Deep Learning- CSE641

Assignment - 1

Name: Akanksha Shrimal	**Roll No:** MT20055
Name: Vaibhav Goswami	**Roll No:** MT20018
Name: Shivam Sharma	**Roll No**: MT20121

FILES SUBMITTED : submitted .py file , .ipynb file and readme.pdf and output.pdf

1. Perceptron Training Algorithm:

PRE-PROCESSING: Data for points (input and output) was provided for each gate.

1(a) and (b) COMPUTING PTA - AND , OR, NOT

Methodology :-

1. **AND Gate:**
   Initial weights are set as w1 = 2, w2 = 1, bias (or w0) = -3.
   <u>Truth Table:</u>

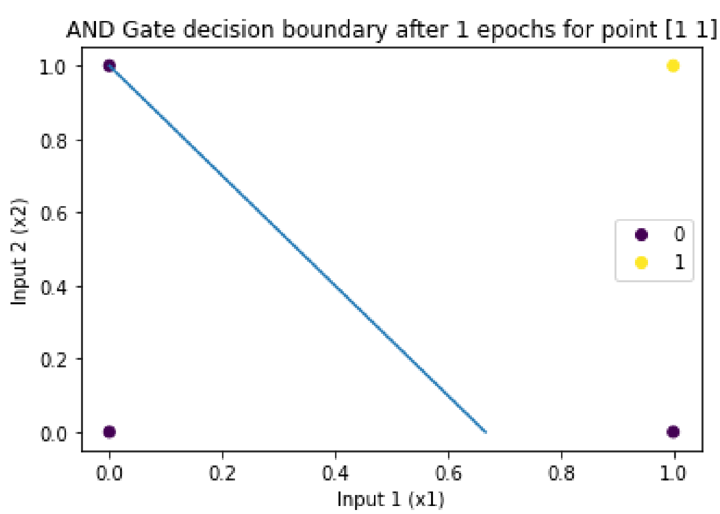
Input (x1)	Input (x2)	Output (y)
0	0	-1
0	1	-1
1	0	-1
1	1	1

Note: Output is 1 and -1 instead of 1 and 0 for mathematical convenience.

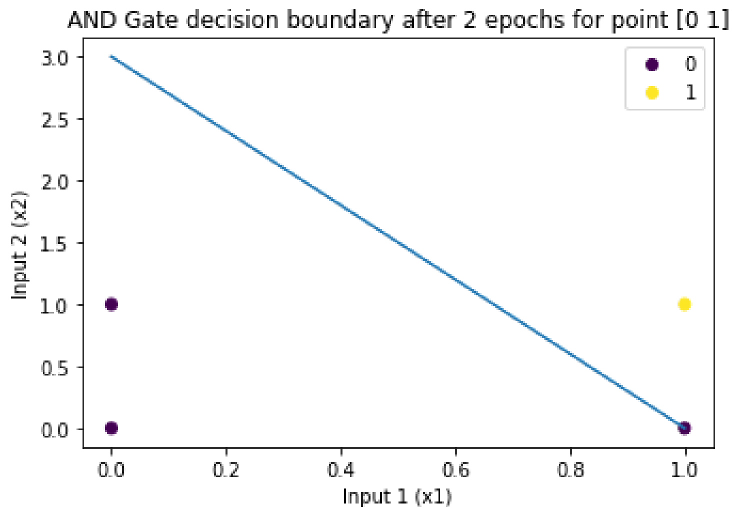
Convergence steps:- For AND gate, PTA took <u>7 steps</u> for convergence.

Graphs:-

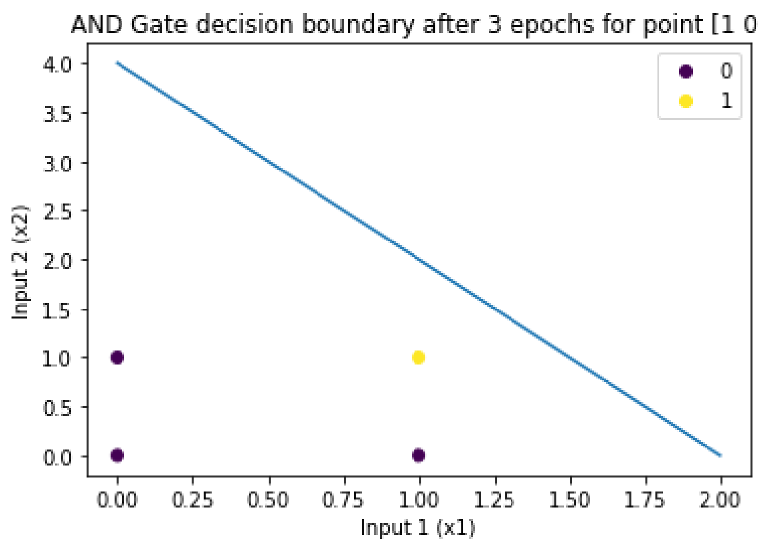
Step 1:



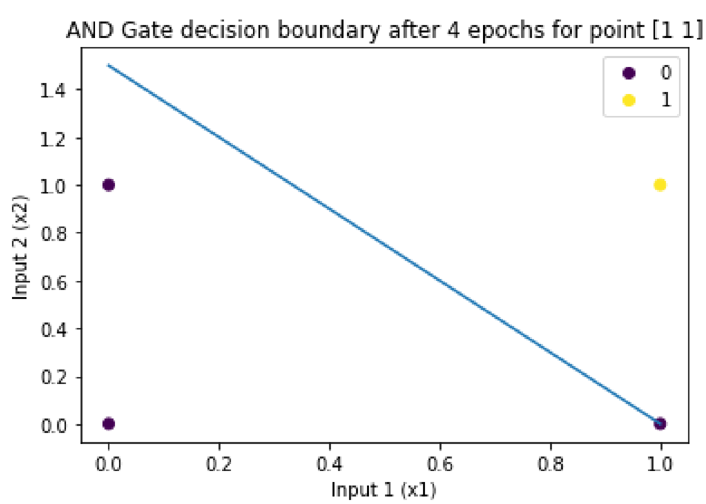
Step 2:



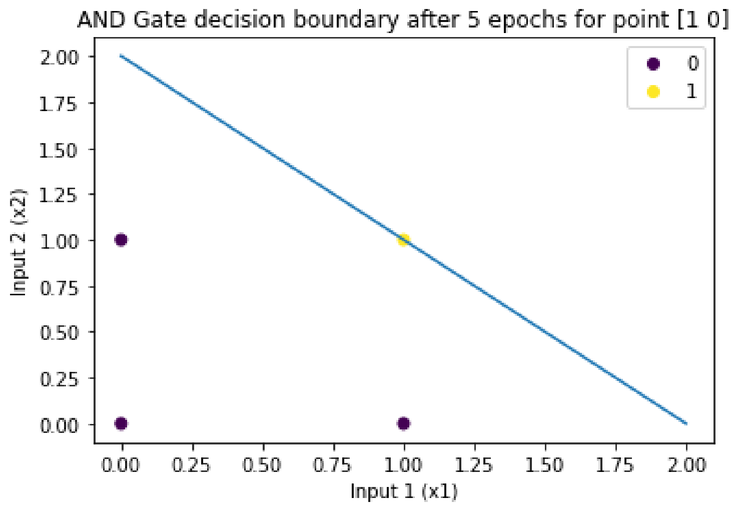
Step 3:



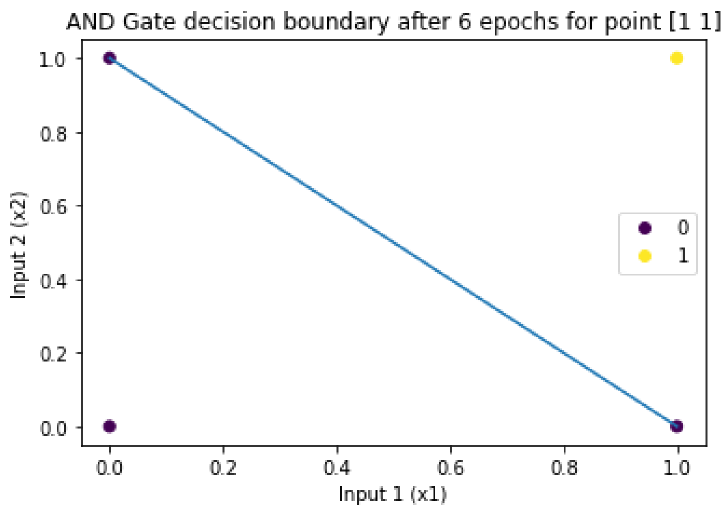
Step 4:



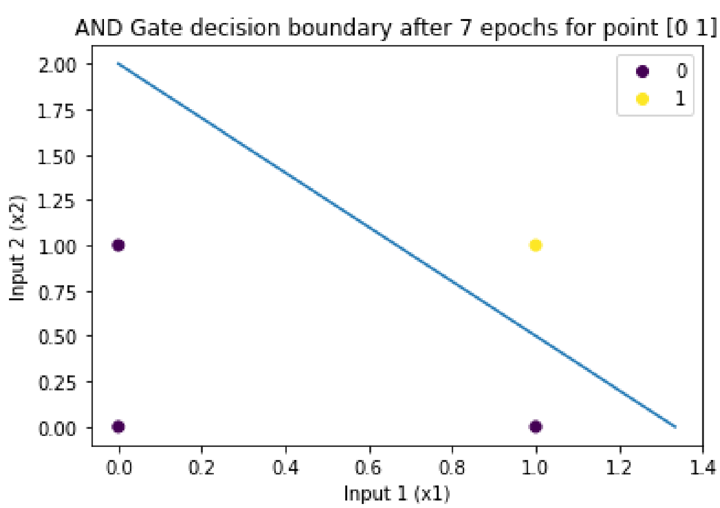
Step 5:


AND Gate decision boundary after 5 epochs for point [1 0]
Input 2 (x2)
Input 1 (x1)
0
1

Step 6:


AND Gate decision boundary after 6 epochs for point [1 1]
Input 2 (x2)
Input 1 (x1)
0
1

Step 7:

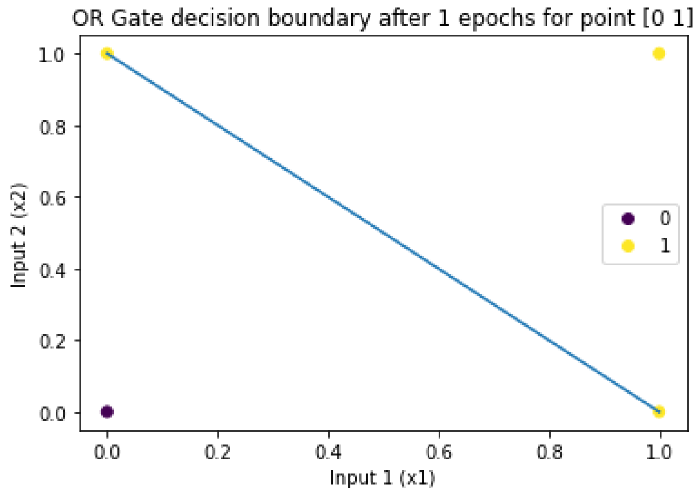

AND Gate decision boundary after 7 epochs for point [0 1]
Input 2 (x2)
Input 1 (x1)
0
1

2. **OR Gate:**
Initial weights are set as w1 = 2, w2 = 1, bias (or w0) = -3.
<u>Truth Table:</u>

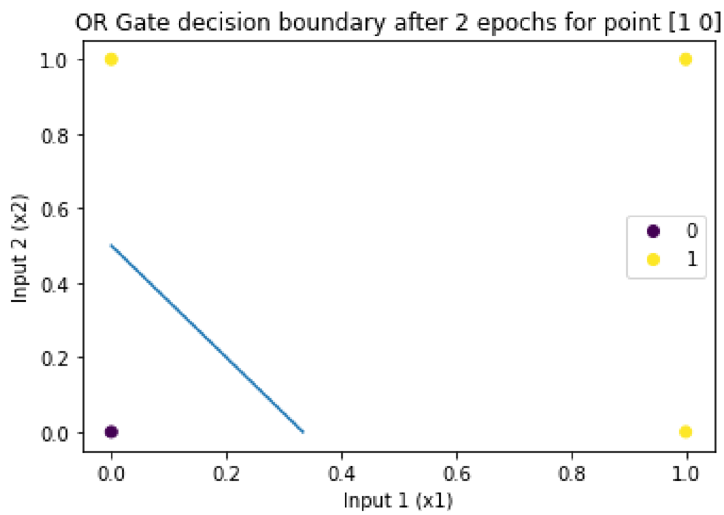
Input (x1)	Input (x2)	Output (y)
0	0	-1
0	1	1
1	0	1
1	1	1

Convergence steps:- For OR gate, PTA took <u>2 steps</u> for convergence.
Graphs:-

Step1:



Step 2:



3. **NOT**

Initial weights are set as w1 = 1.5, bias (or w0) = 0.5.

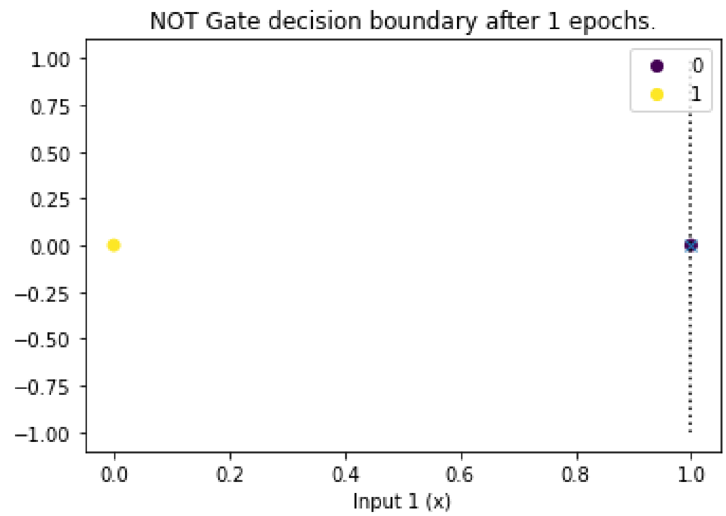
Truth Table:

Input(x)	**Output(y)**
0	1
1	-1

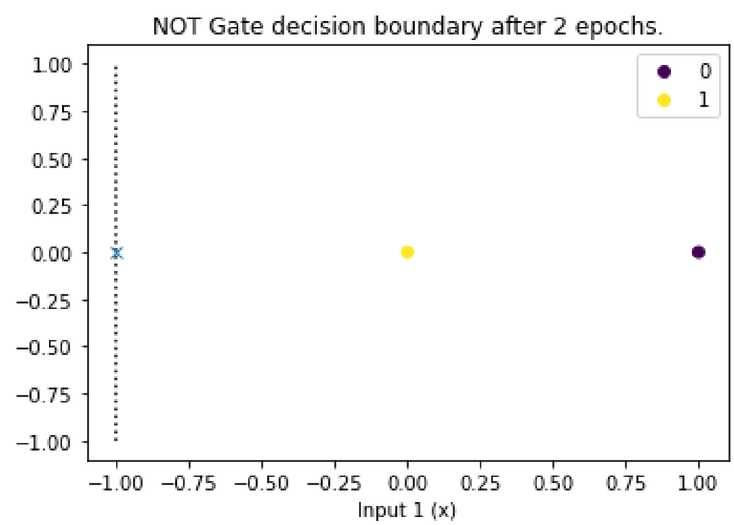
Convergence steps:- For NOT gate, PTA took 6 steps for convergence.

Graphs:-

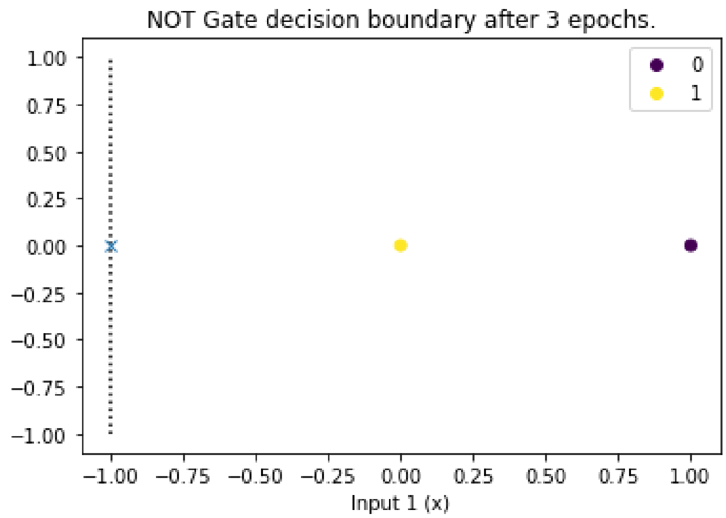
Step 1:



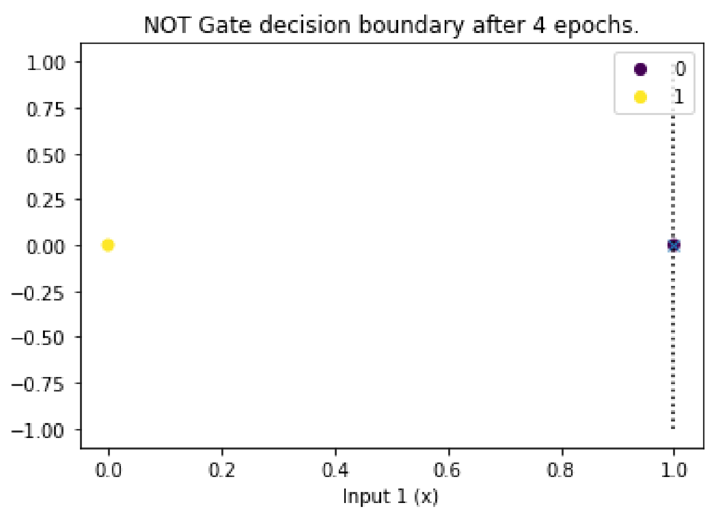
Step 2:



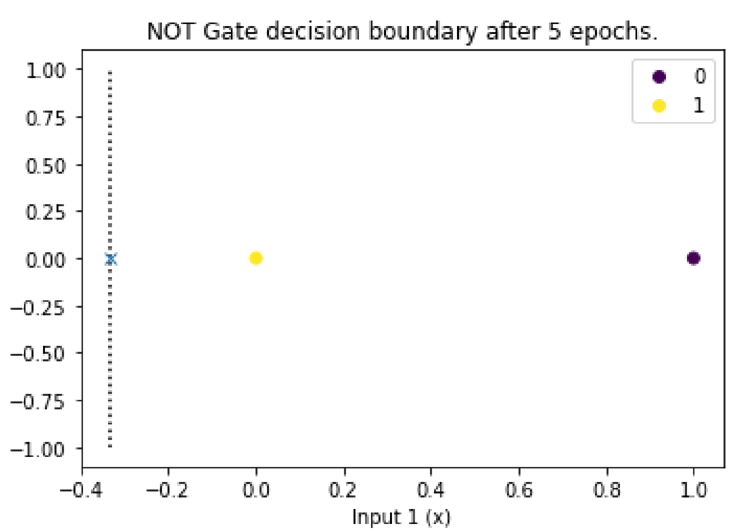
Step 3:



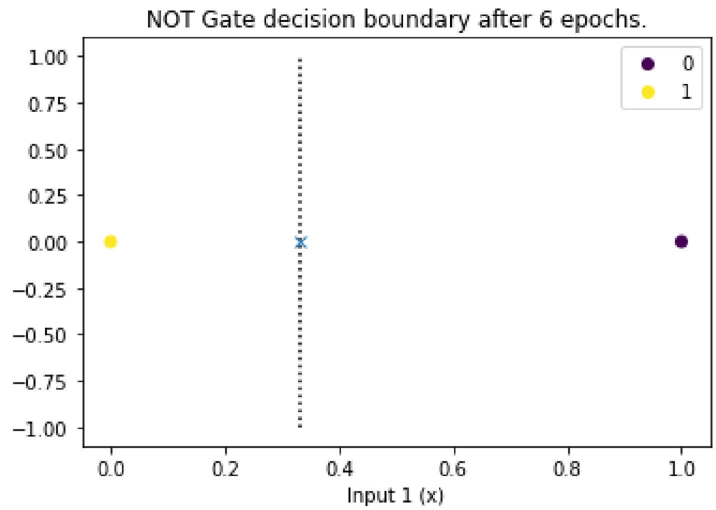
Step 4:



Step 5:



Step 6:



Note:

- Data for NOT gate is 1-dimensional. Therefore, points lie on the same line.
- As data is 1-D, there would be a separating point rather than a boundary. In the plots, point ‘x’ denotes the same.
- Dotted line is just for a better understanding of separation between two classes.

The following observations are made :

- If data is linearly separable it can be classified using perceptron.

1(c) XOR using PTA

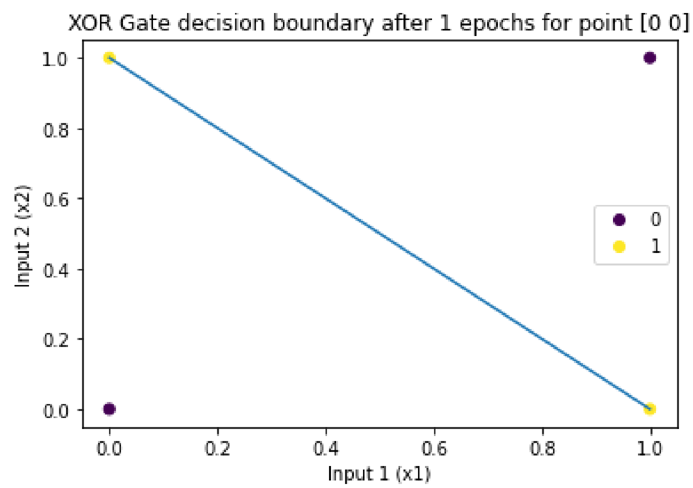
Initial weights are set as w1 = 0.5, w2 = 0.5, bias (or w0) = 0.5.
Maximum epochs used are 10.
<u>Truth Table:</u>

Input(x1)	**Input(x2)**	**Output(y)**
0	0	0
0	1	1
1	0	1
1	1	0

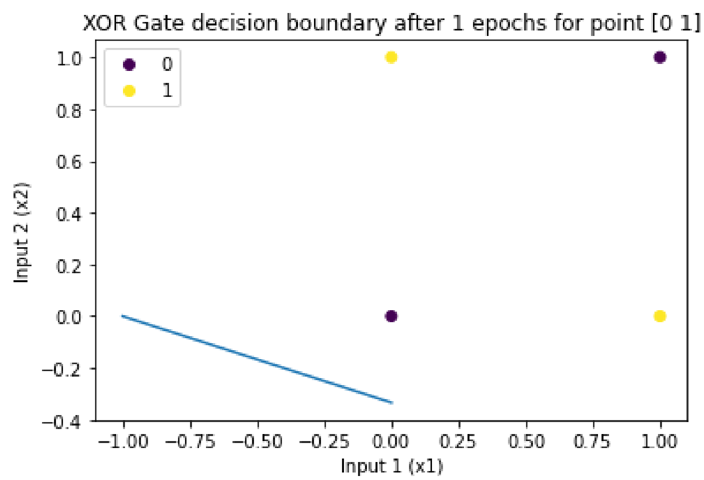
Note: XOR is not a linearly separable data (in 2-dimensional plane).

<u>Graphs:</u>

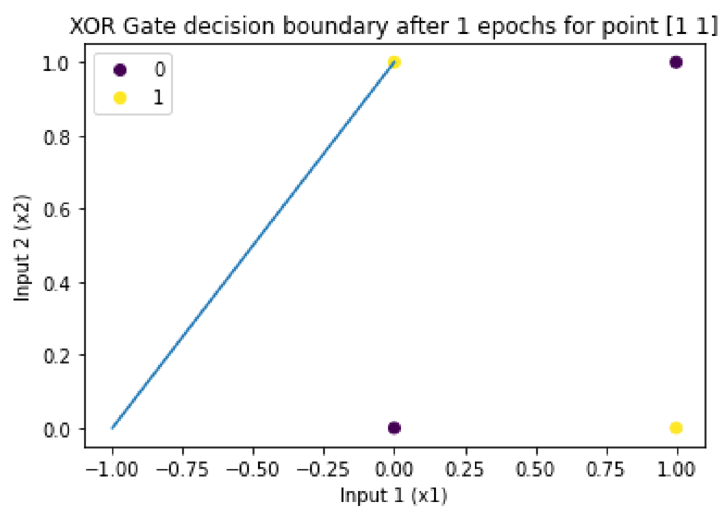
Step 1:



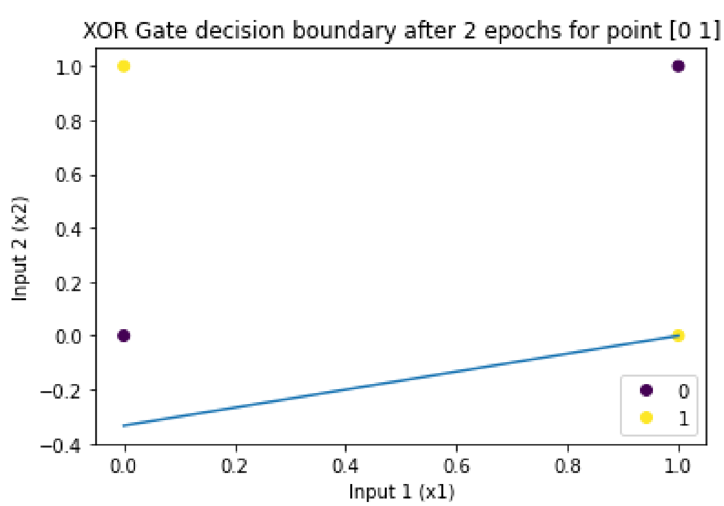
Step 2:



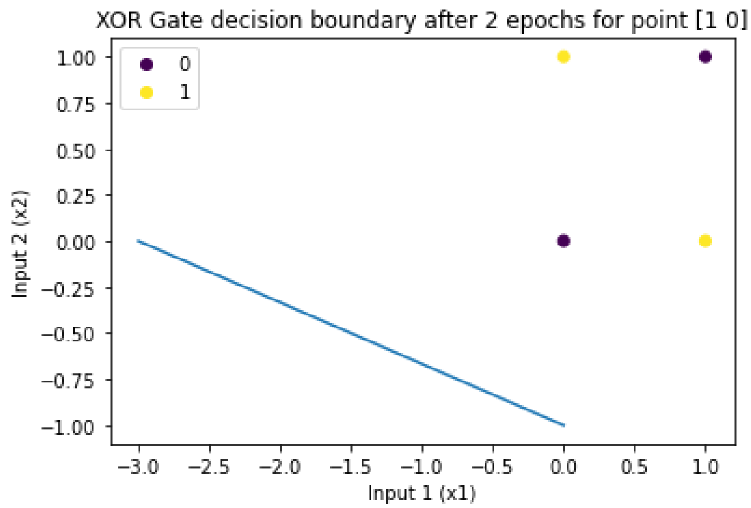
Step 3:



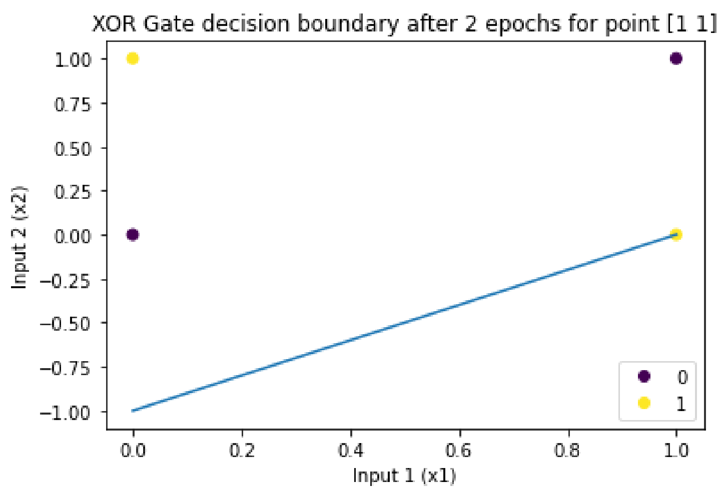
Step 4:



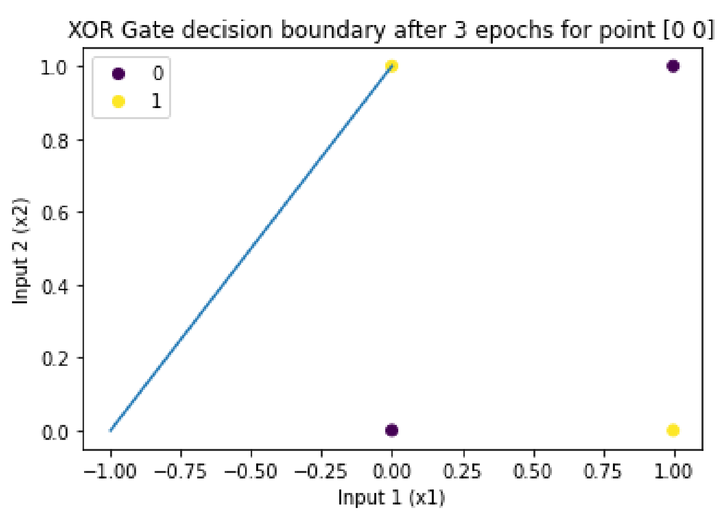
Step 5:


XOR Gate decision boundary after 2 epochs for point [1 0]
Input 2 (x2)
Input 1 (x1)
0
1

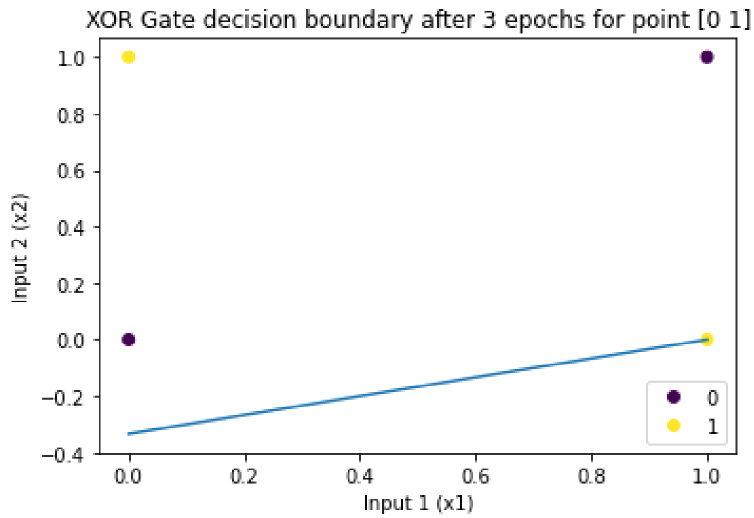
Step 6:


XOR Gate decision boundary after 2 epochs for point [1 1]
Input 2 (x2)
Input 1 (x1)
0
1

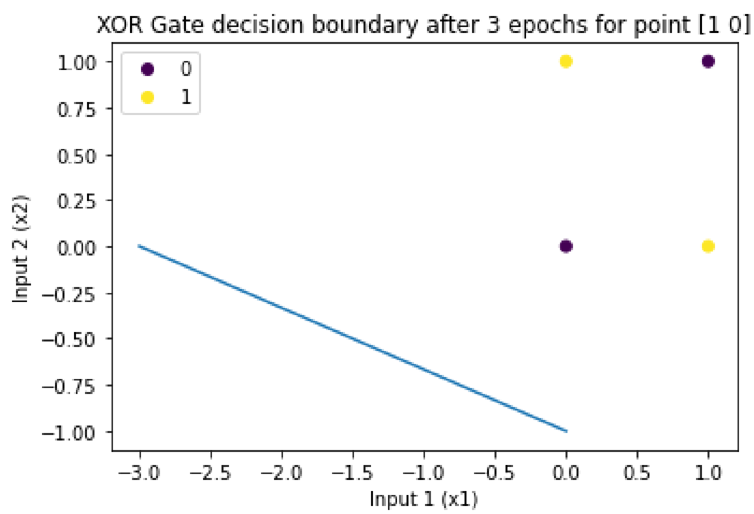
Step 7:


XOR Gate decision boundary after 3 epochs for point [0 0]
Input 2 (x2)
Input 1 (x1)
0
1

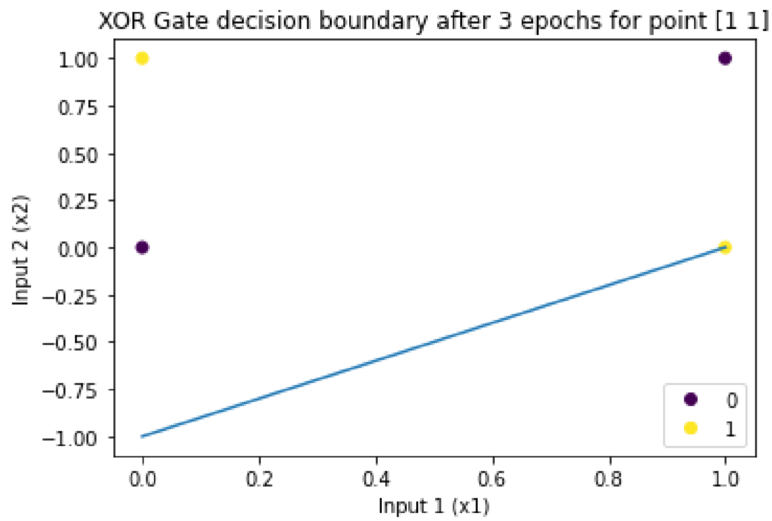
Step 8:



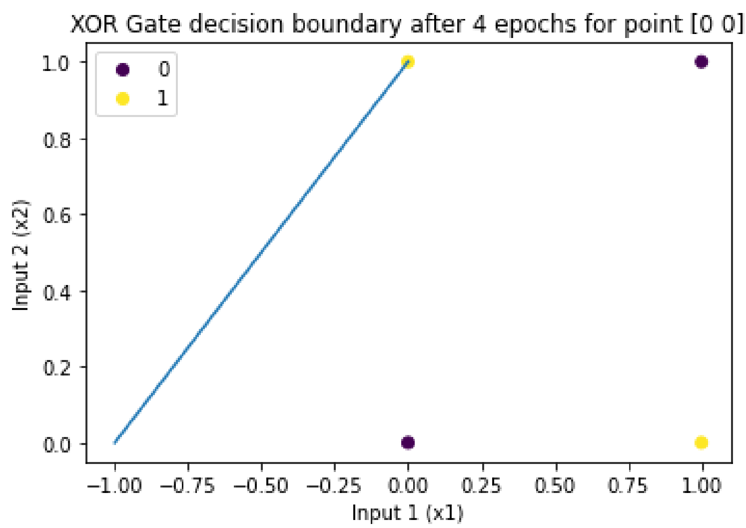
Step 9:



Step 10:



Step 11:



And so on…

<u>Observations:</u>

- The data (XOR) is not linearly inseparable.
- Cycling Theorem states that if the training data is not linearly separable, the algorithm will eventually repeat the same set of weights and, hence, the same set of lines and enter an infinite loop.
- From the plots above, it can be clearly seen that the Cycling Theorem holds as same set of lines is being repeated in a pattern and the algorithm never converges (infinite loop).
- The minimum number of steps required to prove the same is 10.
- It is because steps 3-6 are the set of steps that start repeating (set of 4).
- Therefore, steps 7-10 will verify the pattern as they will match steps 3-6 and verify the Cycling Theorem.
- The set of steps will keep on repeating in an infinite loop.

2. Madaline Learning Algorithm:

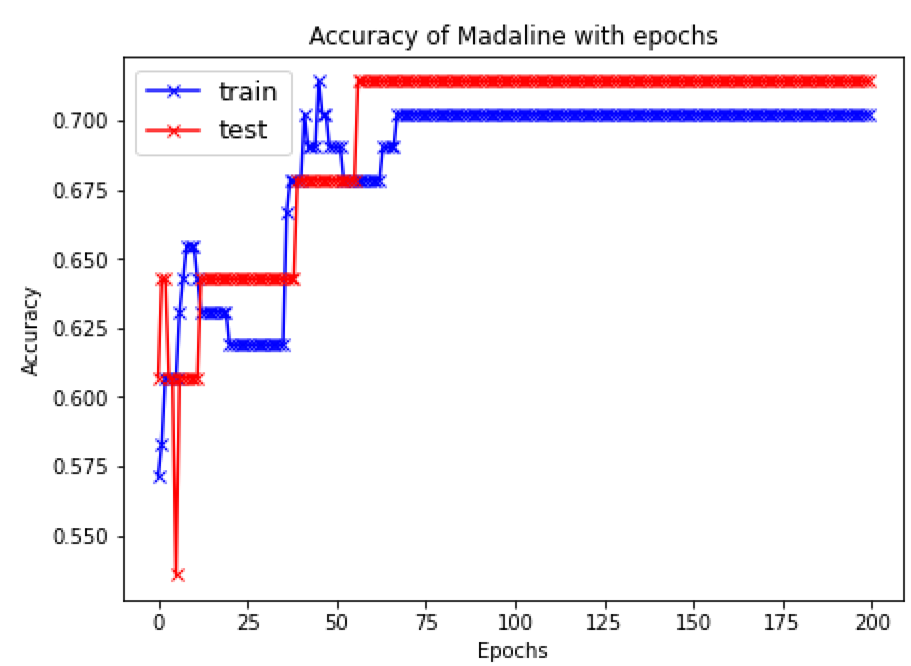
2(a) f1(x1,x2)

Network Details / No of neurons
I have used a two hidden layer network for this function. A two hidden layer network is necessary to learn the given function because of the nature of threshold function (only tells on which side of boundary positive sample is but not where or how much far)
NETWORK DETAILS
Input layer
Hidden layer 1 = 8 perceptrons
Hidden layer 2 = 2 perceptrons
Output layer = 1 perceptron
TOTAL = 11 perceptrons (excluding input layer)

DATA - 112 samples (Train : 126 Test: 42 -> stratified splitting used)
Epoc wise weight updated on each misclassified instance

```
---------------------------------------- 193 -----------------------------------------
Incorrectly classified
Neuron found -----> Updated
Incorrectly classified
Neuron found -----> Updated
Incorrectly classified
Neuron found -----> Updated
Incorrectly classified
Neuron found -----> Updated
Incorrectly classified
Neuron found -----> Updated
Correctly classified
Incorrectly classified
Neuron found -----> Updated
Incorrectly classified
Neuron found -----> Updated
Incorrectly classified
Neuron found -----> Updated
Correctly classified
Incorrectly classified
Neuron found -----> Updated
Incorrectly classified
Neuron found -----> Updated
Incorrectly classified
Neuron found -----> Updated
Correctly classified
Correctly classified
Correctly classified
```

Epoc vs Accuracy



Results and Accuracy

TRAINING CLASSIFICATION REPORT

```
TRAIN
              precision    recall  f1-score   support

        -1.0       1.00      0.36      0.53        39
         1.0       0.64      1.00      0.78        45

    accuracy                           0.70        84
   macro avg       0.82      0.68      0.66        84
weighted avg       0.81      0.70      0.66        84
```

TESTING CLASSIFICATION REPORT

```
TEST
              precision    recall  f1-score   support

        -1.0       1.00      0.38      0.56        13
         1.0       0.65      1.00      0.79        15

    accuracy                           0.71        28
   macro avg       0.83      0.69      0.67        28
weighted avg       0.81      0.71      0.68        28
```

Observation

- It is observed that weight initialization plays a very crucial role in weight updation using a madaline algorithm.
- Here it can be observed that the algorithm gives low accuracy on training as well as test dataset even after 200 epocs and both train and test accuracy remains constant after 100 epocs and no significant learning is happening after it.
- As train and test accuracy is low, the madaline algorithm is underfitting the given function. This is because the function is complex as it involves two hidden layers and madaline does not consider multiple neurons together while weight updation.
- Thus the above results clearly show that the madaline algorithm is not successful in learning this function and underfits the given function.

2(b) f2(x1,x2)

Network Details / No of neurons

I have used a two hidden layer network for this function. A two hidden layer network is necessary to learn the given function because of the nature of threshold function (only tells on which side of boundary positive sample is but not where or how much far)

NETWORK DETAILS

Input layer
Hidden layer 1 = 8 perceptrons
Hidden layer 2 = 4 perceptrons
Output layer = 1 perceptron
TOTAL = 13 perceptrons (excluding input layer)

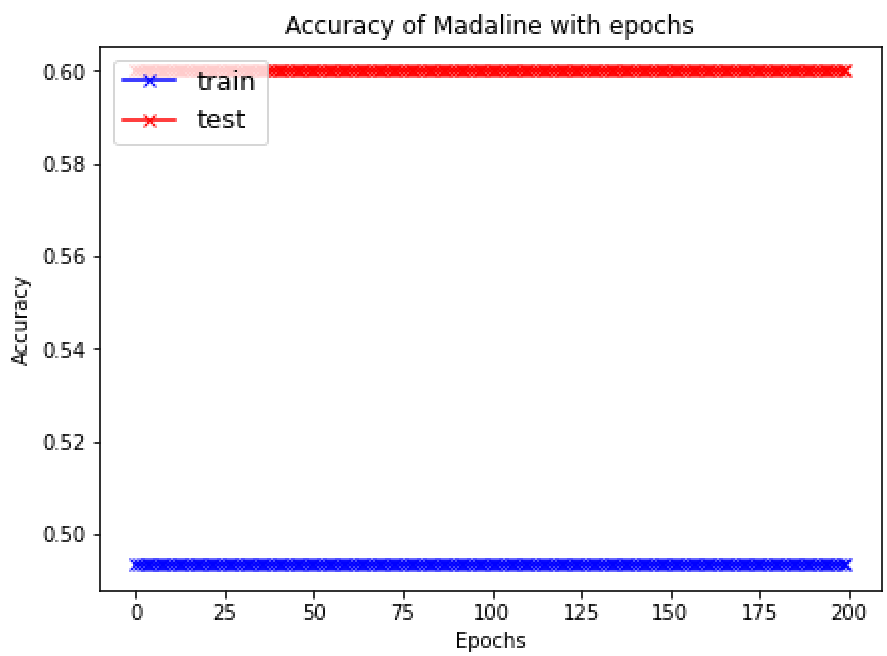
MADALINE FAILED TO LEARN THIS FUNCTION FROM THE GIVEN INITIALIZATION

DATA - 100 samples (Train : 75 Test: 25 -> stratified splitting used)
Epoc wise weight updated on each misclassified instance
(None of the weights were updated during complete 200 epocs)

```
---------------------------------------- 191 -----------------------------------------
Correctly classified
Incorrectly classified
Correctly classified
Incorrectly classified
Correctly classified
Correctly classified
Correctly classified
Correctly classified
Incorrectly classified
Correctly classified
Correctly classified
Incorrectly classified
Correctly classified
Correctly classified
Incorrectly classified
Incorrectly classified
Incorrectly classified
Correctly classified
Incorrectly classified
Incorrectly classified
Correctly classified
Incorrectly classified
Correctly classified
Incorrectly classified
Correctly classified
Correctly classified
Incorrectly classified
Incorrectly classified
Incorrectly classified
Correctly classified
Correctly classified
```

Epoc vs Accuracy

Note:- None of the weights got updated and this accuracy thus remains the same which is calculated over initialised weights.



Results and Accuracy

(Results are over the initial weights as madaline was unable to update weights)

TRAINING CLASSIFICATION REPORT

TRAIN

	precision	recall	f1-score	support
-1.0	1.00	0.06	0.12	63
1.0	0.52	1.00	0.68	63
accuracy			0.53	126
macro avg	0.76	0.53	0.40	126
weighted avg	0.76	0.53	0.40	126

TESTING CLASSIFICATION REPORT

TEST

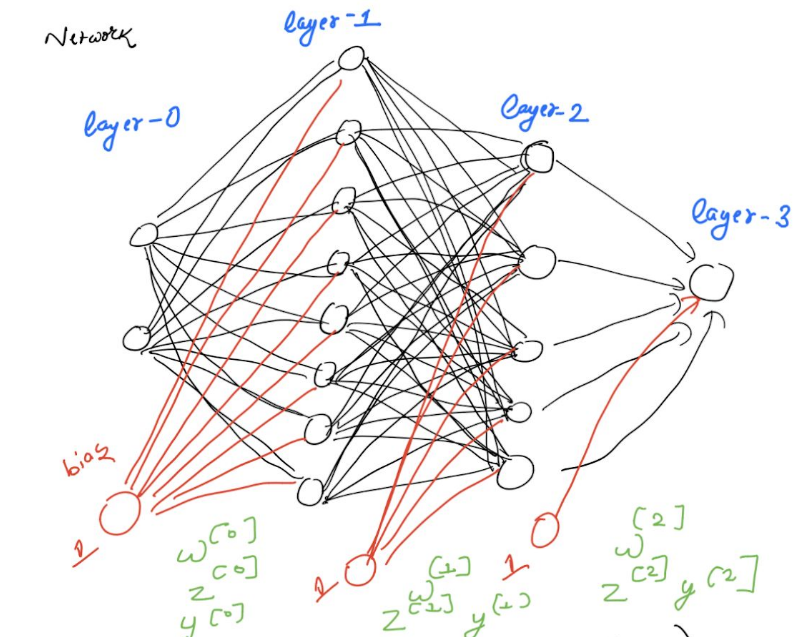
	precision	recall	f1-score	support
-1.0	1.00	0.19	0.32	21
1.0	0.55	1.00	0.71	21
accuracy			0.60	42
macro avg	0.78	0.60	0.52	42
weighted avg	0.78	0.60	0.52	42

Observation

- It is observed that weight initialization plays a very crucial role in weight updation using a madaline algorithm.
- Madaline was unable to learn the given function because it was complex and at any particular step flipping just one adeline unit did not make any change in the final output. This happened because multiple neurons together must have been flipped to obtain the correct output but as madaline does not consider multiple neurons together, It fails to learn this particular function.
- Thus the above results clearly show that the madaline algorithm is not successful in learning this function.
- Detailed Calculation are shown below.

CALCULATIONS

1. Network Explanation



All initialised to 0.5 (weights + biases)

1 initialization

$$\left.\begin{aligned} w^{[0]} &= (2\times 8) \\ b^{[0]} &= (1\times 8) \\ w^{[1]} &= (8\times 4) \\ b^{[1]} &= (1\times 4) \\ w^{[2]} &= (4\times 1) \end{aligned}\right\} \text{all } 0.5$$

$$b^{[2]} = (1\times 1)$$

2. Epoc-one (positive sample taken)

Weights are all initialized to 0.5 and the first positive sample taken which is correctly classified so no updates made in weights required.

epoc = 1 (I) (3,0)   label = 1

$$X = \underset{1\times 2}{[3 \;\; 0]} \qquad \underset{2\times 8}{w^{[0]}} = \begin{bmatrix} 0.5 & 0.5 & \cdots \\ 0.5 & 0.5 & \cdots \end{bmatrix} \qquad \underset{1\times 8}{b^{[0]}} = [0.5 \;\; 0.5 \;\; 0.5 \cdots]$$

So

$$z^{[0]} = [1.5+0.5 \quad 1.5+0.5 \quad 1.5+0.5 \quad 1.5+0.5 \quad 1.5+0.5 \quad 1.5+0.5 \quad 1.5+0.5 \quad 1.5+0.5]$$

$$y^{[0]} = [1 \;\; 1 \;\; 1 \;\; 1 \;\; 1 \;\; 1 \;\; 1 \;\; 1]_{1\times 8}$$

$$\underset{8\times 4}{w^{[1]}} = \begin{bmatrix} 0.5 & 0.5 & \cdots \\ 0.5 & 0.5 & \cdots \\ \vdots & \vdots & \end{bmatrix} \qquad \underset{1\times 4}{b^{[1]}} = [0.5 \;\; 0.5 \;\; 0.5 \;\; 0.5]$$

$$z^{[1]} = [4.5 \;\; 4.5 \;\; 4.5 \;\; 4.5]$$

$$y^{[1]} = [1 \;\; 1 \;\; 1 \;\; 1]$$

$$\underset{5\times 1}{w^{[2]}} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} \qquad b^{[2]} = [0.5]$$

$$z^{[2]} = [2.5]$$

$y^{[2]} = 1$ (label matched do nothing)

3. Epoc-one (negative sample taken)
Here feedforward is done over a negative sample which is incorrectly classified, then all affine values flipped and checked that flipping any single adeline unit affine value does not make the classification correct. Further just by changing a few adeline units flips together , one could easily observe that the network was properly able to classify negative samples. But madaline does not incorporate, or allow updating multiple adeline units together. This fails to classify this particular function.

II $(0, 6)$ label $= -1$

$$x_{1\times 2} = \begin{bmatrix} 0 & 6 \end{bmatrix} \qquad w^{[0]}_{2\times 8} = \begin{bmatrix} 0.5 & 0.5 & \cdots \\ 0.5 & 0.5 & \cdots \end{bmatrix} \qquad b^{[0]}_{1\times 8} = \begin{bmatrix} 0.5 & 0.5 & 0.5 & \cdots \end{bmatrix}$$

So

$$z^{[0]} = \begin{bmatrix} 3.5 & 3.5 & 3.5 & 3.5 & 3.5 & 3.5 & 3.5 & 3.5 \end{bmatrix}$$

$$y^{[0]} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}_{1\times 8}$$

$$w^{[1]}_{8\times 4} = \begin{bmatrix} 0.5 & 0.5 & \cdots \\ 0.5 & 0.5 & \cdots \\ \vdots & \vdots & \end{bmatrix} \qquad b^{[1]}_{1\times 4} = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix}$$

$$z^{[1]} = \begin{bmatrix} 4.5 & 4.5 & 4.5 & 4.5 \end{bmatrix}$$

$$y^{[1]} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

$$w^{[2]}_{5\times 1} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} \qquad b^{[2]} = \begin{bmatrix} 0.5 \end{bmatrix}$$

$$z^{[2]} = \begin{bmatrix} 2.5 \end{bmatrix}$$ (label Not matched)

$$y^{[2]} = 1$$

changing $Z^{(i)}$ Values & checking

Z [0]	flipped y [0]	Z [1]	y [1]	Z[2]	y[2]
3.5	1 −1	3.5−0.5+0.5	1	3	1
3.5	1	3.5	1		
3.5	1	3.5	1		
3.5	1	3.5	1		
3.5	1		1		
3.5	1				
3.5	1				
3.5	1				

Similarly flipping any one does not work in layer [0]

Z [0]	y [0]	Z [1]	y [1]	Z[2]	y[2]
3.5	1	4.5	1 −1	0.25	1
3.5	1	4.5	1		
3.5	1	4.5	1		
3.5	1	4.5	1		
3.5	1		1		
3.5	1				
3.5	1				
3.5	1				

flipping any one does not work in layer [1]

Multiple flips together will work,
but not captured using Madaline :—

z [0]	flipped y [0]	z [1]	y [1]	z [2]	y [2]
3.5	1 -1	-0.5	-1	-1.5	-1
3.5	1 -1	-0.5	-1		
3.5	1 -1	-0.5	-1		
3.5	1 -1	-0.5	-1		
3.5	1 -1				
3.5	1				
3.5	1				
3.5	1				

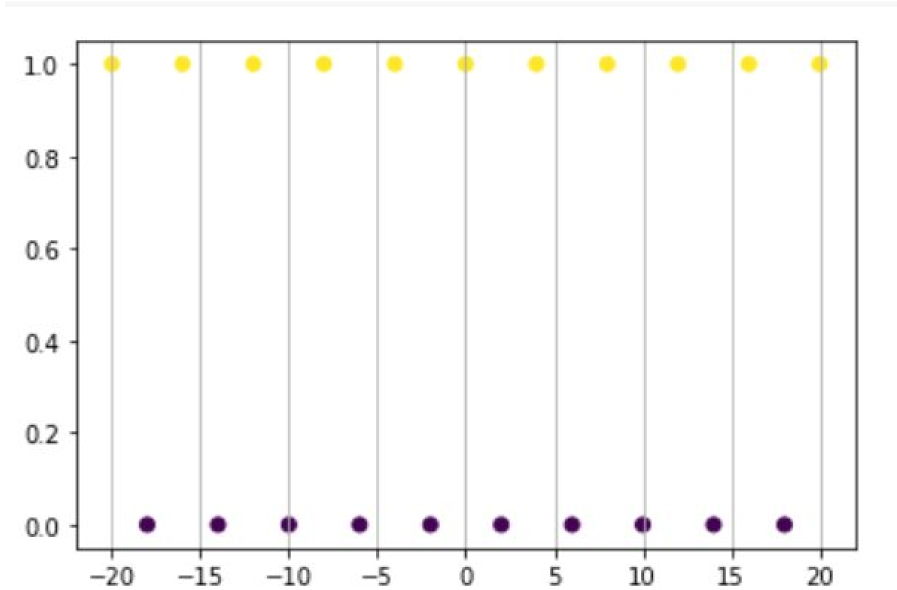
matched. Now apply multiple adalines

CONCLUSION :-

1. Madaline is not able to learn complex functions because of the fact that it fails to update multiple adaline neurons together and considers only one at a time.
2. Madaline is highly affected by initial weights , that is one set of weights may help madaline converge while other may not even update any weights.
3. Madaline is easily able to converge for AND, OR, XOR in a few epochs as they are simpler functions.

3. Generalized Delta Rule

Scatter plot of the dataset

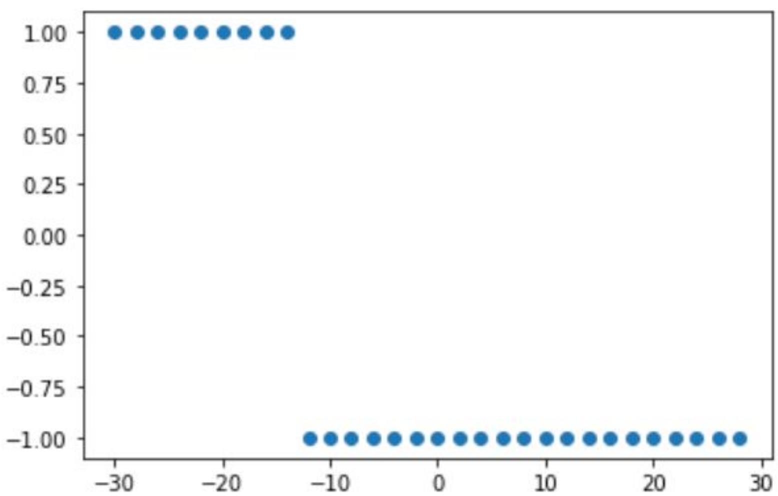


1-D dataset with labels 0 and 1

Observations

1. Sigmoid

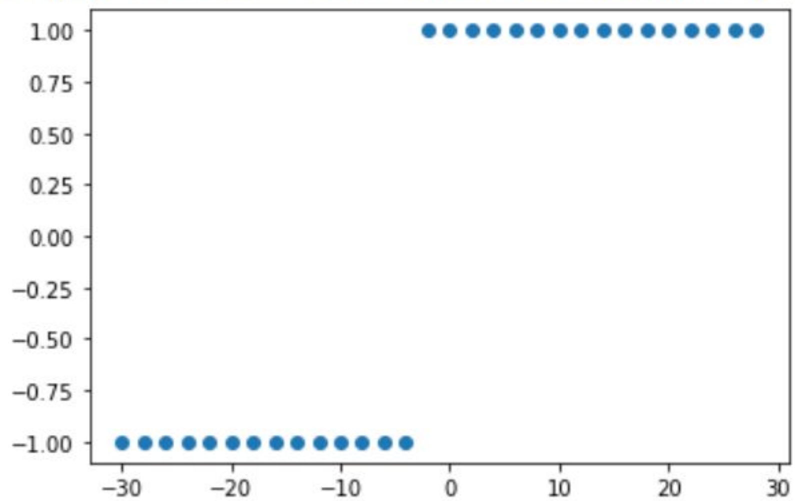
```
Learnt weights are [[-0.02321425 -0.29468309]]
Train Accuracy 59.09090909090909
Test Accuracy 37.5
Dummy sample Accuracy 100.0
<matplotlib.collections.PathCollection at 0x7f2dc11b2358>
```



Sigmoid: Learnt Weights and Accuracy

2. Tanh

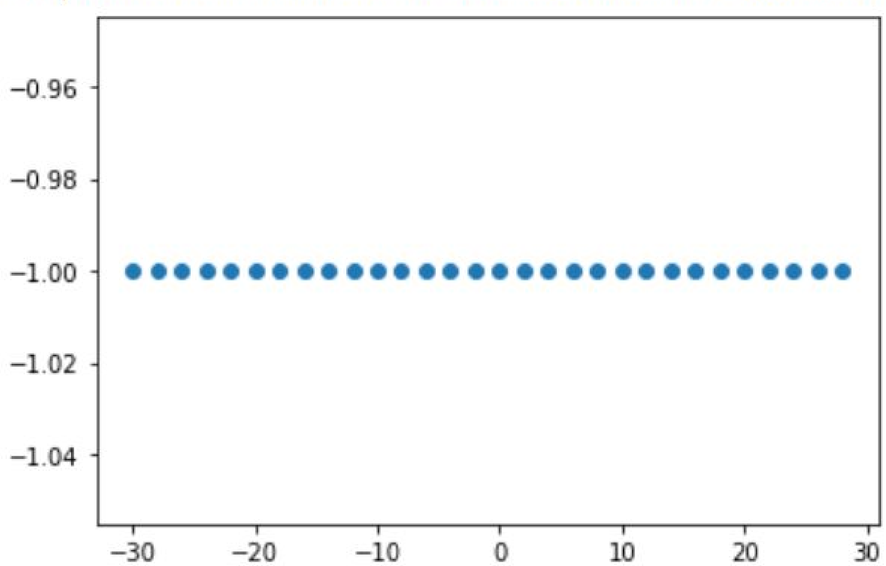
```
Learnt weights are [[0.10434457 0.3381356 ]]
Train Accuracy 50.0
Test Accuracy 50.0
Dummy sample Accuracy 100.0
<matplotlib.collections.PathCollection at 0x7f2dc1558828>
```



Tanh: Learnt Weights and Accuracy

3. Relu

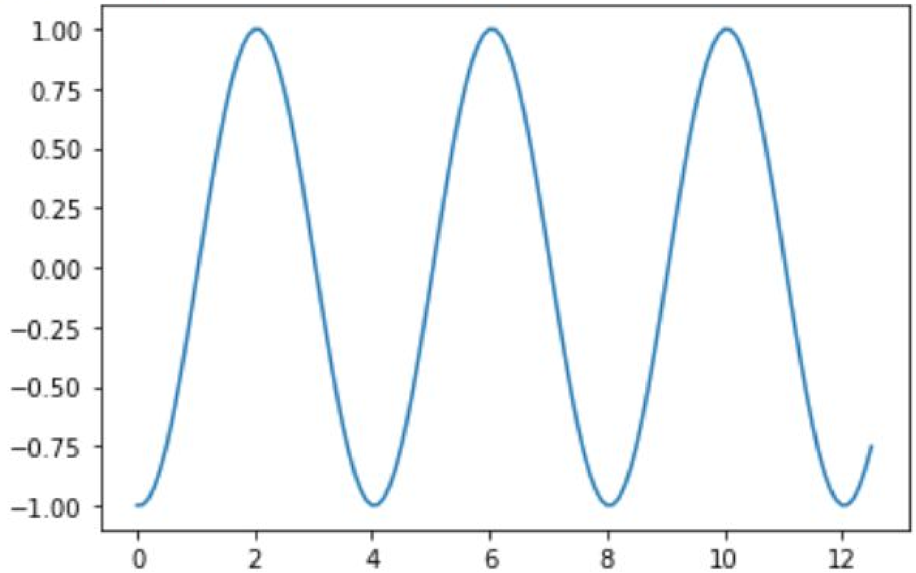
```
Learnt weights are [[ 0.15855732 -8.32313963]]
Train Accuracy 54.54545454545454
Test Accuracy 37.5
Dummy sample Accuracy 0.0
<matplotlib.collections.PathCollection at 0x7f2dc1292ba8>
```



Relu: Learnt Weights and Accuracy

4. Cosine

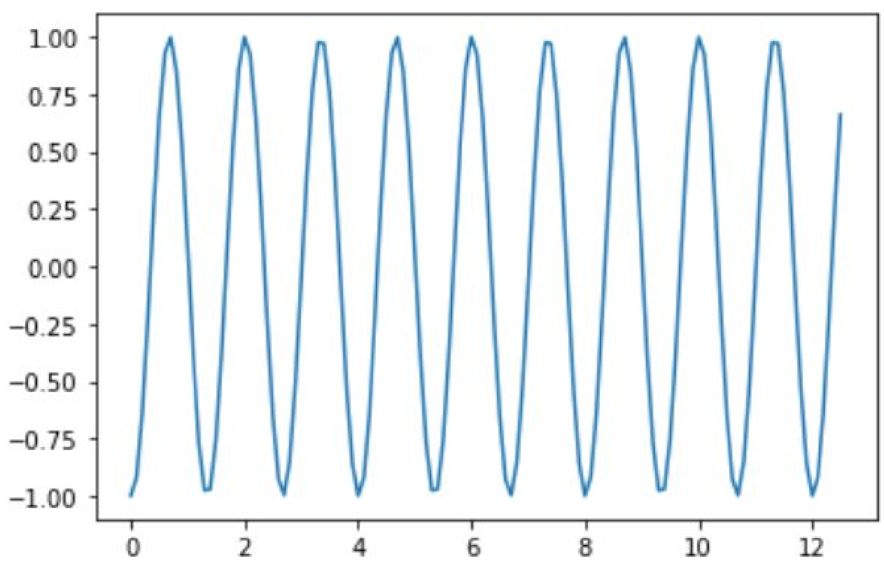
```
Learnt weights are [[-1.57093784  0.06613848]]
Train Accuracy 100.0
Test Accuracy 100.0
Dummy sample Accuracy 100.0
```



Cosine: Learnt Weights and Accuracy

5. Sine

```
Learnt weights are [[4.71253547 1.50175503]]
Train Accuracy 100.0
Test Accuracy 100.0
Dummy sample Accuracy 100.0
```



Sine: Learnt Weights and Accuracy