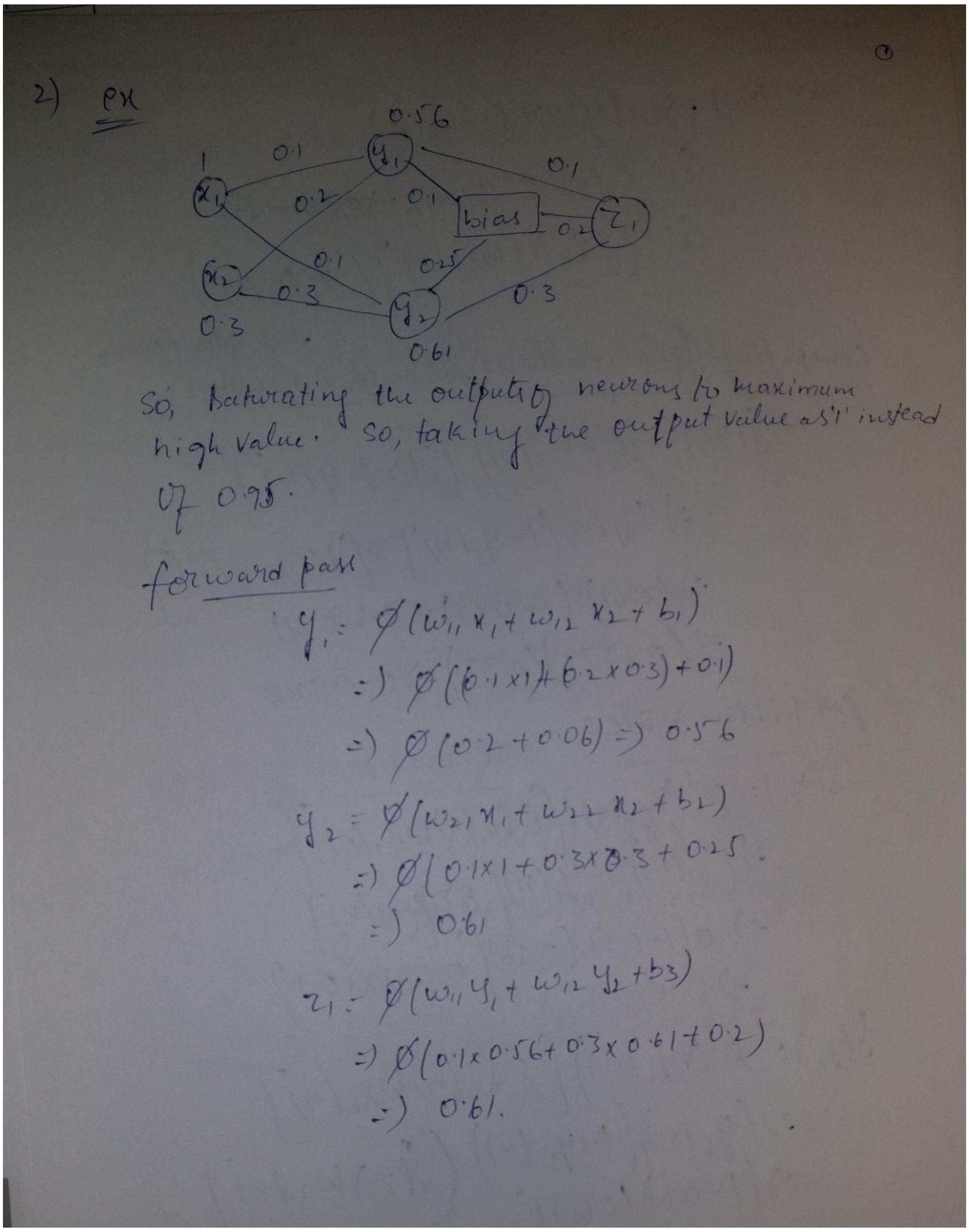
Soft Computing

Assignment 2: Multi-Layer Perceptrons

Vinay Mohan Behara (U00851261)

2. When using the back propagation and either of sigmoidal transfer functions, the output of neurons is not be allowed to saturate to their maximum high or low values because it stops the learning process. The training can't be performed. So, even though there's a change in the inputs, the outputs won't change.

   Given an example:



2) ex

So, saturating the output of neurons to maximum high value. So, taking the output value as '1' instead of 0.95.

forward pass

$y_1 = \phi(w_{11} x_1 + w_{12} x_2 + b_1)$

$\Rightarrow \phi((0.1 \times 1) + (0.2 \times 0.3) + 0.1)$

$\Rightarrow \phi(0.2 + 0.06) \Rightarrow 0.56$

$y_2 = \phi(w_{21} x_1 + w_{22} x_2 + b_2)$

$\Rightarrow \phi(0.1 \times 1 + 0.3 \times 0.3 + 0.25$

$\Rightarrow 0.61$

$z_1 = \phi(w_{11} y_1 + w_{12} y_2 + b_3)$

$\Rightarrow \phi(0.1 \times 0.56 + 0.3 \times 0.61 + 0.2)$

$\Rightarrow 0.61.$

Backward pass

$$w_{ji}(n+1) = w_{ji}(n) + \eta\, \delta_j (y_i(n))$$

$$\delta_j = e_j(n)\, \phi'(v_j(n)) \rightarrow \text{o/p neuron}$$

$$\delta_j = \left[\sum_k \delta_k(n)\, w_{kj}(n)\right]\left[\phi_j(v_j(n))\right)$$

→ hidden neuron

computing local gradient for output neuron :-

$$\delta(z_1) = \phi'(v_{z_1}(n))\, [d(n) - y(n)]$$

$$\Rightarrow y_j(n)\,[1 - y_j(n)] \times [d(n) - y(n)]$$

$$\Rightarrow (0.61)(1-0.61) \times (1-0.61)$$

$$\Rightarrow (0.2379)(0.39) \Rightarrow 0.0928$$

for hidden nodes.

$$\delta(y_1) = \phi'(v_{y_1}(n))\left[\sum \delta_k(n)\, w_{k1}(n)\right]$$

$$\Rightarrow (y_1(n))(1 - y_1(n))\left[\delta(z_1)\, w_{11}(n)\right]$$

$$\Rightarrow (1-0.56)(0.56)\left[0.0928 \times 0.1\right]$$

$$\Rightarrow 0.0022$$

$$\delta(y_2) = \phi'(v_{y_2}(n))\left[\sum \delta_k(n)\, w_{k2}(n)\right]$$

$$\Rightarrow (y_2(n))(1 - y_2(n))\left(\delta(z_1)\, w_{12}(n)\right]$$

$$\Rightarrow (1-0.61)(0.61)\left[0.0928 \times 0.3\right]$$

$$\Rightarrow 0.0066$$

updating weights

$w'_{11}(y_1 \text{ to } z_1) \Rightarrow w_{11}(y_1 \text{ to } z_1) + \eta\, \delta_{z_1} y_1(n)$

$\Rightarrow 0.1 + 0.05 \times 0.0928 \times 0.61$

$\Rightarrow 0.102$

$w'_{12}(y_2 \text{ to } z_1) = w_{12}(y_2 \text{ to } z_1) + \eta\, \delta_{z_1} y_1(n)$

$\Rightarrow 0.3 + 0.05 \times 0.0928 \times 0.61$

$\Rightarrow 0.30$

bias updating

$0.2 + 0.05 \times 0.0928 \times 0.61$

$\Rightarrow 0.2 + 0.002$

$\Rightarrow 0.2002$

updating inner weights

$w'_{11}(x_1 \text{ to } y_1) = w'_{11}(x_1 \text{ to } y_1) + \eta\, \delta_{y_1} y_1(n)$

$\Rightarrow 0.1 + 0.05 \times 0.0021 \times 0.56$

$\Rightarrow 0.10006$

$w'_{12}(x_2 \text{ to } y_1) = w'_{12}(x_2 \text{ to } y_1) + \eta\, \delta_{y_1}(y_1(n))$

$\Rightarrow 0.2 + 0.006$

$\Rightarrow 0.206$

$$bias_1' \Rightarrow bias_1 + \eta \, \delta y_1 \, y_1(n)$$
$$\Rightarrow 0.1 + 0.05 \times 0.0022 \times 0.56$$
$$\Rightarrow 0.1$$

$$w_{21}'(x_1 \text{ to } y_2) = w_{21}(x_1 \text{ to } y_2) + \eta \, \delta y_2 \, y_2(n)$$
$$\Rightarrow 0.1 + 0.05 \times 0.0066 \times 0.61$$
$$\Rightarrow 0.1002$$

$$w_{22}'(x_2 \text{ to } y_2) = w_{22}(x_2 \text{ to } y_2) + \eta \, \delta y_2 \, y_2(n)$$
$$\Rightarrow 0.3 + 0.05 \times 0.0066 \times 0.61$$
$$\Rightarrow 0.3$$

$$bias_2' \Rightarrow bias_2 + \eta \, \delta y_2 \, (y_2(n))$$
$$\Rightarrow 0.25 + 0.0002$$
$$\Rightarrow 0.25002$$

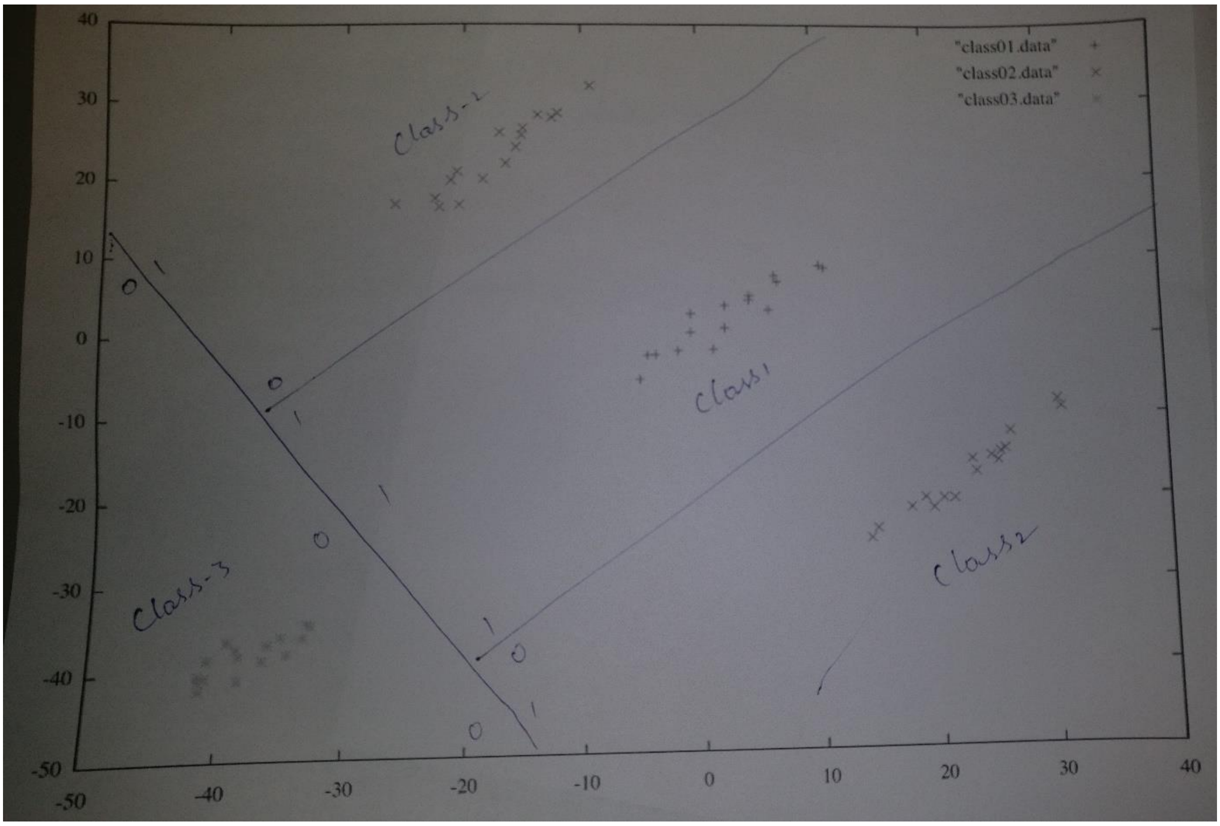
So, the weights and the bias are not gettin updated. So, the learning is stopped.

3. Inputs to multilayer perceptron are normalized so that the values of x lie in the range [-1.0 to 1.0] before presenting them to neural network for training. This is because, normalizing reduces the chances of getting struck in local minima and it helps the neural networks to learn faster. The weight initializations can be easily made in which we need to initialize the weights in a sweet spot where the whole performance will be better with good generalization.

   For example if we consider a neural network, the learning is done using the error vector which is multiplied by the learning rate. If we don't normalize the data, the learning rate would cause corrections that are different from one another because they are of different ranges. This makes us difficult to reach the global maxima and this makes the learning slow which takes more time to reach the global optima.

The other example to consider is that if we scale the input features, we get a range of the input features for which we can randomize the initialization of weights in a specific sweet spot of the range with respect to the input values and this gives us the better performance with generalization.

4. Given three classes to design a multi-layer perceptron.



The hidden nodes are 3 and the output nodes are two. if we have one neuron in the output layer, it can only classify two classes by making one class to zero and the other to one. The reason I took two classes in the output layer is because, we have three classes to classify and two neurons can easily classify four classes. So, the minimum number of output neurons in the neural network is two. The reason I took three neurons in the hidden layer is because, the minimum neurons that can classify the given three classes is three, where two neurons can't classify given three classes.

In the implementation part, I have set the threshold error value as 0.01 and I have reached to that error in 66149 iterations and in 16.19 seconds

The outputs of the implementation is show below

```
(2, 3, 2)
[array([[ 2.14214571, -0.71282406,  1.67222731],
       [ 2.26671734, -1.24843647, -1.44517414],
       [-0.31992005,  0.54765495, -1.27369656]]), array([[ 2.25570244, -0.73888163, -0.28122038,  1.68667188],
       [ 0.51758331,  1.85189162, -1.71891832, -0.06828427]])]
Iterations 0    Error: 54.452920
Iterations 100  Error: 12.830020
Iterations 200  Error: 12.513705
Iterations 300  Error: 9.103003
Iterations 400  Error: 10.710230
Iterations 500  Error: 10.436404
Iterations 600  Error: 9.086858
Iterations 700  Error: 8.876241
Iterations 800  Error: 8.751773
Iterations 900  Error: 8.661232
Iterations 1000 Error: 8.591384
```

```
Iterations 64700        Error: 0.001073
Iterations 64800        Error: 0.001068
Iterations 64900        Error: 0.001063
Iterations 65000        Error: 0.001057
Iterations 65100        Error: 0.001052
Iterations 65200        Error: 0.001047
Iterations 65300        Error: 0.001042
Iterations 65400        Error: 0.001037
Iterations 65500        Error: 0.001032
Iterations 65600        Error: 0.001027
Iterations 65700        Error: 0.001022
Iterations 65800        Error: 0.001017
Iterations 65900        Error: 0.001012
Iterations 66000        Error: 0.001007
Iterations 66100        Error: 0.001002
Minimum error reached at Iteration: 66149
time taken in seconds
16.1993638399872
```

1.

1) Given transfer function for first hidden layer

$$\phi_1(v) = \frac{1-e^{-2v}}{1+e^{-2v}} \quad \text{(hyperbolic)}$$

The activation for the next layer

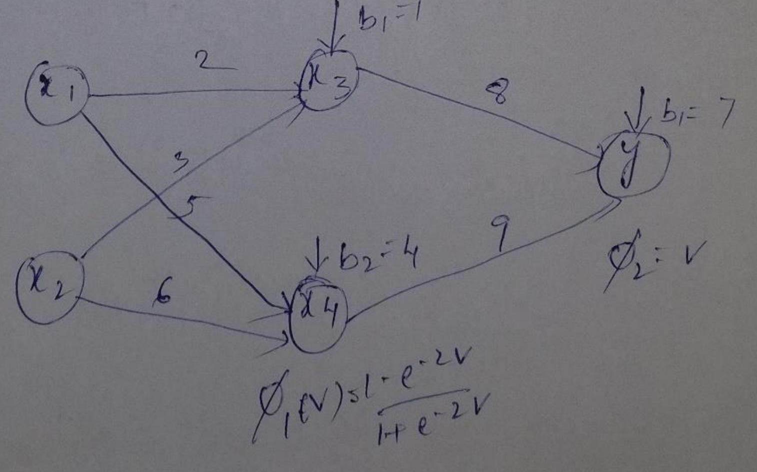
$$\phi_2(v) = v \quad \text{(linear)}$$

weight matrices

$$\text{layer 1} \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ b_1 & w_{11} & w_{12} \\ b_2 & w_{21} & w_{22} \end{bmatrix}$$

$$\text{layer 2} \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ b_1 & w_{11} & w_{12} \end{bmatrix}$$

$$\text{Input} \Rightarrow [1 \;\; x_1 \;\; x_2]^T \Rightarrow \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$$



5. A.Training a (2,4,2) 2 input 4 hidden and 2 output network for 20 times

training	Epoch where we have zero error
1	36445
2	111183
3	86537
4	42611
5	111599
6	48921
7	33561
8	16699
9	17457
10	25396
11	80617
12	93207
13	41327
14	39519
15	73928
16	40382
17	26424
18	83269
19	49021
20	28469

Average time in epochs = 54328

b. Training a (2,8,2) 2 input 8 hidden and 2 output network for 20 times

training	Epoch where we have zero error
1	44844
2	35580
3	37357
4	101169
5	24148
6	68326
7	78914
8	17234
9	81393
10	15635
11	51645
12	94565
13	38125
14	48654
15	51645
16	16115
17	15118
18	21225
19	14128
20	36694

Average time in epochs = 44625

C. Training a 2 input 2 hidden and 2 output network for 20 times

For this network, the zero error is not obtained even though I did train for many number of epochs. I have reached 485000 epochs but the error is still 9.8. So with two hidden neurons, getting a zero error is time taking. This time is greater than the one computed in a and b

```
Iterations 335000      Error: 9.877737
Iterations 340000      Error: 10.082026
Iterations 345000      Error: 9.895268
Iterations 350000      Error: 9.882769
Iterations 355000      Error: 9.889490
Iterations 360000      Error: 9.881883
Iterations 365000      Error: 7.740832
Iterations 370000      Error: 9.894237
Iterations 375000      Error: 9.883321
Iterations 380000      Error: 9.862817
Iterations 385000      Error: 9.883795
Iterations 390000      Error: 9.876668
Iterations 395000      Error: 9.883917
Iterations 400000      Error: 9.877570
Iterations 405000      Error: 9.901831
Iterations 410000      Error: 9.881658
Iterations 415000      Error: 9.865429
Iterations 420000      Error: 9.885158
Iterations 425000      Error: 9.880743
Iterations 430000      Error: 9.875971
Iterations 435000      Error: 9.883956
Iterations 440000      Error: 9.878875
Iterations 445000      Error: 9.867075
Iterations 450000      Error: 9.884160
Iterations 455000      Error: 9.875865
Iterations 460000      Error: 9.900075
Iterations 465000      Error: 9.880936
Iterations 470000      Error: 9.864505
Iterations 475000      Error: 9.874302
Iterations 480000      Error: 9.879872
Iterations 485000      Error: 9.874052
```

D.

network	Average epoch time
(2,4,2)	54328
(2,8,2)	44625
(2,2,2)	More than (2,4,2) and (2,8,2)