Soft Computing

Assignment 1: Rosenblatt Perceptron and Linear Adaptive Filters

Vinay Mohan Behara (U00851261)

1. a.

   In Inductive reasoning, the conclusion of an inductive argument is based on the evidence given. In linear regression, it gives the relationship between a dependent variable with at least one independent variable. The linear adaptive filter is usually trained with a dataset and after applying the LMS optimization algorithm, we get the best fit for a training dataset. So, here comes the problem. If in case the dataset has some almost similar points, the hypothesis does not alter a lot. When given a larger dataset with different points, like we have a data point which is completely different from the previous set of input points, the model completely changes. So, the performance of the model is not satisfactory if it was given a large data set with outliers. This problem is caused by induction.

   b.

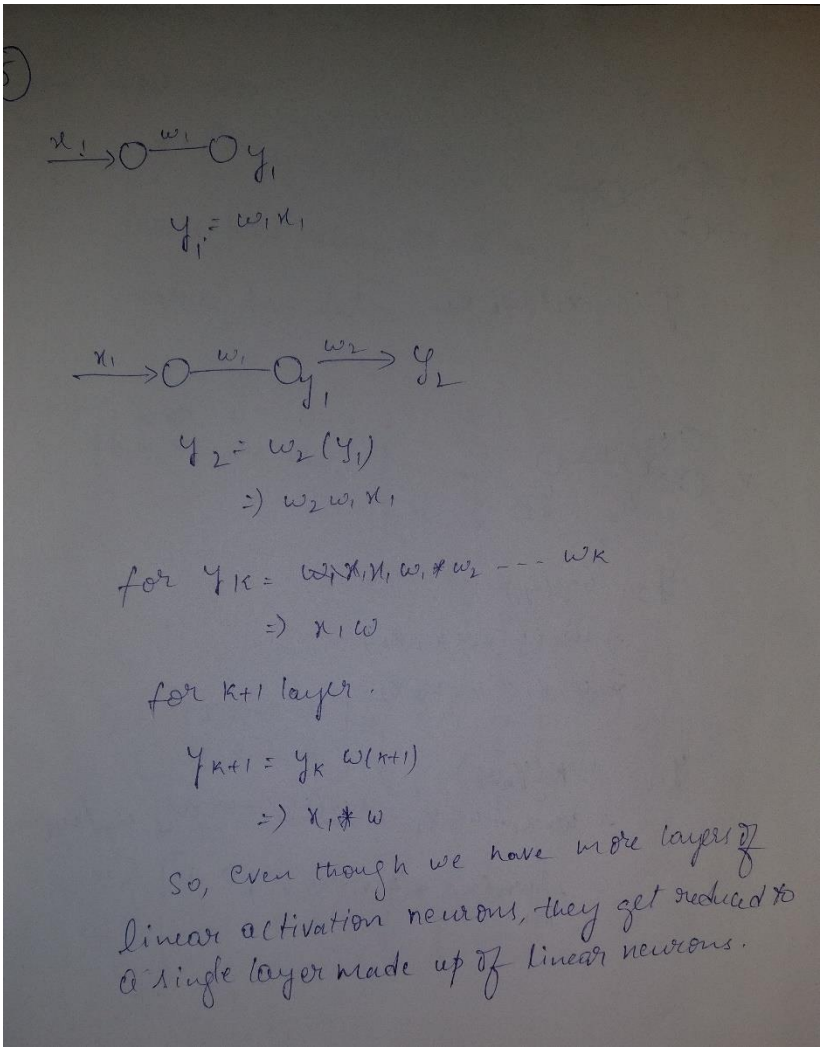
   The Rosenblatt's perceptron is built for a nonlinear neuron. It is the process of linear combination followed by applying the signum function. It classifies the data into two or more classes. For example, a fully trained Rosenblatt perceptron classifies the data into two classes, then if we have a set of input points that are different from the two possible classes, the trained perceptron, instead of classifying it as a new class, it classifies as one of the class which we have as output earlier. This is a consequence of the problem of induction.

2. **Hebb's Rule:** This rule states that learning is performed by change in synaptic gap between the dendrites and the same theory is used in neural networks. Like we can change the weights of neurons, the neurons learns from the input given. We know that unsupervised learning is nothing but there is no training database. So, the neuron learns from the input but not from the training database. Hebb's rule also states that by changing the synaptic gap between the dendrites, the neuron A triggers an information to neuron B if they are connected. So, Hebb's rule modifies the connections to neurons based on the inputs from which it can be considered as an example for unsupervised learning.

   General Delta Rule: In this method, we calculate the difference between the actual output and the calculated output. This is the error rate. So, a training database is given which is used to compare the output obtained from the neural network. Supervised learning is all about having a training data and comparing the obtained output with training data which has the input and output pairs. So, the General Delta Rule can be considered as an example of a supervised learning.

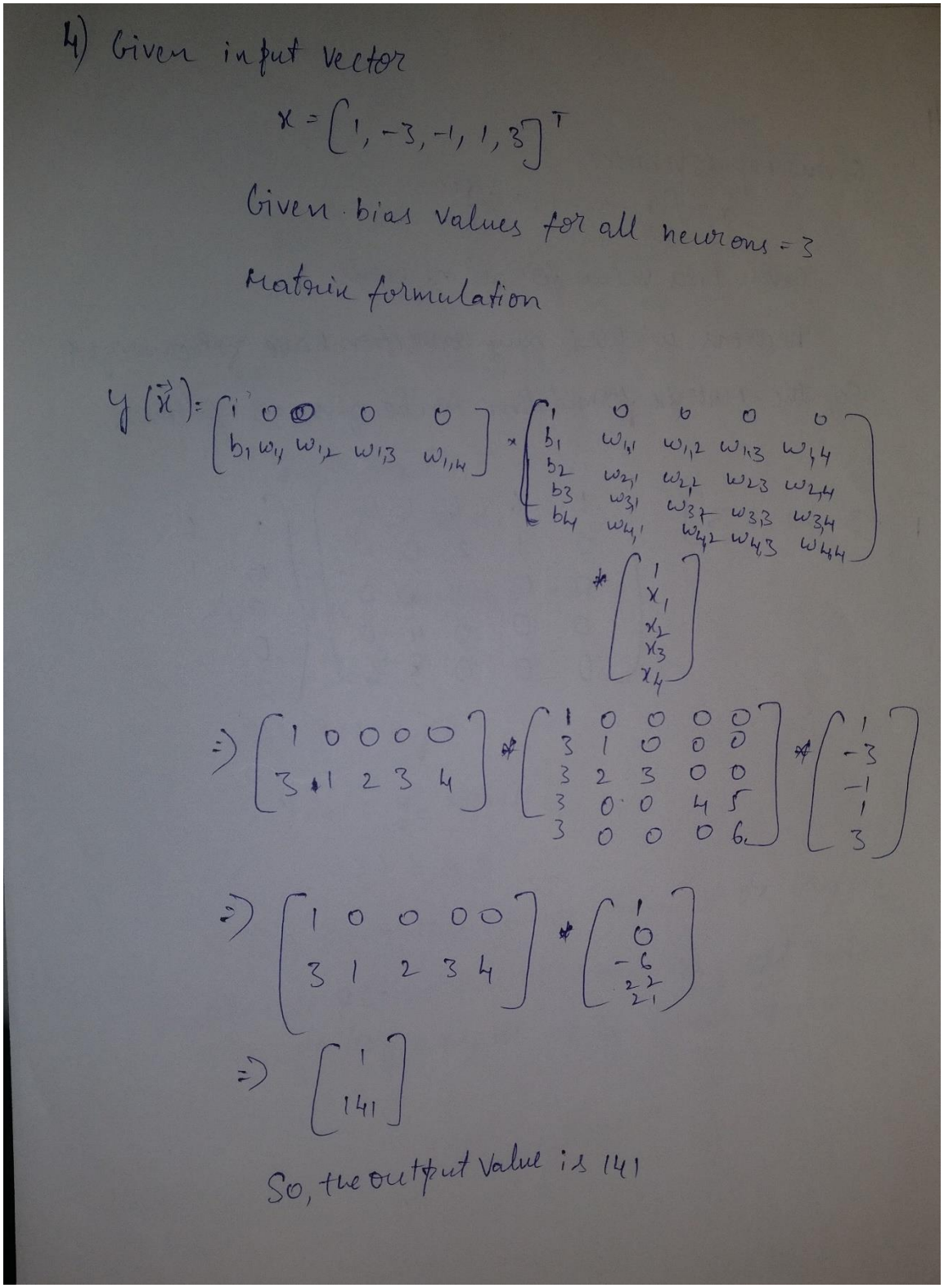
5. The most important aspect of Artificial Neural networks lies in its activation function. The complex behaviour of artificial neural networks is obtained by non-linearity. The linear activation function always gives a linear output which reduces the whole network to a single layer even we try adding additional layers of linear activation function neurons to the pre-existing network.

   Even though we have many layers, if all the layers are linear, the final activation function is a linear function of input of the first layer. So, multiple layers of linear transfer function are pointless.



6. Python notebooks were downloaded.

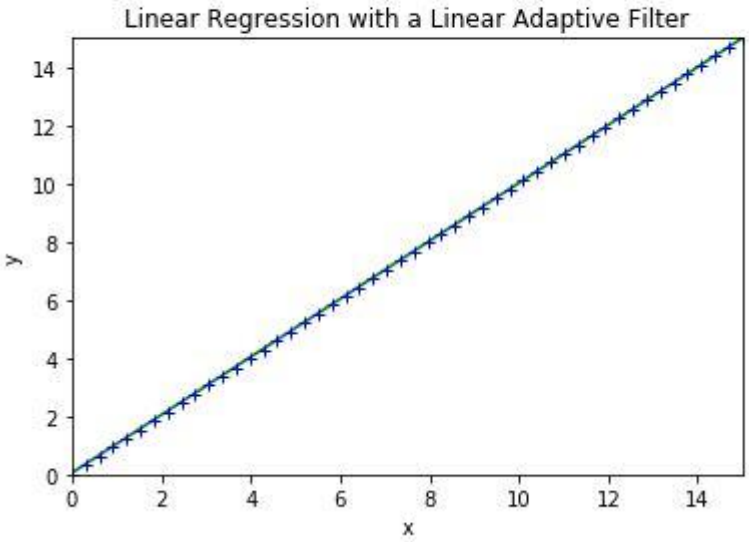
4.


4) Given input vector
$x = [1, -3, -1, 1, 3]^T$
Given bias values for all neurons = 3
Matrix formulation
$y(\vec{x}) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ b_1 & w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ b_1 & w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ b_2 & w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ b_3 & w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \\ b_4 & w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} \end{bmatrix} * \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$
$\Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 3 & 1 & 2 & 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 & 0 \\ 3 & 2 & 3 & 0 & 0 \\ 3 & 0 & 0 & 4 & 5 \\ 3 & 0 & 0 & 0 & 6 \end{bmatrix} * \begin{bmatrix} 1 \\ -3 \\ -1 \\ 1 \\ 3 \end{bmatrix}$
$\Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 3 & 1 & 2 & 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \\ -6 \\ 22 \\ 21 \end{bmatrix}$
$\Rightarrow \begin{bmatrix} 1 \\ 141 \end{bmatrix}$
So, the output value is 141

8. LMS learning will not continue to work in all cases if I were to change the linear activation function with ReLu activation function. ReLu is nonlinear in nature. Having normalized weights gives us a network which yields mostly zero because of the condition given by the ReLu function. So, this indicates only few neurons dies and the learning stops. Because of this the gradient can lead to a zero value since the weights gets adjusted during the gradient descent. So, the neurons which die because of this condition doesn't make the LMS algorithm to learn. Though the neurons which are still active participate in the learning process of the algorithm. In order to avoid this problem, a slight change that is to change the $x < 0$ condition which equals to $0.01x$ instead of 0. This is called a leaky ReLu and this helps the neurons not to die completely and continue with the learning process unlike ReLu which makes the neurons to die.

7. LMS algorithm with y points along a perfect line

```
Inital Error 9.204304435374896
Final error after 51 epochs of training = 0.000982095366689205
```



a. The line which is uncommented is :

   Y_points = [x + random.uniform(-2.0, 2.0) for x in X_points]

   The line which is commented is :

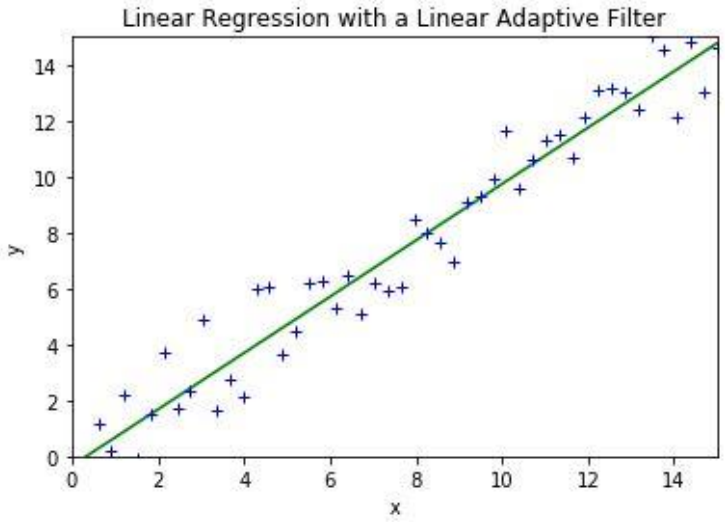
   Y_points = [x for x in X_points]

b. Running the cell several times

The output table is given as follows

iterations	Initial error	Final error	No of epochs
1	15.48	1.5	1000
2	5.54	1.48	1000
3	57.99	1.30	1000
4	2.11	1.11	1000

```
Inital Error 2.114952523832338
Final error after 10000 epochs of training = 1.119001655815869
```



The graph plotted above shows the minimum error obtained for 1000 epochs and with the randomized values, the error alters significantly.

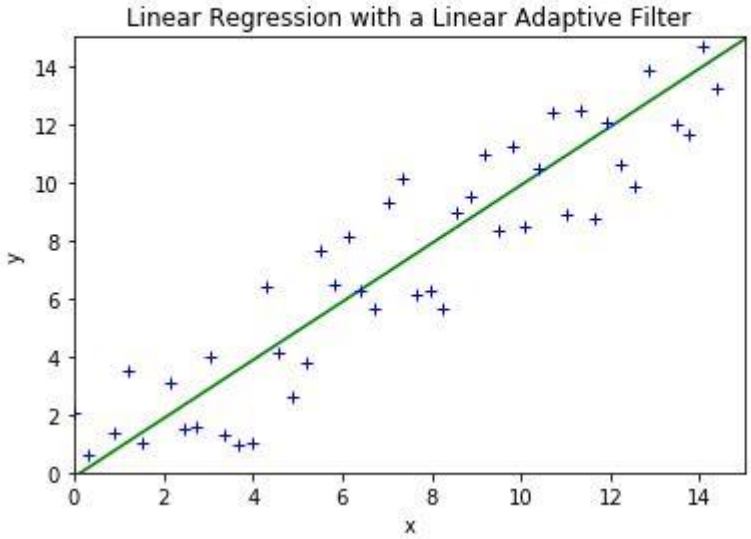
c. Modified code

Y_points = [x + random.uniform(-3.0, 3.0) for x in X_points]

d. The output for more randomness is given as follows

iterations	Initial error	Final error	No of epochs
1	66.35	5.6	1000
2	28.14	3.13	1000
3	66.38	3.02	1000
4	10.98	3.01	1000

```
Inital Error 10.984046266154513
Final error after 10000 epochs of training = 2.9315962132514244
```



The above graph shows the obtained error with increased randomness and it clearly shows the error has increased.

e. In the given code, the loop for updating weights using learning rate and epochs might me varied in order to get some better output. In this case we may improve the termination condition by increasing the mean square error value. The condition should be in such a way that the error value obtained while running the epochs should be compared with the errors obtained in the last iteration and it should stop at a given boundary condition by us. It should be a very small value.

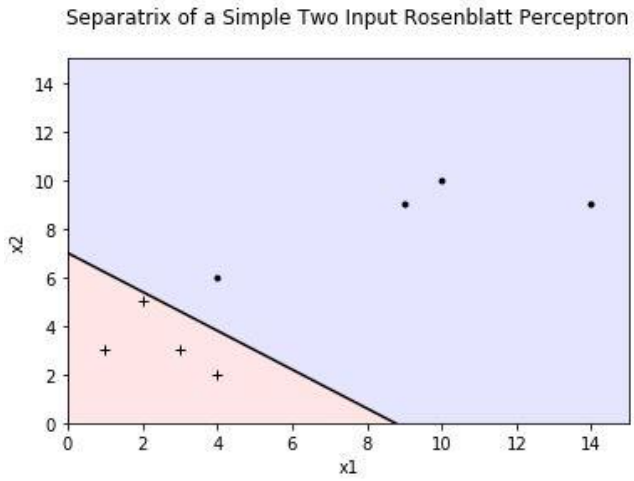
f. I have calculated the error values for 40000 epochs and then kept on reducing the epochs where I got the optimum error value for 12000 epochs which is 1.001

3. a. The weights and bias are taken as follows

bias = -35

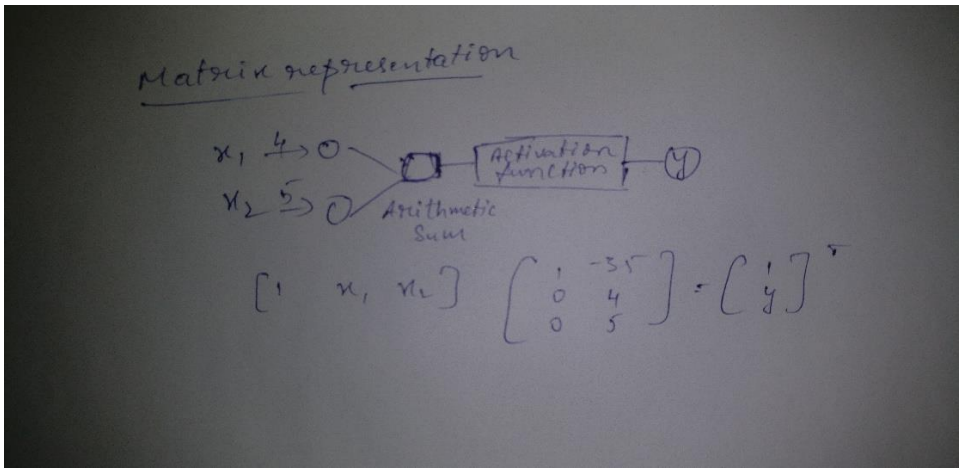
weight_1 = 4

weight_2 = 5



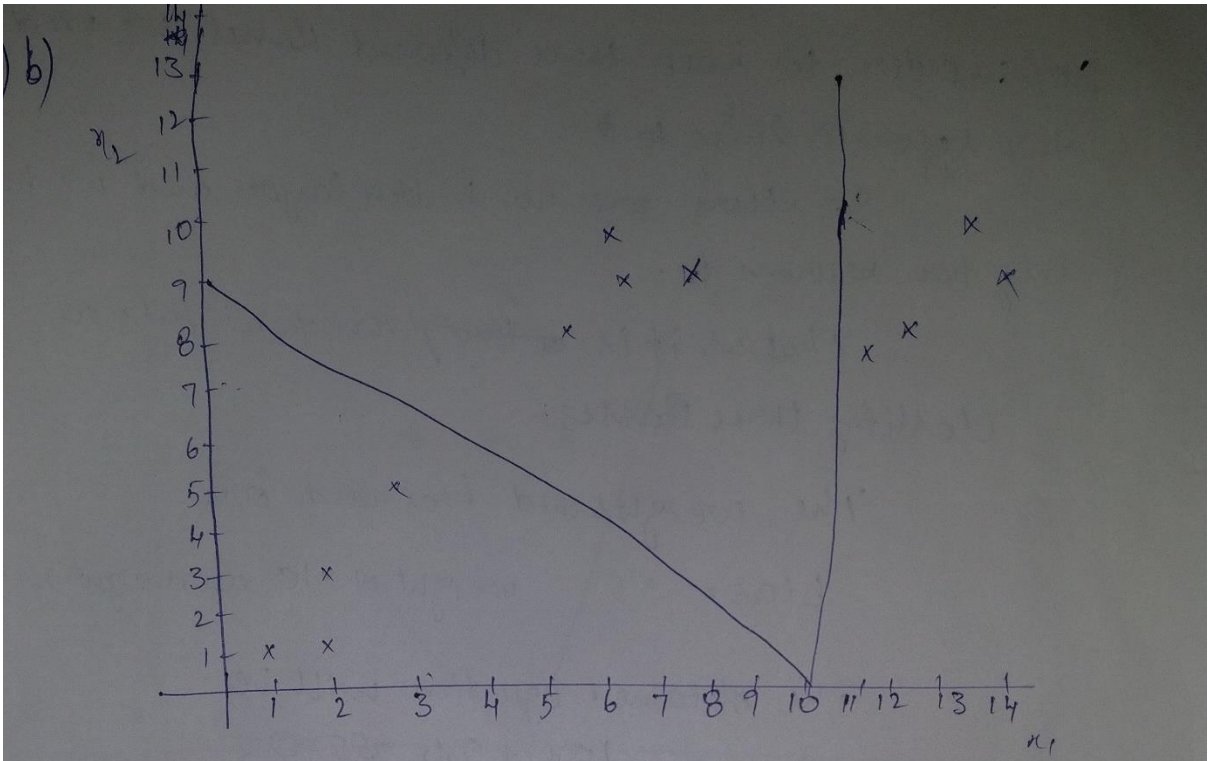

```
Class A input pair [1.0, 1.0]      belongs to filter class -1.0
Class A input pair [1.0, 3.0]      belongs to filter class -1.0
Class A input pair [4.0, 2.0]      belongs to filter class -1.0
Class A input pair [3.0, 3.0]      belongs to filter class -1.0

Class B input pair [10.0, 10.0]    belongs to filter class 1.0
Class B input pair [8.0, 6.0]      belongs to filter class 1.0
Class B input pair [14.0, 12.0]    belongs to filter class 1.0
Class B input pair [9.0, 9.0]      belongs to filter class 1.0
```

The classification is done correctly is based on the bias and weights taken. The final equation can be given as y = signum(4x1+5x2-35)



3.b & 3.c

b)



In order to classify two classes, one neuron is sufficient. But for three classes we should have three neurons.

The considered points are:

x_1	x_2	class
1	1	A
2	1	A
2	3	A
3	5	A
6	8	B
7	9	B
7	10	B
8	9	B

x_1	x_2	class
11	7	C
12	8	C
13	10	C
14	9	C

So, inorder to have three different classes, we need two seperate lines.

So, there are no hidden layers and we ha only two neurons.

That is it is given two inputs to classify three classes.

The weights and bias used are

bias = -55 weight-1 = 10 and weight-2 = 9

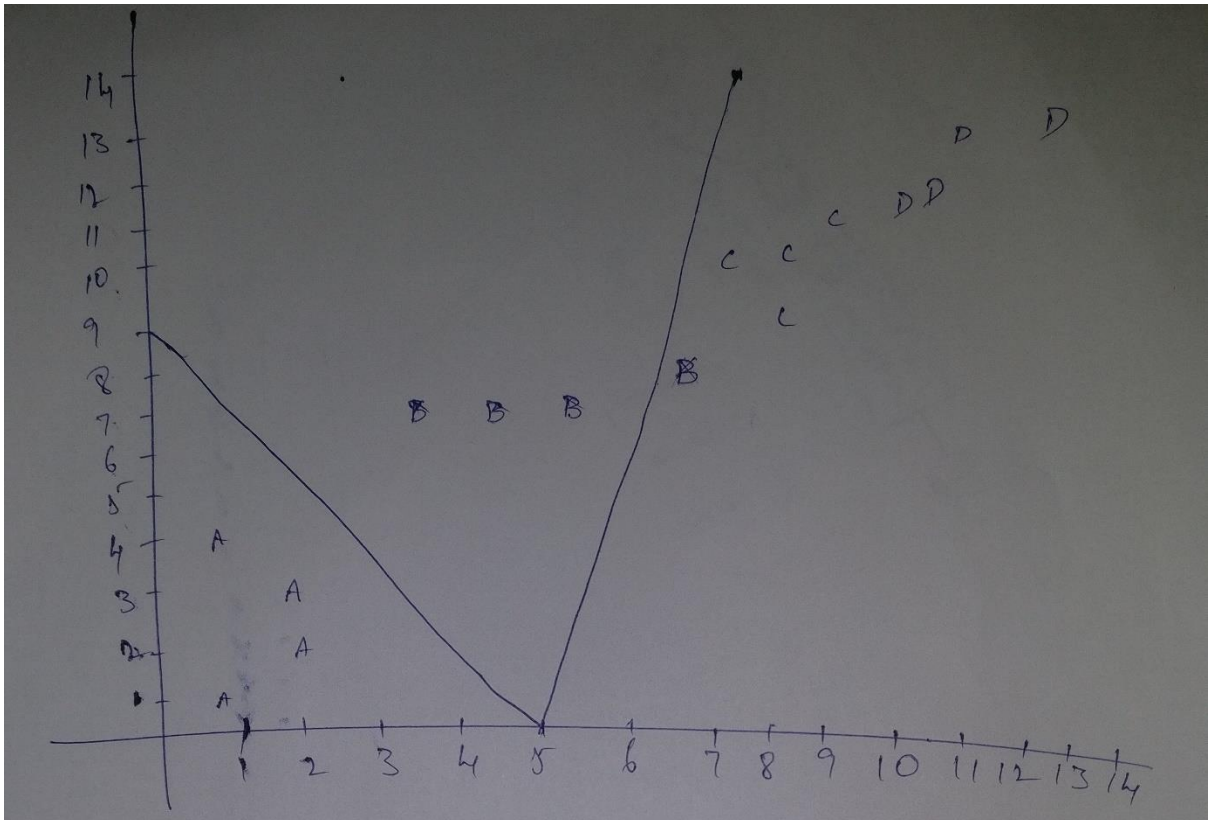
The obtained equation will be:

$10x_1 + 9x_2 - 55 = 0$.

3.c) In this case we will take four different classes

x_1	x_2	class.
1.	1	A
1	4	A
2	2	A
2	3	A
4	7	B
5	7	B
6	7	B
7	8	B

x_1	x_2	class.
7.	10	C
9	10	C
8	9	C
9	11	C
10	11	D
11	13	D
10	12	D
14	13	D



from the above graph, it is not possible to classify four classes using two neurons. with the changing bias and weights, given points cannot be classified into four classes.