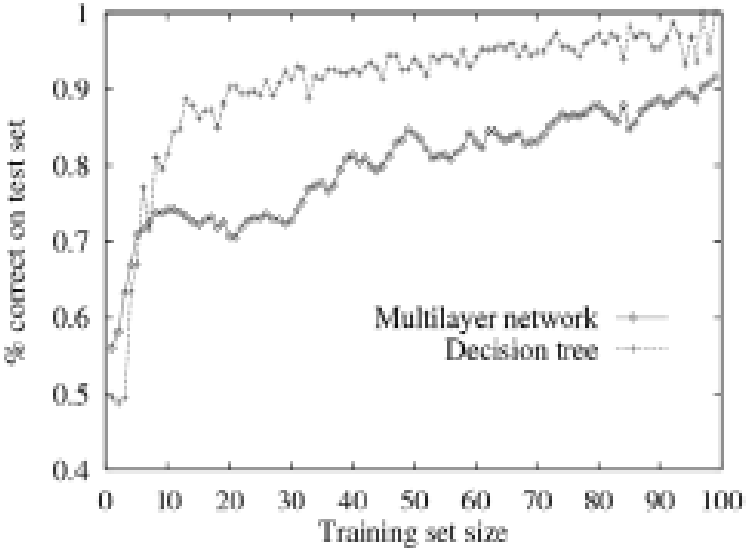


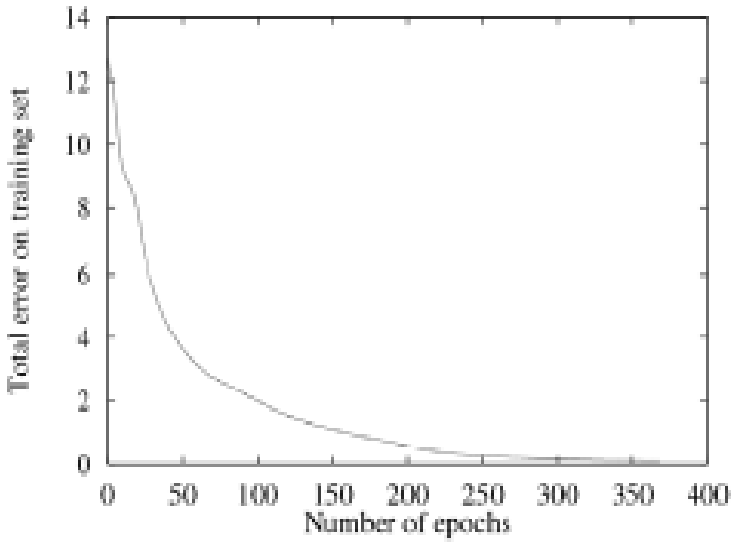
Choosing the Network Structure

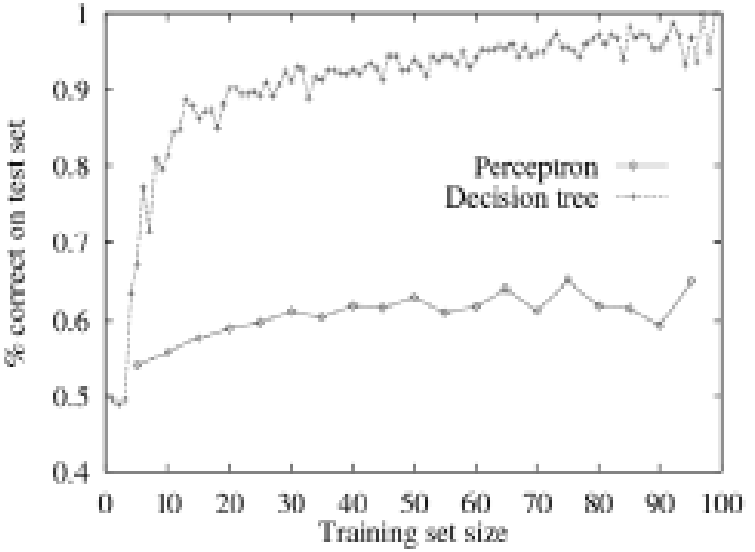
- For any particular network structure it is hard to characterise exactly which functions can be represented and which cannot
- Determining an appropriate network structure for a particular problem usually involves some experimentation to find the number of layers (and hidden nodes for multi-layer networks) that work best
- A typical approach is to try several different networks and keep the best
- If we limit ourselves to full-connected networks the only choices we have to make are the number of hidden layers and their sizes
- For example, for the restaurant problem, a single hidden layer containing four nodes approaches the performance of the decision tree

Example


Output units O_i
$W_{j,i}$
Hidden units a_j
$W_{k,j}$
Input units I_k


% correct on test set
Training set size
Multilayer network
Decision tree


Total error on training set
Number of epochs


% correct on test set
Training set size
Perceptron
Decision tree

Non Fully-conected Networks

- We can produce non fully-connected networks by starting with a fully connected network and removing links or nodes
- After the network is trained for the first time, we identify those connections (and units) which can be removed
- The network is retrained and if the performance has not decreased, the process is repeated
- Other approaches have been proposed for “growing” a larger network from a smaller one

Summary

- Neural networks work well for some problems which decision trees find hard
- Perceptrons have simple structure and it's often easy to to see how to encode a learning problem as a perceptron
- However if we need to learn a function which is not linearly seperable, the network must have one or more hidden layers – often not clear what network structures are appropriate
- If we choose a network which is too big it will be able to 'memorise' all the examples, and will not necessarily generalise to inputs that have not been seen before
- Neural networks may need significant computation to train – the multi-layer restaurant network needed 350 passes through the training set; the decision tree is induced in one pass