

COMP4302/COMP5322, Lecture 2  
**NEURAL NETWORKS**

**Neuron Model and Network Architectures.  
Perceptron.**

Course web page has been moved to:  
<http://www.cs.usyd.edu.au/~comp4302>

---

---

---

---

---

---

---

---

**Neuron Models and Network Architectures - Outline**

- **Single-input neuron**
- **Transfer functions**
- **Multiple-input neuron**
- **Layer of neurons**
- **Multilayer network**
- **An illustrative example**

---

---

---

---

---

---

---

---

**Notation**

- scalars - small *italic* letters:  $a, b, c$
- vectors - small **bold non-italic** letters:  $\mathbf{a}, \mathbf{b}, \mathbf{c}$
- matrices - capital **BOLD non-italic** letters:  $\mathbf{A}, \mathbf{B}, \mathbf{C}$

---

---

---

---

---

---

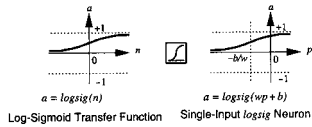
---

---



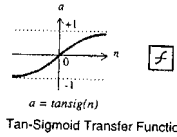
## Sigmoid Transfer Functions

- **Log-sigmoid and tan-sigmoid transfer functions**
  - used in multilayer networks trained with backpropagation



$$a = \frac{1}{1 + e^{-n}}$$

- Output: between 0 and 1



$$a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$$

- Output: between -1 and 1

- Transfer functions - summary; see the handout
- To experiment with a single-input neuron: try `nnd2n1!`

---

---

---

---

---

---

---

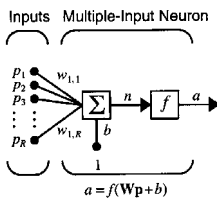
---

---

---

## Multiple-Input Neuron

- A neuron with R inputs



### Weight indices convention:

- 1st index - neuron destination
- 2d index - source of the signal fed to the neuron
- e.g.  $w_{1,2}$  - represents the connection to the 1st neuron from the 2d source

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \dots + w_{1,R}p_R + b$$

$$n = \mathbf{Wp} + b \quad (\text{in matrix form})$$

$$a = f(\mathbf{Wp} + b)$$

---

---

---

---

---

---

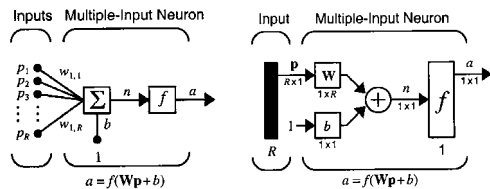
---

---

---

---

## Multiple-Input Neuron – Abbreviated Notation



### Abbreviated Notation

- you can tell immediately if the variable is a scalar, a vector or a matrix & what its dimensionality is
- To experiment with 2-input neuron: try `nnd2n2!`

---

---

---

---

---

---

---

---

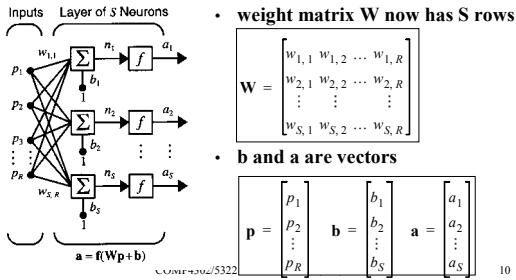
---

---

## Network Architectures

- 1 neuron, even with many inputs may not be sufficient...

### A Layer of Neurons




---

---

---

---

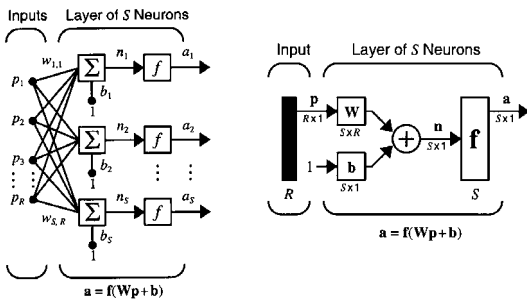
---

---

---

---

### A Layer of Neurons - Abbreviated Notation




---

---

---

---

---

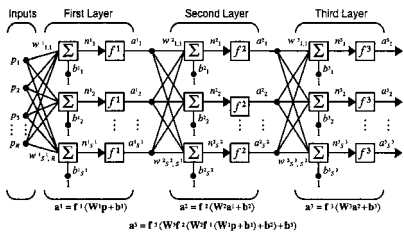
---

---

---

### Multilayer Network

- each layer has its own weight matrix  $W$ , a bias vector  $b$ , a net input vector  $n$  and an output vector  $a$ 
  - to distinguish between the layers - use superscripts:
  - $W^1$  - weight matrix of the 1st layer,  $S^2$  - neurons in the 2d layer, etc.
  - 1 output and 2 hidden layers




---

---

---

---

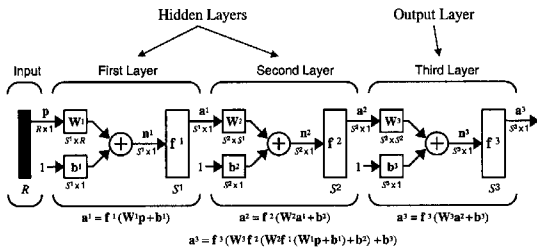
---

---

---

---

## Multilayer Network - Abbreviated Notation



COMP4302/5322 Neural Networks, w2, s2 2003

13

---

---

---

---

---

---

---

---

---

---

## Perceptron - Outline

- Perceptron's neuron model
- Investigation of Perceptron's boundaries
- Perceptron's learning rule
- Example
- Capabilities and limitations

COMP4302/5322 Neural Networks, w2, s2 2003

14

---

---

---

---

---

---

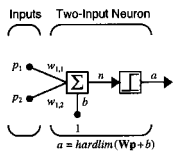
---

---

---

---

## Single-Neuron Perceptron



- 2 inputs  $p1$  and  $p2$  (may have one or more inputs)
- each input is weighted with a weight ( $w11$  and  $w12$ )
- an additional weight  $b$  (bias) associated with the neuron
- the sum of the weighted inputs, together with an input of 1 transmitted through the bias, is sent to a step function
- 2 possible outputs: 0 and 1 (or -1 and 1)

COMP4302/5322 Neural Networks, w2, s2 2003

15

---

---

---

---

---

---

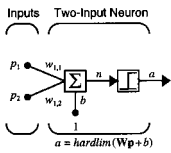
---

---

---

---

## Single-Neuron Perceptron - Investigation of the Boundaries



**• 1. Output of the net:**

$$a = \text{hardlim}(w\mathbf{p} + b) = \text{hardlim}(w_{1,1}p_1 + w_{1,2}p_2 + b)$$

**• 2. Decision boundary:**

$$n = w\mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b = 0$$

**• 3. Assign values for the weights & bias:**

$$w_{1,1} = 1; w_{1,2} = 1; b = -1; \text{MP4302/5322 Neural Networks, w2, s2 2003}$$

**• 4. The decision boundary is:**

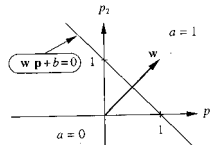
$$n = w\mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b = p_1 + p_2 - 1 = 0$$

i.e. a line in the input space

**• 5. Draw the decision boundary:**

$$p_1 = 0 \Rightarrow p_2 = 1; (p_2 \text{ intersect})$$

$$p_2 = 0 \Rightarrow p_1 = 1; (p_1 \text{ intersect})$$



16

---

---

---

---

---

---

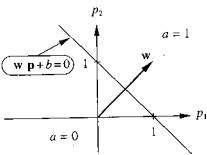
---

---

---

---

## Single-Neuron Perceptron - Investigation of the Boundaries – cont.



**• Properties of the decision boundary:**

- it's always orthogonal to w
- w always points toward the region where the neuron output is 1

**• 6. Find the side corresponding to an output of 1:**

$$\mathbf{p} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \quad a = \text{hardlim}(w\mathbf{p} + b) = \text{hardlim}([1 \ 1] \begin{bmatrix} 2 \\ 0 \end{bmatrix} - 1) = 1$$

i.e. shaded area

To experiment with decision boundaries, try nnd4db!

---

---

---

---

---

---

---

---

---

---

## Perceptron Learning Rule – Derivation. Test Problem

**• Supervised learning:**

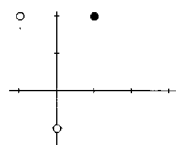
- a set of training examples:  $\{p_1, t_1\}, \{p_2, t_2\}, \dots, \{p_n, t_n\}$
- random initial weights w and b
- after each example, the learning rule adjusts the weights and biases to move the network output closer to the target output
- goal: classify all examples correctly

**• Simple test problem & experimentation with possible rules**

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, t_1 = 1 \right\}$$

$$\left\{ \mathbf{p}_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, t_2 = 0 \right\}$$

$$\left\{ \mathbf{p}_3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, t_3 = 0 \right\}$$



**• infinite number of boundaries**

---

---

---

---

---

---

---

---

---

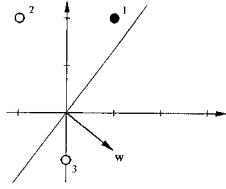
---

### Test Problem - Starting Point

- A perceptron with 2 inputs and no bias
  - Random initial weight vector
- $$w = [1 \quad -0.8]$$

- Presenting the patterns to the perceptron

$$\{p_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, t_1 = 1\}$$



$$a = \text{hardlim}(w p_1) = \text{hardlim}([1 \quad -0.8] \begin{bmatrix} 1 \\ 2 \end{bmatrix}) = \text{hardlim}(-0.6) = 0$$

**Incorrect classification!**

---

---

---

---

---

---

---

---

---

---

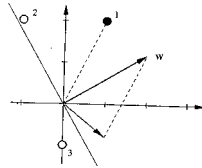
### Test Problem - Tentative Learning Rule

- we need to alter the weight vector so that it points more toward  $p_1$ , so that in the future it has a better chance of classifying it correctly
  - add  $p_1$  to  $w$  - repeated presentations of  $p_1$  would cause the direction of  $w$  to approach the direction of  $p_1$
  - tentative learning rule (rule 1):

$$\text{If } t = 1 \text{ and } a = 0, \text{ then } w^{\text{new}} = w^{\text{old}} + p^T$$

- applying the rule:

$$w^{\text{new}} = w^{\text{old}} + p_1^T = [1 \quad -0.8] + [1 \quad 2] = [2 \quad 1.2]$$




---

---

---

---

---

---

---

---

---

---

### Test Problem - Second Input Vector

$$\{p_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, t_2 = 0\}$$

$$a = \text{hardlim}(w p_2) = \text{hardlim}([2 \quad 1.2] \begin{bmatrix} -1 \\ 2 \end{bmatrix}) = \text{hardlim}(0.4) = 1$$

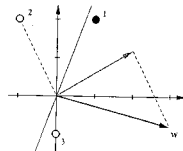
**Incorrect classification!**

- we would like to move the weight vector  $w$  away from the input => rule 2:

$$\text{If } t = 0 \text{ and } a = 1, \text{ then } w^{\text{new}} = w^{\text{old}} - p^T$$

- applying the rule:

$$w^{\text{new}} = w^{\text{old}} - p_2^T = [2 \quad 1.2] - [-1 \quad 2] = [3 \quad -0.8]$$




---

---

---

---

---

---

---

---

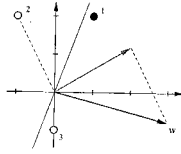
---

---

### Test Problem - Third Input Vector

$$\{p_3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, t_3 = 0\}$$

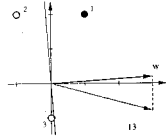
$$a = \text{hardlim}(\mathbf{w}p_3) = \text{hardlim}([3 \quad -0.8] \begin{bmatrix} 0 \\ -1 \end{bmatrix}) = \text{hardlim}(0.8) = 1$$



- apply rule 2:

$$\text{If } t = 0 \text{ and } a = 1, \text{ then } \mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} - \mathbf{p}^T$$

$$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} - p_3^T = [3 \quad -0.8] - [0 \quad -1] = [3 \quad 0.2]$$



Patterns are now correctly classified!

- rule 3:  $\text{If } t = a, \text{ then } \mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}}$

---

---

---

---

---

---

---

---

---

---

### Unified Learning Rule

- covers all combinations of output and target values (0 and 1):

$$\begin{aligned} \text{If } t = 1 \text{ and } a = 0, \text{ then } \mathbf{w}^{\text{new}} &= \mathbf{w}^{\text{old}} + \mathbf{p}^T \\ \text{If } t = 0 \text{ and } a = 1, \text{ then } \mathbf{w}^{\text{new}} &= \mathbf{w}^{\text{old}} - \mathbf{p}^T \\ \text{If } t = a, \text{ then } \mathbf{w}^{\text{new}} &= \mathbf{w}^{\text{old}} \end{aligned}$$

- define:  $e = t - a$

$$\begin{aligned} \text{If } e = 1, \text{ then } \mathbf{w}^{\text{new}} &= \mathbf{w}^{\text{old}} + \mathbf{p}^T \\ \text{If } e = -1, \text{ then } \mathbf{w}^{\text{new}} &= \mathbf{w}^{\text{old}} - \mathbf{p}^T \\ \text{If } e = 0, \text{ then } \mathbf{w}^{\text{new}} &= \mathbf{w}^{\text{old}} \end{aligned}$$

- unified rule:

$$\begin{aligned} \mathbf{w}^{\text{new}} &= \mathbf{w}^{\text{old}} + e\mathbf{p}^T = \mathbf{w}^{\text{old}} + (t-a)\mathbf{p}^T \\ \mathbf{b}^{\text{new}} &= \mathbf{b}^{\text{old}} + e \end{aligned}$$

---

---

---

---

---

---

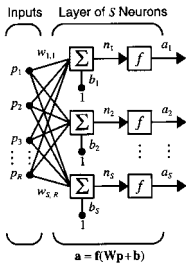
---

---

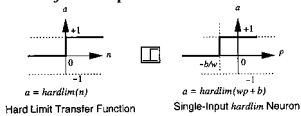
---

---

### A Layer of Perceptrons



where  $f$  is a step function:



- in matrix form for a layer of perceptrons:

$$\begin{aligned} \mathbf{W}^{\text{new}} &= \mathbf{W}^{\text{old}} + e\mathbf{p}^T \\ \mathbf{b}^{\text{new}} &= \mathbf{b}^{\text{old}} + e \end{aligned}$$

---

---

---

---

---

---

---

---

---

---

## Perceptron Learning Law – Summary

1. Initialize weights (including biases) to small random values
2. Choose a random input-output pair  $\{p,t\}$  from the training set
3. Let the network to operate on the input to generate output  $a$
4. Compute the output error  $e=t-a$
5. Update weights:
  - Add a matrix  $\Delta W$  to the weight matrix  $W$ , which is proportional to the product  $ep^T$  between the error vector and the input:
 
$$W^{new} = W^{old} + ep^T$$
  - Add a vector  $\Delta b$  to the bias vector  $b$ , which is proportional to the error vector:
 
$$b^{new} = b^{old} + e$$
6. Choose another random pair and do the correction again
7. Continue until the stopping criteria is satisfied:
  - all examples are correctly classified or a maximum number of epochs is reached

---

---

---

---

---

---

---

---

---

---

## Perceptron – Stopping Criteria

- Stopping criteria is checked at the end of each epoch:
- epoch - one pass through the training set (i.e. each training example is passed once) involving weight adaptation
  - the numbering starts from 1: epoch 1, epoch 2, etc.
- To check if all examples are correctly classified at the end of the epoch:
- all training examples are passed again, the actual activation is calculated and it is compared with the target activation.
  - note: this does not count for another epoch as there is no weight adaptation

---

---

---

---

---

---

---

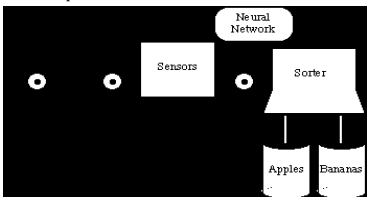
---

---

---

## An Example - Apple/Banana Sorter

- A produce dealer has a warehouse that stores a variety of fruits. He wants a machine that will sort the fruit according to the type...
- There is a conveyor belt on which the fruit is loaded... it is then passed through a set of sensors, which measure 3 properties of the fruit: shape, texture and weight.
- the sensors are somehow primitive :-):
  - shape sensor: -1 if the fruit is round, 1 - if it is more elliptical
  - texture sensor: -1 if the surface is smooth, 1 - if it is rough
  - weight sensor: -1 if the fruit is > 500g, 1 - if < 500g
- The sensor outputs will then be input to a NN...
- NN's purpose: fruit type recognition, so that it is directed to the correct storage bin
- For simplicity - only 2 kinds of fruit




---

---

---

---

---

---

---

---

---

---

## Apple/Banana Example

- How many inputs?
- How many outputs?
- How many perceptrons?
- What are the input vectors?

Training set

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, t_1 = 1 \right\} - \text{banana} \quad \left\{ \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, t_2 = 0 \right\} - \text{apple}$$

Initial weights (random):

$$W = [0.5 \quad -1 \quad -0.5], \quad b = 0.5$$

---

---

---

---

---

---

---

---

---

---

## Apple/Banana Example – First Iteration

Applying  $\mathbf{p}_1$ :

$$a = \text{hardlim}(W\mathbf{p}_1 + b) = \text{hardlim} \left( \begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} + 0.5 \right) =$$

$$= \text{hardlim}(-0.5) = 0$$

$$e = t_1 - a = 1 - 0 = 1$$

Updating the weights:

$$W^{new} = W^{old} + e\mathbf{p}^T = \begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix} + (1) \begin{bmatrix} -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} -0.5 & 0 & -1.5 \end{bmatrix}$$

$$b^{new} = b^{old} + e = 0.5 + 1 = 1.5$$

---

---

---

---

---

---

---

---

---

---

## Apple/Banana Example - Second Iteration

Applying  $\mathbf{p}_2$ :

$$a = \text{hardlim}(W\mathbf{p}_2 + b) = \text{hardlim} \left( \begin{bmatrix} -0.5 & 0 & -1.5 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + 1.5 \right) =$$

$$= \text{hardlim}(2.5) = 1$$

$$e = t_2 - a = 0 - 1 = -1$$

$$W^{new} = W^{old} + e\mathbf{p}^T = \begin{bmatrix} -0.5 & 0 & -1.5 \end{bmatrix} + (-1) \begin{bmatrix} 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} -1.5 & -1 & -0.5 \end{bmatrix}$$

$$b^{new} = b^{old} + e = 1.5 + (-1) = 0.5$$

- End of epoch; check if the stopping criteria is satisfied

---

---

---

---

---

---

---

---

---

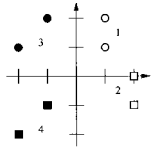
---





### Question

- Can this problem be solved by a perceptron network?



$$\text{class1: } \left\{ p_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, p_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right\}$$

$$\text{class2: } \left\{ p_3 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, p_4 = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \right\}$$

$$\text{class3: } \left\{ p_5 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, p_6 = \begin{bmatrix} -2 \\ 1 \end{bmatrix} \right\}$$

$$\text{class4: } \left\{ p_7 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, p_8 = \begin{bmatrix} -2 \\ -2 \end{bmatrix} \right\}$$

---

---

---

---

---

---

---

---

---

---

### Layer of Perceptrons - Capabilities

- Each perceptron defines one decision boundary, i.e. one line
- A layer of perceptrons can classify input vectors into many categories
  - Each category is represented by a different output vector
  - Maximum number of categories  $2^S$ , where S is the number of perceptrons in the layer
- A layer of perceptrons can be used to solve more difficult linearly separable problems
  - guaranteed to converge to a solution in a finite number of steps, if a solution exist (i.e. the input patterns are separated with several lines)

---

---

---

---

---

---

---

---

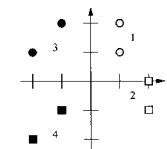
---

---

### For Homework

- Solve this classification problem with the perceptron rule. Apply each input vector in order, for as many repetitions as it takes to ensure that the problem is solved. Draw a graph of the problem only after you have found a solution. Use the following initial weights and bias:

$$W(0) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad b(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$



$$\text{class1: } \left\{ p_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, p_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right\}$$

$$\text{class2: } \left\{ p_3 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, p_4 = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \right\}$$

$$\text{class3: } \left\{ p_5 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, p_6 = \begin{bmatrix} -2 \\ 1 \end{bmatrix} \right\}$$

$$\text{class4: } \left\{ p_7 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, p_8 = \begin{bmatrix} -2 \\ -2 \end{bmatrix} \right\}$$

---

---

---

---

---

---

---

---

---

---