

# NEURAL NETWORKS

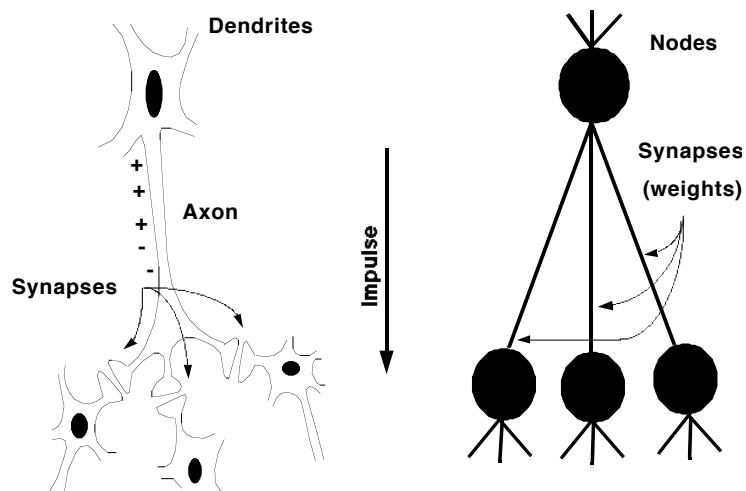
## Today

- Reading
  - AIMA 18.6 – 18.7
  
- Goals
  - Perceptrons and perceptron networks
  - Feed-forward neural networks
  - Backpropagation algorithm

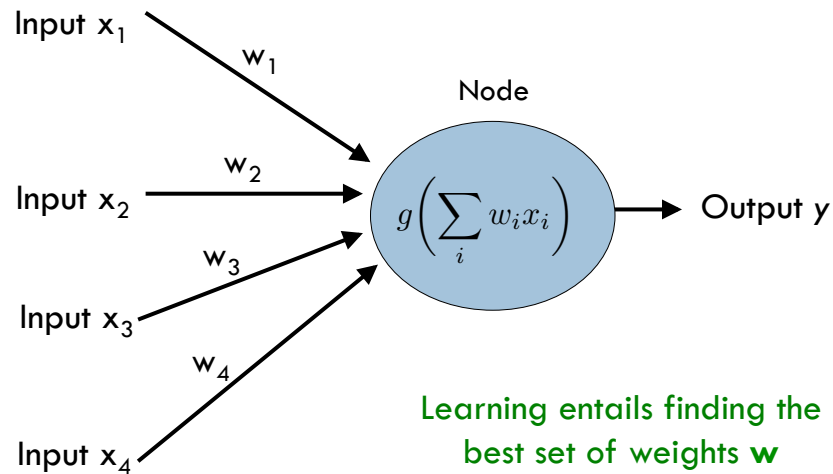
## Announcements

- HW4/5 passed back some time this week
- HW6 due Wednesday so I can give feedback on project proposals by Friday
- Second test is scheduled for December 1st

## Motivation: Our Nervous System

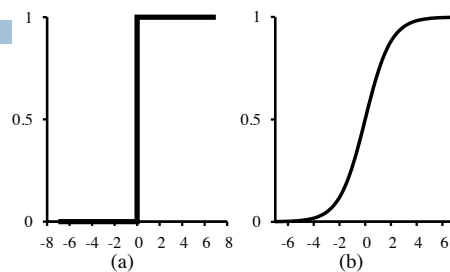


## A single perceptron



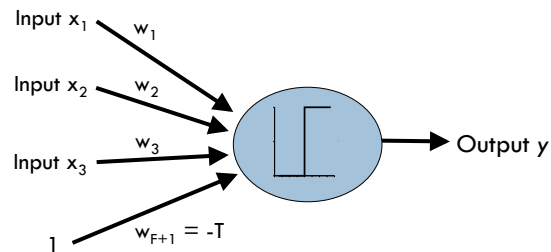
## Activation function

$$g\left(\sum_i w_i x_i\right)$$



- 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

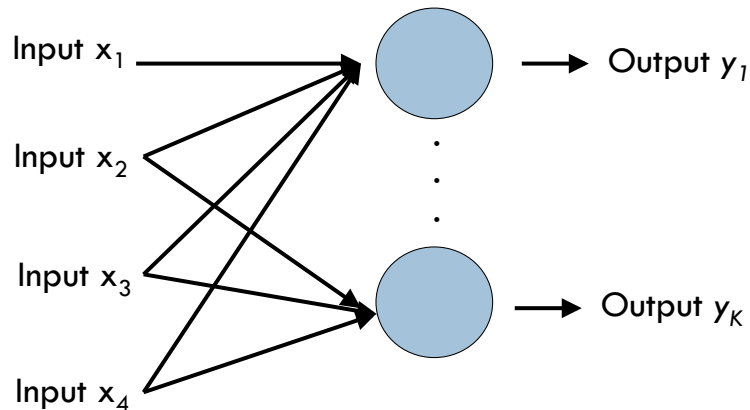


- Having a threshold  $T$  is equivalent to creating a “dummy” variable with value always 1

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

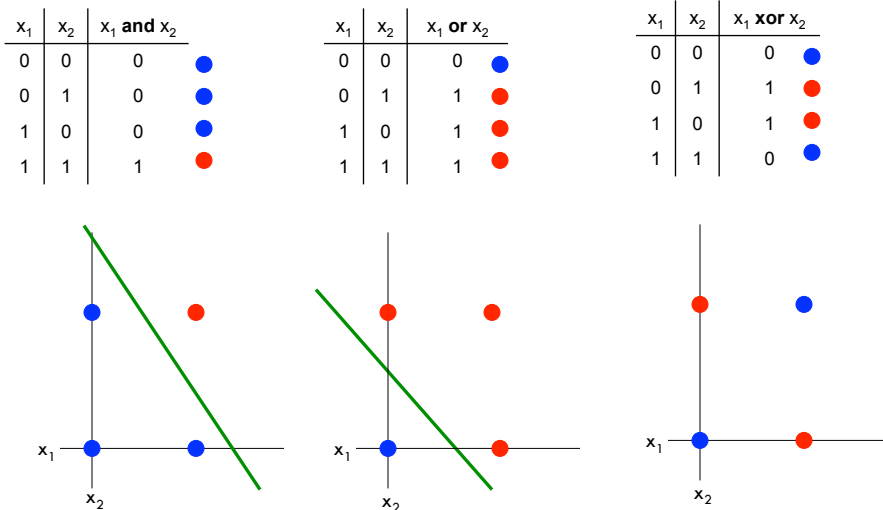
$$\sum_i x_i w_i - T \geq 0 \implies 1$$

## A perceptron network



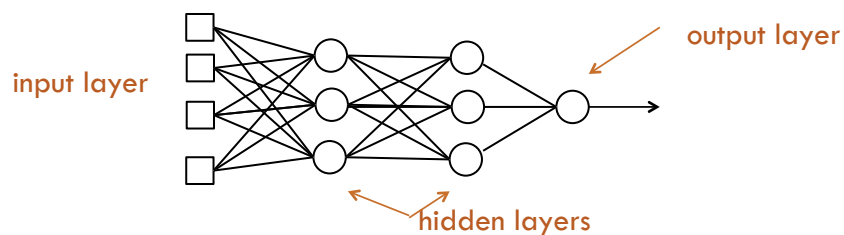
Reduces to  $K$  independent perceptrons

## Perceptrons: Linearly separable functions



## 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



## 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 on board)

## 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:  $\Delta$ , 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 (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  $\ell$  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

```

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