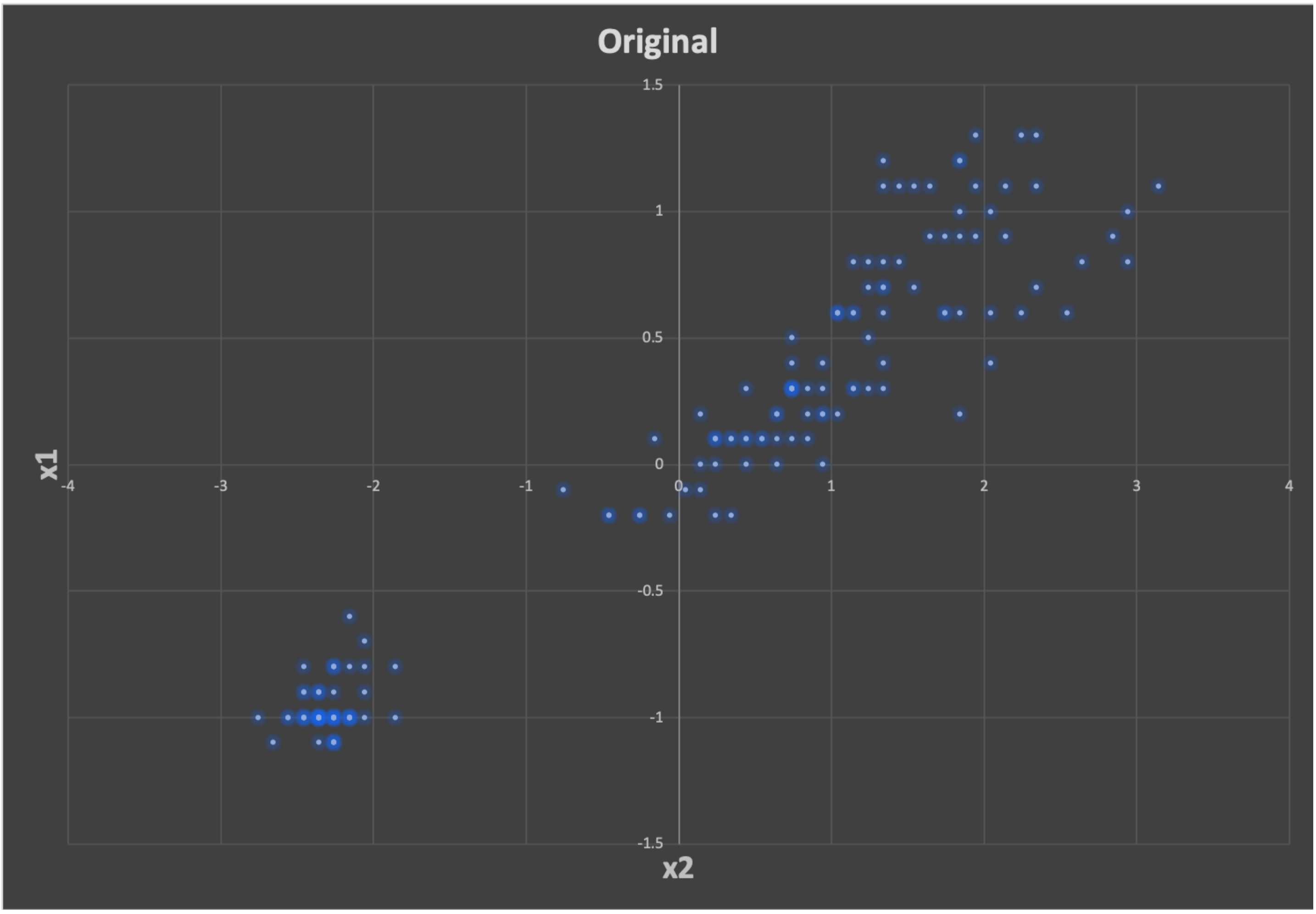
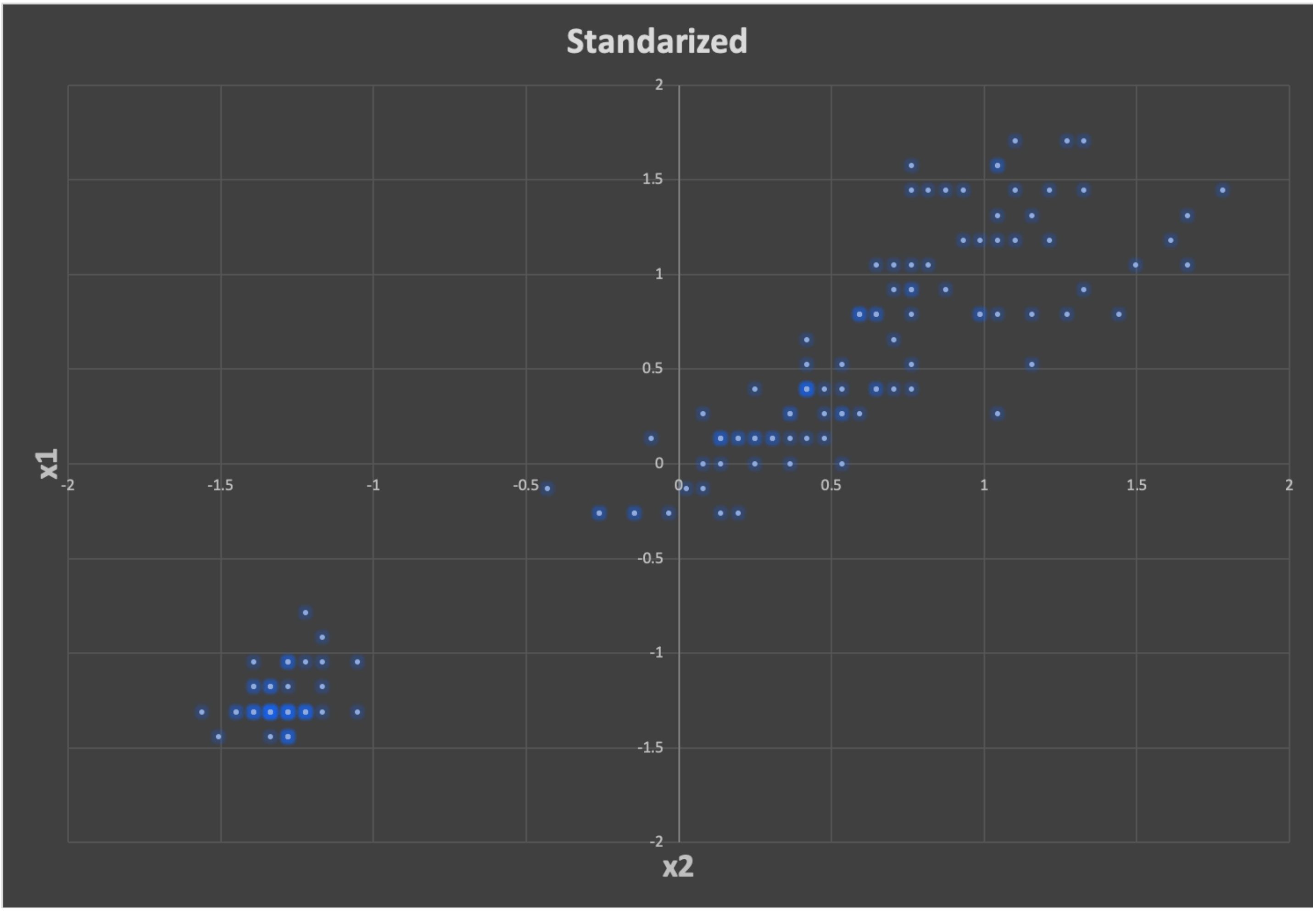
Artificial Intelligence Final Report

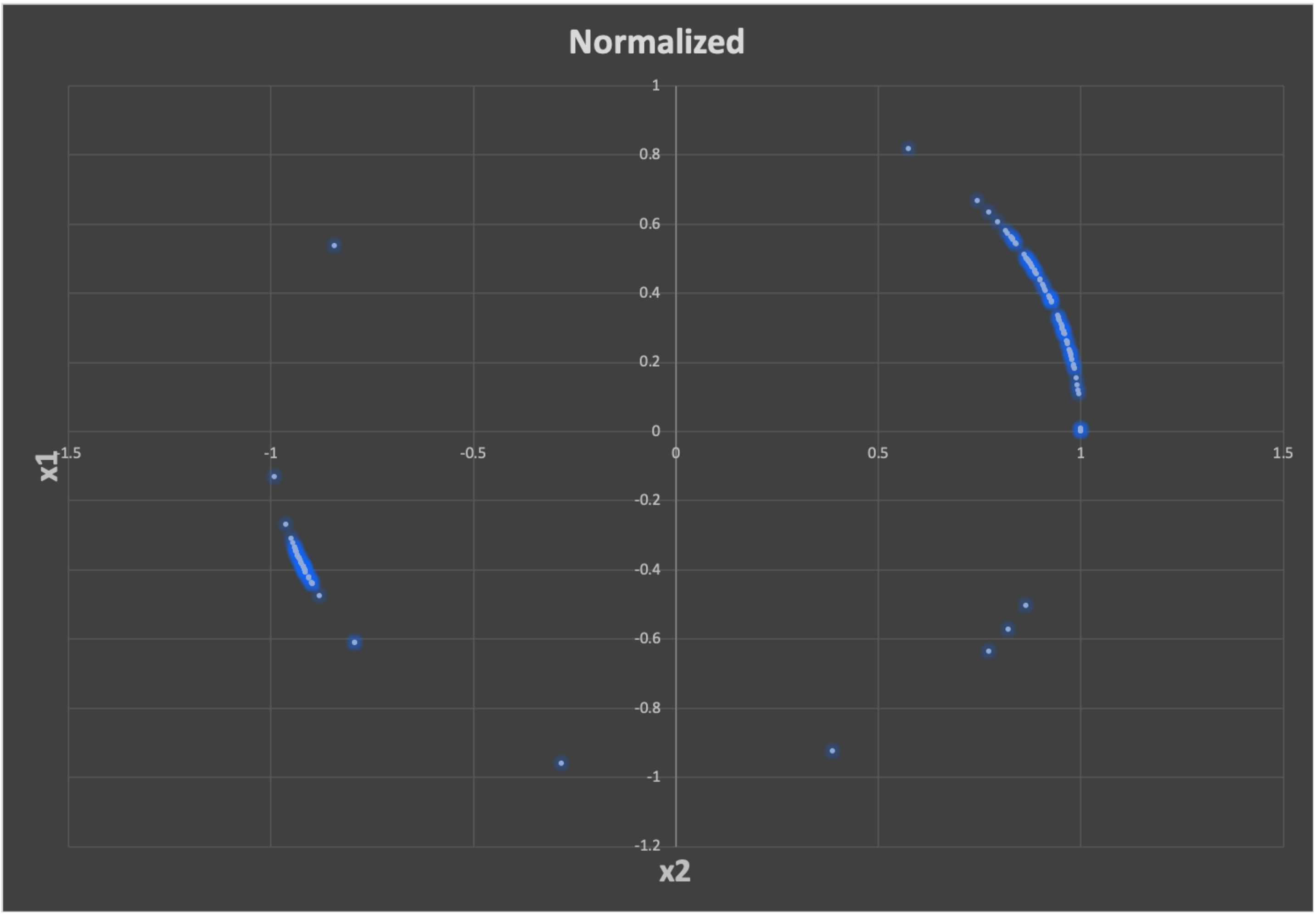
2.1. Competitive learning

Data Exploration

First step to solving this problem was to visualize the provided dataset. Before graphing the data, to get better understanding, I used two methods to change the format of the provided dataset. First was standardization, and second was normalization. Below are the three resulting graphs:




Standarized
x1
x2
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


Normalized
x1
x2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.5
-1
-0.5
0
0.5
1
1.5

From this, we can tell that the data has a positive trend with slope of about 5/12 and is clustered to two major areas. Since standardized data is very similar to that of the original, I will only perform Kohonen's algorithm to original and normalized data.

Set Up

When starting weights are random, whether it be for 2, 3, or 7 neurons, there was always a higher risk of algorithm picking a point where either point density is originally low, or non existent at all (as shown in the images below). This is the case even when iteration is over 200 and learning rates adjusted. Thus, for all results that is labeled as manual weighting below have a pre-selected weights that best describe the given data. They are as follows and are in format of (x1, x2):

For original:
2 Neurons:

1. (-2.5, -1.0)
2. (0.43, 0.43)

3 Neurons:

1. (-2.5, -1.0)
2. (0.43, 0.43)
3. (2.0, 1.0)

7 Neurons:

1. (-2.5, -1.0)
2. (0.43, 0.43)
3. (2.0, 1.0)
4. (-2.2, -1.0)
5. (0.2, 0.2)
6. (1.3, 0.8)
7. (1.9, 1.2)

For Normalized:
2 Neurons:

1. (-0.9, -0.4)
2. (0.8, 0.4)

3 Neurons:

1. (-0.9, -0.4)
2. (0.8, 0.4)
3. (1.0, 1.0)

7 Neurons:

1. (-0.9, -0.4)
2. (0.8, 0.4)
3. (1.0, 1.0)
4. (0.75, 0.5)
5. (-0.85, -0.36)
6. (0.95, 0.2)

7. (-0.75, -0.42)

For all cases, learning constant was set at 0.25 and iteration was set at 120 iterations. This is because setting the initial weights of the neurons manually while maintaining above 100 iteration yielded most constant results.

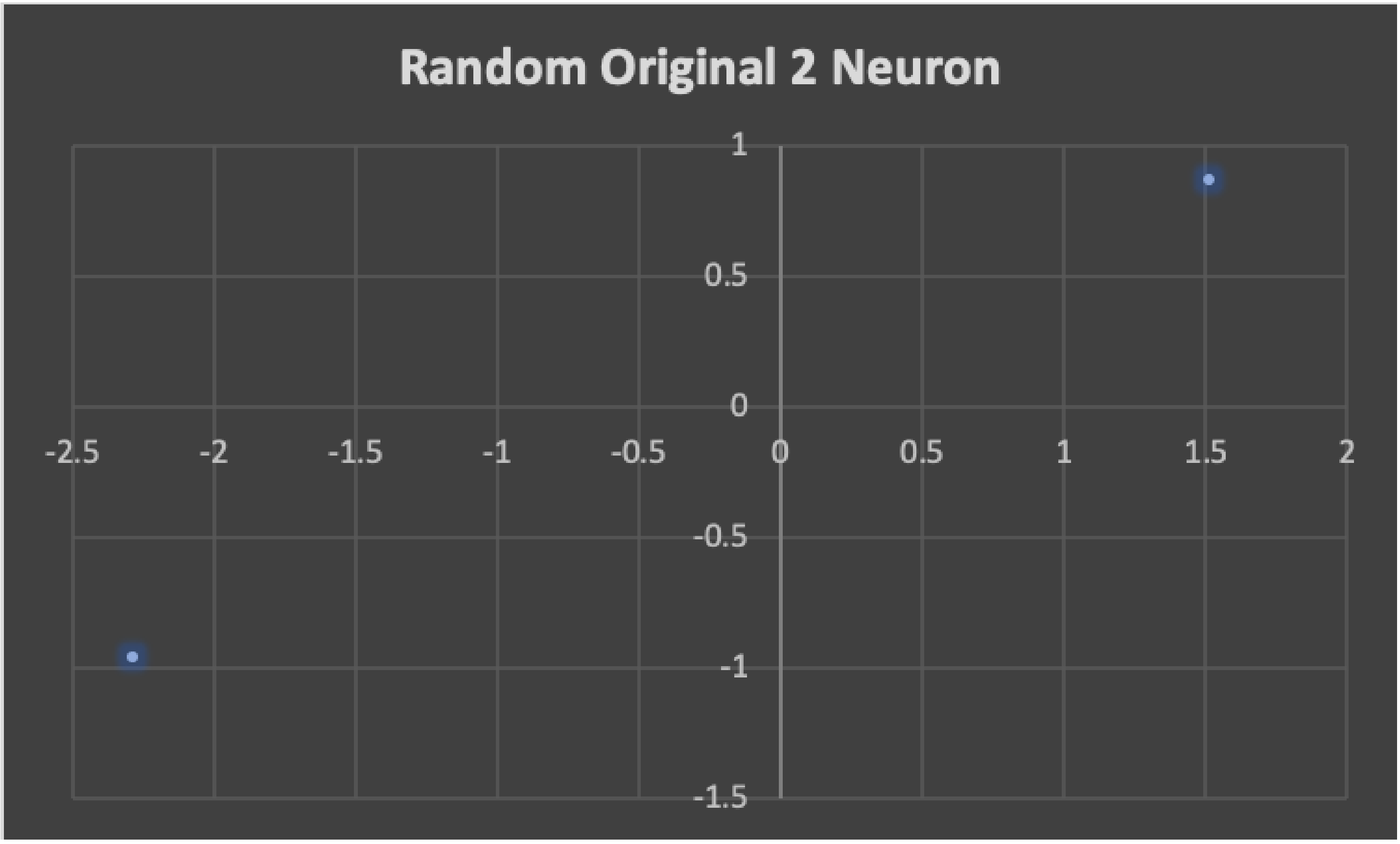
Results

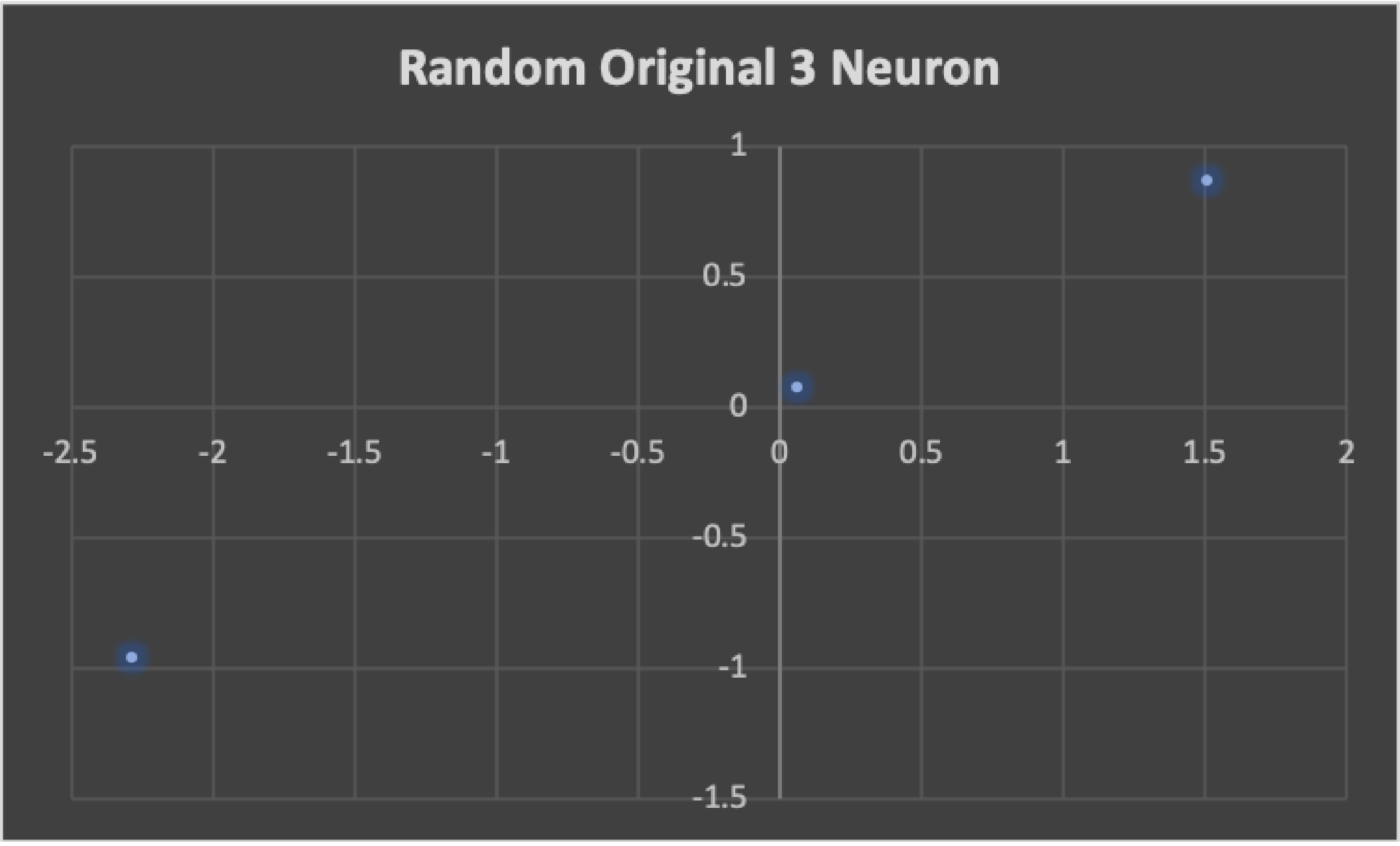
```
// Original dataset

For 2 neurons:

-2.287026339313784, -0.9576085857229641

1.507977470146223, 0.8734214441190423

For 3 neurons:

-2.287026339313784, -0.9576085857229641

1.2498605052413307, 0.6264038046200225

1.7339488432800634, 1.0875072556668575

For 7 neurons:

-2.4022423317049544, -0.9694348056217195

0.5665437740949107, 0.20802880523749756

2.416247569901489, 0.9394699152922313

-2.1652026352522924, -0.9334280763551498

-0.18069304728413738, -0.06288949856104088

1.2965397930756226, 0.6469079531996834

1.6612424881943286, 1.0673134478332993

Random - For 2 neurons:

1.507977470146223, 0.8734214441190423

-2.287026339313784, -0.9576085857229641

Random - For 3 neurons:

0.05877280956085648, 0.08145493834525411

-2.287026339313784, -0.9576085857229641

1.5079789050986494, 0.8734217444124367

Random - For 7 neurons:
```

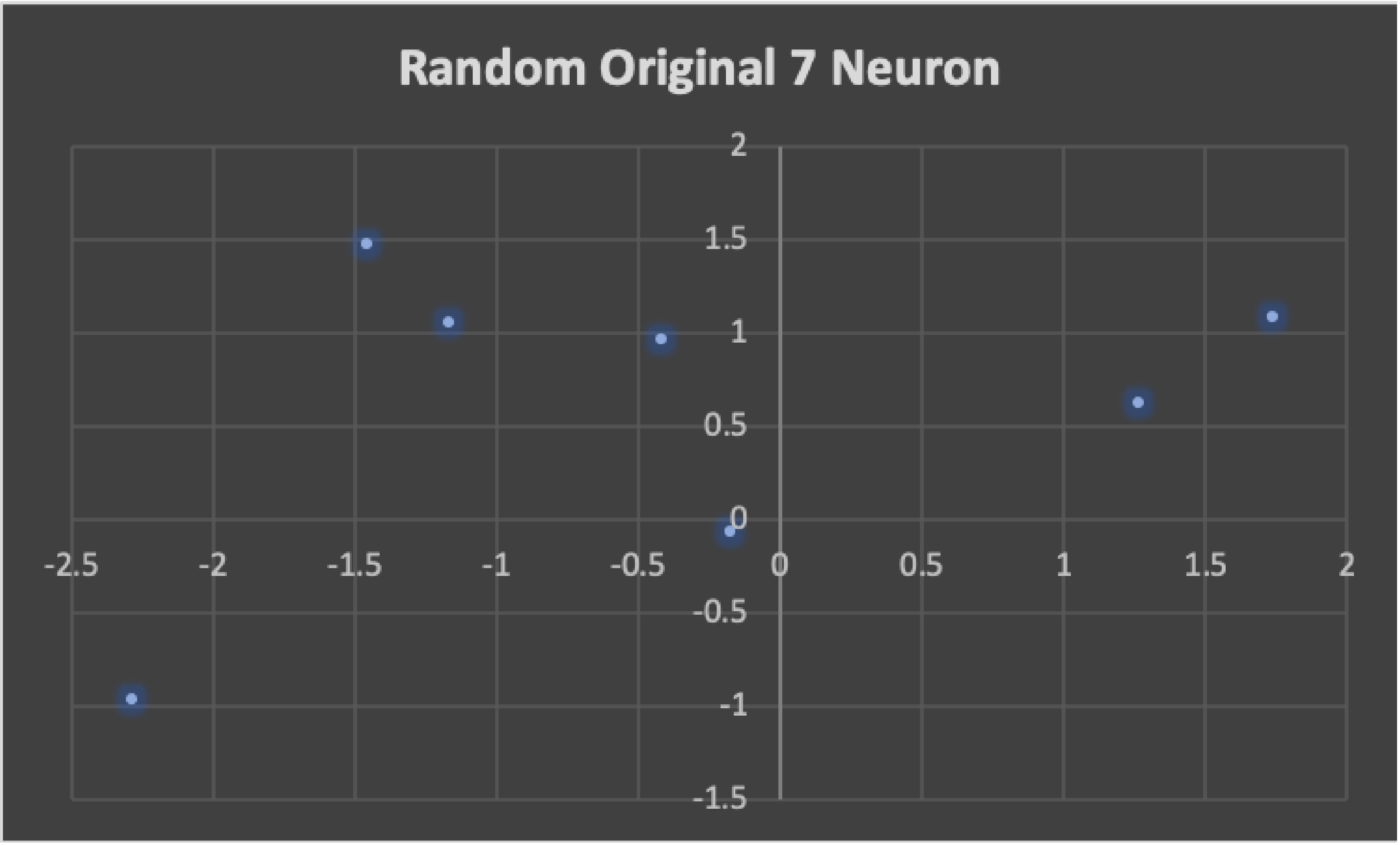
```
-0.18069304728413738, -0.06288949856104088
-2.287026339313784, -0.9576085857229641
-0.4214825923209142З, 0.9733963127500687
1.7339505982921208, 1.0875103226550271
-1.1713879172991644, 1.0651706737024673
-1.4601650946487488, 1.4808441849464242
1.26084640717894, 0.6287045445850057

// Normalized dataset
For 2 neurons:
-0.9124106995321392, -0.4092758426529381
0.8687667539720089, 0.49522149306440555
For 3 neurons:
-0.9124106995321392, -0.4092758426529381
0.8687667529185821, 0.4952214949124316
0.6903280637942227, -0.723496485366805
For 7 neurons:
-0.8427986647157201, 0.5382289575574687
0.9978472355969397, 0.06558120471251999
0.8666244612539634, 0.49896096355955316
0.9614112419990597, 0.2751152917557028
-0.2832087053056739, -0.9590583033575613
0.6903280637942227, -0.723496485366805
-0.9221850403677364, -0.38674895128746806
Random - For 2 neurons:
0.8687667539720089, 0.49522149306440555
-0.9124106995321392, -0.4092758426529381
Random - For 3 neurons:
0.8687667529185821, 0.4952214949124316
0.4197936141629962, -0.90761959074712
-0.956847269377298, -0.2905912990528254
Random - For 7 neurons:
```

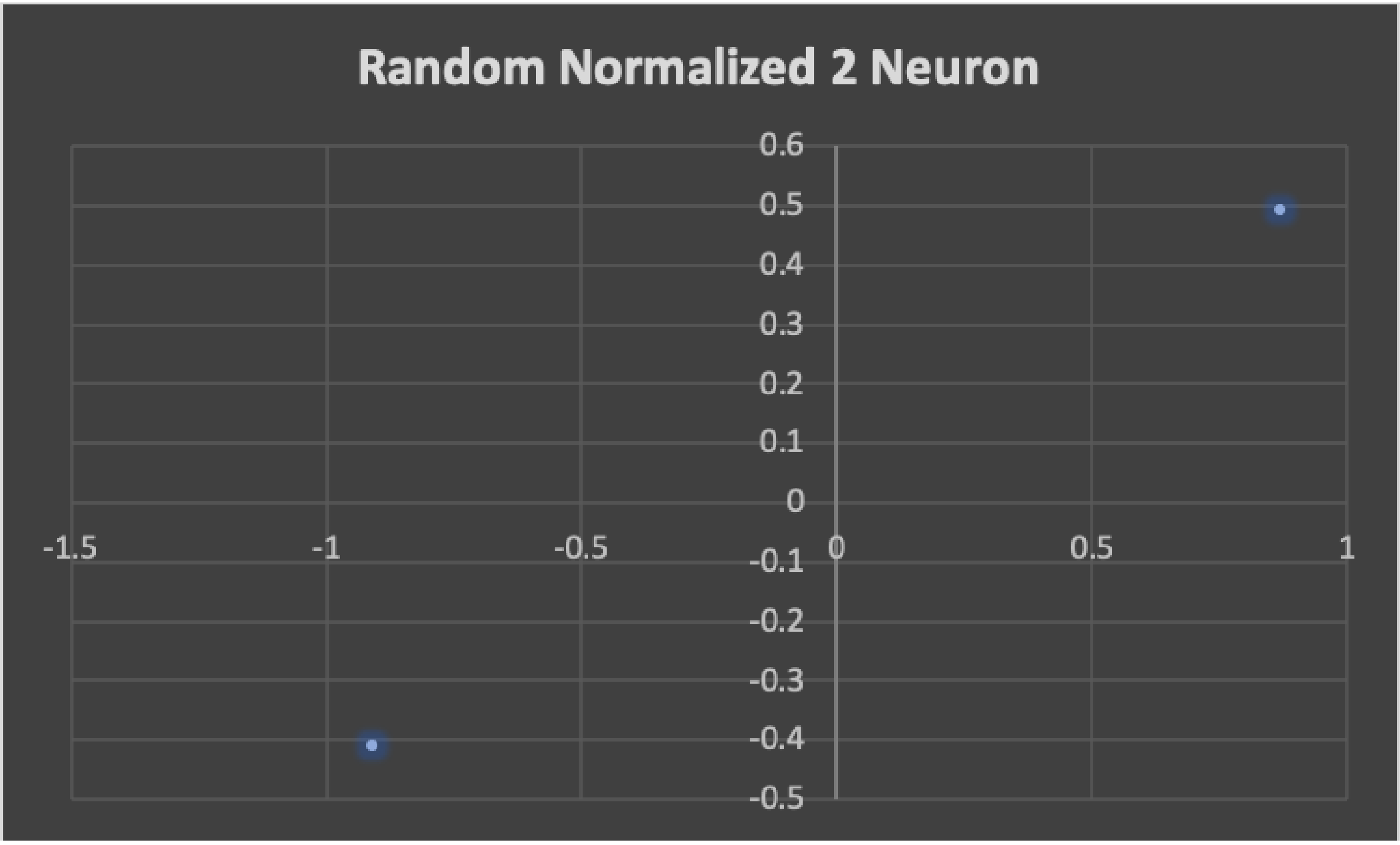
```
-0.6667873872097506, 0.7452479991640327

-0.9221850403677364, -0.38674895128746806

-0.8427986647157188, 0.5382289575574706

-0.28320870530575826, -0.9590583033575364

0.8666275380033286, 0.49895561964396107

0.6903280637942227, -0.723496485366805

0.9690954026403715, 0.24668623914052504
```

Random Weighting

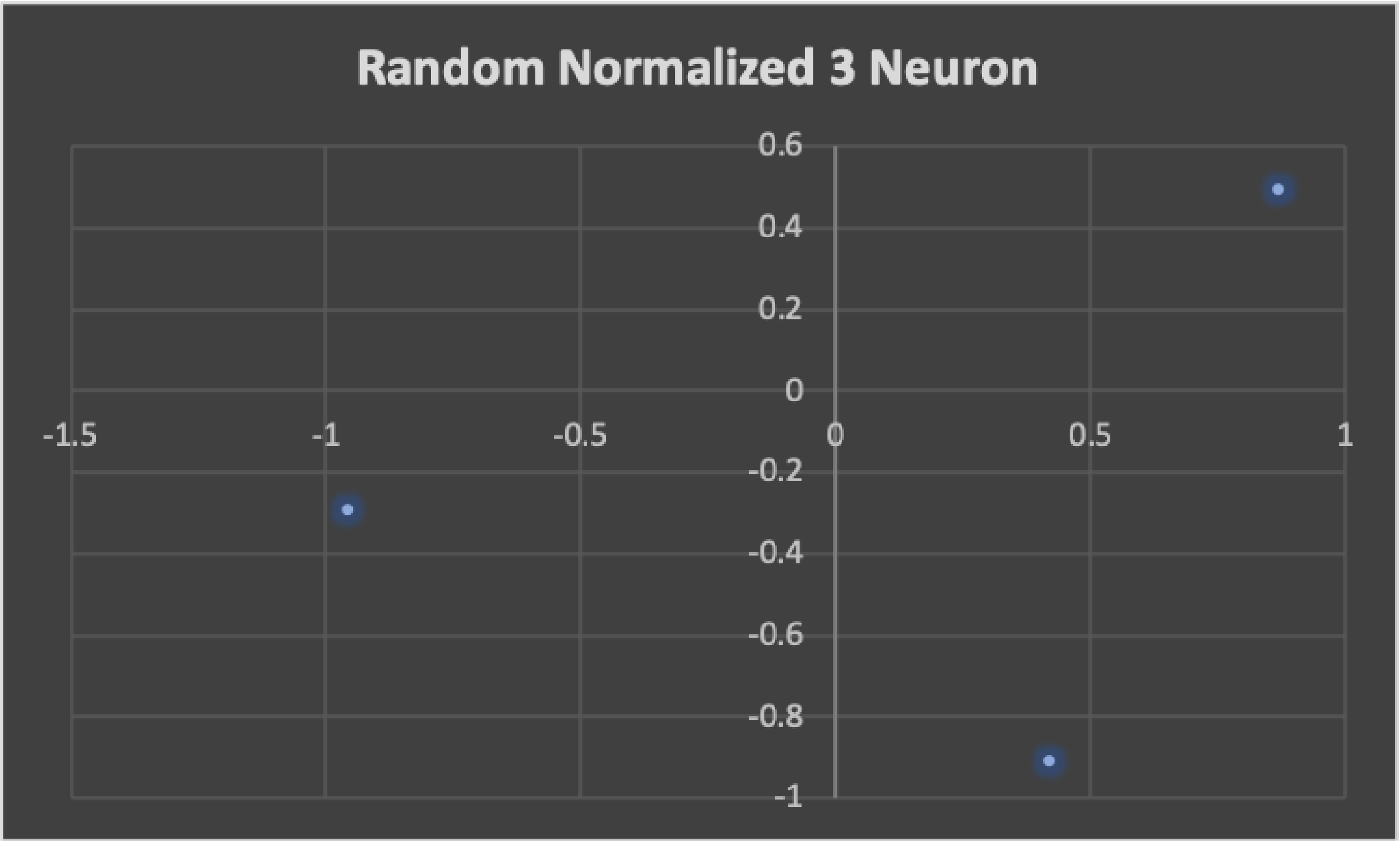



Random Original 3 Neuron
1
0.5
0
-0.5
-1
-1.5
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

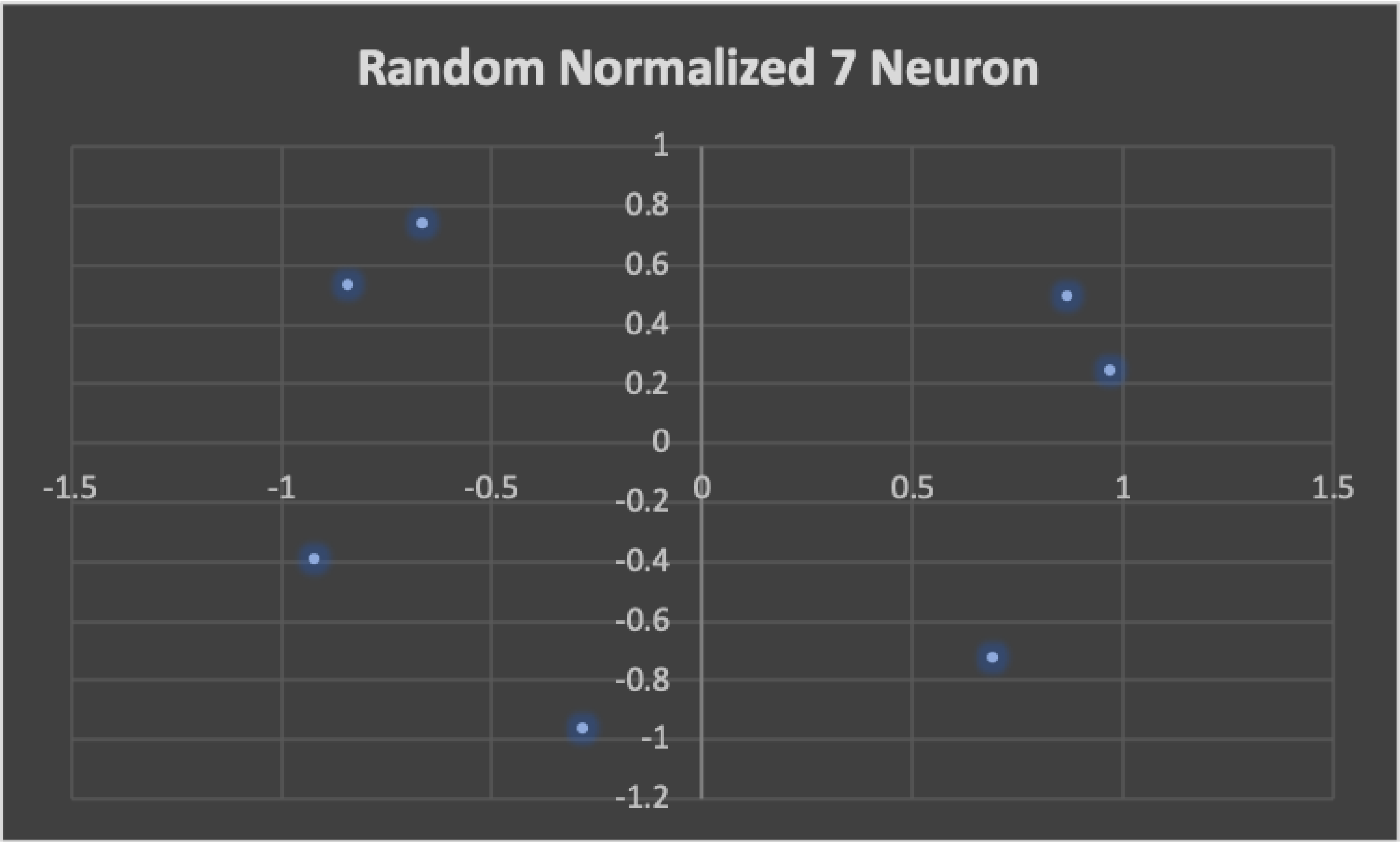

Random Original 7 Neuron
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2



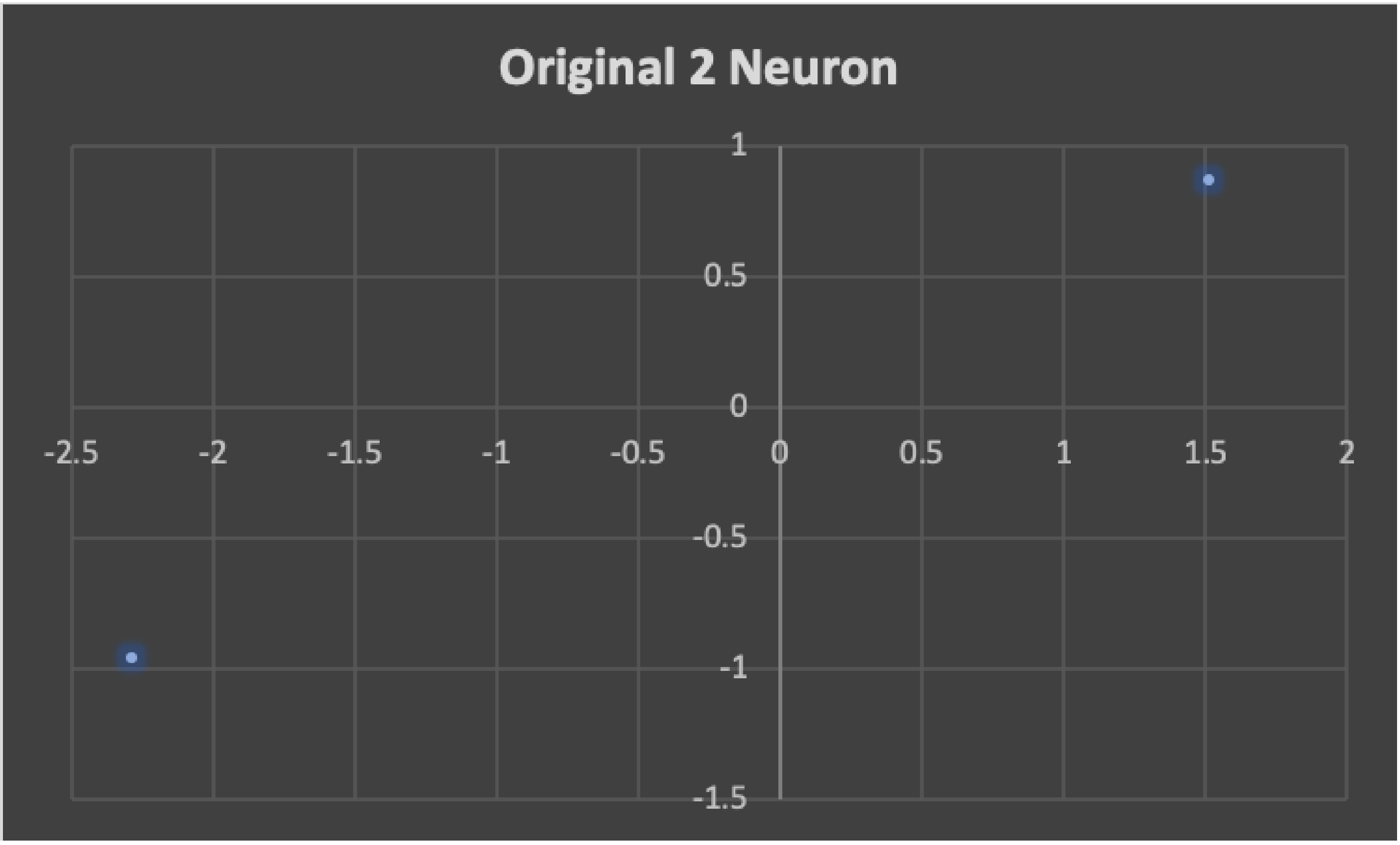
Random Normalized 2 Neuron

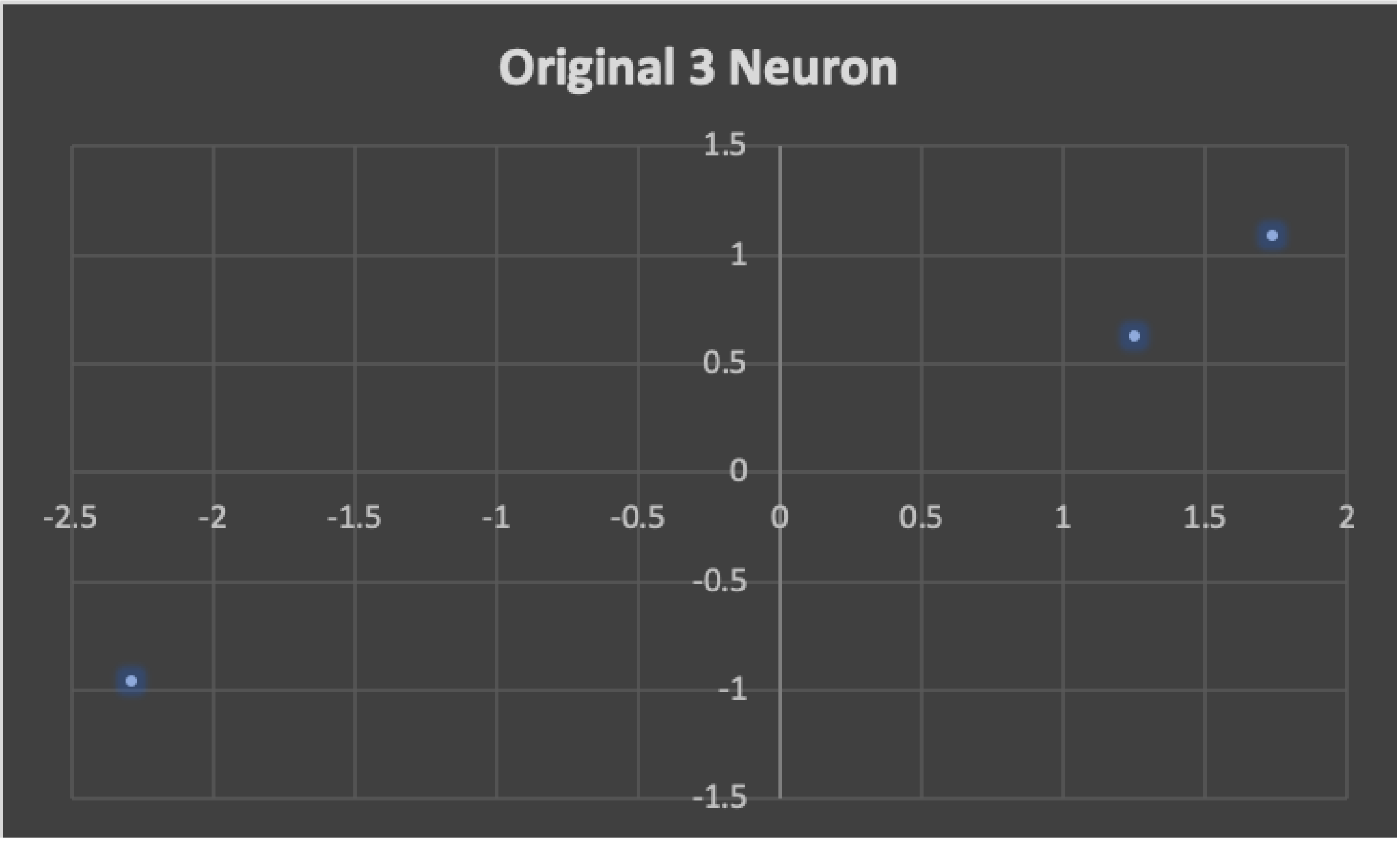


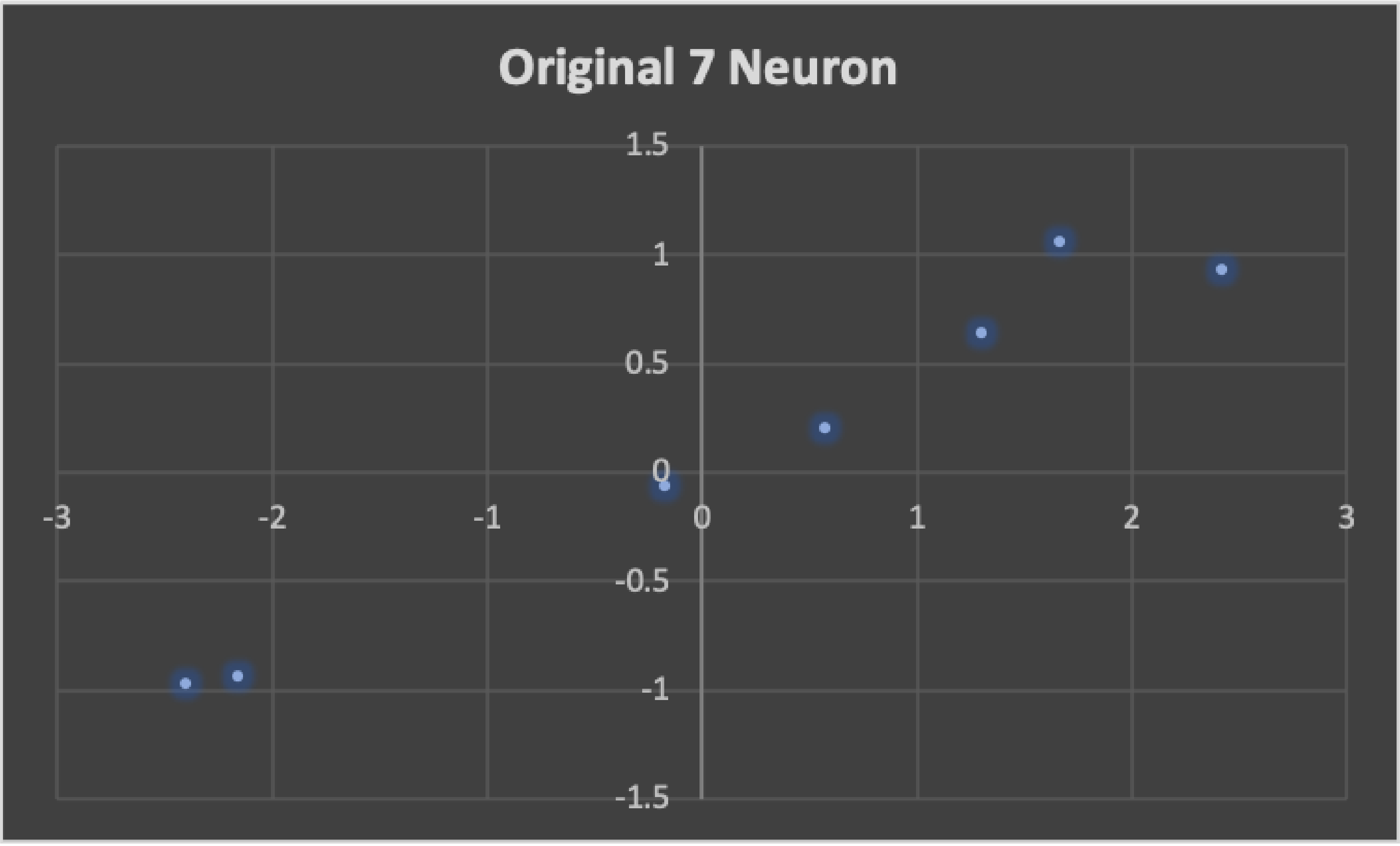
Random Normalized 3 Neuron

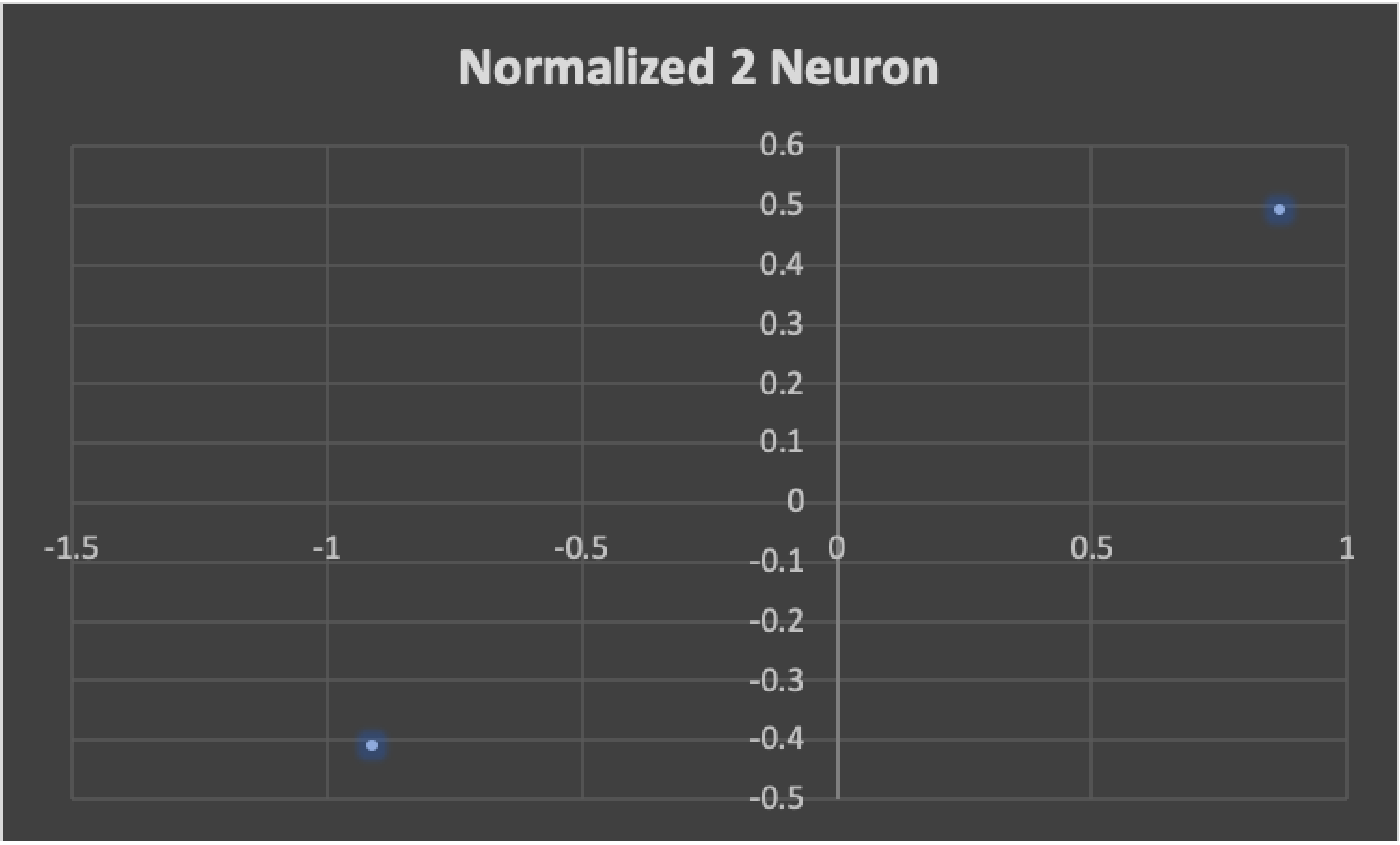


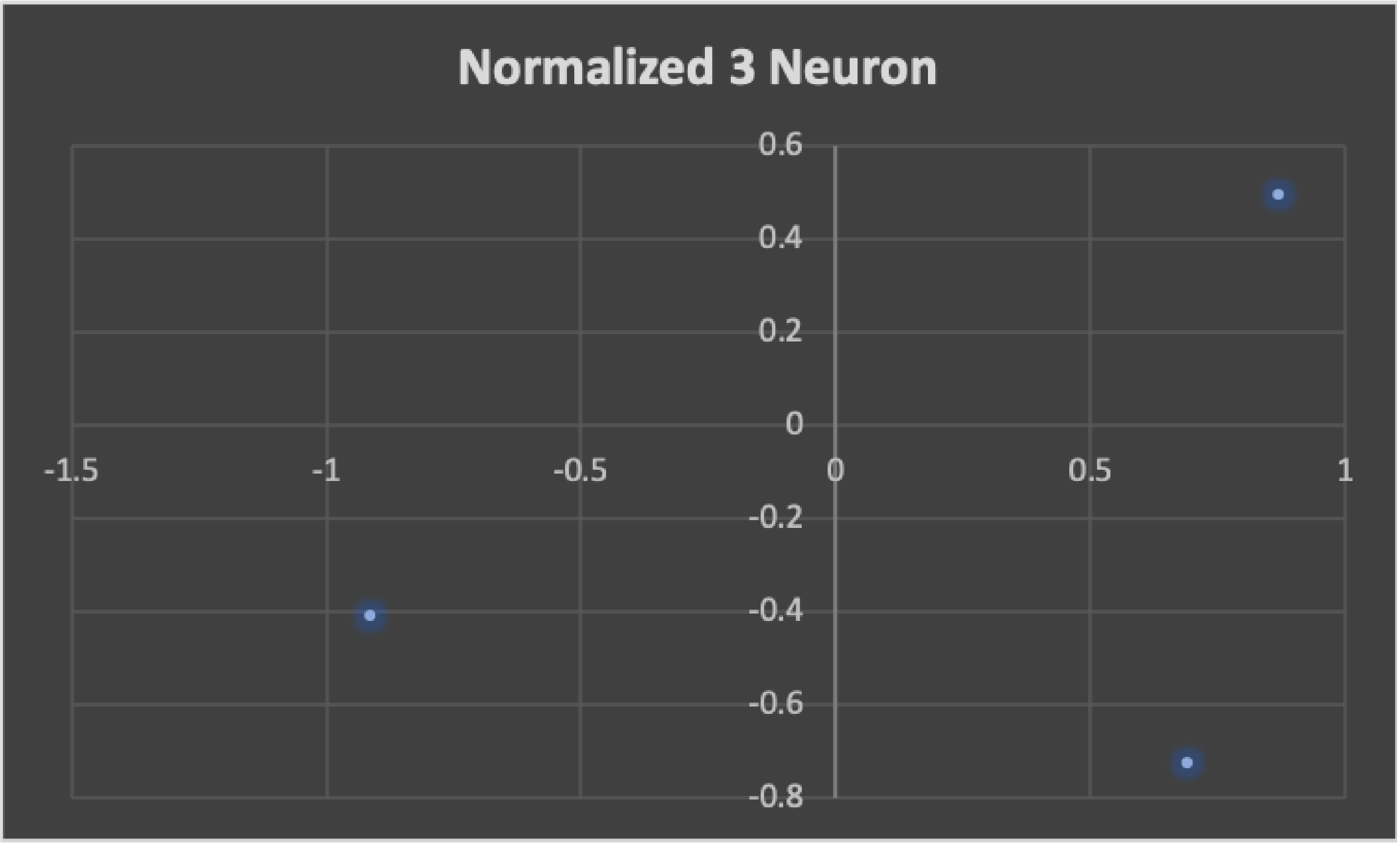
Manual Weighting

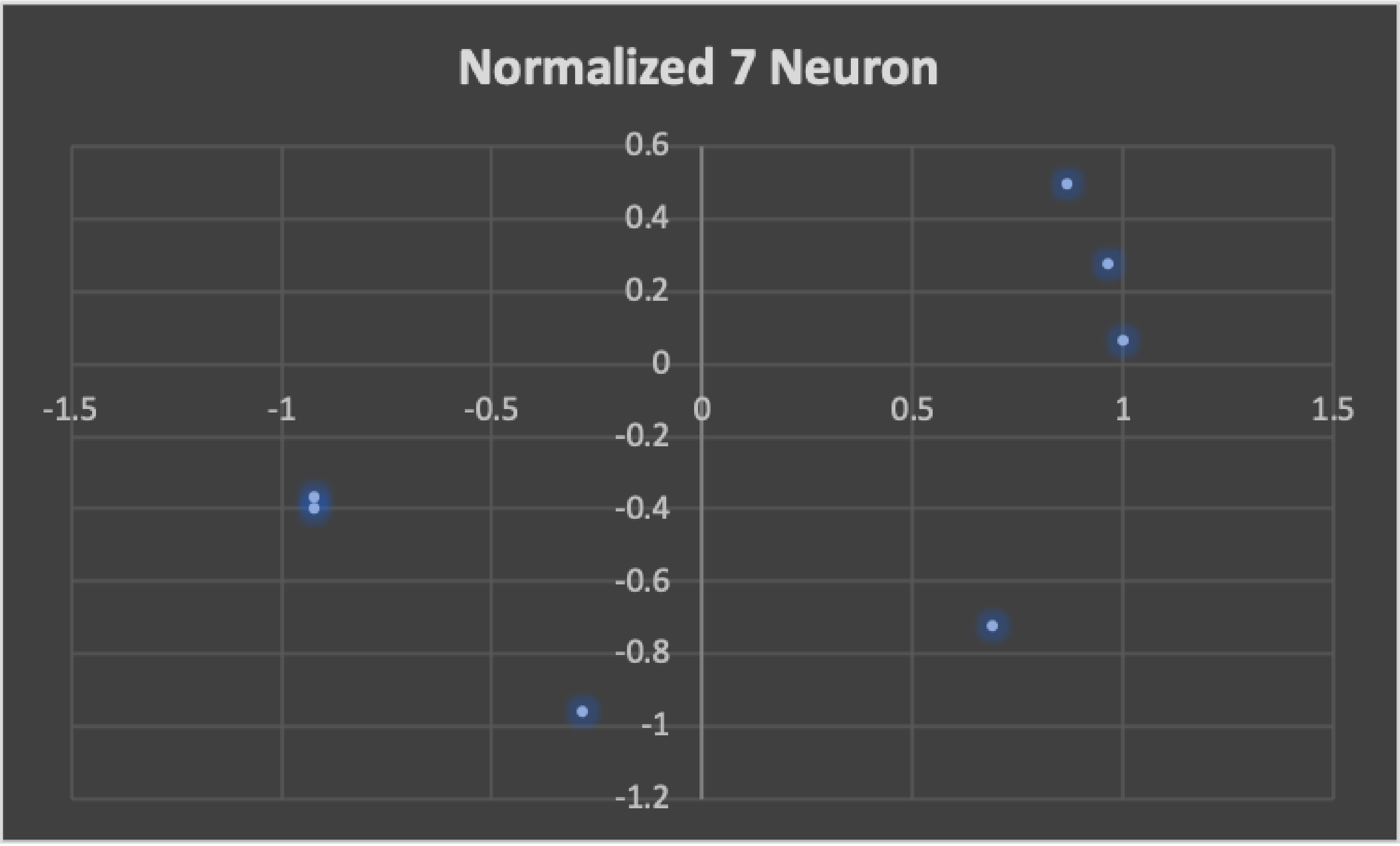



Original 3 Neuron
1.5
1
0.5
0
-0.5
-1
-1.5
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


Original 7 Neuron
1.5
1
0.5
0
-0.5
-1
-1.5
-3
-2
-1
0
1
2
3


Normalized 2 Neuron
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1.5
-1
-0.5
0
0.5
1


Normalized 3 Neuron
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1.5
-1
-0.5
0
0.5
1



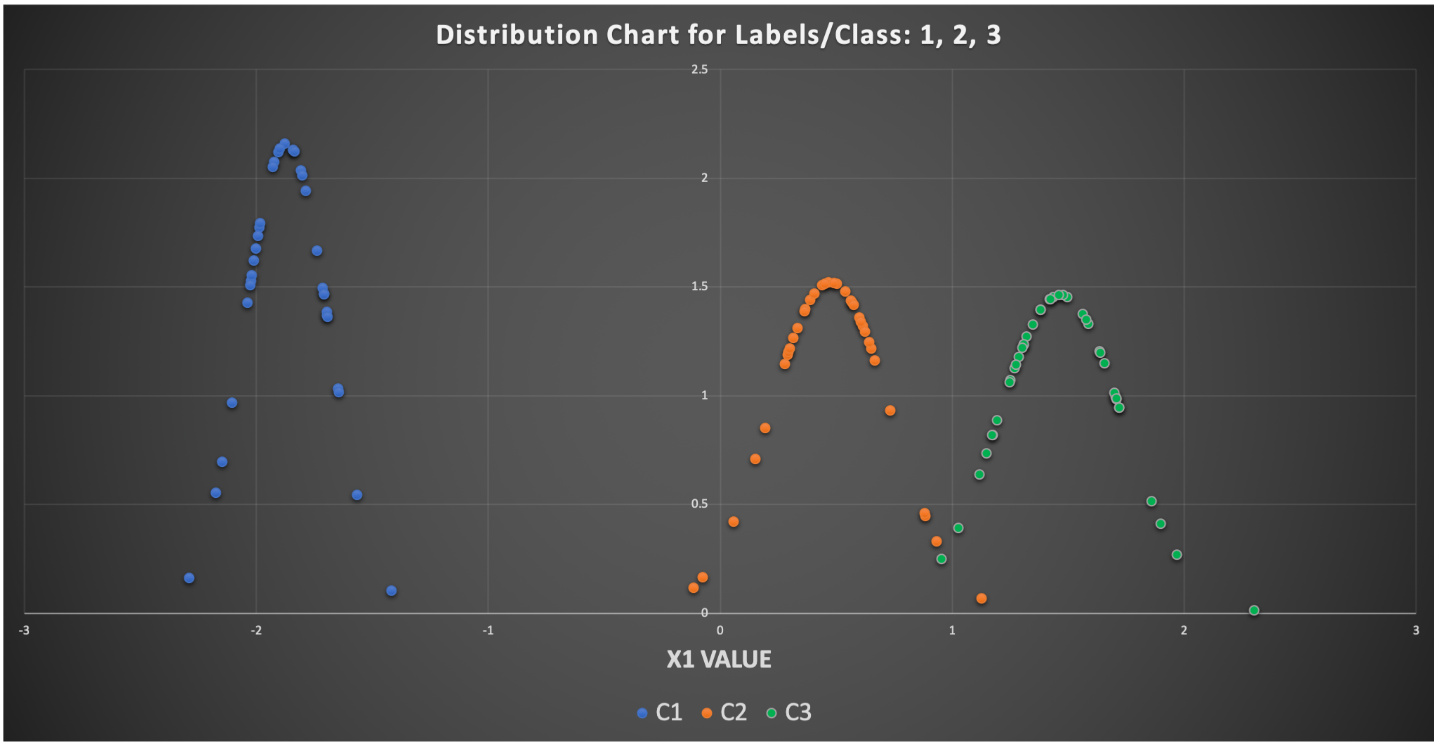
Discussion

From the comparisons above, for both the normalized and original, it is clear that when dealing with neurons, the importance of initial weights are paramount. When weights are manually assigned as in the set up section, the final position of the neurons comes out to be very close to that of their starting weights observed from the initial data exploration. In addition, defined weights also reduce the need for large iterations, for any iteration over 2 will yield desired results.

2.2. Bayesian classifier

Data Exploration

First step to create an effective bayesian classifier was to further explore the training dataset to get a better grasp of what the data looks like. I first started by creating a plot for each class to compare and contrast them visually. Below is the distribution graph for each class mapped in one graph using excel.



To get better detail for each class, I ran a code that shows us some basic stats for each class:

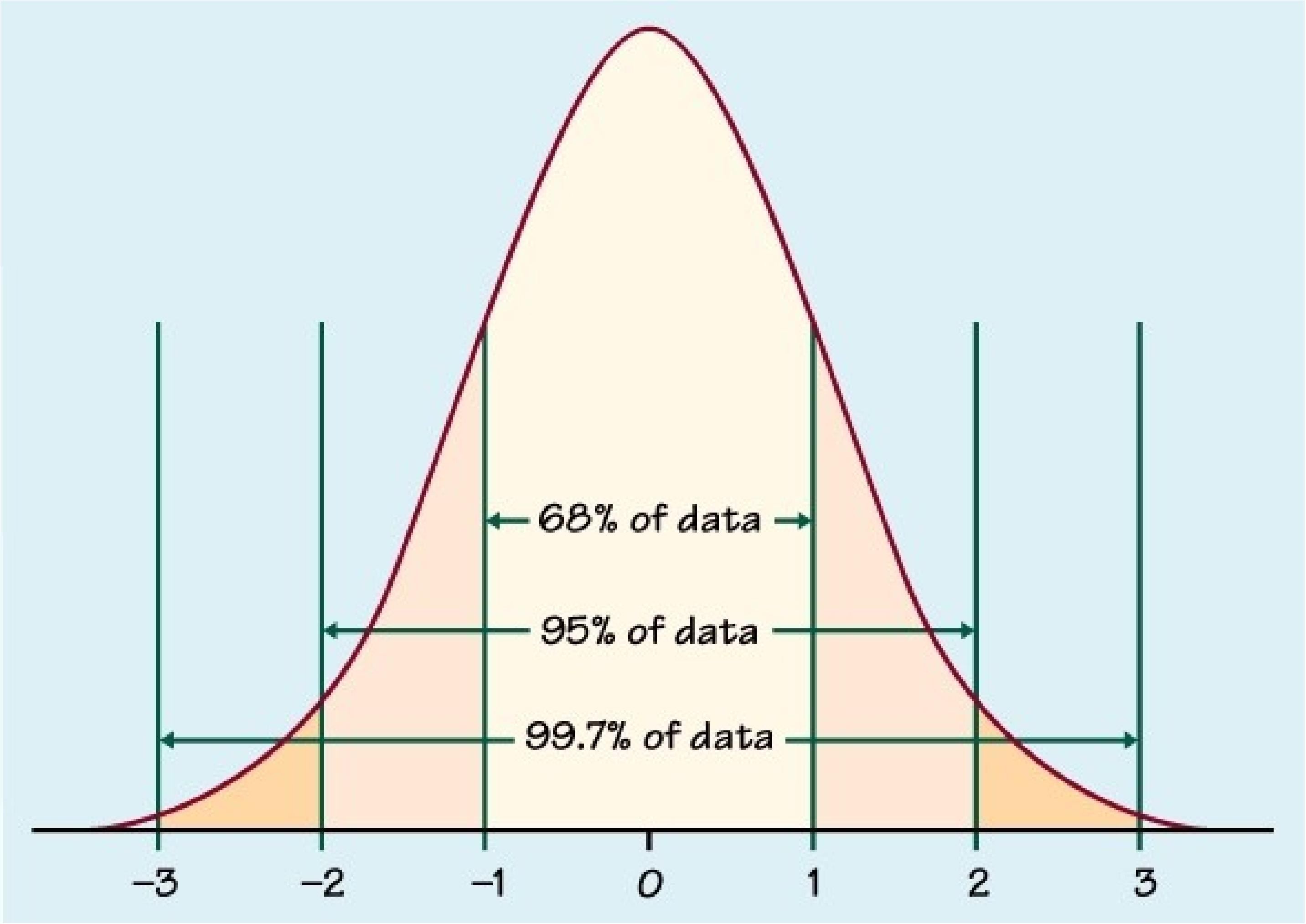
```
Train report -----------------------------------------

Total count for training dataset is 112

Includes 35 of label 1

Includes 36 of label 2

Includes 41 of label 3

Class statistics: Class label: 1, mean: -1.8726114285714288, range: -2.2917 ~
-1.4179, stdv: 0.1845953919428397

Class label: 2, mean: 0.47275088888888889, range: -0.11823 ~ 1.1244, stdv:
0.2616939989055047

Class label: 3, mean: 1.4641917073170734, range: 0.95286 ~ 2.2981, stdv:
0.2721431533613795

Probability of a randomly selected entry being in:

Label 1: 0.3125

Label 2: 0.32142857142857145

Label 3: 0.36607142857142855
```

From above, we can make few observations:

1. Each class are pretty distinctively separated from each other. Especially the first class.
2. Each class's entry distribution are close to normal distribution.

3. Every class has similar probability to be randomly selected at around 30~36%

Thanks to the semi-normal distribution for all classes and clear separation between classes, not counting slight overlap between second and third class, we should get pretty high accuracy. Also, since the data is pretty much normally distributed, we can assume that 3 standard deviation should cover 99.7% of each class.



Testing the classifier

Running a trained Bayesian classifier on test data set yielded the following result:

```
Test report -------------------------------------------

Total count for test dataset is 38

Number of correct classification is 37

Correct percentage is 0.9736842105263158
```

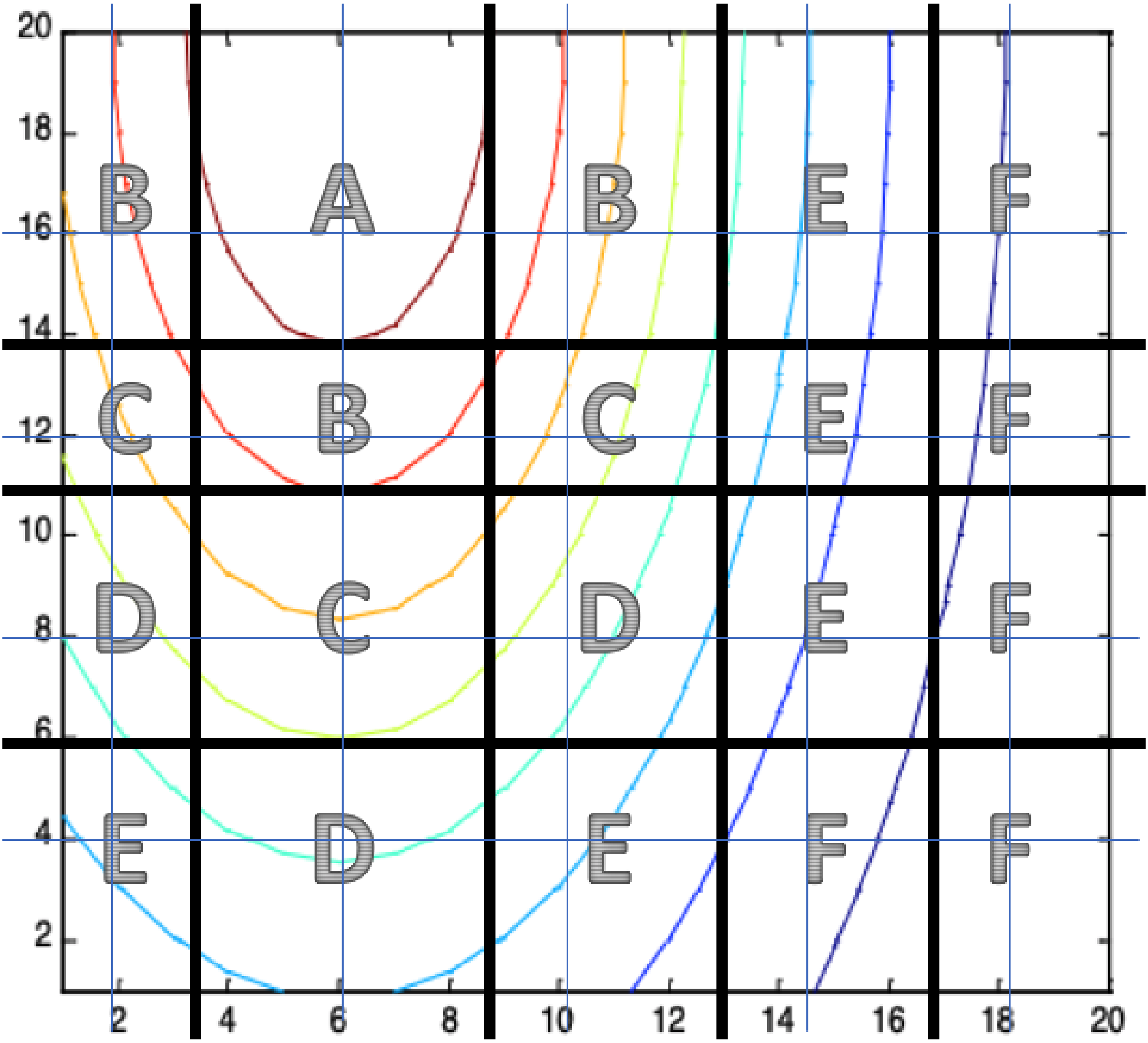
As can be seen, accuracy of the algorithm is quite high at around 97% as expected. It only misses 1 classification due to overlap between classes 2 and 3.

2.3. Designing controller

Creating sections and memberships

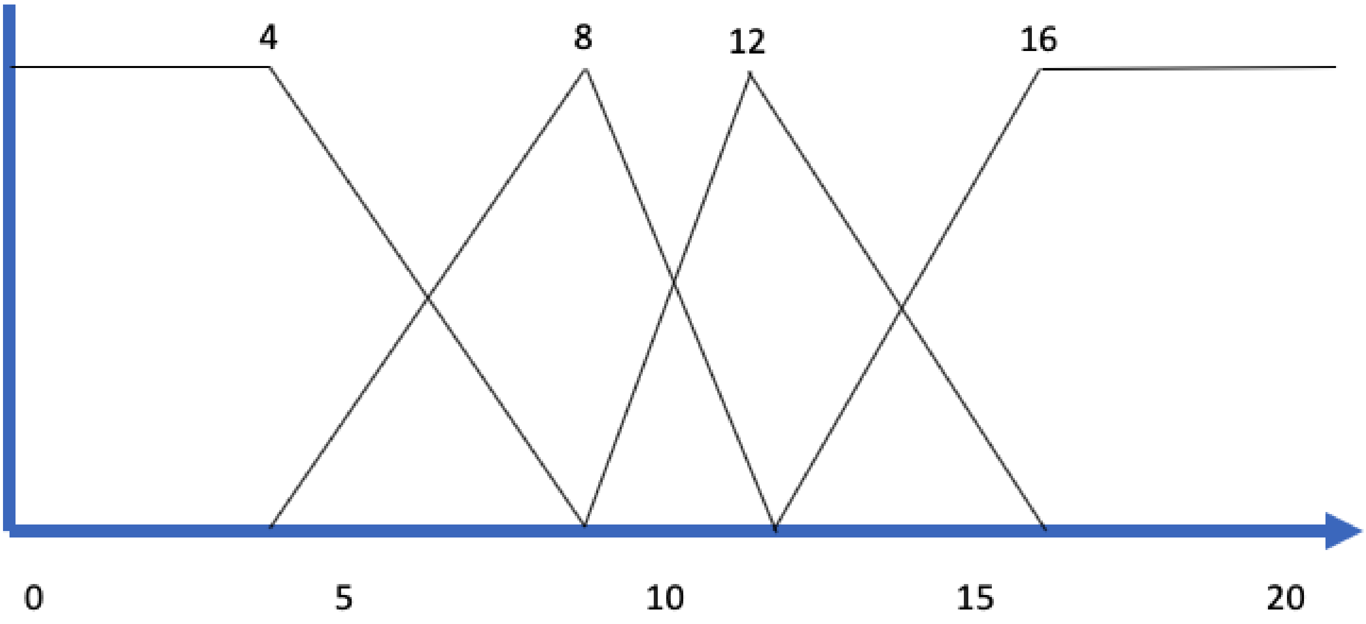
Fuzzy controller was designed as follows:

Contour map was divided into sections depending on their height.

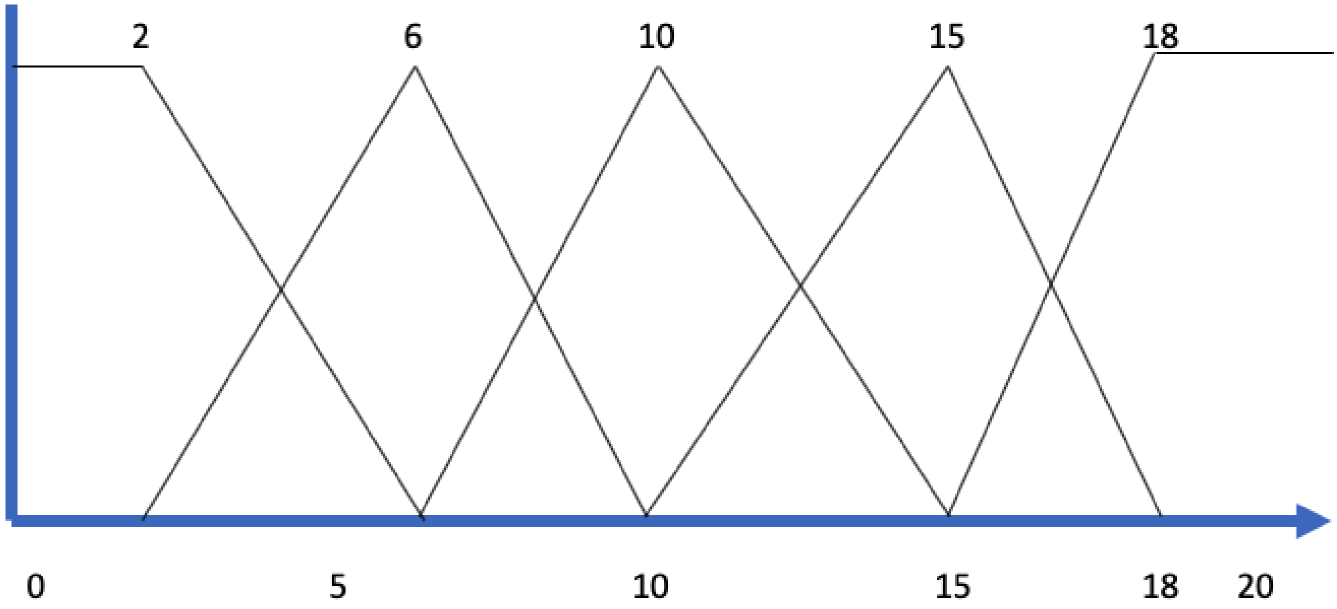


Triangular membership functions were created as follows:

Direction Y

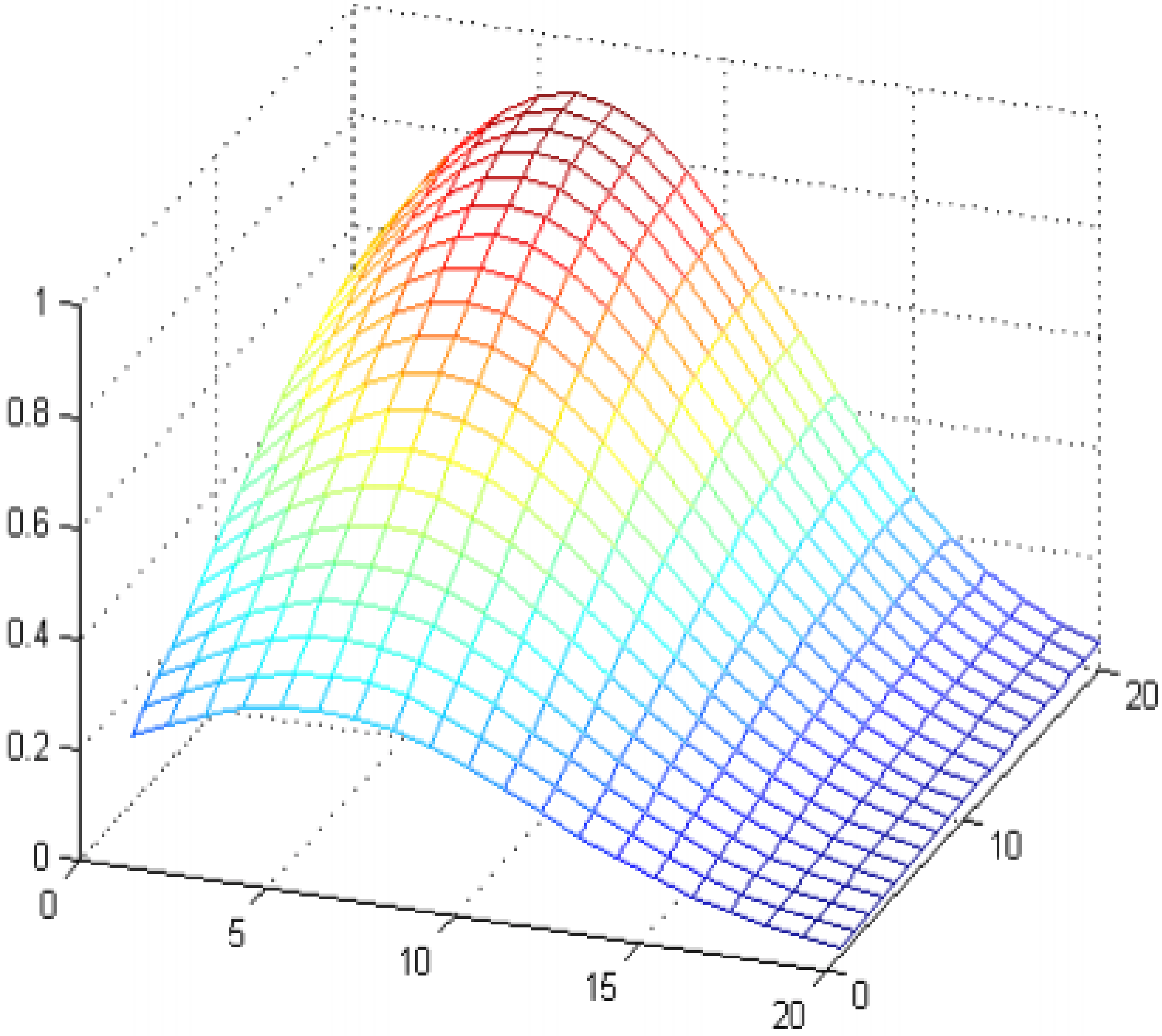


Direction X



Contour Map

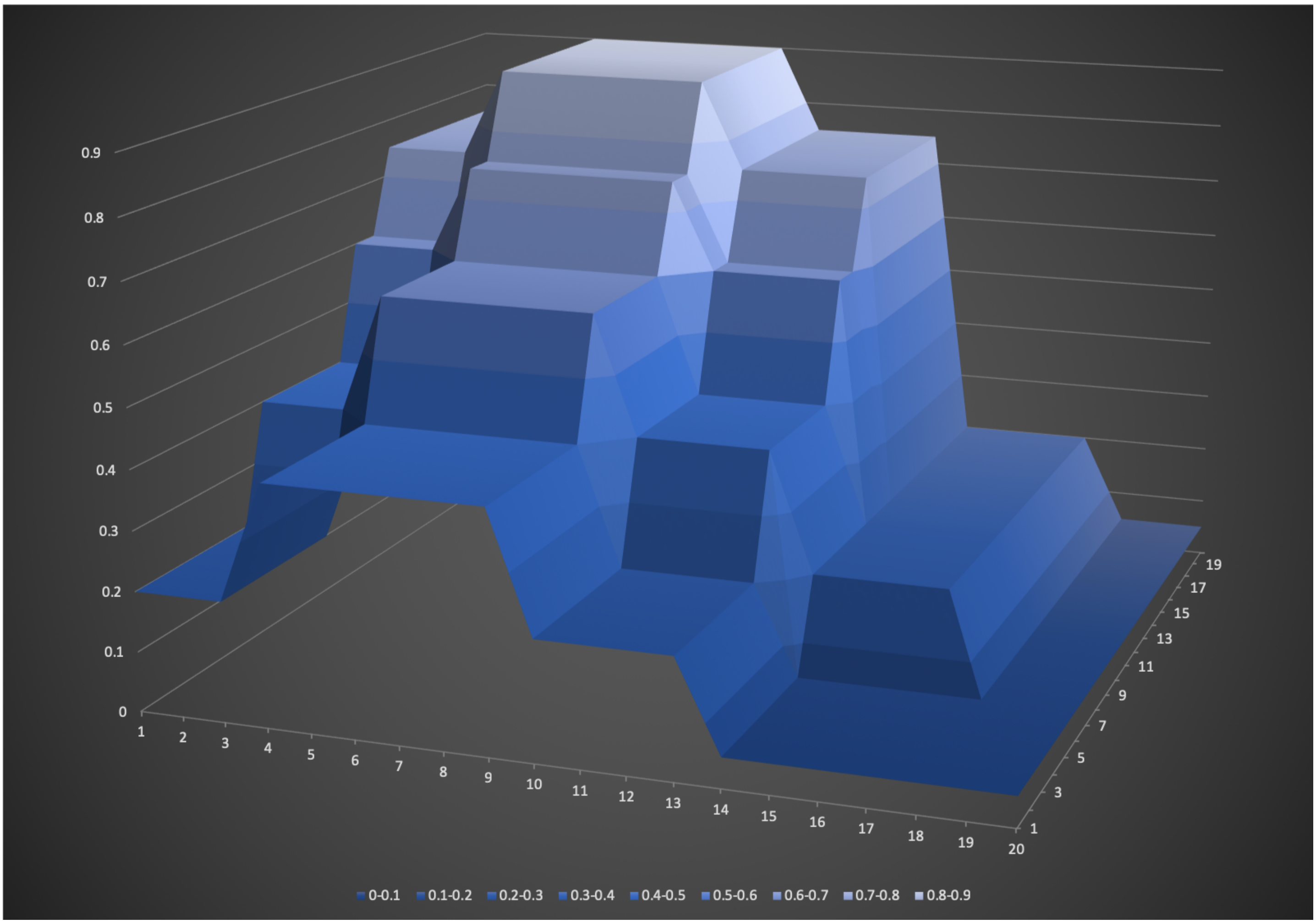
Below is the contour map of the control surface.



With this provided contour map, I have assigned following singleton values to each of the sections.

- A = 0.9
- B = 0.75
- C = 0.6
- D = 0.4
- E = 0.2
- F = 0.05

The following controller graph used above information.

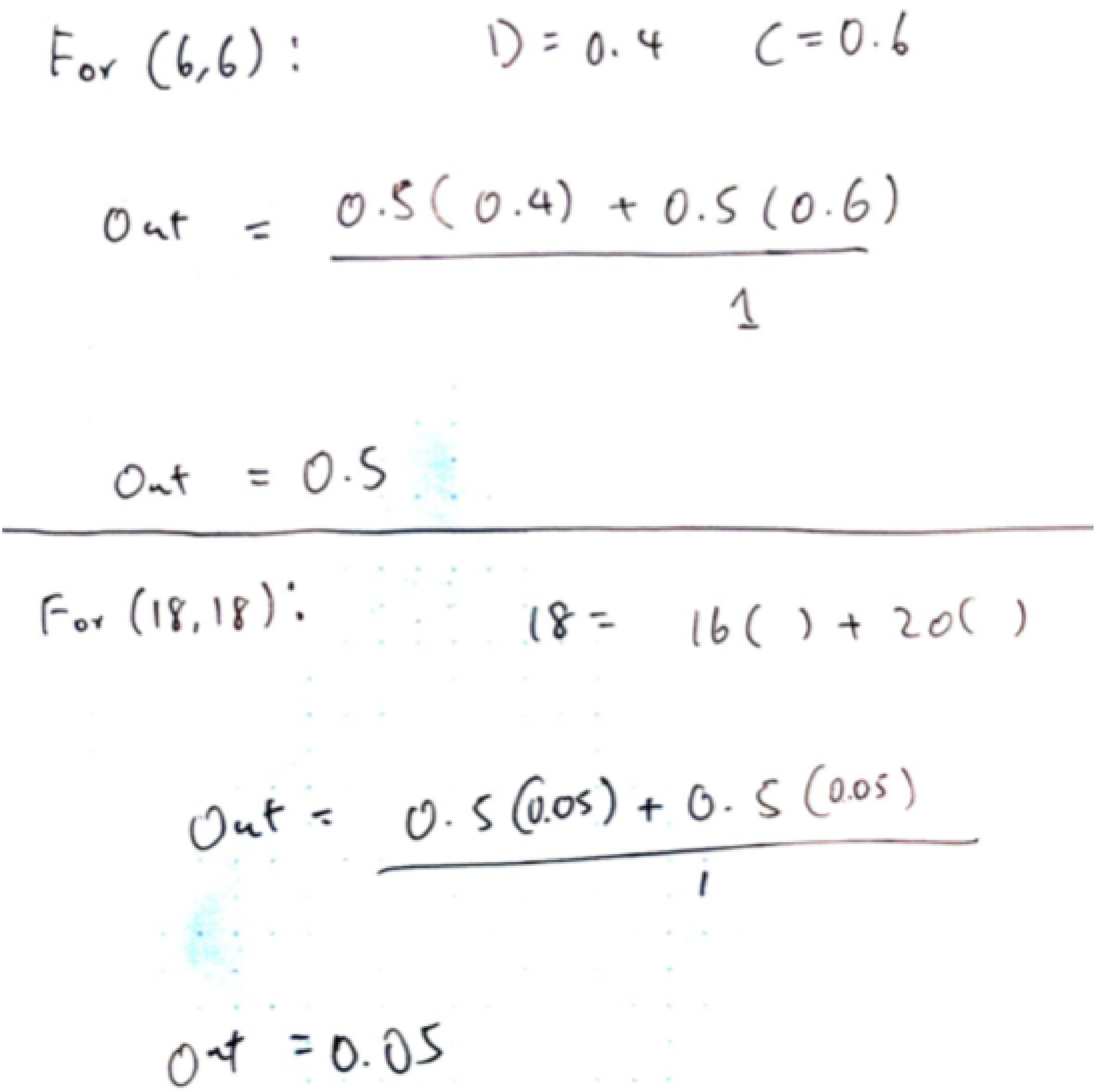


When we compare these two graphs, we can notice that the controller is much more blocky due to it's categorization into sections. The best way to get a smoother controller that more closely represents the control surface would be to have more categories and have more grid sections.

Testing

For (6, 6):
For x, since 6 is exactly on 6, it will be 1 * 0
For y, since 6 is exactly between 4 and 8, we give 0.5 for each. Such that: max D is 0.5 as well as max C. Expected is 0.49 and my output was 0.5. Making error rate of 2%.

For (18, 18):
For x, 18 is again exactly on18. Making 1 * 0
For y, since 18 is between 16 and 20. We can again use 0.5 for each. However, since both falls into section F, output will be 0.05. Expected is 0.098 and my output was 0.05. Making error rate of 49%.



For (6,6): $D = 0.4$ $C = 0.6$

$$Out = \frac{0.5(0.4) + 0.5(0.6)}{1}$$

$$Out = 0.5$$

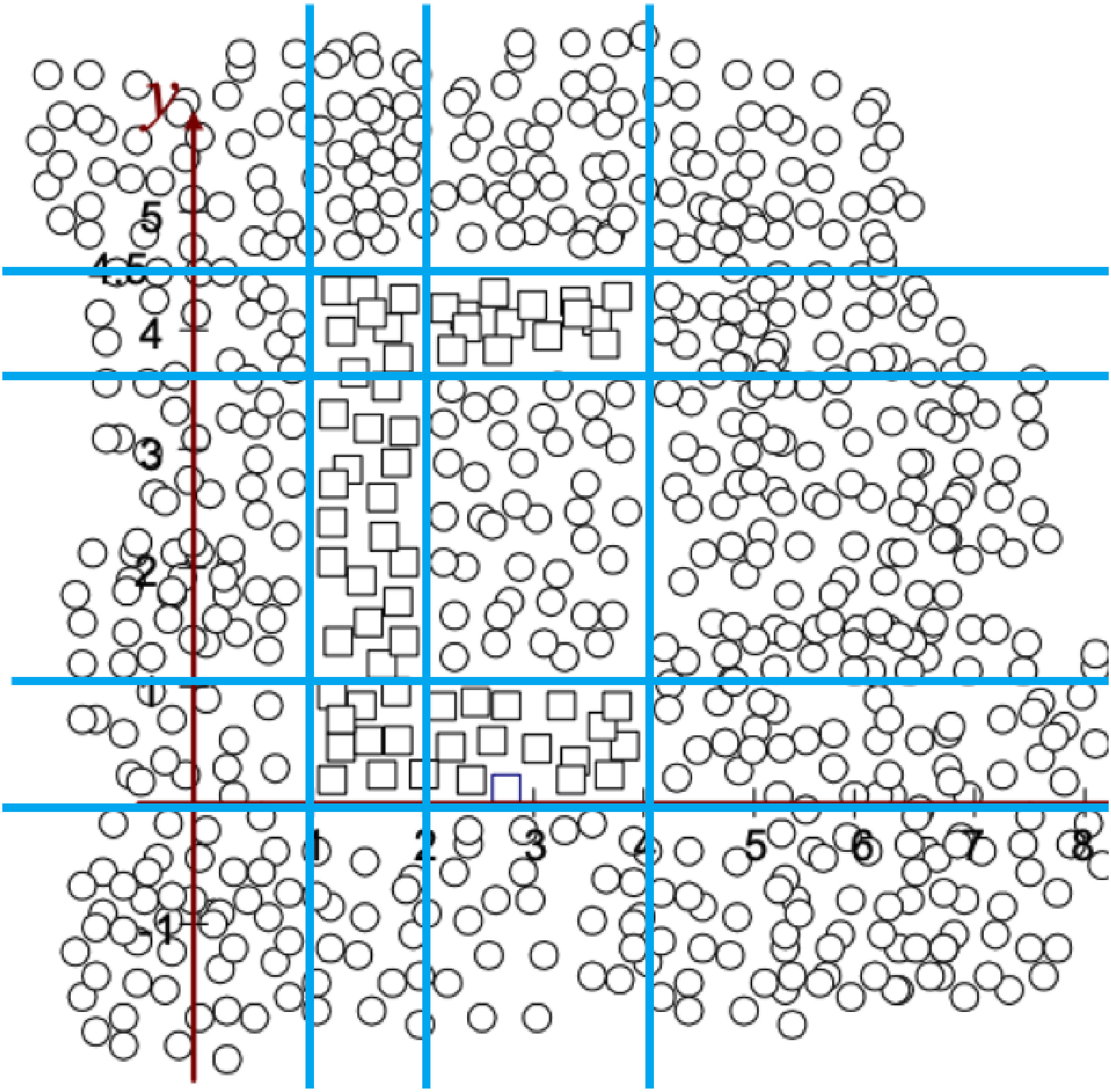
For (18,18): $18 = 16(\ \ ) + 20(\ \ )$

$$Out = \frac{0.5(0.05) + 0.5(0.05)}{1}$$

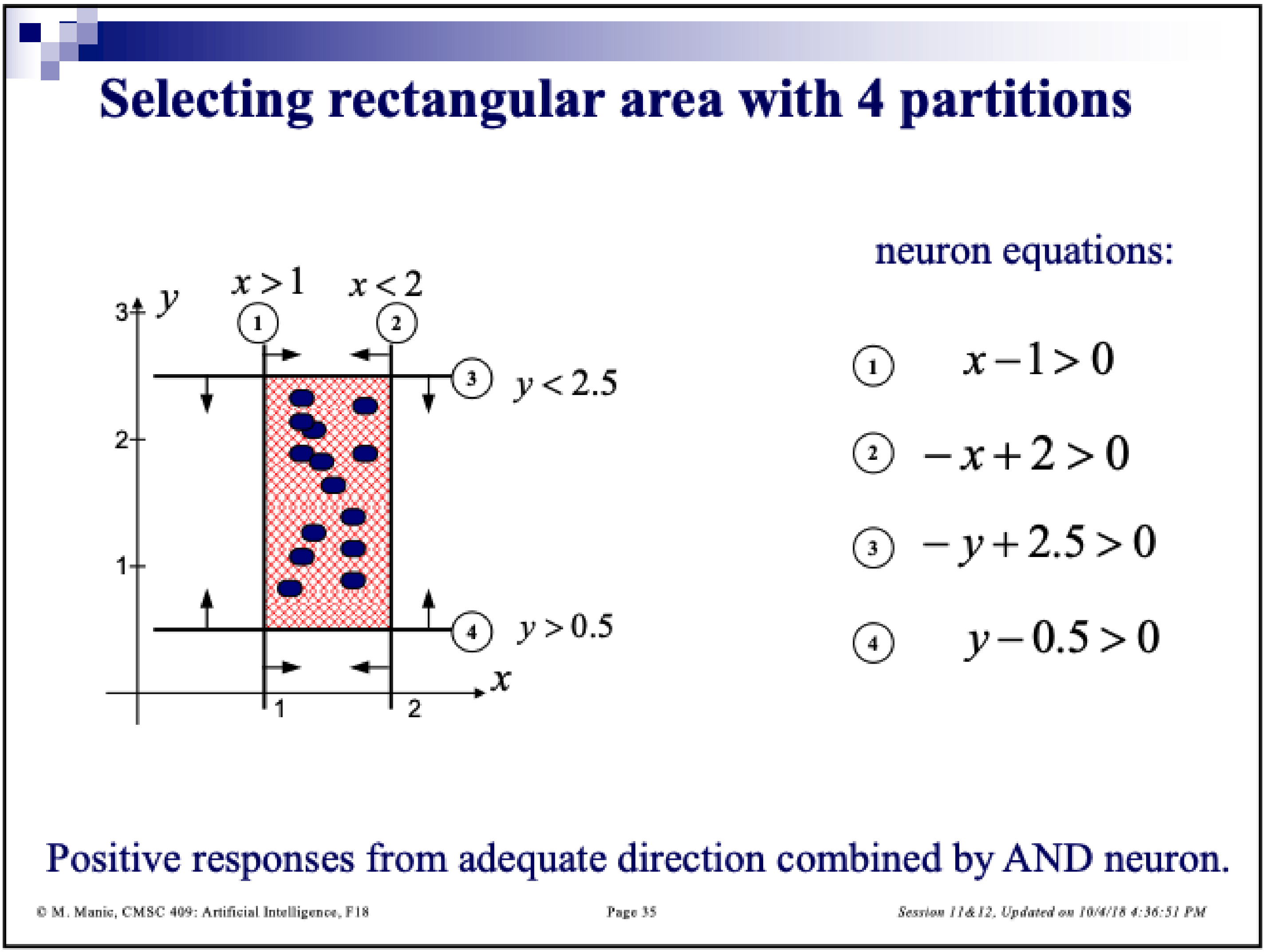
$$Out = 0.05$$

Creating neural network via design

The main idea for this problem is to design a network that can create a separation line depending on the types of data (circle default square points). When we perform this and create a separation line, it should loo something similar to below:



One way we can extract squares is by comparing each lines against each other to see if the "C" shape is formed. I am not sure how the threshold should be set up or how the separation lines would go through the square bits without getting confused. However, one idea to think about is to set slope limit where the first 4 neuron will chose best separation line with slope of 0 while the 3 vertical lines would need to have a slope of 1. And to set their threshold to match the y and x coordinate value. To better explain my idea, below is the similar example we did during class:



Selecting rectangular area with 4 partitions

neuron equations:

① $x - 1 > 0$

② $-x + 2 > 0$

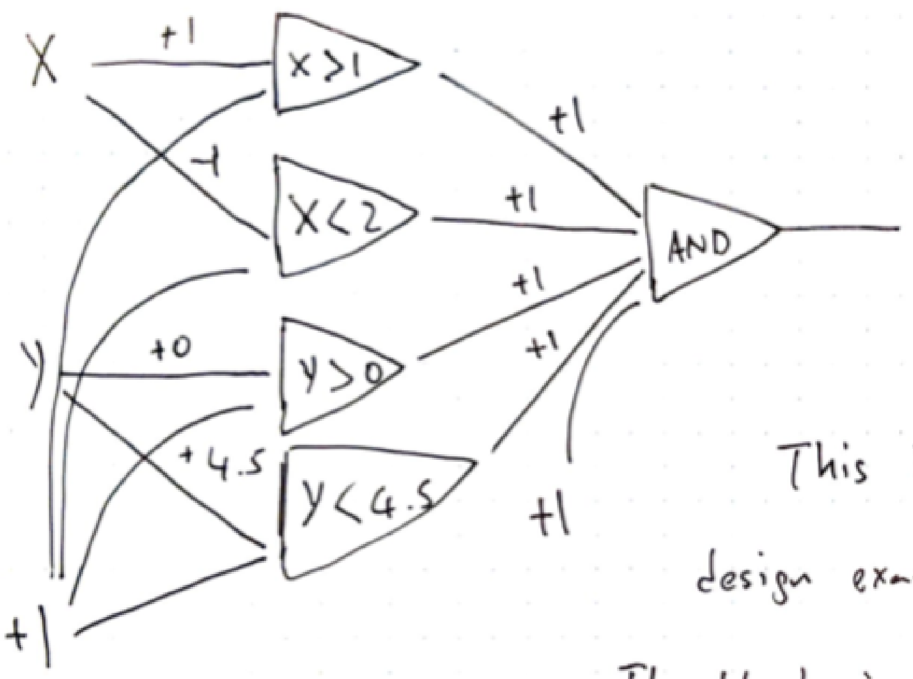
③ $-y + 2.5 > 0$

④ $y - 0.5 > 0$

Positive responses from adequate direction combined by AND neuron.

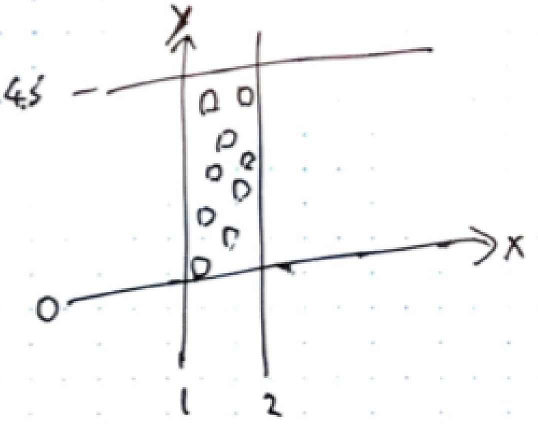
© M. Manic, CMSC 409: Artificial Intelligence, F18 Page 35 *Session 11&12, Updated on 10/4/18 4:36:51 PM*

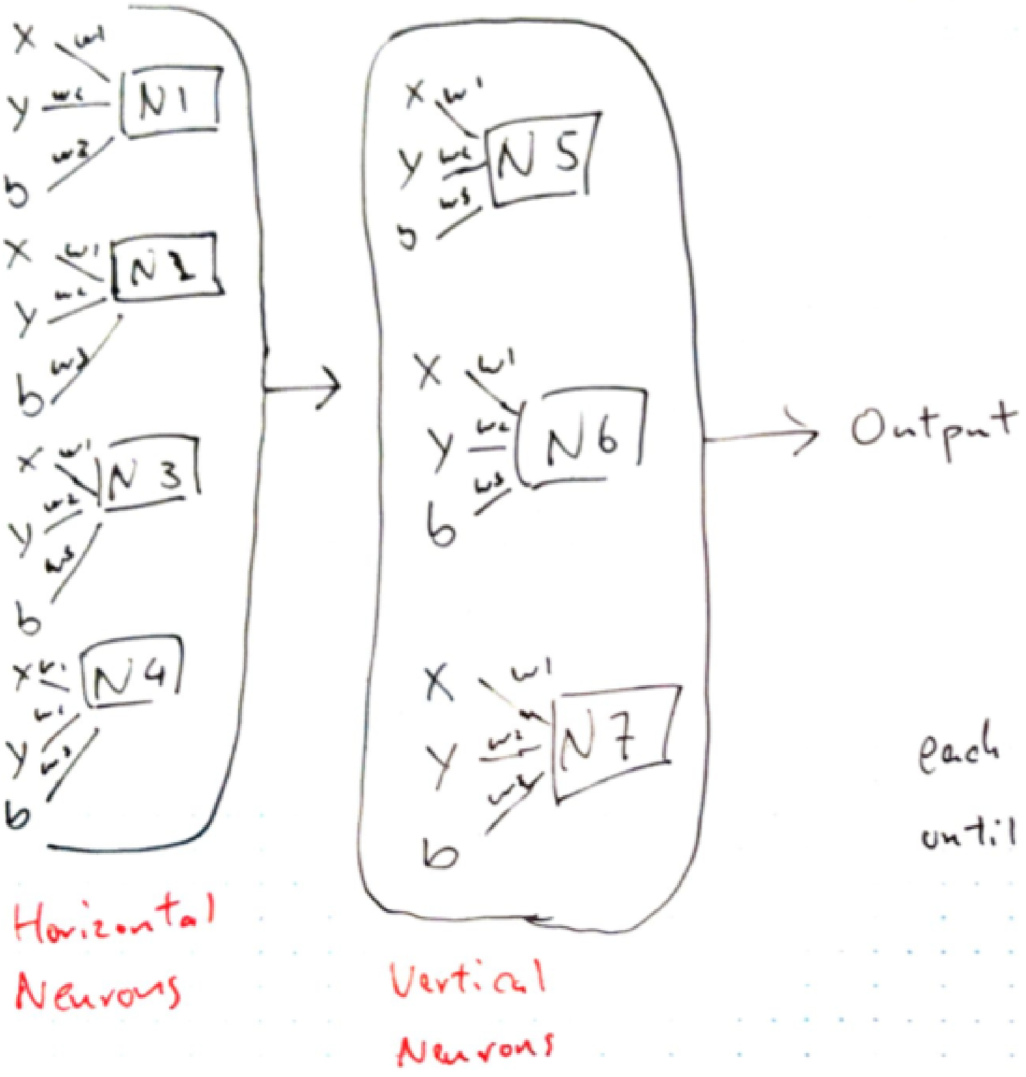
I will be using a sigmoid activation function with neuron definition of `n = w1*x + w2*y + w0` . Where x and y are the inputs and w's are the weights. The network architecture diagram should be something similar to below:



This is the neuron design example for one block.

The block is below. Where the block is vertical.





each step will loop until best fit is found.