Deep Learning Assignment 1

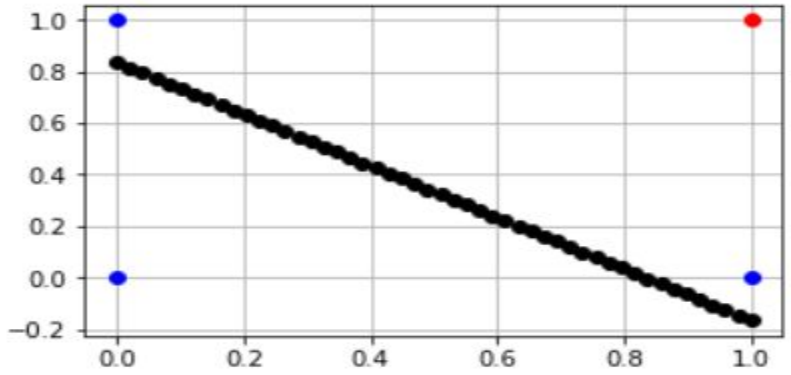
Q1 a)
Compute 2-variables AND, OR, and NOT (1-variable) operations and report the number of steps (number of weight-updates) required for the convergence.

i) AND GATE
- Converges in 4 Epochs
- W = [0.4, 0.4]
- T =0.5

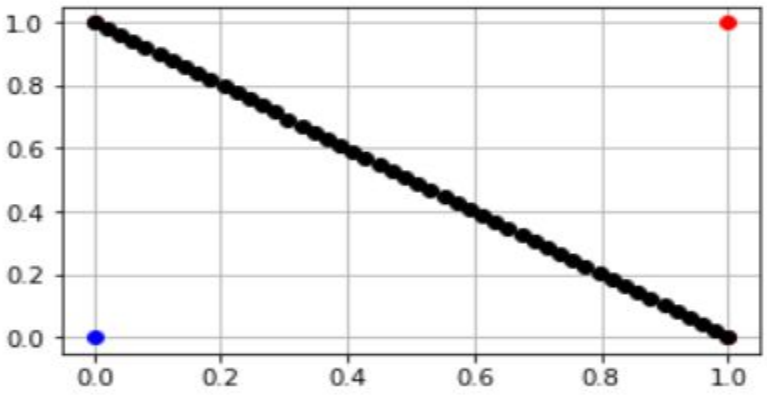
ii) OR GATE
- Converges in 4 Epochs
- W =[1.5, 1.5]
- T= 1.5

iii) NOT GATE
- Converges in 2 Epochs
- W=-1.0
- T=0.0

Q1 b)
Draw the decision boundary
i) AND GATE



ii) OR GATE



Q1 c) Prove that XOR can't be computed using PTA.

SOLUTION

- A perceptron gets the affine combination of input and if the sum is greater than a certain threshold then the output is 1 else 0.
- XOR Gate takes 2 inputs and gives output 0 if both inputs are equal else 1.

X1	X2	X1 (XOR) X2
0	0	0
0	1	1
1	0	1
1	1	0

- Why can't perceptron learn XOR?
  Let the inputs be X1 and X2, and a single perceptron, with weights w1 and w2.
  Y=w1*X1+w2*X2
  If (Y>t): Z=1 else 0

  Let us assume both X1, X2=0
  Y=0
  For Z to be 0, t has to be greater than 0, i.e
  t>0 ……. (i)

  If both inputs are 1, Y=w1+w2
  For Z to be 0 ; **t > w1+w2 ……. (ii)**

  But if only 1 input is 0, then **t < w1 and t < w2 ……. (iii)**
  Which is contradicting (ii)

So we can see that we cannot fit the equation on a 2d plane and hence we cannot fit the equation using a single perceptron.

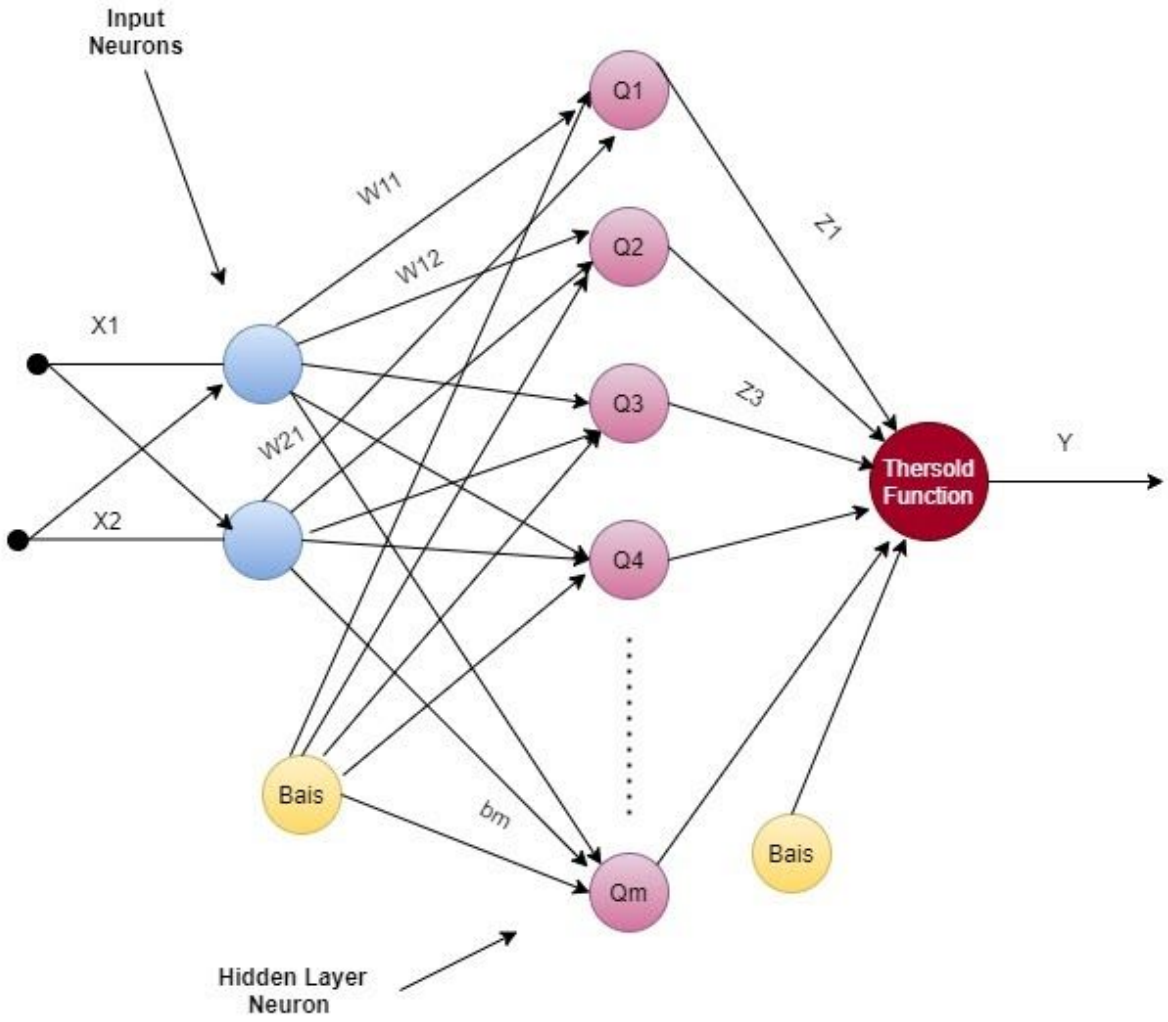
Q1 c_i) How many minimum steps do you need to prove it?

Sol)
The points on the plane are (0,0),(0,1),(1,0),(1,1). If we draw a hyperplane, the maximum distance it can have from any of the points is
Since decision boundary is not possible λ=0, since the maximum number of mistakes possible is 1/ λ, it should converge ∞ epochs. To show experimentally we will run our perceptron algorithm and observe that the loss oscillates.

---

2. Using the Madaline learning algorithm compute the following two functions. Shaded regions are 1, rest are 0. Report the number of neurons for each case.

Solution) : Following is the architecture used in implementation of Madaline Algorithm



For both parts of the answer there are 2 input layers and 1 output layer.

Solution a):

The Madaline algorithm is not converging for function ***f1(x1,x2)*** following are the parameters used in Training the Model.

Number of training sample : 11
Number of Input Neurons : 2
Number of Output Neurons : 1
Learning rate : 0.5
Number of iterations : 100
Number of hidden neurons : 20
Initial default baised is 0.5 for all neuron
Initial default weight for input layer is 0.5
Initial default weight for hidden layer is 0.25

We also try for different numbers of hidden neurons ranging from 12 to 100 but for some of the testing samples the desired output is wrong.

When we try to further increase the number of hidden neurons and the iteration the task is computationally too expensive to execute. So we concluded that for function f1(x1,x2) and with the given parameters(training set, learning rate, number of neurons) the Madaline did not work.

Following are the input parameters with their output.
inputs = np.array([[0,0],[1,1],[2,3],[4,3],[3,3],[5,3],[3,5],[1,5],[5,1],[6,6],[1,3]])
outputs = np.array([[-1,-1,1,1,1,1,1,-1,-1,-1,1]])
Here is the screenshot of output.

```
PS C:\Users\reshanriq\Documents\sem_4\DL\Assignment\madaline-master> python one.py

 Enter the bias of hidden to output layer:
Initial Weights:
[[0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25
  0.25 0.25 0.25 0.25 0.25 0.25]
 [0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25
  0.25 0.25 0.25 0.25 0.25 0.25]]

Initial Bias:
[0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 0.5 0.5]
No. of epochs: 100

 Final Weights:
[[ 2.93369621e+39 -5.31535015e+38 -1.18081255e+38  4.68561046e+38
  -2.94254806e+38  6.20316164e+39 -1.07515341e+38 -1.49068044e+39
   2.93956146e+39  6.32889522e+39 -1.73748952e+39 -1.73259036e+39
   3.82167619e+38 -1.64375232e+39 -1.04164545e+39  9.79010214e+38
  -3.75434302e+38 -5.58529942e+38 -1.29873656e+39 -4.02645427e+38]
 [ 1.83985071e+39 -2.84310439e+39 -8.72652296e+38  3.14983115e+39
  -2.54608322e+39  9.70062097e+39 -9.39529511e+38 -8.61952338e+39
   2.48551046e+39  3.21358852e+39 -1.08721384e+40 -9.26718092e+39
   9.71734853e+38 -1.56104278e+40 -1.76463895e+39  1.25178228e+39
  -3.85365555e+39 -3.53465945e+39 -9.60032222e+39 -2.98641701e+39]]

 Final Bias:
[ 6.71061859e+38 -5.33216732e+38 -1.65121368e+38  1.05281524e+39
 -4.83704884e+38  1.94715489e+39 -1.78447664e+38 -1.72741288e+39
  9.01284829e+38  1.16474671e+39 -2.04806375e+39 -1.73804641e+39
  3.59173512e+38 -2.98266791e+39 -3.36780358e+38  4.58921098e+38
 -7.40752365e+38 -6.65721613e+38 -1.81652666e+39 -5.65029260e+38]
```

```
 Enter number of test cases:4
Enter test case input
0
0
Output:
[[-1.]]
Enter test case input
50
50
Output:
[[-1.]]
Enter test case input
2.5
2.5
Output:
[[-1.]]
Enter test case input
5
1
Output:
[[-1.]]
```

Here the output for the 3rd case is wrong .

Solution b):

The Madaline algorithm is not converging for function ***f2(x1,x2)*** following are the parameters used in Training the Model.
Number of training sample : 11
Number of Input Neurons : 2
Number of Output Neurons : 1
Learning rate : 0.5
Number of iterations : 100
Number of hidden neurons : 20
Initial default baised is 0.5 for all neuron
Initial default weight for input layer is 0.5
Initial default weight for hidden layer is 0.25

We also try for different numbers of hidden neurons ranging from 12 to 100 but for some of the testing samples the desired output is wrong.
When we try to further increase the number of hidden neurons and the iteration the task is computationally too expensive to execute. So we concluded that for function f2(x1,x2) and with the given parameters(training set, learning rate, number of neurons)the Madaline did not work.

Following are the input parameters with their output.
inputs = np.array([[0,0],[1,1],[1,3],[3,3],[3.5,3.5],[5,3],[3,5],[6,6],[3,8],[8,3],[3,1]])
outputs = np.array([[-1,-1,1,-1,-1,-1,1,1,-1,-1,-1,1]])

Here is the screenshot of output.

```
PS C:\Users\reshanriq\Documents\sem_4\DL\Assignment\madaline-master> python one.py

 Enter the bias of hidden to output layer:
Initial Weights:
[[0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25
  0.25 0.25 0.25 0.25 0.25 0.25]
 [0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25
  0.25 0.25 0.25 0.25 0.25 0.25]]

Initial Bias:
[0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 0.5 0.5]
No. of epochs: 100

 Final Weights:
[[-6.04006860e+24 -1.33688423e+23 -6.07711270e+22 -9.61270282e+23
  -2.62418365e+24  6.51482919e+23 -7.13804119e+23 -8.92893253e+22
  -3.39885992e+24 -6.02316381e+23  4.42904342e+23  2.62766771e+23
  -1.24120305e+24 -7.19017076e+22  2.57151566e+23 -7.73373989e+24
  -6.66907144e+23 -2.28169636e+24 -5.65382222e+23  1.27189656e+23]
 [-6.13135027e+24 -1.42417981e+23 -1.16644972e+23 -9.47091300e+23
  -2.58531992e+24  3.81751680e+23 -7.24553712e+23 -8.25074787e+22
  -3.45336220e+24 -6.24341755e+23  2.53935343e+23  4.58239407e+23
  -1.19685980e+24 -7.79207913e+22 -8.78440743e+22 -7.58098412e+24
  -1.06075328e+24 -2.31600395e+24 -5.42109975e+23  2.18463126e+22]]

 Final Bias:
[-1.00111276e+24 -4.72880312e+22 -2.00497438e+22 -1.56658144e+23
 -4.27648091e+23  1.27443500e+23 -1.18076015e+23 -2.97601313e+22
 -5.63032648e+23 -9.48927840e+22  8.59064426e+22  8.88897941e+22
 -1.90504427e+23 -2.59684666e+22  4.66287437e+22 -1.26202677e+24
 -2.02261526e+23 -3.77425397e+23 -8.59025224e+22  4.16608610e+22]
```

```
 Enter number of test cases:4
Enter test case input
-5
-5
Output:
[[1.]]
Enter test case input
0
0
Output:
[[-1.]]
Enter test case input
3
3
Output:
[[-1.]]
Enter test case input
3
5
Output:
[[-1.]]
```

From the following test case the test case number 1 and 4 are not desired output what we want.

Solution c) : No we can not compute f2(x1,x2) in <= 2 more neuron with respect to f1(x1,x2). Because the Madaline algorithm did not work for both function and hence it is not feasible . Also we check our Algorithm for XOR with 2 hidden neuron and it work well and hence it justify that our algorithm is correct but it is not feasible to compute the f1 and f2.

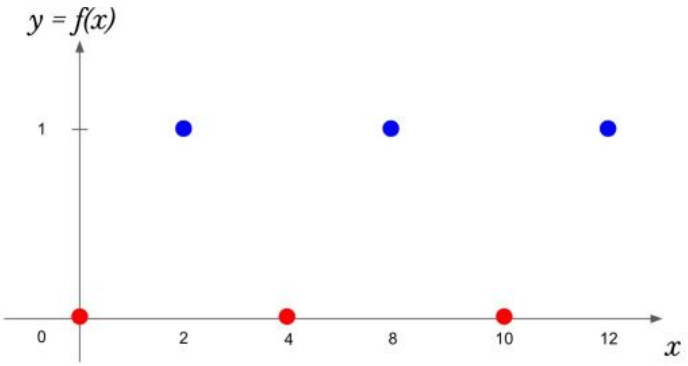
Here is the screenshot of XOR with Madaline Algorithm.

```
PS C:\Users\reshanriq\Documents\sem_4\DL\Assignment\madaline-master> python one.py

 Enter the bias of hidden to output layer:
Initial Weights:
[[0.25 0.25]
 [0.25 0.25]]

Initial Bias:
[0.5 0.5]
No. of epochs: 100

 Final Weights:
[[-1.625  0.875]
 [ 0.875 -1.625]]

 Final Bias:
[-0.875 -0.875]
```

```
 Enter number of test cases:4
Enter test case input
1
1
Output:
[[-1.]]
Enter test case input
1
-1
Output:
[[1.]]
Enter test case input
-1
1
Output:
[[1.]]
Enter test case input
-1
-1
Output:
[[-1.]]
```

---

3. **Implement a single-neuron neural network to compute the following function** $y = f(x)$**. You can use the generalized delta rule for learning. If you think it's not possible, justify your claims properly.** [15 points]



Ans.

Assuming Typo for x -axis as (0,2,4,8,10,12) being corrected to (0,2,4,6,8,10)

Since the output y fluctuates between 0 and 1.

Therefore, a sinusoidal function can serve as an activation function.

But, Sine & Cosine oscillate between [-1,1].

Hence, we use $(\sin(x))^2$ as the activation function.

Overall mathematics of the neuron will then be:

Adder output,

$$v = W0.1 + W1.x = W0 + W1x$$

Output of activation unit,

$$y = A(v) = \sin^2(v) \text{ , which will range in } [0,1]$$

As per Generalized delta rule, weight update will be

$$del(W) = lr * (d-y) * A'(v) * x$$

where, del(W) is the update in weight vector W,

lr = learning rate parameter in [0,1]

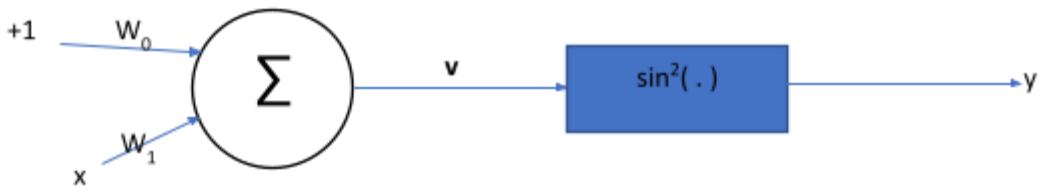
d = desired output

y = actual output of neuron

x = input vector

A'(v) = derivative of neuron output y w.r.t. adder output v

For activation function $A= \sin^2(.)$, the derivative $A'(v) = 2*\sin(v)*\cos(v)$



Steps:

1. Randomly initialized weights of the neuron
2. Trained the neuron using given 6 examples: (0,0), (2,1), (4,0), (6,1), (8,0), (10,1)
    a. Learning rate = 1.0
    b. Training iterations = 1000
3. Weight update equation, W = W + del(W), where del(W) is computed as stated above.

Result:

Trained weights = [W0, W1] = [0.01379515, 0.78335993]

```
Input: 0.0 Desired output: 0 Neuron output: 0.0001902940579096695
Input: 2.0 Desired output: 1 Neuron output: 0.9999055501063187
Input: 4.0 Desired output: 0 Neuron output: 3.1834356545053885e-05
Input: 6.0 Desired output: 1 Neuron output: 0.9999975483914563
Input: 8.0 Desired output: 0 Neuron output: 6.3036027430313914e-06
Input: 10.0 Desired output: 1 Neuron output: 0.9999566099168984
```
