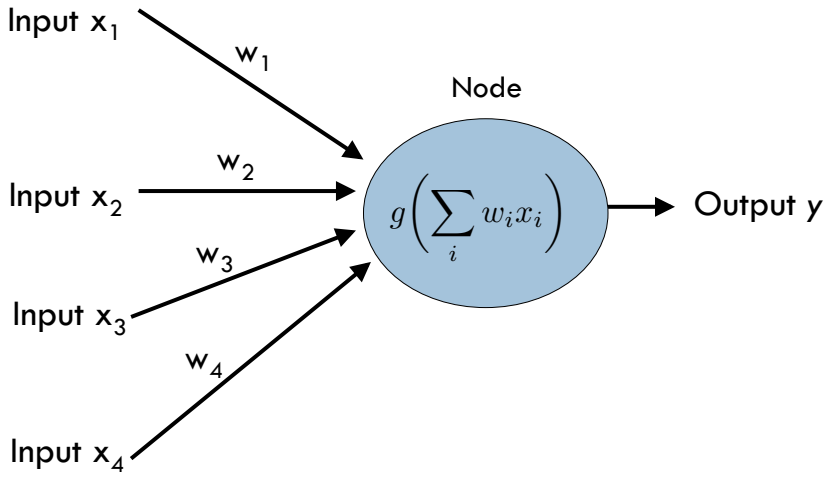
NEURAL NETWORKS 2

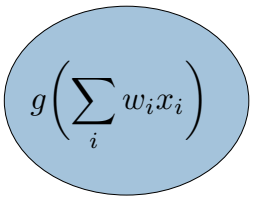
Today

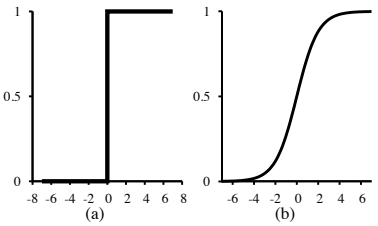
- Reading
  - AIMA 18.6-18.8
- Goals
  - Delta Algorithm for Perceptrons
  - Feed-forward neural networks
  - Backpropagation

A single perceptron



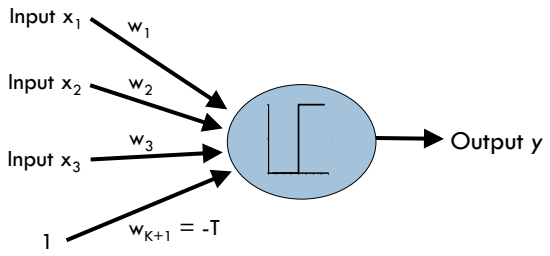
Activation function





- The activation function determines if the "electrical signal" entering the neuron is sufficient to cause it to fire
  - Threshold function – range is {0,1}
  - Sigmoid function – range [0,1]
  - Hyperbolic tangent function – range [-1,1]

Threshold versus "dummy" variable



- Having a threshold T is equivalent to creating a "dummy" variable with value always 1

$$\sum_i x_i w_i \geq T \Longrightarrow 1$$

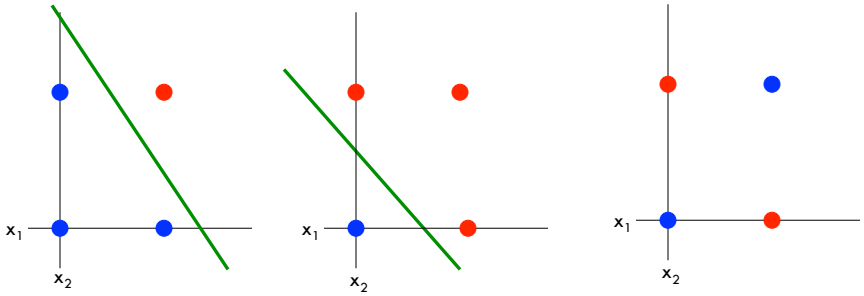
$$\sum_i x_i w_i - T \geq 0 \Longrightarrow 1$$

Perceptrons: Linearly separable functions

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	x_1 or x_2
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	x_1 xor x_2
0	0	0
0	1	1
1	0	1
1	1	0



Perceptron Training rule

- Need an algorithm for finding a set of weights w such that
  - The predicted output of the neural network matches the true output for all examples in the training set
  - Predicts a reasonable output for inputs not in the training set

Perceptron Training Rule

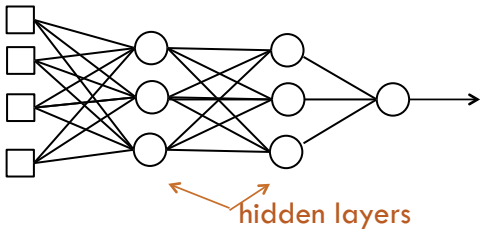
1. Begin with randomly initialized weights
2. Apply the perceptron to each training example (each pass through examples is called an epoch)
3. If it misclassifies an example **modify the weights**
4. Continue until the perceptron classifies all training examples correctly

(Derive gradient-descent update rule)

Beyond perceptrons

- Feed-forward neural network
  - Forms a directed acyclic graph (DAG) structure
  - *Any continuous function of the inputs can be represented using a sufficiently large hidden layer*
- Recurrent neural network
  - The output is fed back into the inputs
  - Creates a dynamical system that can have "short-term memory"

Interesting project idea!



Backpropagation

1. Begin with randomly initialized weights
2. Apply the neural network to each training example (each pass through examples is called an epoch)
3. If it misclassifies an example **modify the weights**
4. Continue until the neural network classifies all training examples correctly

(Derive gradient-descent update rule)

Backpropagation

function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network
inputs: *examples*, a set of examples, each with input vector **x** and output vector **y**
network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
local variables: Δ, 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 (**x**, **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 ℓ **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 ℓ **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*

Figure 18.24 The back-propagation algorithm for learning in multilayer networks.