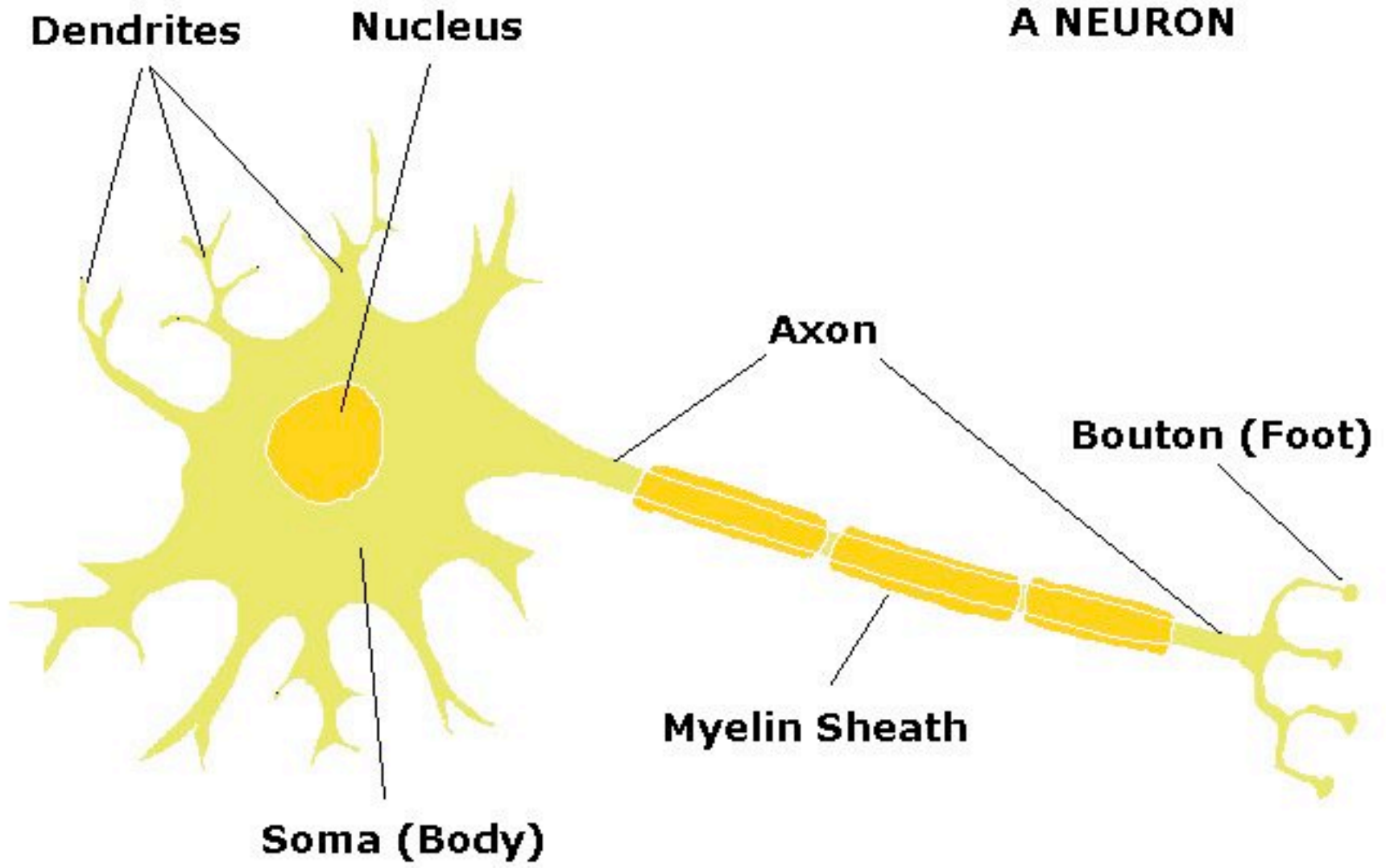


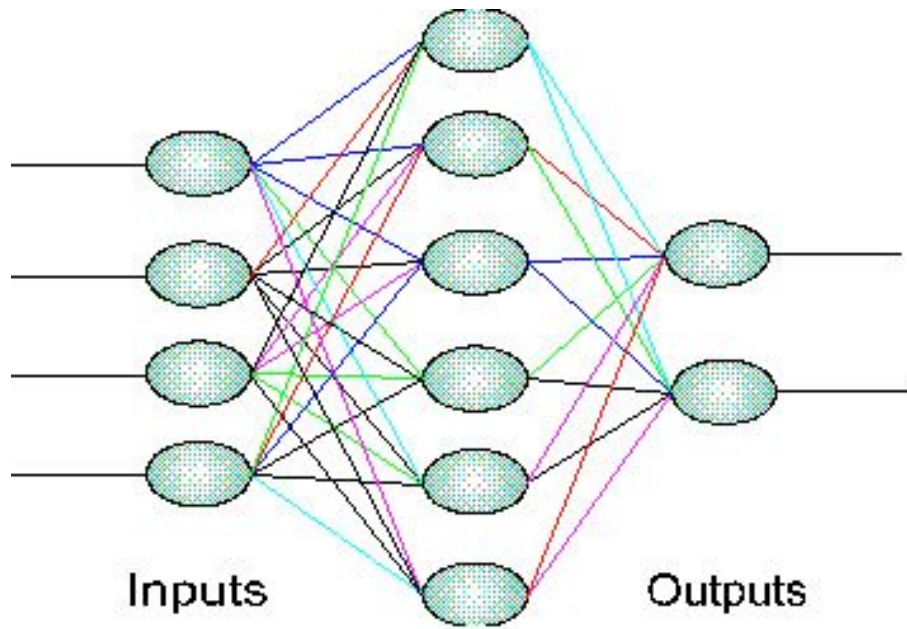
# Neural Nets

# A NEURON

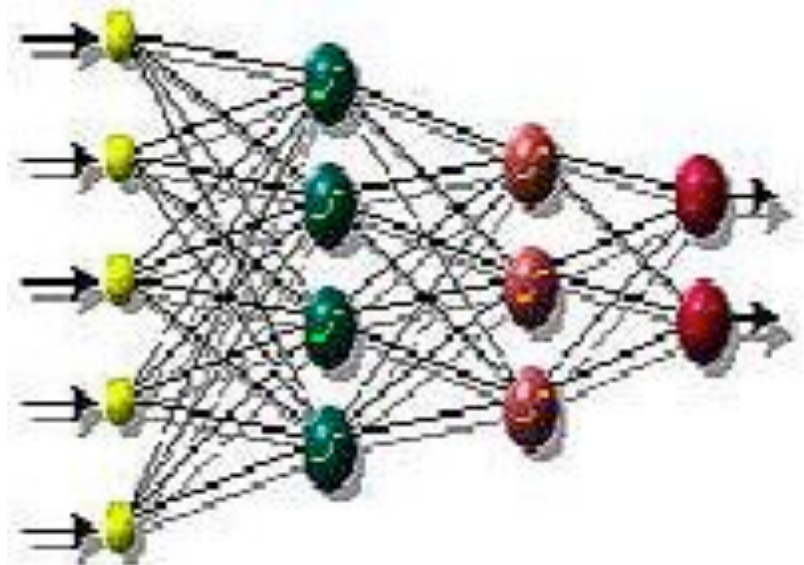


# Neural Networks: Some First Concepts

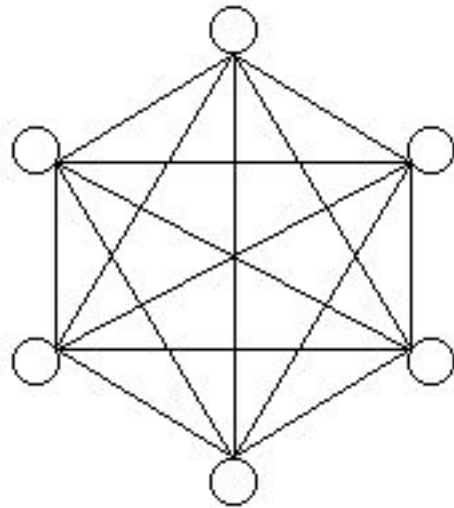
- Each neural element is loosely based on the structure of neurons
- A neural net is a collection of neural elements connected by weighted links
- We think of some set of neurons as “input elements”; these are linked to “output elements” which can be interpreted as a classification of the input pattern
- Standard formats: perceptrons, multilayer feedforward networks, autoassociative networks



Standard diagrams of  
feedforward networks



# An example of a recurrent network



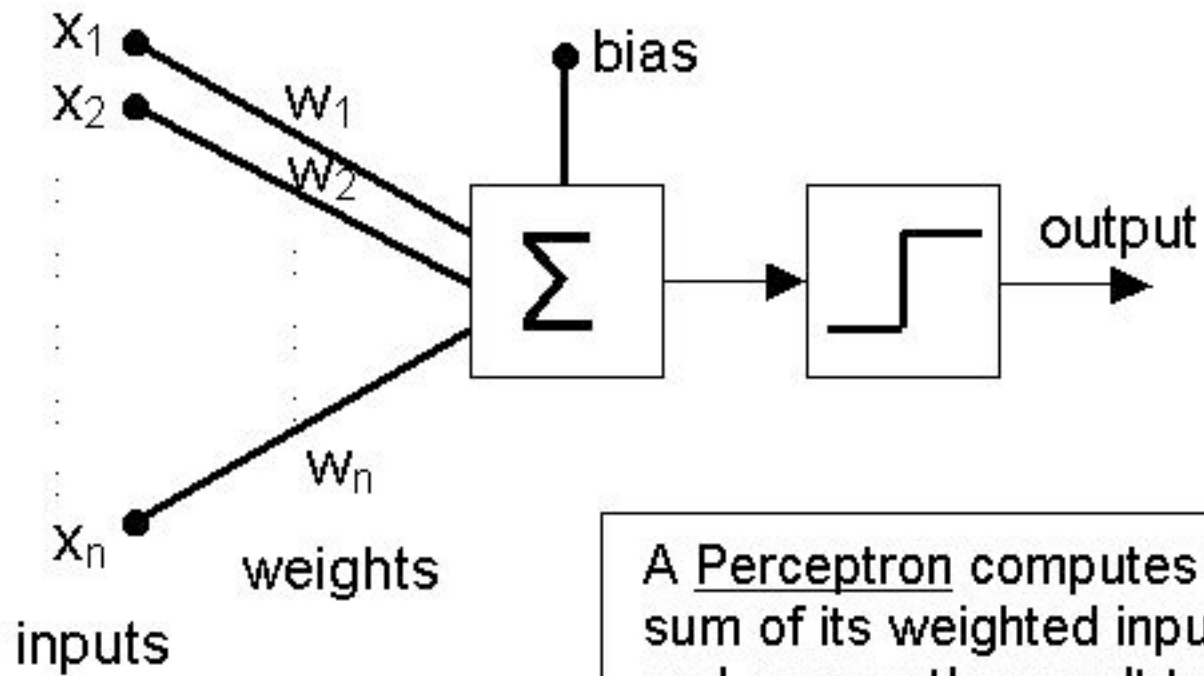
Patterns are overlayed on units.  
Common regularities  
across patterns modify the  
connection weights.

# Structure of an (artificial) neuron

- Think of this as a very simple computational element: it receives numeric input values, sums those values, and compares them to a threshold.
- If the sum of the inputs is greater than the threshold, the neuron outputs a 1; otherwise it outputs a zero.
- The output of this neuron is connected (as usual, via weighted links) to subsequent neurons in the net.

# Perceptrons: the Simplest Neural Network

- Perceptrons are two-layer networks: one layer of inputs directly connected to a layer of outputs.
- For simplicity, we can look at a perceptron with a single output node



A Perceptron computes the sum of its weighted inputs and passes the result to a hard-limit threshold function.

# Training a Perceptron by Adjusting its Weights

- Overall error is the (squared) value of the difference between what we wanted and what we got from our perceptron.
- Once we see that our perceptron is in error, we can adjust each of the weights leading to the output node. We'll adjust each weight in such a way as to make the error value smaller.

# The Update Rule for a Weighted Edge of a Perceptron

To update the weight from a node  $j$  leading to an output node, we adjust the weight according to the following formula:

$$W_j \leftarrow W_j + (\alpha \text{Err} g'(\text{in}) x_j)$$

Here,

Err is the difference between what we wanted and what we got from the output.

$G'(\text{in})$  is the derivative of our output function

$X_j$  is the output from node  $j$  leading to us

$\alpha$  is a “rate parameter”

# Multi-Layer Neural Networks

Now, we expand our network architecture so that it has several (or more) layers of neural elements. We think of the layers as numbered in columns left to right, with the leftmost layer as “layer 1” (input) and the rightmost layer as “layer n” (output).

## Adjusting Weights for Multilayer Networks

Here's the perceptron rule, again:

$$W_j \leftarrow W_j + (\alpha \text{Err} g'(in) x_j)$$

Note that this is the update rule to adjust a weight from neuron  $j$  to an output neuron. Let's rewrite this for our new multilayer network as:

$$W_{j \rightarrow k} \leftarrow W_{j \rightarrow k} + (\alpha \Delta_k a_j)$$

That is, we call the output neuron  $k$  and we combine the error term and the derivative-at- $k$  term into one symbol,  $\Delta$ . We also relabel the output coming from neuron  $j$  as  $a_j$ . Think of the  $\Delta$  term as “how much node  $k$  wants its output to change”. For an output node, this is just the product of the error at  $k$  and the rate at which error changes for a small change in input at  $k$ .

# How Much Does a Hidden Node Want to Change Its Output?

Okay, what about a hidden-layer node: what should its value of  $\Delta$  be?

It is the product of its own derivative and the weighted sum of how much its own targets “want” to change:

$$\Delta_j = g'(in_j) \sum_m W_{j \rightarrow m} \Delta_m = a_j(1 - a_j) \sum_m W_{j \rightarrow m} \Delta_m$$

Again, think of these terms as “how fast I will change my output for a little change in input” and “the sum over all my targets of how much they want to change, weighted by my influence upon them”

# Backpropagation: Step 1

Start at the output layer. For each neuron in that layer, we compute:

$$\begin{aligned}\Delta_{\text{output-}i} &= g'(\text{in}_{\text{output-}i}) \text{Error}_{\text{output-}i} \\ &= a_{\text{output-}i}(1 - a_{\text{output-}i}) \text{Error}_{\text{output-}i}\end{aligned}$$

## Backpropagation: Step 2

Now, for each neuron  $j$  in the previous layer, compute  $\Delta_j$  as follows:

$$\Delta_j = a_j (1 - a_j) \sum_i W_{j \rightarrow i} \Delta_i$$

Now, use each of these  $\Delta$  values to compute the  $\Delta$  values for the layer before that, and so on until you reach the input layer.

# Backpropagation: Final Step

We now have, for each node, a measure of how much that node wishes to change its own output value. We now adjust the weights on each edge as follows:

$$W_{j \rightarrow i} \leftarrow W_{j \rightarrow i} + (\alpha a_j \Delta_i)$$

# A Sampler of Additional Issues

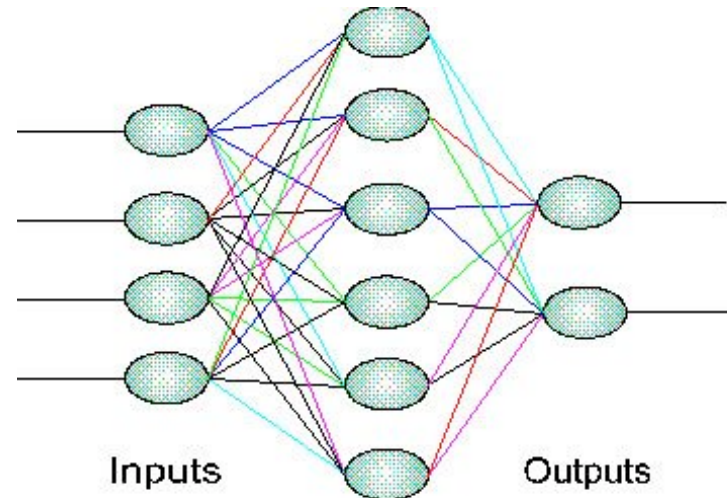
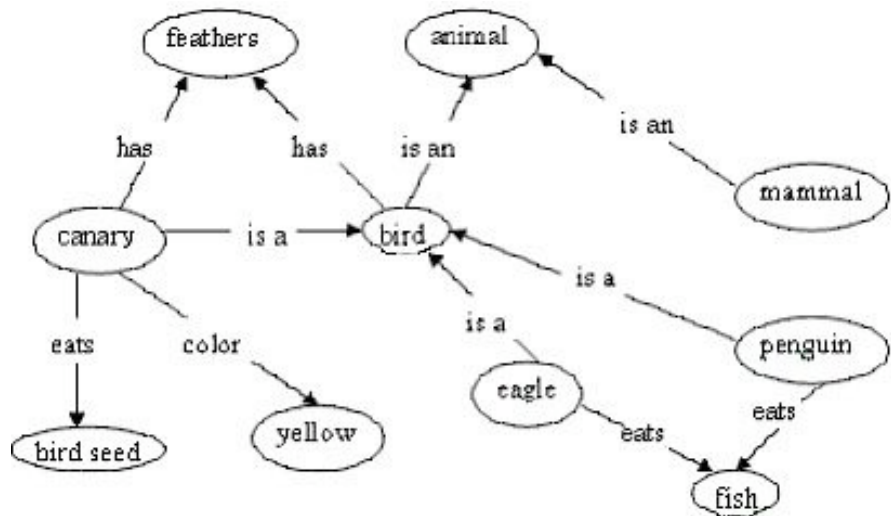
- How do we choose an appropriate network size to train? Too small a network, and we may not be able to represent our concept; too big, and we run the risk of *overfitting*.
- How do we choose a training rate parameter? (The notion of “simulated annealing”.)
- Training partial nets (e.g., one output at a time).

# From Feedforward Networks to Autoassociative Networks

- An autoassociative network is one that just tries to reproduce its input (as in Hinton's "principal components learning" example)
- Completion of partial patterns
- Using mutual constraints to help resolve ambiguities



*Worth* 1000.com "Wait! Wait! ... Cancel that, I guess it says 'help.'"



Some natural points of comparison:

The meaning or significance of a node or edge

The types of psychological phenomena being modeled

The types of uses to which these models may be put

# Neural Nets: What Do People Like?

## *The pros:*

Utility for a variety of pattern recognition tasks  
(handwriting, spoken word, face, etc.)

Simplicity of programming (reinforcement  
learning)

“Graceful degradation”

An elegant model of emergence in programming:  
the idea that large numbers of simple  
programming elements are collectively capable of  
interesting, higher-level behaviors.

# What Do People Dislike?

*The cons:*

A vague feeling that we haven't really learned anything about these pattern recognition skills

Neural nets seem to be a far less natural means for demonstrating “explicit rule learning”