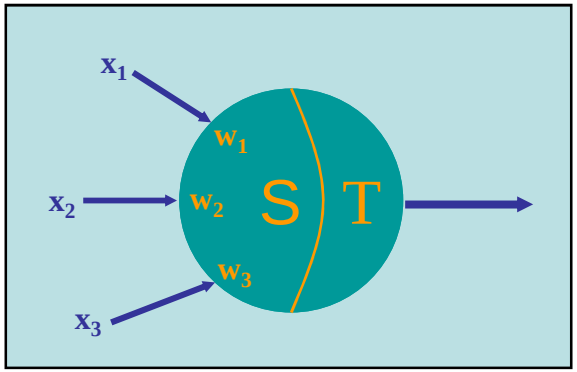
Review

- Features of Neural Nets
- Learning in Neural Nets
- McCulloch/Pitts Neuron and Hebbian Learning

Review – MCP Neuron

- One of the first neuron models to be implemented
- Its output is 1 (fired) or 0
- Each input is weighted with weights in the range -1 to + 1
- It has a threshold value, T



The neuron fires if the following inequality is true:

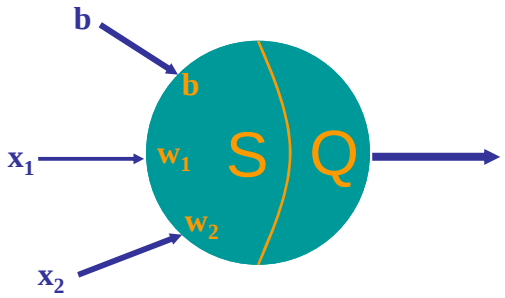
$$x_1w_1 + x_2w_2 + x_3w_3 > T$$

OUTLINE

- Introduction to the Perceptron
- Adaline Network

The Perceptron

- The perceptron was suggest by Rosenblatt in 1958.
- It uses an iterative learning procedure which can be proven to converge to the correct weights for linearly separable data
- It has a bias and a threshold function



Activation function:

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

Perceptron Learning Rule

- Weights are changed only when an error occurs

- The weights are updated using the following:

$$w_i(new) = w_i(old) + \alpha t x_i$$

t is either +1 or -1
α is the learning rate

If an error does not occur, the weights are not changed

Limitations of the Perceptron

The perceptron can only learn to distinguish between classifications if the classes are *linearly separable*.

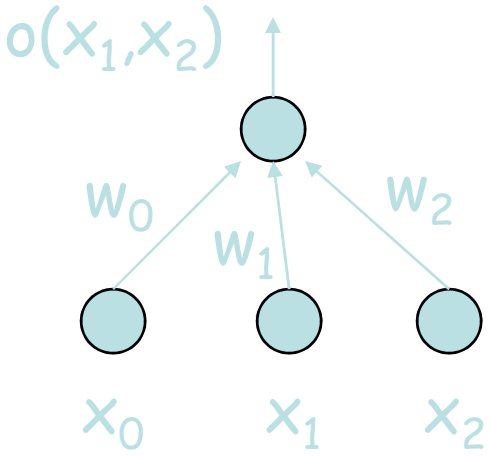
If the problem is not linearly separable then the behaviour of the algorithm is not guaranteed.

If the problem is linearly separable, there may be a number of possible solutions.

The algorithm as stated gives no indication of the quality of the solution found.

XOR Problem

- A perceptron network can not implement an XOR function



$$w_0 + 0.w_1 + 0.w_2 \le 0$$

$$w_0 + 0.w_1 + 1.w_2 > 0$$

$$w_0 + 1.w_1 + 0.w_2 > 0$$

INPUTS ARE DIFFERENT – 1 OUT

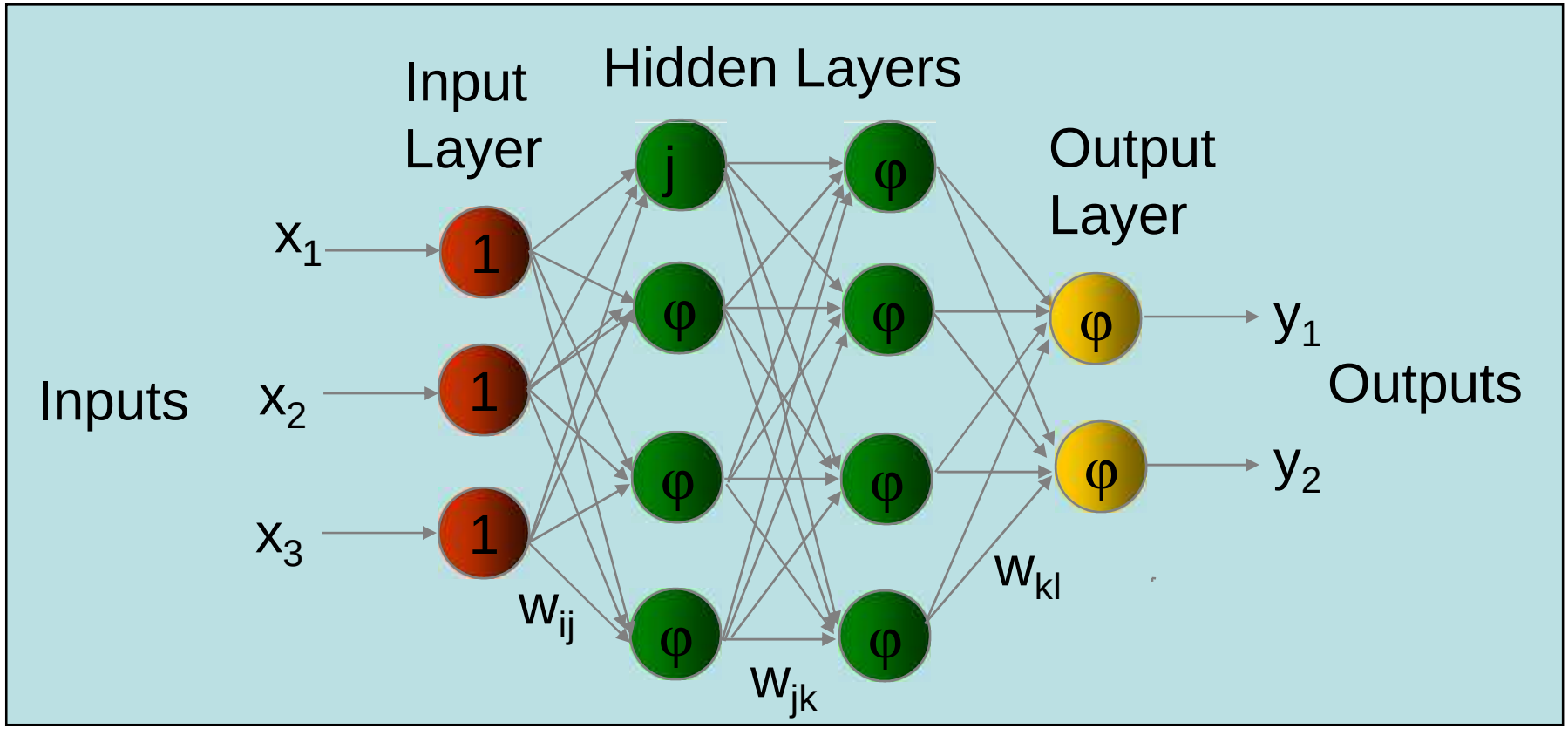
$$w_0 + 1.w_1 + 1.w_2 \le 0$$

XOR(x_1,x_2)

There is no assignment of values to w_0,w_1 and w_2 that satisfies above inequalities. XOR cannot be represented!

MultiLayer Perceptrons

- Perceptrons can be improved if placed in a multilayered network



Adaline Network

- Variation on the Perceptron Network
  - inputs are +1 or -1
  - outputs are +1 or -1
  - uses a bias input

- Differences
  - trained using the Delta Rule which is also known as the least mean squares (LMS) or Widrow-Hoff rule
  - the activation function, during training is the identity function
  - after training the activation is a threshold function

Adaline Algorithm

- **Step 0: initialize the weights to small random values and select a learning rate,** α
- **Step 1: for each input vector s, with target output, t set the inputs to s**
- **Step 2: compute the neuron inputs**
- **Step 3: use the delta rule to update the bias and weights**
- **Step 4: stop if the largest weight change across all the training samples is less than a specified tolerance, otherwise cycle through the training set again**

Neuron input

$$y\_in = b + \Sigma x_i w_i$$

Delta rule

$$b(new) = b(old) + \alpha(t - y\_in)$$
$$w_i(new) = w_i(old) + \alpha(t - y\_in)x_i$$

The Learning Rate, a

- The performance of an ADALINE neuron depends heavily on the choice of the learning rate
  - if it is too large the system will not converge
  - if it is too small the convergence will take to long

- Typically, α is selected by trial and error
  - typical range: $0.01 < \alpha < 10.0$
  - often start at 0.1
  - sometimes it is suggested that:
    $0.1 < n\alpha < 1.0$
    where n is the number of inputs

Running Adaline

- One unique feature of ADALINE is that its activation function is different for training and running

- When running ADALINE use the following:
  - initialize the weights to those found during training
  - compute the net input
  - apply the activation function

Neuron input

$$y\_in = b + \sum x_i w_i$$

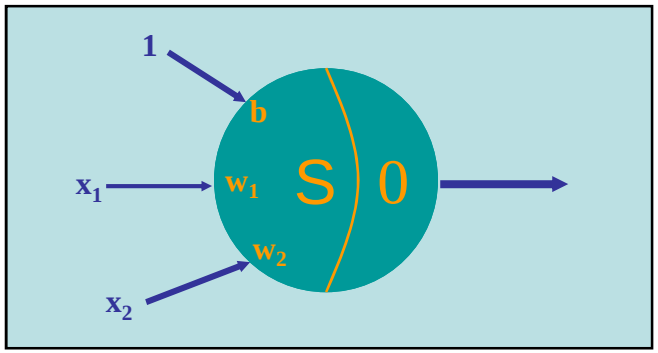
Activation Function

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

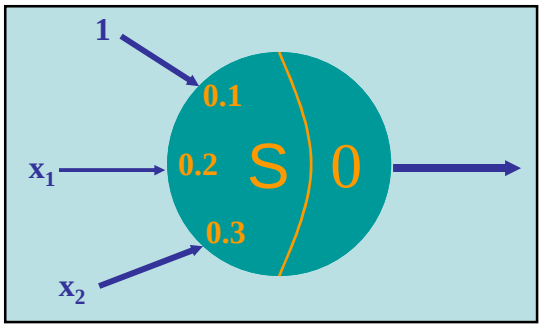
Example – AND function

- Construct an AND function for a ADALINE neuron
  - let $\alpha = 0.1$

x1	x2	bias	Target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

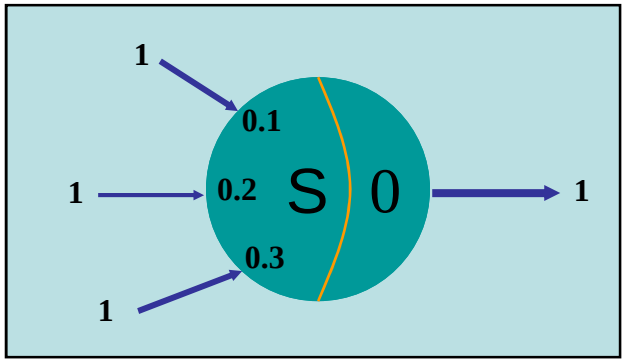


Initial Conditions: Set the weights to small random values:



First Training Run

- Apply the input (1,1) with output 1



The net input is:

$$y\_in = 0.1 + 0.2*1 + 0.3*1 = 0.6$$

The new weights are:

$$b = 0.1 + 0.1(1-0.6) = 0.14$$
$$w_1 = 0.2 + 0.1(1-0.6)1 = 0.24$$
$$w_2 = 0.3 + 0.1(1-0.6)1 = 0.34$$

The largest weight change is 0.04

Neuron input

$$y\_in = b + \Sigma x_i w_i$$

Delta rule

$$b(new) = b(old) + \alpha(t - y\_in)$$
$$w_i(new) = w_i(old) + \alpha(t - y\_in)x_i$$

Second Training Run

- Apply the second training set (1 -1) with output -1

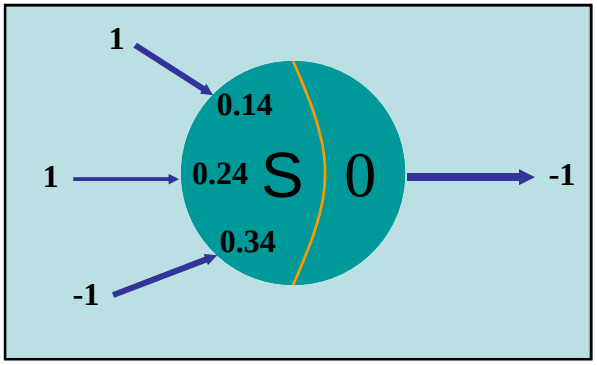
The net input is:

$$y\_in = 0.14 + 0.24*1 + 0.34*(-1) = 0.04$$

The new weights are:

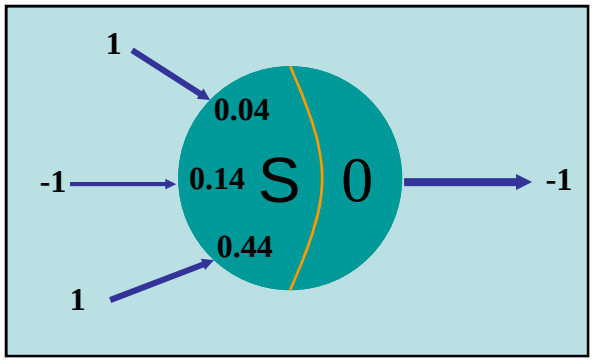
$$b = 0.14 - 0.1(1+0.04) = 0.04$$
$$w_1 = 0.24 - 0.1(1+0.04)1 = 0.14$$
$$w_2 = 0.34 + 0.1(1+0.04)1 = 0.44$$

The largest weight change is 0.1



Third Training Run

- Apply the third training set (-1 1) with output -1



The net input is:

$$y\_in = 0.04 - 0.14*1 + 0.44*1 = 0.34$$

The new weights are:

$$b = 0.04 - 0.1(1+0.34) = -0.09$$

$$w_1 = 0.14 + 0.1(1+0.34)1 = 0.27$$

$$w_2 = 0.44 - 0.1(1+0.34)1 = 0.31$$

The largest weight change is 0.13

Fourth Training Run

- Apply the fourth training set (-1 -1) with output -1

The net input is:

$$y\_in = -0.09 - 0.27*1 - 0.31*1 = -0.67$$

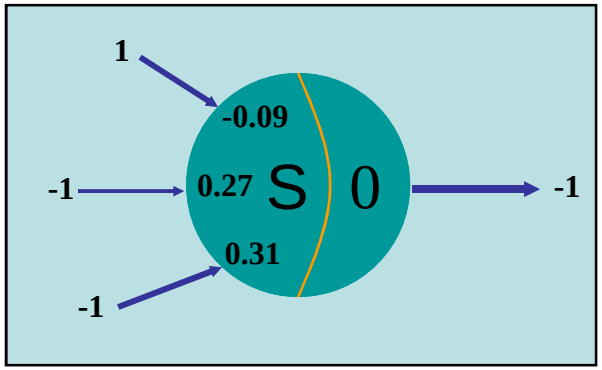
The new weights are:

$$b = -0.09 - 0.1(1+0.67) = -0.27$$

$$w_1 = 0.27 + 0.1(1+0.67)1 = 0.43$$

$$w_2 = 0.31 + 0.1(1+0.67)1 = 0.47$$

The largest weight change is 0.16

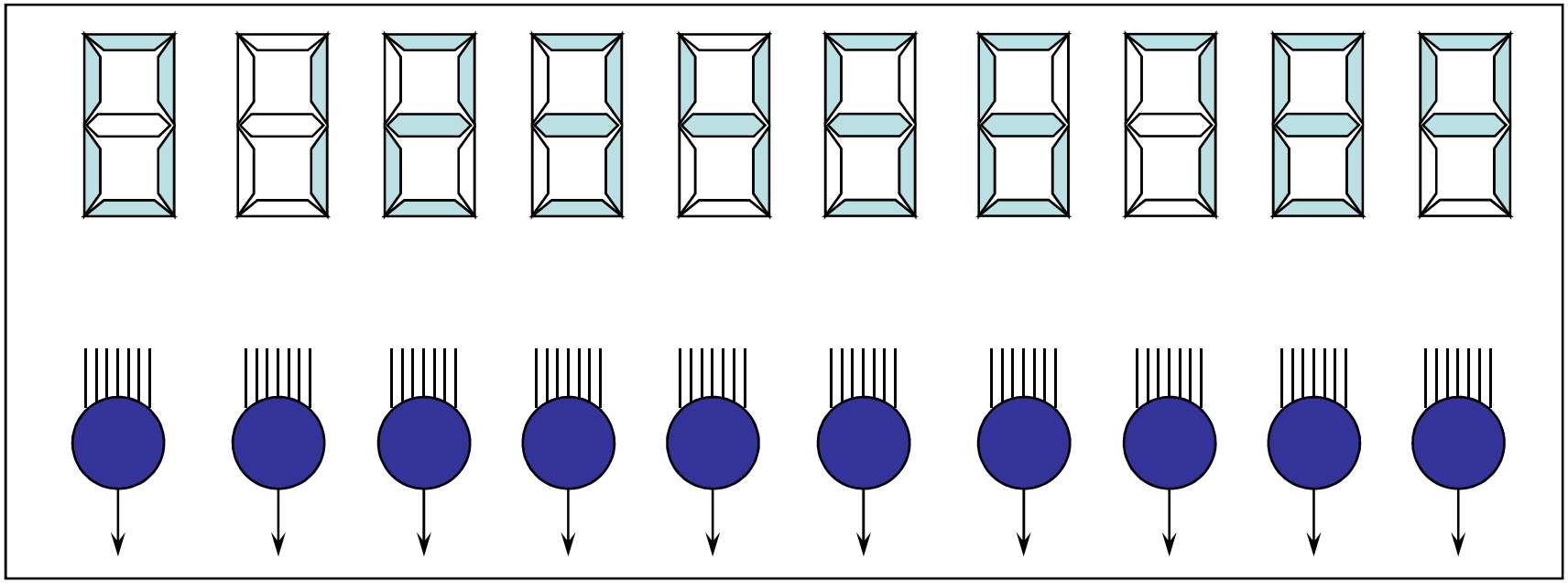


Conclusion

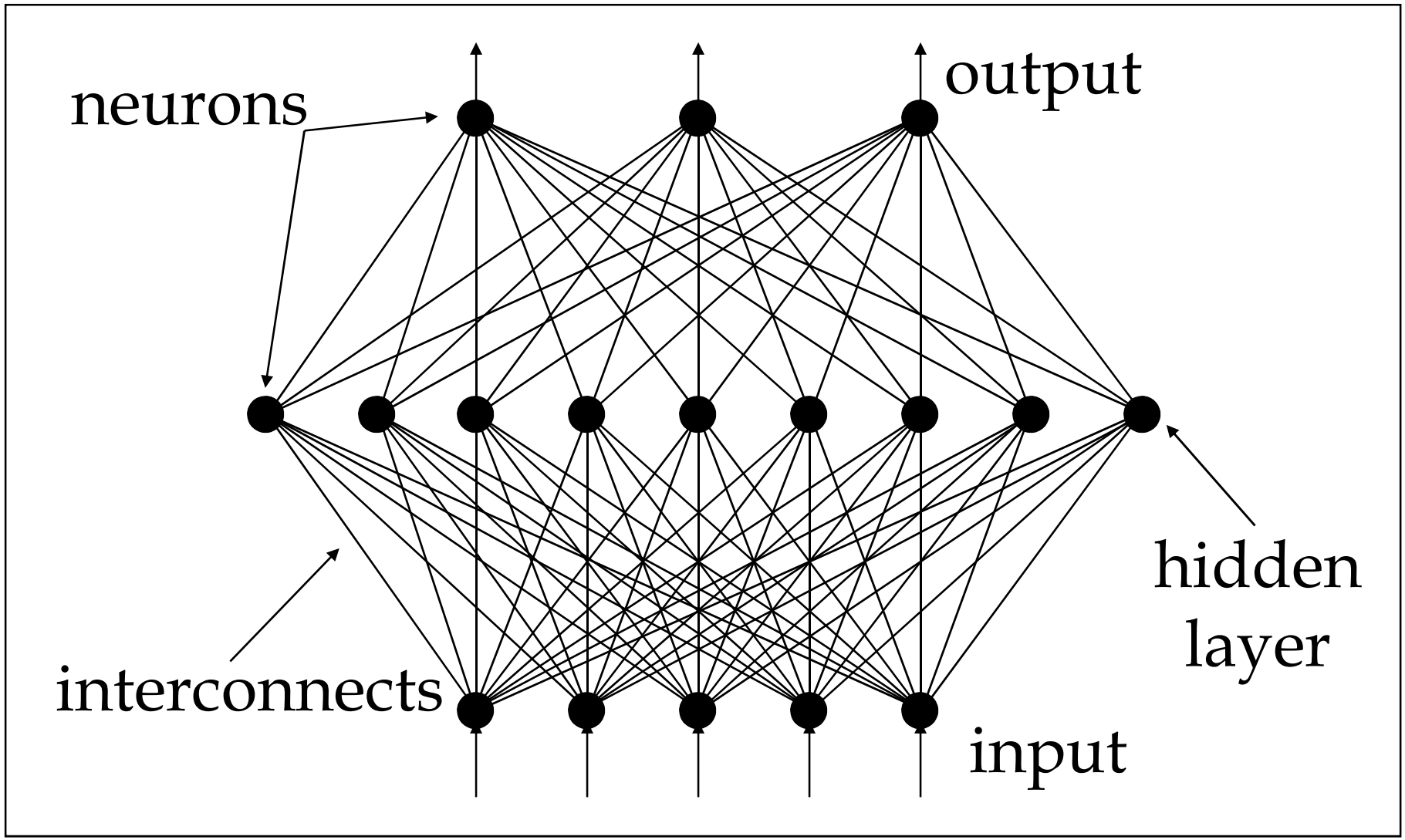
- **Continue to cycle through the four training inputs until the largest change in the weights over a complete cycle is less than some small number (say 0.01)**

- **In this case, the solution becomes: $b = -0.5$, $w_1 = 0.5$, $w_2 = 0.5$**

Example – Recognize Digits



Multi-Layered Net (again)



Properties

- Patterns of activation are presented at the inputs and the resulting activation of the outputs is computed.

- The values of the weights determine the function computed. A network with one hidden layer is sufficient to represent **every Boolean function**. With real weights **every real valued function** can be approximated with a single hidden layer.

Training

- Given Training Data,
  - input vector set :

$$\{ \boldsymbol{i}_n \mid 1 < n < N \}$$

  - corresponding output (target) vector set:

$$\{ \boldsymbol{t}_n \mid 1 < n < N \}$$

- Find the weights of the interconnects using training data to ***minimize the error*** in the test data

Error

- Input, target & response
  - input vector set : { $\boldsymbol{i}_n \mid 1 < n < N$ }
  - target vector set: { $\boldsymbol{t}_n \mid 1 < n < N$ }
  - $\boldsymbol{o}_n$ = neural network output when the input is $\boldsymbol{i}_n$

- Error

$$b = \frac{1}{2} \sum (o_n - t_n)^2$$

Error Minimization Techniques

- The error is a function of the
  - fixed training and test data
  - neural network weights

- Find weights that minimize error (Standard Optimization)
  - conjugate gradient descent
  - random search
  - genetic algorithms
  - steepest descent (error backpropagation)

Possible Quiz

What is a limitation of the perceptron?

What are the names of the layers in a multilayer net?

What is the role of training?

SUMMARY

- **Introduction to the Perceptron**
- **Adaline Network**