AUTUMN MID SEMESTER EXAMINATION-2023

School of Computer Engineering
Kalinga Institute of Industrial Technology, Deemed to be University
Subject Name: Computational Intelligence
[Subject Code: CS 3031]

Time: 1 1/2 Hours **Full Mark: 40**

Answer Any four Questions including Question No. 1 which is compulsory.
The figures in the margin indicate full marks. Candidates are required to give their answers in their own words as far as practicable and all parts of a question should be answered at one place only.

1. Answer all the questions. [2 x 5]

a) Differentiate hard computing and soft computing?

Hard Computing	Soft Computing
1. Hard computing approaches aim to provide precise, deterministic solutions to problems.	Soft computing approaches can tolerate uncertainty, ambiguity imprecision, vague, and partial truth inputs and provide approximate solutions to problems.
2. Hard computing often employs well-defined symbolic logic, mathematical equations, and algorithms.	Soft computing techniques aim to find approximate or probabilistic solutions.
3. Hard computing methods typically do not adapt or learn from data. They follow predefined rules and algorithms without modification.	Soft computing systems are adaptive and can learn from data. They improve their performance over time and can handle changing environments.
4. It uses two-valued logics to handle precise data.	It uses multi-valued fuzzy logic to handle imprecise and uncertainty data.
5. It can deal with precise data.	It can deal with noisy data.
6. It has a stochastic nature.	It has a deterministic nature.

b) Consider a fully connected neural network consisting of two hidden layers, each containing 5 neurons. The input to this network is represented by a 4D feature vector and is designated for a classification task with 3 classes. Calculate the number of connection weights and biases in the network.

Ans: Neural Networks model = [4: 5: 5: 3] has 4 layers. The number of parameters (P) are weights and biases in the network. We can use following formula to find out number of nodes in the network.

$$P = \sum_{i=1}^{L-1} p_{i+1}(p_i + 1)$$

Where p_i is the number of nodes in the i^{th} $(1 \leqslant i \leqslant L)$ layer.

P = 5 * (4 + 1) + 5 * (5 + 1) + 3 * (5 + 1) = 73

c) How do you describe ReLU non-linearity function?
   a) Continuous and Non-differentiable
   b) Discontinuous and Non-differentiable
   c) Continuous and Differentiable
   d) Discontinuous and Differentiable

   Ans: (a) or (c)

   Note# The derivative of ReLU is:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$ and undefined at $x=0$

   The reason for it being undefined at $x=0$, is that its left- and right derivative are not equal.

d) Given input signals, x1=-0.11, x2 = 0.22, desired response, d= -1, synaptic weights, w1 = -0.17, w2 = 0.53, learning rate = 0.65, make necessary correction to the weights using LMS weight update rule.
   Ans: Weights correction using LMS algorithm as follows:

$$\mathbf{w}_{new} = \mathbf{w}_{old} + \eta.e.\mathbf{x}$$
$$\mathbf{w}_{new} = \mathbf{w}_{old} + \eta.(d - y).\mathbf{x}$$
$$\mathbf{w}_{new} = \mathbf{w}_{old} + \eta.(d - w^T x).\mathbf{x}$$
$$\mathbf{w}_{new} = \begin{bmatrix} \mathbf{-0.17} \\ \mathbf{0.58} \end{bmatrix} + 0.65.(-1 - [-0.17 - 0.58] \begin{bmatrix} \mathbf{-0.11} \\ \mathbf{0.22} \end{bmatrix}). \begin{bmatrix} \mathbf{-0.11} \\ \mathbf{0.22} \end{bmatrix}$$
$$\mathbf{w}_{old} = \begin{bmatrix} \mathbf{-0.17} \\ \mathbf{0.58} \end{bmatrix} \text{ and } \mathbf{w}_{new} = \begin{bmatrix} \mathbf{-0.089} \\ \mathbf{0.368} \end{bmatrix}$$

e) What factors influence the number of epochs needed to achieve convergence in a Perceptron training process?
   Ans: Following factors affect the number of epochs to achieve convergence.
   1. The learning rate determines the step size the Perceptron takes during each update.
   2. The scale of the input features can affect convergence.
   3. The initial values of the weights can affect convergence.
   4. The stopping criteria used in the training process can also influence the number of epochs.

2. What is linearly separable problem? Discuss the rules for adapting synaptic weights of perceptron. Write down the Perceptron learning algorithm and explain each step. Find the new weights after epoch-1 to classify OR function with bipolar input and targets. Assume initial weight w1=w2=b=0. learning rate = 1 and threshold = 0.

[10 Marks]

Linearly Separable: if the data points can be separated using a line, linear function, or flat hyper-plane are considered linearly separable.

Algorithm 1 Perceptron Learning

```
w = [w0, w1, w2, . . . , wn]
x = [1, x1, x2, . . . , xn]
P ← input with labels 1;
N ← input with labels -1;
Initialize w randomly;
while !convergence do
      Pick random x ∈ P ∪ N
      if x ∈ P and w^T x < 0 then
            w = w + x
      if x ∈ N and w^T x ≥ 0 then
            w = w − x
end
```

Here the perceptron learning rule can be written as

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \le y_{in} \le \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

If $y \neq t$, then

$$w(\text{new}) = w(\text{old}) + \alpha t x \quad (\alpha - \text{learning rate})$$

else, we have

$$w(\text{new}) = w(\text{old})$$

Algorithm 2: Perceptron Convergence Algorithm

TABLE 1.1 Summary of the Perceptron Convergence Algorithm

Variables and Parameters:

$\mathbf{x}(n)$ = $(m + 1)$-by-1 input vector
$= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$
$\mathbf{w}(n)$ = $(m + 1)$-by-1 weight vector
$= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T$
b = bias
$y(n)$ = actual response (quantized)
$d(n)$ = desired response
η = learning-rate parameter, a positive constant less than unity

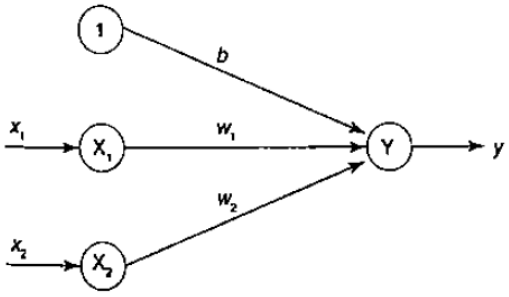
1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \ldots$.
2. *Activation.* At time-step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response.* Compute the actual response of the perceptron as
$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$
where sgn(·) is the signum function.
4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain
$$\mathbf{w}(n + 1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$
where
$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$
5. *Continuation.* Increment time step n by one and go back to step 2.

The perceptron learning rule can be written as

W(new) = w(old) + learning_rate * (d – y) * x

Note# any one algorithm written by the students may be considered for full mark.

Bipolar OR



x1	x2	x1 OR x2
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

Perceptron Learning Rule: Wnew = Wold + Δw where Δw = η * t *x

Input			Target	net input	calculated output	weight changes			Weights		
x0 or b	x1	x2	t or d	yin	y	Δw0 or Δb	Δw1	Δw2	w0 or b	w1	w2
1	?	?	?	w0*x0 + w1*x1 + w2*x2	f(yin)	η * t *x0	η * t *x1	η * t *x2	b + Δb	w1 + Δw1	w2 + Δw2
									0	0	0
1	-1	-1	-1	0	0	-1	1	1	-1	1	1
1	-1	1	1	-1	-1	1	-1	1	0	0	2
1	1	-1	1	-2	-1	1	1	-1	1	1	1
1	1	1	1	3	1	0	0	0	1	1	1

x1	x2	x1 OR x2
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

Perceptron Learning Rule: Wnew = Wold + Δw where Δw = η * t *x

Input			Target	net input	calculated output	weight changes			Weights		
x0 or b	x1	x2	t or d	yin	y	Δw0 or Δb	Δw1	Δw2	w0 or b	w1	w2
1	?	?	?	w0*x0 + w1*x1 + w2*x2	f(yin)	η * t *x0	η * t *x1	η * t *x2	b + Δb	w1 + Δw1	w2 + Δw2
									0	0	0
1	1	1	1	0	0	1	1	1	1	1	1
1	1	-1	1	1	1	0	0	0	1	1	1
1	-1	1	1	1	1	0	0	0	1	1	1
1	-1	-1	-1	-1	-1	0	0	0	1	1	1

As per the perceptron convergence algorithm

Bipolar OR

x1	x2	x1 OR x2
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

Perceptron Learning Rule: Wnew = Wold + Δw where Δw = η * (t-y) *x

Input			Target	net input	calculated output	weight changes			Weights		
x0 or b	x1	x2	t or d	yin	y	Δw0 or Δb	Δw1	Δw2	w0 or b	w1	w2
1	?	?	?	w0*x0 + w1*x1 + w2*x2	f(yin)	η * (t-y) *x0	η * (t-y) *x1	η * (t-y) *x2	b + Δb	w1 + Δw1	w2 + Δw2
									0	0	0
1	-1	-1	-1	0	0	-1	1	1	-1	1	1
1	-1	1	1	-1	-1	2	-2	2	1	-1	3
1	1	-1	1	-3	-1	2	2	-2	3	1	1
1	1	1	1	5	1	0	0	0	3	1	1
1	-1	-1	-1	1	1	-2	2	2	1	3	3
1	-1	1	1	1	1	0	0	0	1	3	3

x1	x2	x1 OR x2
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

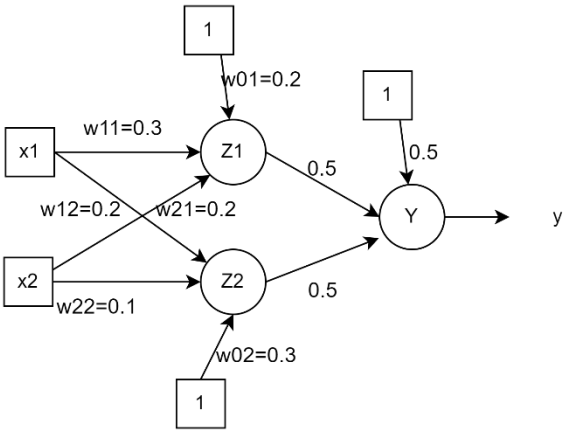
Perceptron Learning Rule: Wnew = Wold + Δw where Δw = η * (t-y) *x

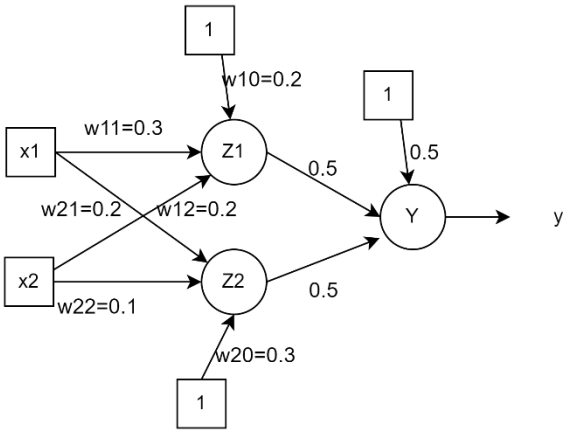
Input			Target	net input	calculated output	weight changes			Weights		
x0 or b	x1	x2	t or d	yin	y	Δw0 or Δb	Δw1	Δw2	w0 or b	w1	w2
1	?	?	?	w0*x0 + w1*x1 + w2*x2	f(yin)	η * (t-y) *x0	η * (t-y) *x1	η * (t-y) *x2	b + Δb	w1 + Δw1	w2 + Δw2
									0	0	0
1	1	1	1	0	0	1	1	1	1	1	1
1	1	-1	1	1	1	0	0	0	1	1	1
1	-1	1	1	1	1	0	0	0	1	1	1
1	-1	-1	-1	-1	-1	0	0	0	1	1	1

3. Describe a step-by-step manual numerical execution of the MADALINE (Multiple Adaline) neural network for the implementation of the XOR function using MR-I training algorithm. Initialize weights and bias as follows. The weights for first hidden neuron: $w_{10} = 0.2$, $w_{11} = 0.3$, and for second hidden neuron: $W_{12} = 0.2$, $W_{20} = 0.3$, $W_{21} = 0.2$, $W_{22} = 0.1$ with learning rate parameter $\eta = 0.5$.

[10 Marks]

MADALINE Architecture (MR-I)





Note# Students may follow any notation given above.

x1	x2	t = x1 XOR x2
1	1	-1
1	-1	1
-1	1	1
-1	-1	-1

- $z\_in1 = 1 \times w01 + x1 \times w11 + x2 \times w21$
  $= 1 \times .2 + 1 \times .3 + 1 \times .2 = .7$
- $z\_out1 = 1$
- $z\_in2 = 1 \times w02 + x1 \times w12 + x2 \times w22$
  $= 1 \times .3 + 1 \times .2 + 1 \times .1 = .6$
- $z\_out2 = 1$
- $y\_in = 1 \times 0.5 + z\_out1 \times 0.5 + z\_out2 \times 0.5$
  $= 1 \times .5 + 1 \times .5 + 1 \times .5 = 1.5$
- $y\_out = 1$
- $w01\ (new) = w01\ (old) + h \times (-1 - z\_in1)$
  $= .2 + .5 \times (-1 - .7)$
  $= .2 - .85$
  $= -.65$

Step 5. Propagate the input signals through the net to the output unit Y.

5.1 Compute net inputs to the hidden units.

$$z\_in_1 = 1 \times w_{01} + x_1 \times w_{11} + x_2 \times w_{21}$$

$$z\_in_2 = 1 \times w_{02} + x_1 \times w_{12} + x_2 \times w_{22}$$

5.2 Compute activations of the hidden units z_out_1 and z_out_2 using the bipolar step function

$$z\_out = \begin{cases} 1, & \text{if } z\_in \geq 0 \\ -1, & \text{if } z\_in < 0. \end{cases}$$

5.3 Compute net input to the output unit

$$y\_in = 1 \times v_0 + z\_out_1 \times v_0 + z\_out_2 \times v_2$$

5.4 Find the activation of the output unit $y\text{-}out$ using the same activation function as in Step 5.2, *i.e.*,

$$y\_out = \begin{cases} 1, & \text{if } y\_in \geq 0 \\ -1, & \text{if } y\_in < 0. \end{cases}$$

Step 6. Adjust the weights of the hidden units, if required, according to the following rules:

i) If ($y\_out = t$) then the net yields the expected result. Weights need not be updated.

ii) If ($y\_out \neq t$) then apply one of the following rules whichever is applicable.

Case I: $t = 1$

Find the hidden unit z_j whose net input z_in_j is closest to 0. Adjust the weights attached to z_j according to the formula

$$w_{ij}\ (\text{new}) = w_{ij}\ (\text{old}) + h \times (1 - z\_in_j) \times x_i, \text{ for all } i.$$

Case II: $t = -1$

Adjust the weights attached to those hidden units z_j that have positive net input.

$$w_{ij}\ (\text{new}) = w_{ij}\ (\text{old}) + h \times (-1 - z\_in_j) \times x_i, \text{ for all } i.$$

												w10	w11	w12	w20	w21	w22	
		x0	x1	x2	t	z_in1	z_in2	z_out1	z_out2	y_in	y_out	w01	w11	w21	w02	w12	w22	eta
												0.2	0.3	0.2	0.3	0.2	0.1	0.5
Epoch 1	t!= y, Case 2	1	1	1	**-1**	**0.7**	**0.6**	1	1	1.5	**1**	-0.65	-0.55	-0.65	-0.50	-0.60	-0.70	
	t!=y, Case 1	1	1	-1	**1**	-0.55	**-0.4**	-1	-1	-0.5	**-1**	-0.65	-0.55	-0.65	0.20	0.10	-1.40	
	t!=y, Case 1	1	-1	1	**1**	**-0.75**	-1.3	-1	-1	-0.5	**-1**	0.23	-1.43	0.23	0.20	0.10	-1.40	
	t!=y, Case 2	1	-1	-1	**-1**	**1.425**	**1.5**	1	1	1.5	**1**	-0.99	-0.21	1.44	-1.05	1.35	-0.15	
Epoch 2	t!= y, Case 2	1	1	1	**-1**	**0.2375**	**0.15**	1	1	1.5	**1**	-1.61	-0.83	0.82	-1.63	0.78	-0.73	
	t!=y, Case 1	1	1	-1	**1**	-3.25625	**-0.125**	-1	-1	-0.5	**-1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
	t==y	1	-1	1	**1**	0.04375	-3.6875	1	-1	0.5	**1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
	t==y	1	-1	-1	**-1**	-1.59375	-1.1125	-1	-1	-0.5	**-1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
Epoch 3	t==y	1	1	1	**-1**	-1.61875	-1.0125	-1	-1	-0.5	**-1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
	t==y	1	1	-1	**1**	-3.25625	1.5625	-1	1	0.5	**1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
	t==y	1	-1	1	**1**	0.04375	-3.6875	1	-1	0.5	**1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
	t==y	1	-1	-1	**-1**	-1.59375	-1.1125	-1	-1	-0.5	**-1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	

x1	x2	t = x1 XOR x2
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

												w10	w11	w12	w20	w21	w22	
		x0	x1	x2	t	z_in1	z_in2	z_out1	z_out2	y_in	y_out	w01	w11	w21	w02	w12	w22	eta
												0.2	0.3	0.2	0.3	0.2	0.1	0.5
Epoch 1	t!= y, Case 2	1	-1	-1	**-1**	-0.3	0	-1	1	0.5	**1**	0.20	0.30	0.20	0.30	0.20	0.10	
	t==y	1	-1	1	**1**	0.1	0.2	1	1	1.5	**1**	0.20	0.30	0.20	0.30	0.20	0.10	
	t==y	1	1	-1	**1**	0.3	0.4	1	1	1.5	**1**	0.55	0.65	-0.15	0.20	0.10	-1.40	
	t!=y, Case 2	1	1	1	**-1**	**1.05**	-1.1	1	-1	0.5	**1**	-0.48	-0.38	-1.18	0.20	0.10	-1.40	
Epoch 2	t!= y, Case 2	1	-1	-1	**-1**	**1.075**	**1.5**	1	1	1.5	**1**	-1.51	0.66	-0.14	-1.05	1.35	-0.15	
	t!=y, Case 1	1	-1	1	**1**	**-2.3125**	-2.55	-1	-1	-0.5	**-1**	-1.61	-0.83	0.82	-1.05	1.35	-0.15	
	t==y	1	1	-1	**1**	-3.25625	0.45	-1	1	0.5	**1**	-1.61	-0.83	0.82	-1.05	1.35	-0.15	
	t!= y, Case 2	1	1	1	**-1**	-1.61875	**0.15**	-1	1	0.5	**1**	-1.61	-0.83	0.82	-1.63	0.78	-0.73	
Epoch 3	t==y	1	-1	-1	**-1**	-1.59375	-1.675	-1	-1	-0.5	**-1**	-1.61	-0.83	0.82	-1.63	0.78	-0.73	
	t==y	1	-1	1	**1**	0.04375	-3.125	1	-1	0.5	**1**	-1.61	-0.83	0.82	-1.63	0.78	-0.73	
	t!=y, Case 1	1	1	-1	**1**	-3.25625	**-0.125**	-1	-1	-0.5	**-1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
	t==y	1	1	1	**-1**	-1.61875	-1.0125	-1	-1	-0.5	**-1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
Epoch 4	t==y	1	-1	-1	**-1**	-1.59375	-1.1125	-1	-1	-0.5	**-1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
	t==y	1	-1	1	**1**	0.04375	-3.6875	1	-1	0.5	**1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
	t==y	1	1	-1	**1**	-3.25625	**1.5625**	-1	1	0.5	**1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
	t==y	1	1	1	**-1**	-1.61875	-1.0125	-1	-1	-0.5	**-1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
Epoch 5	t==y	1	-1	-1	**-1**	-1.59375	-1.1125	-1	-1	-0.5	**-1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
	t==y	1	-1	1	**1**	0.04375	-3.6875	1	-1	0.5	**1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
	t==y	1	1	-1	**1**	-3.25625	**1.5625**	-1	1	0.5	**1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	
	t==y	1	1	1	**-1**	-1.61875	-1.0125	-1	-1	-0.5	**-1**	-1.61	-0.83	0.82	-1.06	1.34	-1.29	

4. Draw Multilayer Perceptron feed-forward neural network with 2 inputs, 2 hidden and 2 output neurons along with bias. Using Back-propagation algorithm, calculate local gradients of both the hidden layer (δ_1^1, δ_2^1) and output layer neurons (δ_1^2, δ_2^2). Utilize the given weight matrices for the connections from the input to the hidden layer and from the hidden to the output layer and input vector and the corresponding output vector from category 1. Assume learning rate as 0.1.

$$W^1 = \begin{bmatrix} 0.5 & 1.5 & 0.8 \\ 0.8 & 0.2 & -1.6 \end{bmatrix} \quad W^2 = \begin{bmatrix} 0.9 & -1.7 & 1.6 \\ 1.2 & 2.1 & -0.2 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} 0.7 \\ 1.2 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

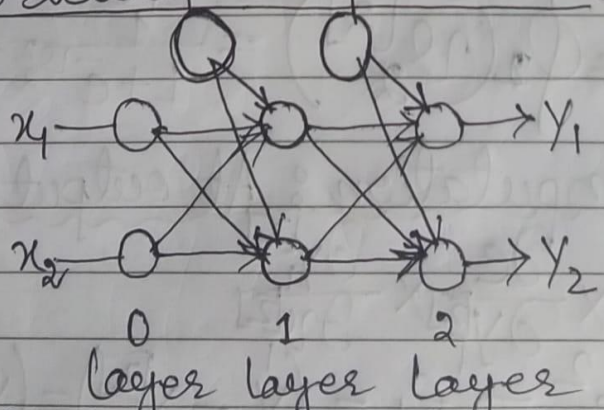
[10 Marks]

Ans:

$W^1 = \begin{bmatrix} 0.5 & 1.5 & 0.8 \\ 0.8 & 0.2 & -1.6 \end{bmatrix}$ $W^2 = \begin{bmatrix} 0.9 & -1.7 & 1.6 \\ 1.2 & 2.1 & -0.2 \end{bmatrix}$

$X = \begin{bmatrix} 0.7 \\ 1.2 \end{bmatrix}$ $Y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

Sol^n: Foreward computation:



* Augmented input Vector by including bias input as 1.

$X' = \begin{bmatrix} 1 \\ 0.7 \\ 1.2 \end{bmatrix}$

Activation at hidden layer:

$v_1^{[1]}$ → linear combination of input and weights in the hidden layer neuron 1.

$y_1^{[1]}$ → Output of the hidden layer neuron 1

$$v_1^{[1]} = \begin{bmatrix} 0.5 & 1.5 & 0.8 \end{bmatrix} \begin{bmatrix} 1 \\ 0.7 \\ 1.2 \end{bmatrix}$$

$$v_1^{[1]} = 0.5 \times 1 + 1.5 \times 0.7 + 0.8 \times 1.2 = 2.51$$

$$v_2^{[1]} = \begin{bmatrix} 0.8 & 0.2 & -1.6 \end{bmatrix} \begin{bmatrix} 1 \\ 0.7 \\ 1.2 \end{bmatrix} = 0.8\times1 + 0.2\times0.7 + (-1.6)\times1.2 = -0.98$$

$$y_1^{[1]} = \varphi_1(v_1^{[1]}) = \frac{1}{1+e^{-v_1}} = \frac{1}{1+e^{2.5}} = 0.82$$

$$y_2^{[1]} = \varphi_2(v_2^{[1]}) = \frac{1}{1+e^{-v_2^1}} = \frac{1}{1+e^{0.98}} = 0.27$$

Activation at output layer:

$$v_1^{[2]} = \begin{bmatrix} 0.9 & -1.7 & 1.6 \end{bmatrix} \begin{bmatrix} 1 \\ 0.82 \\ 0.27 \end{bmatrix} = 0.9 + (-1.7\times0.82) + (1.6\times0.27) = 0.9 - 1.394 + 0.43 = -0.062$$

$$v_2^{[2]} = \begin{bmatrix} 1.2 & 2.1 & -0.2 \end{bmatrix} \begin{bmatrix} 1 \\ 0.82 \\ 0.27 \end{bmatrix} = 1.2 + 2.1\times0.82 + (-0.2\times0.27) = 1.2 + 1.72 - 0.054$$

$$y_1^{[2]} = y_1 = \varphi_1(v_1^{[2]}) = \frac{1}{1+e^{-v_1^2}} = \frac{2.86}{0.856}$$

$$y_2^{[2]} = y_2 = \varphi_2(v_2^{[2]}) = \frac{1}{1+e^{-v_2^2}} = 0.113$$

Backward computation: At output layer:

$$\delta_1^{[2]} = \frac{\partial e_{ee}}{\partial e_1} \times \frac{\partial e}{\partial y_1^{[2]}} \times \frac{\partial y_1^{[2]}}{\partial v_1^{[2]}}$$

$$\frac{\partial e_{ee}}{\partial e_1} = \frac{\partial}{\partial e_1}\frac{1}{2}e_1^2 = e_1 = (y_1 - y_1^{[2]}) = (y_1 - \hat{y}_1)$$

$$\frac{\partial e_1}{\partial \hat{y}_1} = -1 \qquad \frac{\partial y_1^{[2]}}{\partial v_1^{[2]}} = \varphi_1'(v_1^{[2]}) = \hat{y}_1(1-\hat{y}_1)$$

$$\delta_1^{[2]} = -1(y_1 - \hat{y}_1)\hat{y}_1(1-\hat{y}_1) = -(1-\hat{y}_1)\hat{y}_1(1-\hat{y}_1)$$

$$\delta_2^{[2]} = -1(-\hat{y}_2)\hat{y}_2(1-\hat{y}_2) = \hat{y}_2^2(1-\hat{y}_2)$$

$$\delta_1^{[2]} = -(1-0.856)0.856(1-0.856) = 0.017 \qquad \delta_2^{[2]} = 0.011$$

$$\Delta W_{11}^{[2]} = \frac{\partial E_{ee}}{\partial W_{11}^{[2]}} = \delta_1^{[2]} \cdot Y_1^{[1]} = (0.017) \times 0.82 = 0.0139$$

$$W_{11}^{[2]}(new) = W_{11}^{[2]}(old) - \eta \Delta W_{11}^{[2]}$$

$$= -1.7 + 0.1 \times 0.0145 = -1.698$$

$$\Delta W_{12}^{[2]} = \frac{\partial E_{ee}}{\partial W_{12}^{[2]}} = \delta_1^{[2]} y_2^{[1]} = (-0.017) \times 0.27 = -0.0045$$

$$W_{12}^{[2]}(new) = W_{12}^{[2]}(old) - \eta \Delta W_{12}^{[2]}$$

$$= 1.6 + 0.1 \times 0.0045 = 1.6004$$

$$\Delta b_1^{[2]} = \delta_1^{[2]} \Rightarrow b_1^{[2]}(new) = b_1^{[2]}(old) - \eta \Delta b_1^{[2]}$$

$$= 0.9 + 0.1 \times 0.017 = 0.90$$

$$\Delta W_{21}^{[2]} = \frac{\partial E_{ee}}{\partial W_{21}^{[2]}} = \delta_2^{[2]} \cdot y_1^{[1]} = 0.011 \times 0.82$$

$$= 0.009$$

$$W_{21}^{[2]}(new) = W_{21}^{[2]}(old) - \eta \Delta W_{21}^{[2]}$$

$$= 2.1 - 0.1 \times 0.009 = 2.099$$

$$\Delta W_{22}^{[2]} = \frac{\partial E_{ee}}{\partial W_{22}^{[2]}} = \delta_2^{[2]} \cdot y_2^{[1]} = 0.011 \times 0.27 = 0.0029$$

$$W_{22}^{[2]}(new) = W_{22}^{[2]}(old) - \eta \Delta W_{22}^{[2]}$$

$$= -0.2 - 0.1 \times 0.0029 = -0.2002$$

$$\Delta b_2^{[2]} = \delta_2^{[2]} = 0.011$$

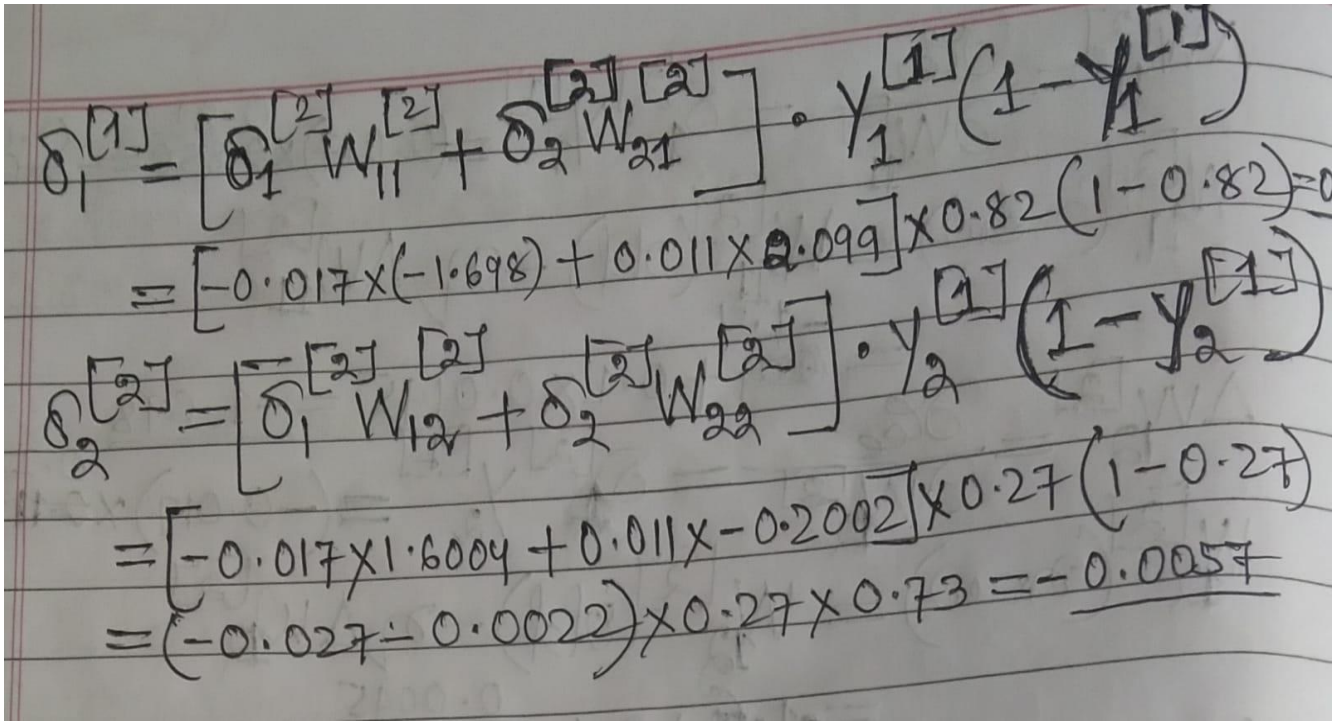
$$b_2^{[2]}(new) = b_2^{[2]}(old) - \eta \Delta b_2^{[2]} = 1.2 - 0.1 \times 0.011 = 1.198$$

$$W^{[2]} = \begin{bmatrix} 0.901 & -1.698 & 1.6004 \\ 1.198 & 2.099 & -0.2002 \end{bmatrix}$$

Backward computation : At hidden layer

$$\Delta W_{11}^{[1]} = \frac{\partial E_{ee}}{\partial W_{11}^{[1]}} = \delta_1^{[1]} x_1$$

$$\delta_1^{[1]} = \sum_{K=1}^{2} \delta_K^{[2]} W_{KJ}^{[2]} \cdot Y_1^{[1]} (1 - Y_1^{[1]})$$



5. Highlight the key differences between a Multilayer Perceptron (MLP) and a Radial Basis Function Network (RBFN) feed-forward neural networks.

[10 Marks]

Ans. Radial-basis function (RBF) networks and Multilayer Perceptrons are examples of nonlinear layered feedforward networks. They are universal function approximators. However, these two networks different from each other in several important points.

1. An RBF network has a single hidden layer, whereas an MLP may have one or more hidden layers.
2. Computational nodes of an MLP, located in a hidden or output layer, share a common neural model. However, in BBFN, the computation nodes in the hidden layer and output layer of an RBF network are quite different. Hidden layer neurons are used for nonlinear mapping and the output layer neurons are used for classification of the network.
3. The hidden layer of an RBF network is nonlinear, whereas the output layer is linear. However, the hidden and output layers of MLP are used for regression or classification, usually all nonlinear.
4. The argument of the activation function of each hidden node in RBF network computes the Euclidean norm (distance) between the input vector and the center of that node. Meanwhile, the activation function of each hidden node in MLP computes the inner product of the input vector and the synaptic weight vector of that node.
5. MLPs construct global approximations to nonlinear input-output mapping. On the other hand, RBF networks construct local approximations to nonlinear input-output mappings.
6. In MLP, the approximation of a nonlinear input-output mapping requires a small number of parameters than the RBF network for same degree of accuracy.