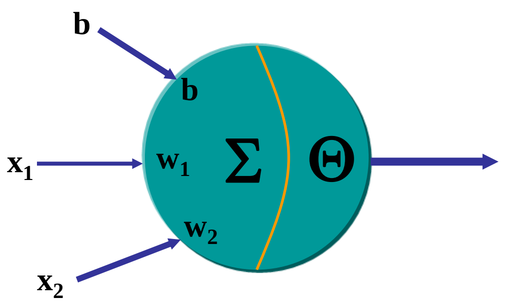
Review – The Perceptron

- The perceptron is a neuron with a bias and a threshold function



Activation function:

$$f(s) = \begin{cases} 1 \text{ if } s > \Theta \\ 0 \text{ if } -\Theta <= s <= \Theta \\ -1 \text{ if } s < -\Theta \end{cases}$$

Review – Hebbian Algorithm

- **Step 0**: initialize all weights to 0
- **Step 1**: Given a training input, s, with its target output, t, set the activations of the input units: $x_i = s_i$
- **Step 2**: Set the activation of the output unit to the target value: $y = t$
- **Step 3**: Adjust the weights: $w_i(new) = w_i(old) + x_i y$
- **Step 4**: Adjust the bias (just like the weights): $b(new) = b(old) + y$

OUTLINE

- **More on Hebbian Learning**
- **Heteroassociative Architecture**
- **Backpropagation**

Alternative View

- **Goal: Associate an input vector with a specific output vector in a neural net**
- **In this case, Hebb's Rule is the same as taking the outer product of the two vectors:**

$$\mathbf{s} = (s_1, \ldots, s_i, \ldots s_n) \text{ and } \mathbf{t} = (t_1, \ldots, t_i, \ldots t_m)$$

$$st = \begin{bmatrix} s_1 \\ \cdot \\ \cdot \\ \cdot \\ s_n \end{bmatrix} [t_1 \ldots t_m] = \begin{bmatrix} s_1t_1 & \cdot & s_1t_m \\ \cdot & \cdot & \cdot \\ s_nt_1 & \cdot & s_nt_m \end{bmatrix} \longleftarrow \text{Weight matrix}$$

Weight Matrix

- To store more than one association in a neural net using Hebb's Rule
  - Add the individual weight matrices

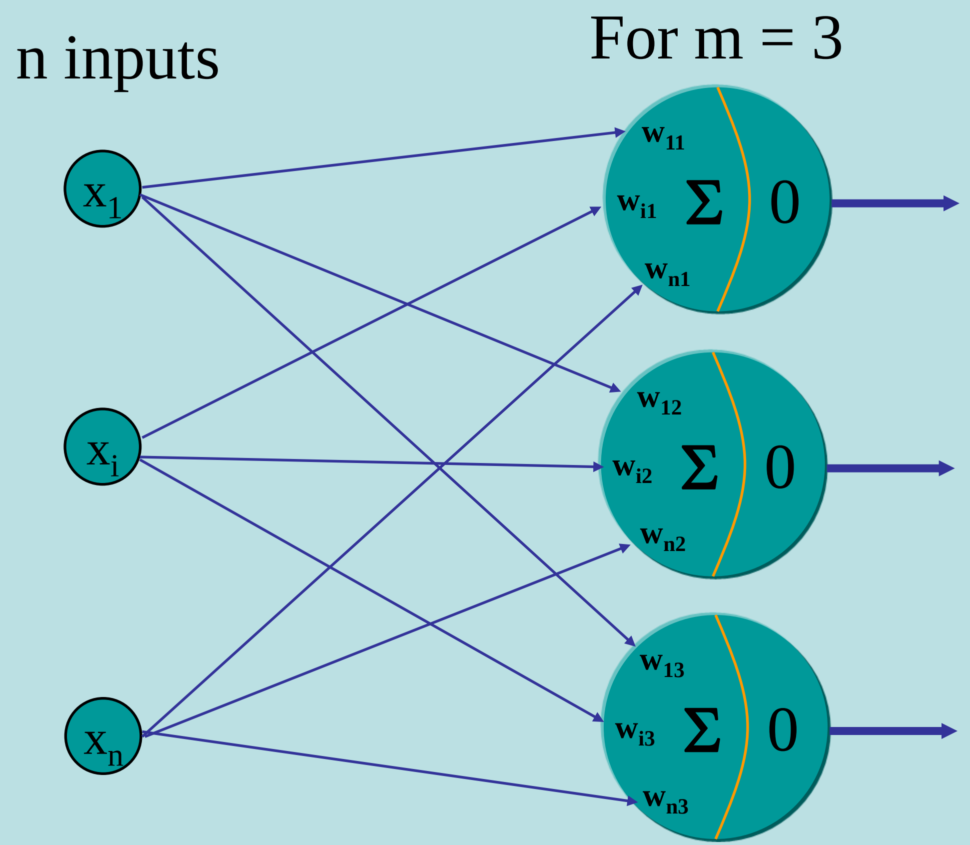
- This method works only if the input vectors for each association are orthogonal (uncorrelated)
  - That is, if their dot product is 0

$\mathbf{s} = (s_1, \ldots, s_i, \ldots s_n)$ and $\mathbf{t} = (t_1, \ldots, t_i, \ldots t_m)$

$$s \bullet t = [s_1 \ldots s_n] \begin{bmatrix} t_1 \\ \cdot \\ \cdot \\ \cdot \\ t_m \end{bmatrix} = 0$$

Heteroassociative Architecture

- There are n input units and m output units with each input connected to each output unit.



Activation Functions:

For bipolar targets:

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$

For binary targets:

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i <= 0 \end{cases}$$

Example

- GOAL: build a neural network which will associate the following two sets of patterns using Hebb's Rule:

$$
\begin{array}{llll}
s_1 = (1 & -1 & -1 & -1) \\
s_2 = (-1 & 1 & -1 & -1) \\
s_3 = (-1 & -1 & 1 & -1) \\
s_4 = (-1 & -1 & -1 & 1)
\end{array}
\qquad
\begin{array}{lll}
t_1 = (1 & -1 & -1) \\
t_2 = (1 & -1 & 1) \\
t_3 = (-1 & 1 & -1) \\
t_4 = (-1 & 1 & 1)
\end{array}
$$

The process will involve 4 input neurons and 3 output neurons

The algorithm involves finding the four outer products and adding them

Algorithm

Pattern pair 1:

$$\begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix}$$

Pattern pair 2:

$$\begin{bmatrix} -1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix}$$

Pattern pair 3:

$$\begin{bmatrix} -1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

Pattern pair 4:

$$\begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & -1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \end{bmatrix}$$

Weight Matrix

- Add all four individual weight matrices to produce the final weight matrix:

$$\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} -1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} + \begin{bmatrix} 1 & -1 & 1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & -1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & -2 & -2 \\ 2 & -2 & 2 \\ -2 & 2 & -2 \\ -2 & 2 & 2 \end{bmatrix}$$

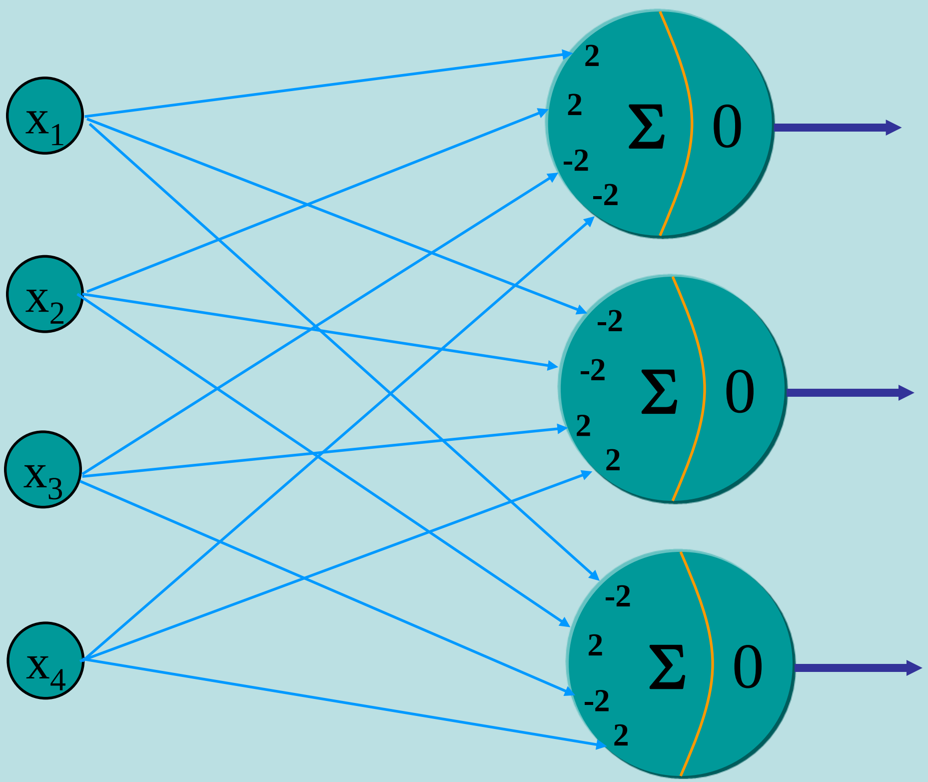
Each column defines the weights for an output neuron

Example Architecture

- General Structure

$$\begin{bmatrix} 2 & -2 & -2 \\ 2 & -2 & 2 \\ -2 & 2 & -2 \\ -2 & 2 & 2 \end{bmatrix}$$

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$



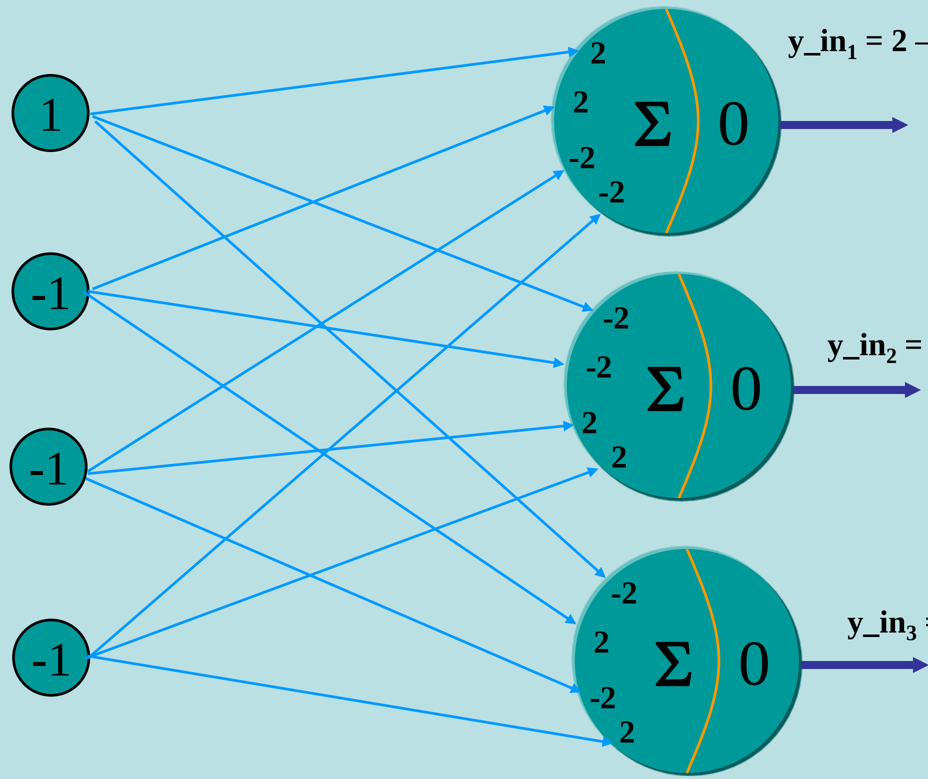
Example Run 1

- Try the first input pattern:

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$

$s_1 = (\ 1\ \ -1\ \ -1\ \ -1)$ $t_1 = (\ 1\ \ -1\ \ -1)$

Where is this information stored?



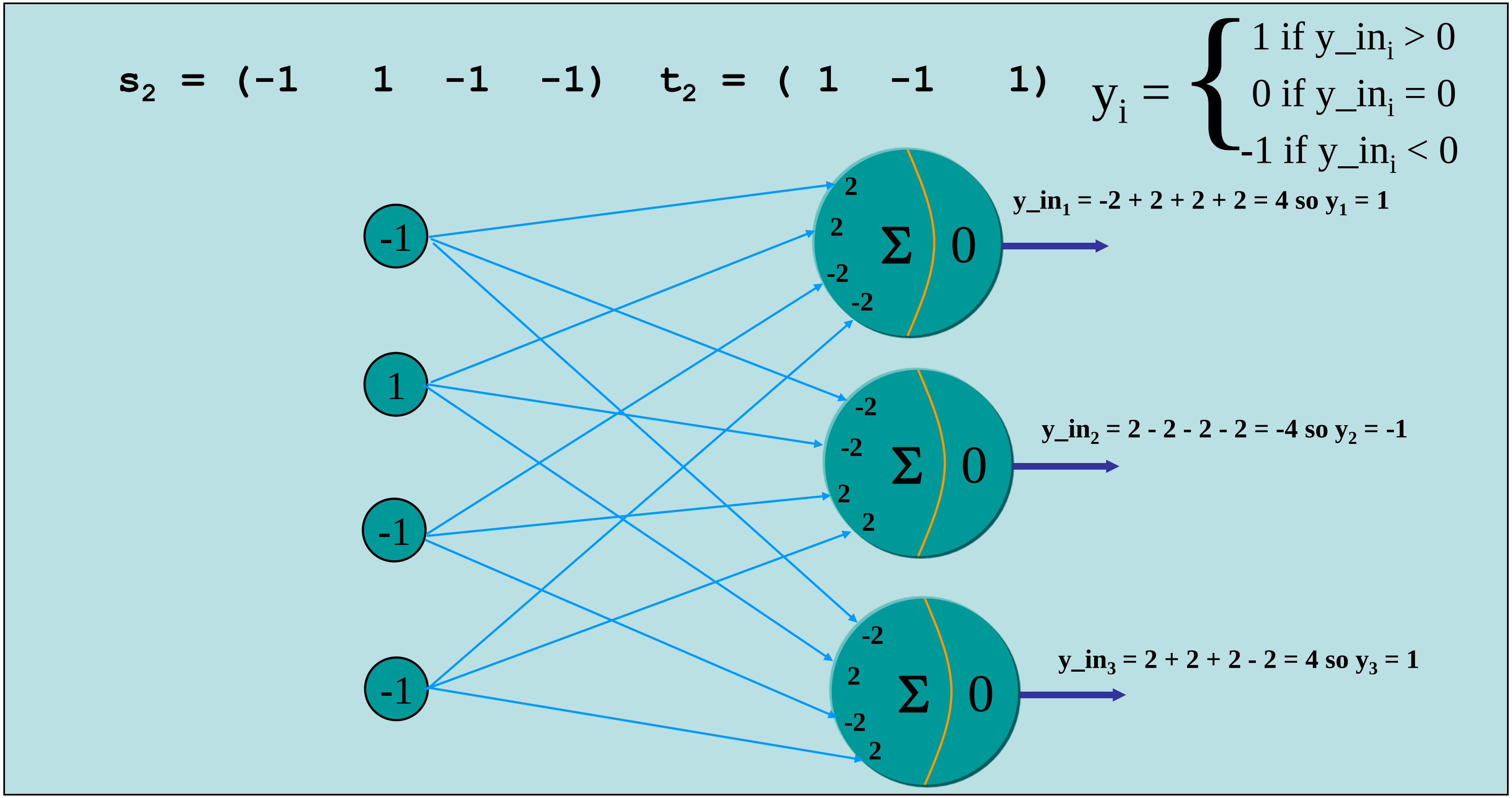
$y\_in_1 = 2 - 2 + 2 + 2 = 4$ so $y_1 = 1$

$y\_in_2 = -2 + 2 - 2 - 2 = -4$ so $y_2 = -1$

$y\_in_3 = -2 - 2 + 2 - 2 = -4$ so $y_3 = -1$

Example Run II

- Try the second input pattern:

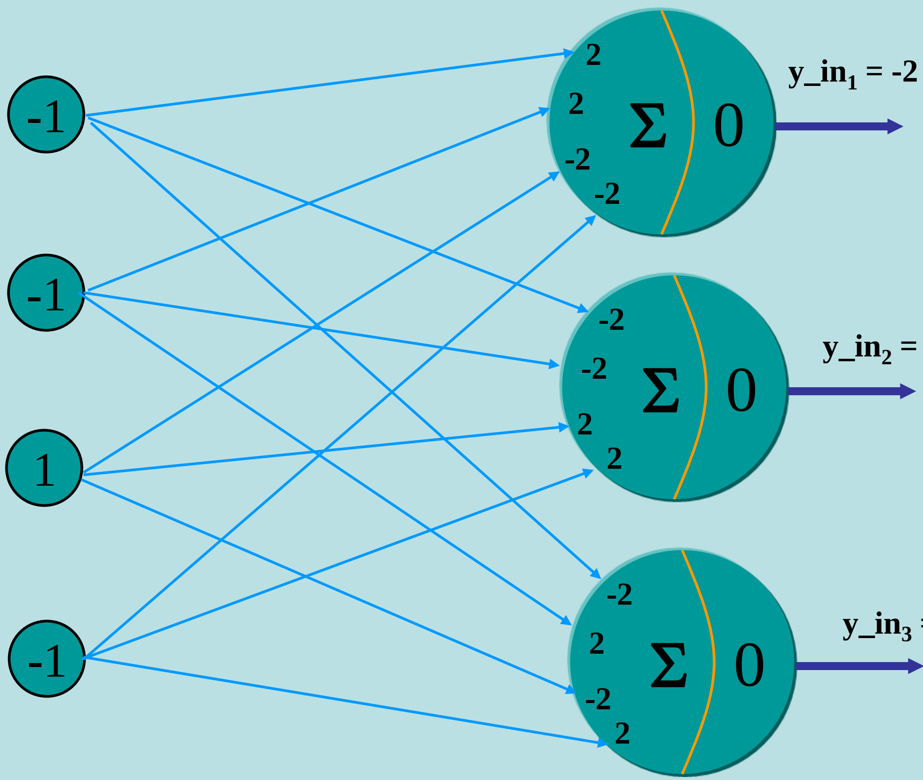


Example Run III

- Try the Third input pattern:

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$

$s_3 = (-1 \quad -1 \quad 1 \quad -1) \quad t_3 = (-1 \quad 1 \quad -1)$



$y\_in_1 = -2 - 2 - 2 + 2 = -4$ so $y_1 = -1$

$y\_in_2 = 2 + 2 + 2 - 2 = 4$ so $y_2 = 1$

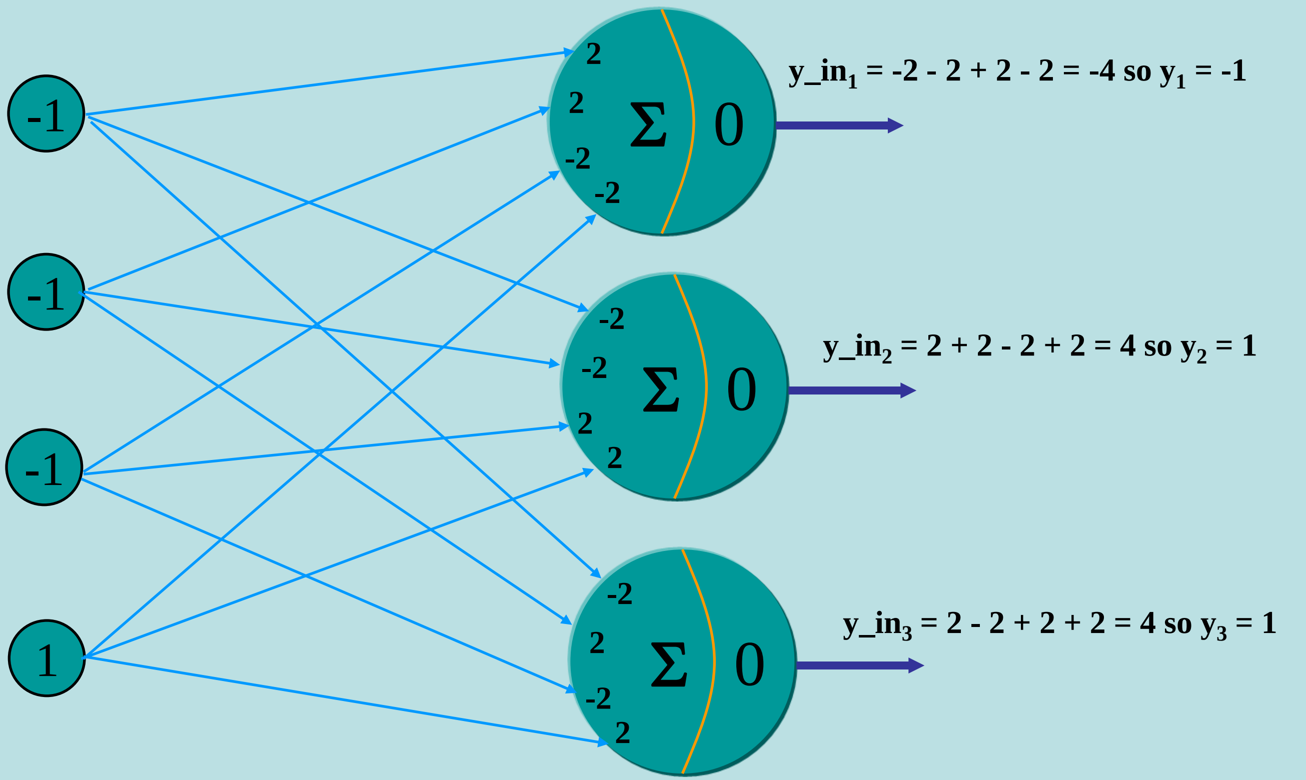
$y\_in_3 = 2 - 2 - 2 - 2 = -4$ so $y_3 = -1$

Example Run IV

- Try the fourth input pattern:

s_4 = (-1 -1 -1 1) t_4 = (-1 1 1)

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$

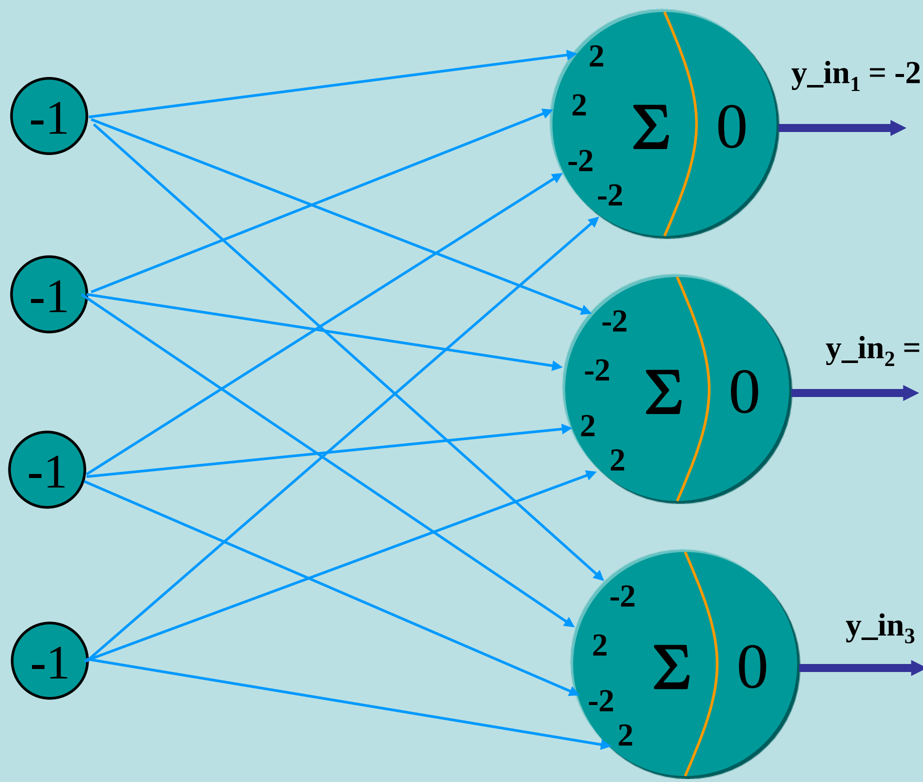


Example Run V

- Try a non-training pattern

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$

What do the 0's mean?



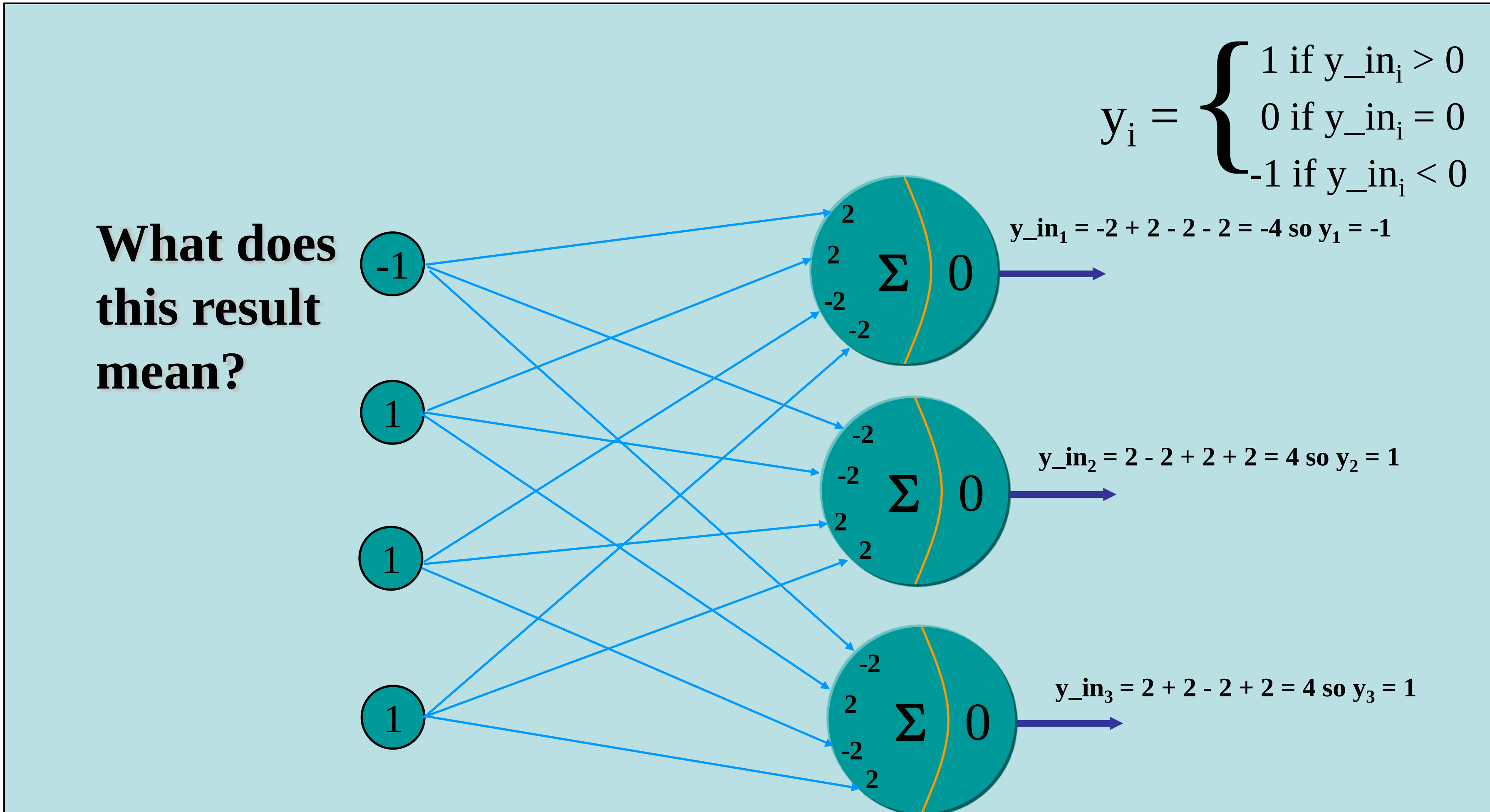
$y\_in_1 = -2 - 2 + 2 + 2 = 0$ so $y_1 = 0$

$y\_in_2 = 2 + 2 - 2 - 2 = 0$ so $y_2 = 0$

$y\_in_3 = 2 - 2 + 2 - 2 = 0$ so $y_3 = 0$

Example Run VI

- Try another non training pattern



Backpropagation

- Backpropagation is the most well know and widely used neural network system

- It is a multi-layered, feedfoward, perceptron-like structure

- Uses the backpropagation rule (or generalized delta rule) for training

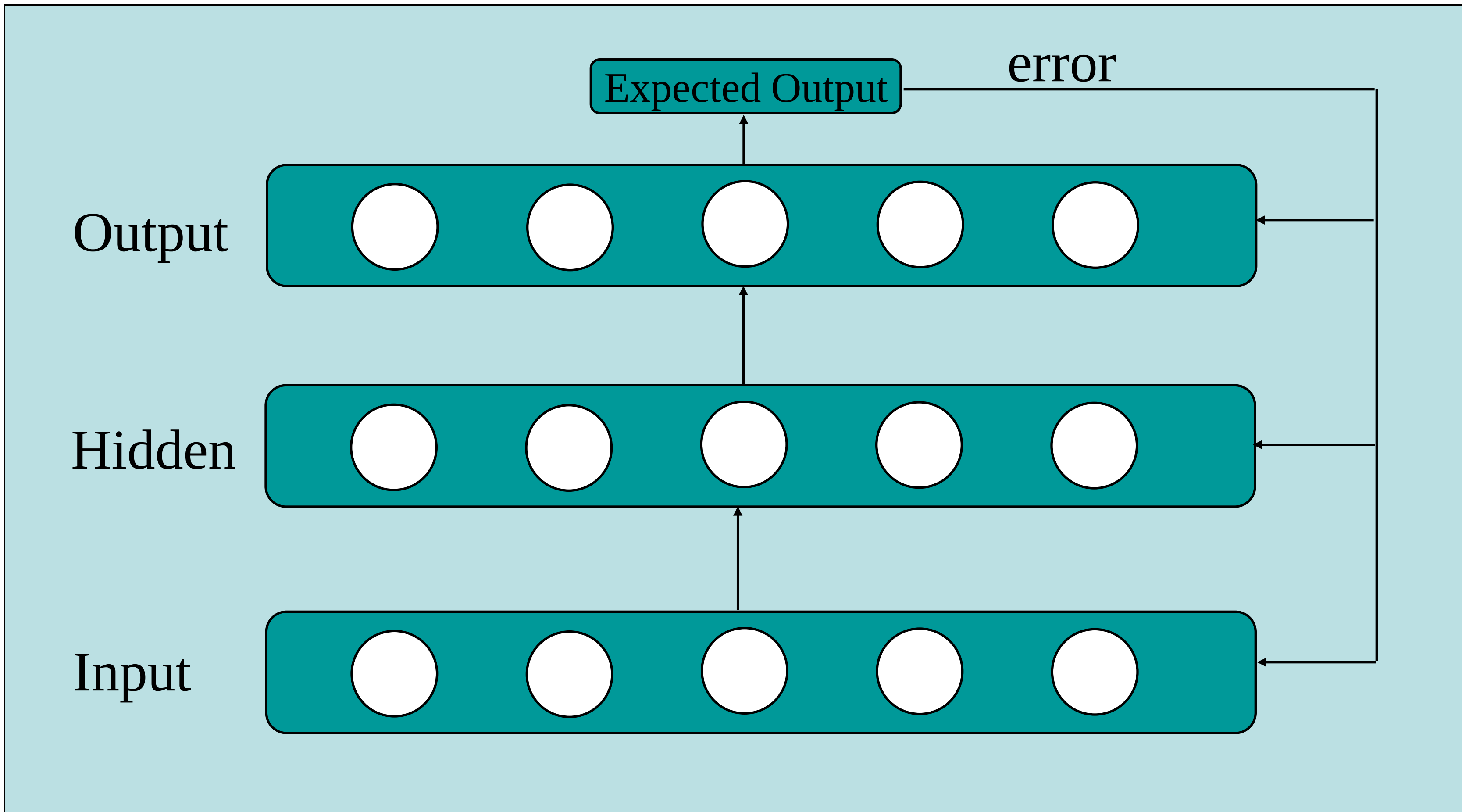
Characteristics

- A multi-layered perceptron has three distinctive characteristics
  - The network contains one or more layers of hidden neurons
  - The network exhibits a high degree of connectivity
  - Each neuron has a smooth (differentiable everywhere) nonlinear activation function, the most common is the sigmoidal nonlinearity:

$$y_j = \frac{1}{1 + e^{s_j}}$$

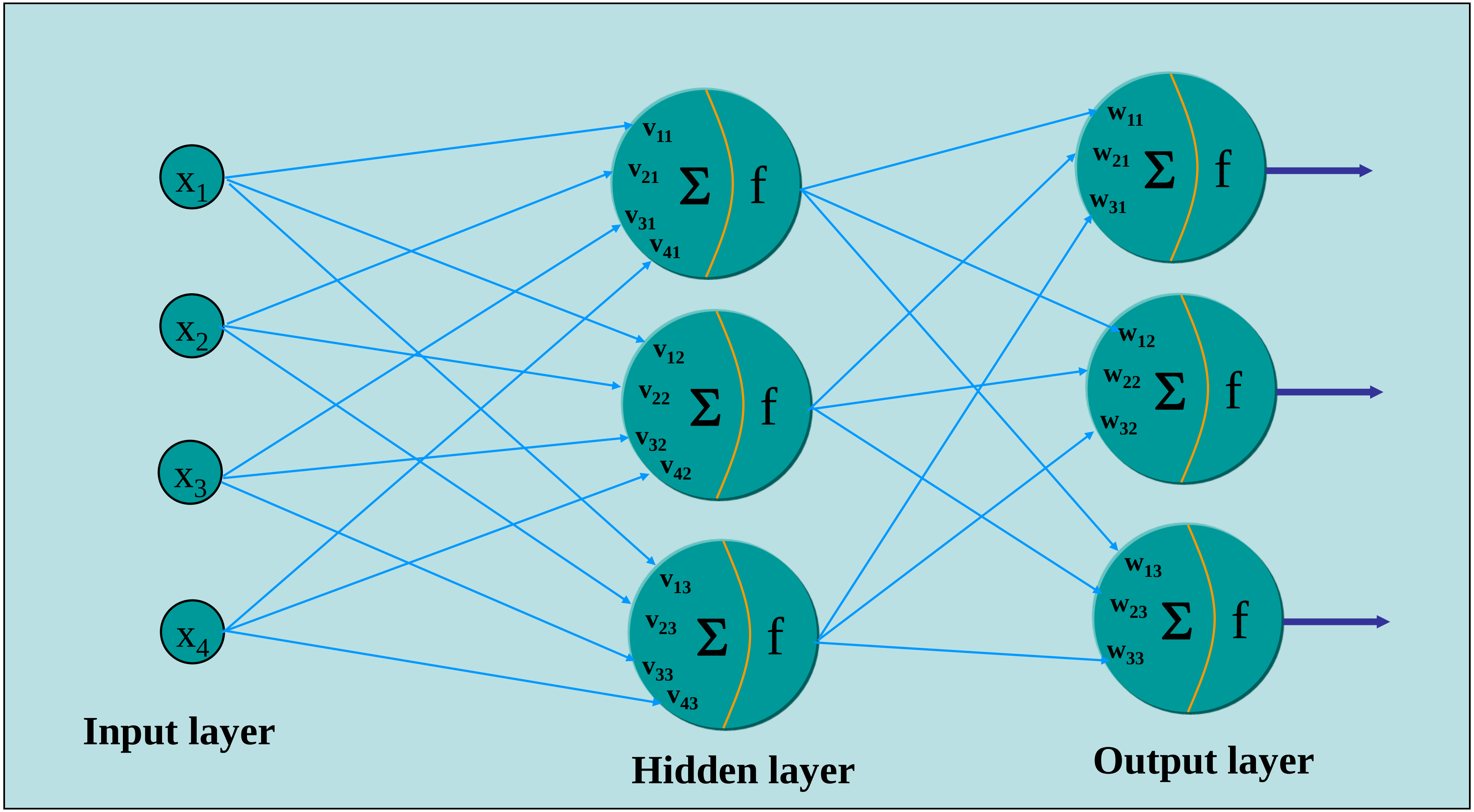
General Structure

- The backpropagation algorithm provides a computational efficient method for training multi-layer networks



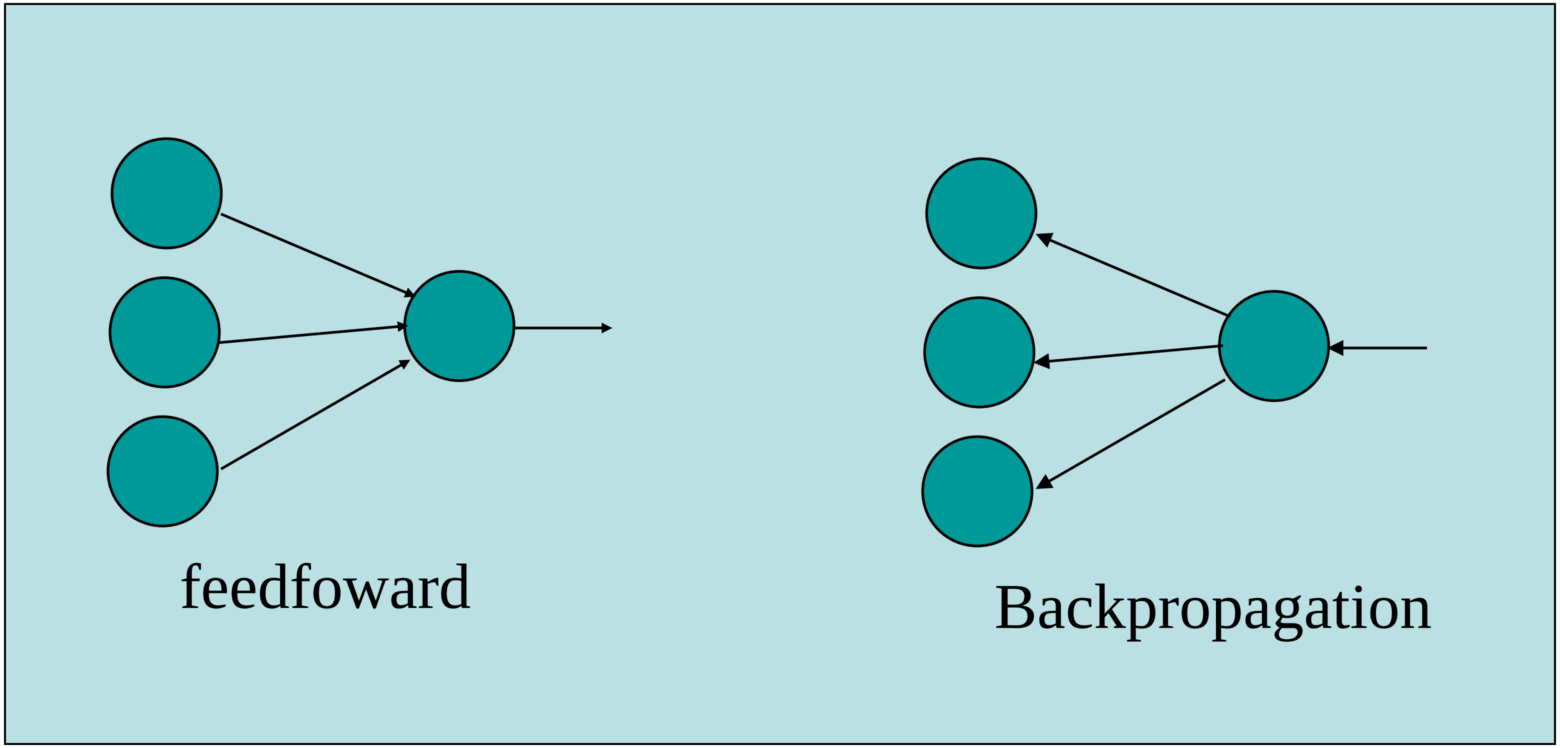
Architecture

- Typical backprop system (bias not shown)



Signal Direction

- Feedforward (for training and normal operation)
- Backpropagation (for error signals)



Activation Function

- The neurons in the BackProp system use a different activation function than the neurons we have studied up to this point
  – Continuous
  – Differentiable
  – Monotonically nondecreasing

- For example, the bipolar sigmoid:

$$f(x) = \frac{2}{1 + e^{-x}} - 1$$

Possible Quiz

What is backpropagation used for?

How do find the final weight matrix?

What kind of neurons are used in a backpropagation system?

SUMMARY

- **More on Hebbian Learning**
- **Heteroassociative Architecture**
- **Backpropagation**