*Philadelphia University*
*Faculty of Engineering*


# Marking Scheme

Exam Paper

BSc CE

## Neural Networks and Fuzzy Logic (630514)

Second Exam Summer semester Date: 16/08/2017

Section 1

Weighting 20% of the module total


Lecturer: Dr. Qadri Hamarsheh

Coordinator: Dr. Qadri Hamarsheh

Internal Examiner: Dr. Mohammed Mahdi

Marking Scheme

# Neural Networks and Fuzzy Logic (630514)

The presented exam questions are organized to overcome course material through 4 questions.
The *all questions* are compulsory requested to be answered.

**Marking Assignments**

**Question 1** This question is attributed with 8 marks if answered properly; the answers are as following:

1) **Input units** of a Neural Network can be **adjusted** during a learning process.

| | | | |
|---|---|---|---|
| **a)** | **True** | **b)** | **False** |

2) State whether **Hebb's law** is **supervised** learning or **unsupervised** type?
   - **a) Supervised**
   - **b) Unsupervised**
   - **c) Either supervised or unsupervised**
   - **d) Can be both supervised and unsupervised**

3) In **Hebbian** learning, the **initial** weights are set?

| | | | |
|---|---|---|---|
| **a)** | **To zero** | **b)** | **Random** |
| **c)** | **Near to target value** | **d)** | **None of the above** |

4) In a three layer network, **shape** of dividing surface (decision boundary) is determined by?

| | |
|---|---|
| **a)** | **Number of units in second layer** |
| **b)** | **Number of units in third layer** |
| **c)** | **Number of units in second and third layer** |
| **d)** | **None of the mentioned** |

5) What is the biggest difference between **Widrow & Hoff's Delta Rule** and the **Perceptron Learning Rule** for learning in a single-layer feedforward network?

| | |
|---|---|
| **a)** | There is no difference. |
| **b)** | The Delta Rule is defined for step activation functions, but the Perceptron Learning Rule is defined for linear activation functions. |
| **c)** | The Delta Rule is defined for sigmoid activation functions, but the Perceptron Learning Rule is defined for linear activation functions. |
| **d)** | The Delta Rule is defined for linear activation functions, but the Perceptron Learning Rule is defined for step activation functions. |

6) What is **gradient descent**?

| | |
|---|---|
| **a)** | **Method to find the absolute minimum of a function** |
| **b)** | **Method to find the absolute maximum of a function** |
| **c)** | **Maximum or minimum, depends on the situation** |
| **d)** | **None of the mentioned** |

7) The number of fundamental memories $M_{max}$ (**Most perfectly retrieved**) that can be stored in the **n-neuron** Hopfield network is limited by

| | | | |
|---|---|---|---|
| **a)** | $M_{max} = 0.15\, n$ | **b)** | $M_{max} = \frac{n}{4 \ln n}$ |
| **c)** | $M_{max} = \frac{n}{2 \ln n}$ | **d)** | **None of above** |

8) What is **asynchronous** update in a network?

| | |
|---|---|
| **a)** | **Update to all units is done at the same time** |
| **b)** | **Change in state of any number of units drive the whole network** |
| **c)** | **Change in state of any one unit drive the whole network** |
| **d)** | **None of the mentioned** |

**Question 2** This question is attributed with 4 marks if answered properly; the answers are as following:

**Solution**

**Step 1: Initialization**

**Set initial weights $w_1$, $w_2$,…, $w_n$ and threshold $\theta$ to random numbers in the range *[-0.5, 0.5]*.**

**Step 2: Activation**

**Activate the perceptron by applying inputs $x_1(p)$, $x_2(p)$,…, $x_n(p)$ and desired output $Y_d$ $(p)$. Calculate the actual output at iteration $p = 1$**

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

**Where n is the number of the perceptron inputs, and step is a step activation function.**

**Step 3: Weight training**

**Update the weights of the perceptron**

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

**where $\Delta w_i(p)$ is the weight correction at iteration *p*.**

$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

**The weight correction is computed by the delta rule:**

**Step 4: Iteration**

**Increase iteration *p* by one, go back to *Step 2* and repeat the process until convergence.**

**Question 3** This question is attributed with 5 marks if answered properly; the answers are as following:

**Solution**

**a) Forward Pass**

**Input to top neuron = (0.1x0.1)+(0.7x0.5)=0.36. Out = 0.589.**

**Input to bottom neuron = (0.1x0.3)+(0.7x0.2) = 0.17. Out = 0.5424.**

**Input to final neuron = (0.589x0.2)+(0.5424x0.1) = 0.17204. Out = 0.5429.**

**b) Reverse Pass**

**Output error $\delta$ =(t-o)(1-o)o = (1-0.5429)(1-0.5429)0.5429 = 0.11343.**

**New weights for output layer**

**w1(+) = w1+( $\delta$ x input) = 0.2+(0.11343x0.589) = 0.2668.**

**w2(+) = w2+( $\delta$ x input) = 0.1+(0.11343x0.5424)=0.16152.**

**Errors for hidden layers:**

**$\delta$ 1 = $\delta$ xw1 = 0.11343x0.2668= 0.030263(x(1-o)o)= 0.007326.**

**$\delta$ 2= $\delta$ xw2= 0.11343x0.16152=0.018321(x(1-o)o)= 0.004547.**

**New hidden layer weights:**

**w3(+)=0.1+(0.007326x0.1)=0.1007326.**

**w4(+) = 0.5+(0.007326x0.7)=0.505128.**

**w5(+)=0.3+(0.004547x0.1)=0.3004547.**

**w6(+) = 0.2+(0.004547x0.7)=0.20318.**

**Question 4** This question is attributed with 3 marks if answered properly; the answers are as following:

$(a)\ AND\ (x_1 \cap$ **Solution**

$$W = x_1x_1^T + x_2x_2^T = \begin{bmatrix}1\\-1\\-1\\1\end{bmatrix}\begin{bmatrix}1 & -1 & -1 & 1\end{bmatrix} + \begin{bmatrix}-1\\1\\1\\-1\end{bmatrix}\begin{bmatrix}-1 & 1 & 1 & -1\end{bmatrix}$$

$$W = \begin{bmatrix}1 & -1 & -1 & 1\\-1 & 1 & 1 & -1\\-1 & 1 & 1 & -1\\1 & -1 & -1 & 1\end{bmatrix} + \begin{bmatrix}1 & -1 & -1 & 1\\-1 & 1 & 1 & -1\\-1 & 1 & 1 & -1\\1 & -1 & -1 & 1\end{bmatrix} = \begin{bmatrix}2 & -2 & -2 & 2\\-2 & 2 & 2 & -2\\-2 & 2 & 2 & -2\\2 & -2 & -2 & 2\end{bmatrix}$$

Since we demand that all wii = 0, we get:

$$W = \begin{bmatrix}0 & -2 & -2 & 2\\-2 & 0 & 2 & -2\\-2 & 2 & 0 & -2\\2 & -2 & -2 & 0\end{bmatrix}$$