Deep Learning and Neural Networks – Multi-Perceptron and Linear Separability

Qing Zhou

1 Introduction

The motivation of this assignment is to develop an automated image classifying algorithm, by implementing the single perceptron algorithm to linearly separable sets. and meanwhile to understand how to implement multi-layer perceptron logarithm to solve sets which are linearly inseparable. This is done by building a neural network from scratch which can solve the easiest inseparable sets postulated by the XOR problem.

2 Analyse distances between images

The center of each cloud C_d can be understood as the average image taking over all points belong in the set defined by C_d ($\boldsymbol{p} \in C_d$). To get an idea of it schematically we can plot the mean images out

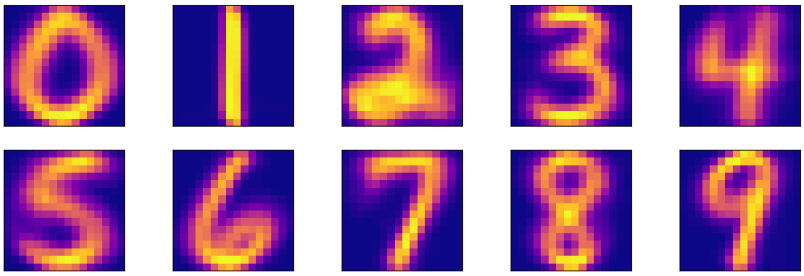

Figure 1: Plotting the function.

And the radius r_d of each cloud is the farthest Euclidean distance from a point in cloud C_d to its center. Since the points and cloud center can be understood as vectors with n features, for a cloud C_d with center $\boldsymbol{c_d}$ and point $\boldsymbol{p}$, the distance between two $n-$dimensional vectors can thus be computed as

$$r_d = \|\boldsymbol{p} - \boldsymbol{c_d}\|_{\max} = \sqrt{\|\boldsymbol{p}\|^2 + \|\boldsymbol{c_d}\|^2 - 2\boldsymbol{p} \cdot \boldsymbol{c_d}}, \text{ for all p} \in \text{C}_\text{d} \tag{1}$$

the radius of each cloud (as well as the total number of points each cloud has) is then

Table 1: The radii and number of points of all the clouds C_d.

C_d	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
r_d	15.89	9.48	14.17	14.74	14.53	14.45	14.03	14.91	13.71	16.14
n_d	319	252	202	131	122	88	151	166	144	132

which has a following distribution

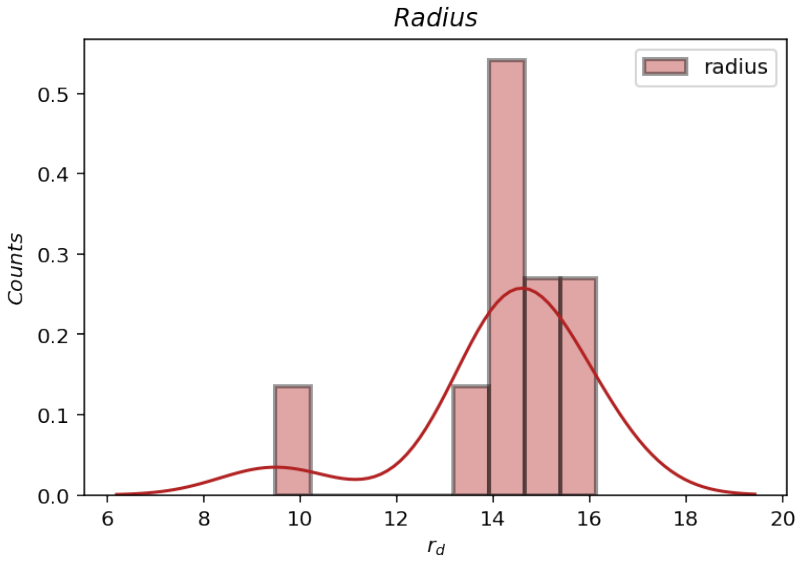

Figure 2: The distribution of the radii.

And likewise, the distance between two centers $\boldsymbol{c_i}$, $\boldsymbol{c_j}$ of clouds C_i and C_j can be computed as

$$d_{c_i,c_j} = \|\boldsymbol{c_i} - \boldsymbol{c_j}\| = \sqrt{\|\boldsymbol{c_i}\|^2 + \|\boldsymbol{c_j}\|^2 - 2\boldsymbol{c_i} \cdot \boldsymbol{c_j}} \tag{2}$$

And naturally there are in total 45 pair-wise distances d_{c_i,c_j} for a community of ten sets C_i (with $i = 0, 1, \cdots, 9$). The pair-wise distances are listed in the plot below, which are color coded by the values of distances

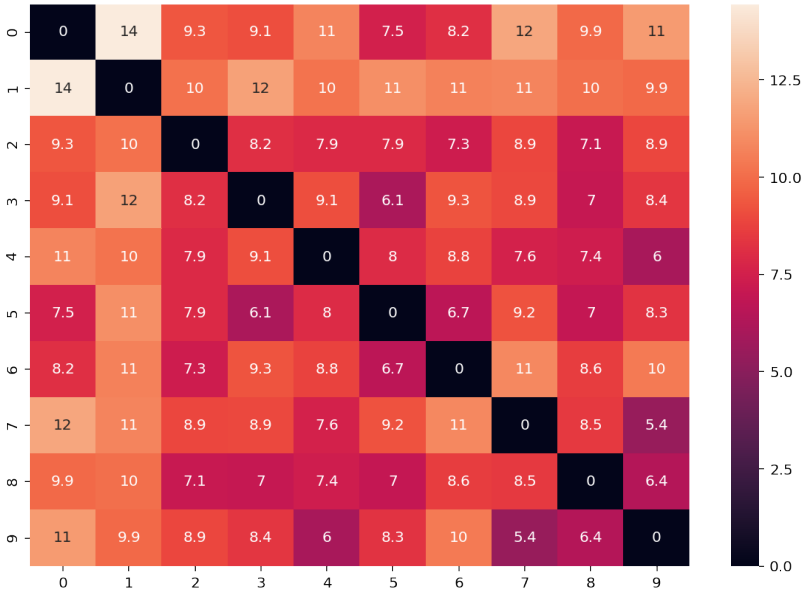

Figure 3: The distribution of the radii, x and y labels indicate the label of clouds and the value in each cube located at i th row j th column gives the distance between cloud C_i and C_j. The matrix is by itself symmetric. The lighter the color, the more separated the two clouds are.

To get an idea of the mean and variance of the distribution of the pair-wise distances, we plot the histogram for it as well

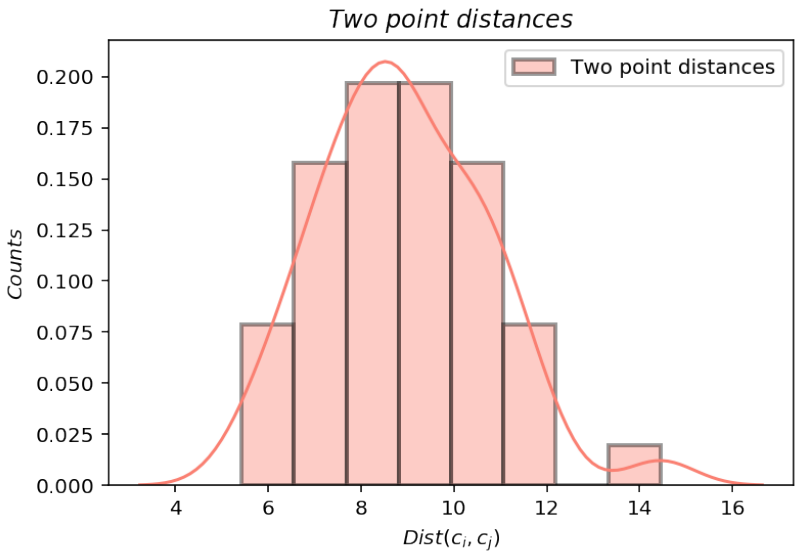

Figure 4: The distribution of the pair-wise distances.

From the two histograms one can infer, that the radius of the cloud is more likely to happen ~ 14, whereas the histogram of the pair-wise distances suggests that, most clouds have separations ~ 8. It can be learnt that, there must be a lot of clouds with separations smaller than the sum of their distances, which leads to the confusion. The hardest pair to separate, inferring from the pair-wise distances, must corresponds to the pair with the smallest value (except for cases where $i = j$). Reading off from the graph, the smallest value is $5, 4$, corresponds to pair 7 and 9, so they must be the most difficult pair to separate, as it appears at first sight, and we are going to show whether this is indeed the case.

3 Implement and evaluate the simplest classifier

The simple classifier is built as follows: for each image, compute its distance to the ten clouds, the closest cloud is the most likely cloud which the belongs to. To compute the confusion matrix, simply implement the above strategy on all images, and for each set, calculate the percentage of the mis-classified ones to set d, which yields the confusion percentage of this current examining set. The confusion matrix can be visualised as below

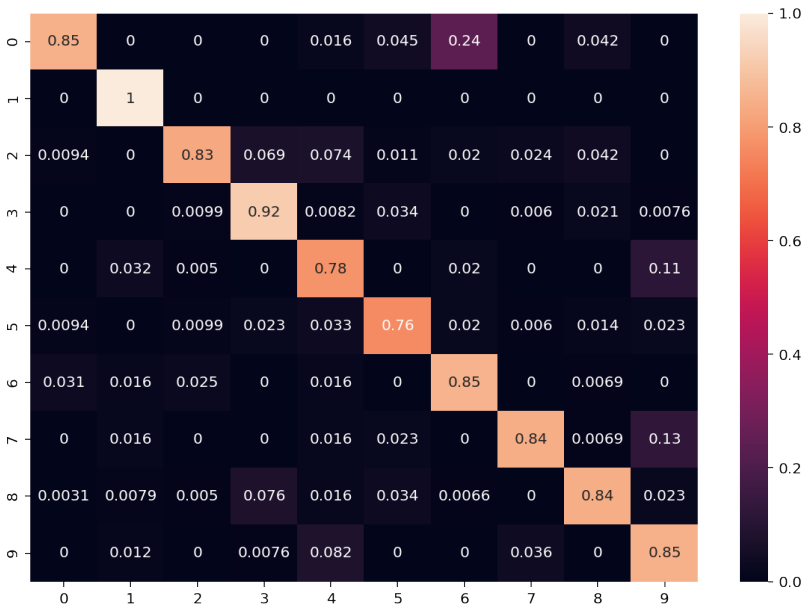

Figure 5: The confusion matrix of the training set, using the simplest classifier with Euclidean distances.

From the above graph we can see, 0.24 which is on the first row, seventh column, is the greatest value in the confusion matrix, except for the diagonal elements (which describes how likely one point is classified to its own set, and should ideally be 1.) This tells us that, contrary to our prediction, despite the pair 7 and 9 have the smallest pair-wise distance, which could potentially leads to the most difficult ones to distinguish – 0 and 6 is the leading source of confusion. Other secondary sources of confusion are: 7 and 9 (0.13), 4 and 9 (0.11). Now implement the same classifier on the test set. Its confusion matrix is shown as follows

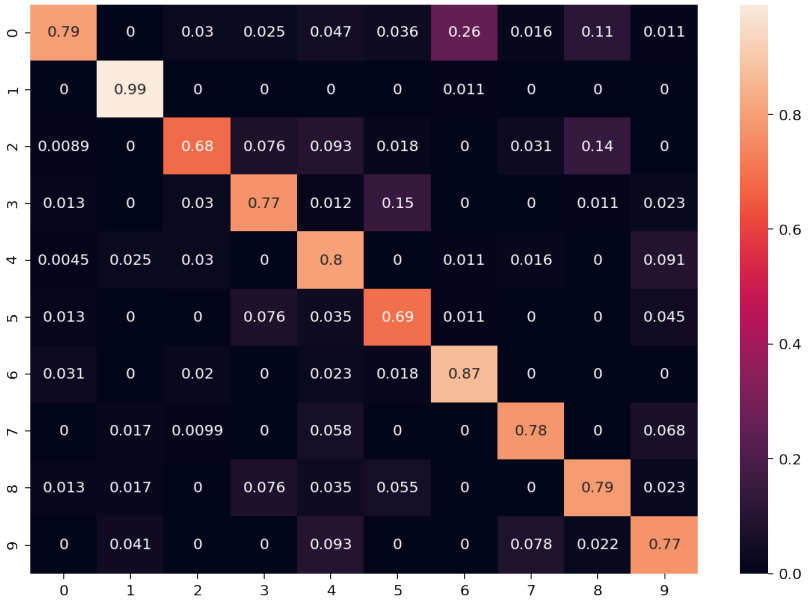

Figure 6: The confusion matrix of the test set, using the simplest classifier with Euclidean distances.

It can be learnt from the graph above, that in general, the accuracy for each point to be classified within its own set is decreasing (which can be seen from the diagonal elements of the matrix, which appear to be of darker shades as compared to that of the confusion matrix of the training

set). It implies that this classifier can not well represent the features of the test set, despite its acceptable behaviour on the training set. Now the most confusing pair remains to be 0 and 6, with secondary sources of confusion to be: 3 and 5 (0.15), 2 and 8 (0.14). Now finally apply the **sklearn.metrics.pairwise.pairwise distances** to get the confusion matrix of the test set, which is shown below.

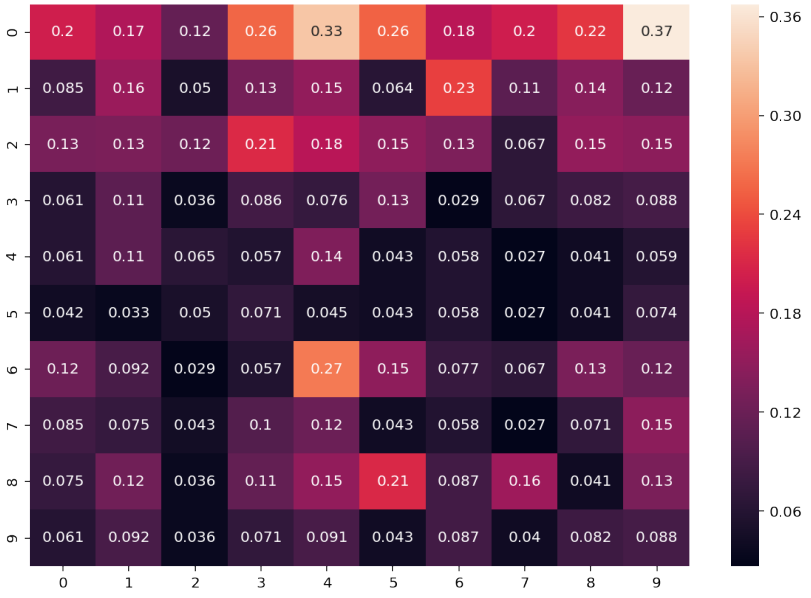

Figure 7: The confusion matrix of the test set, using the a classifier with pair-wise distances.

It can be seen, with pairwise distance the behaviour of the classifier is greatly suppressed. Hence we adopt the classical Euclidean distances in the upcoming sections.

4 Implement a multi-class perceptron algorithm

For a single layer perceptron, for each input, the input data is sent towards a weight matrix, whose shape is $(\text{n\_feature} + 1, \text{n\_node})$, with n_feature describing the number of features of the input data, and n_node the number of node of the layer. After transformation, the output is an n_node dimensional signal whose normalisation represents the probability of this input data belonging to a cloud C_i. With each iteration, the weight matrix is updated, until there is no mis-classified inputs. For each iteration of the weight matrix, we also forward the test set, to see the behaviour of its prediction. Since ultimately what we wanted is the weight which predicts both the training set and test set with some accuracy, the optimum stopping point should be at where the two accuracies diverge. The accuracies as a function of iteraction for the generalised perceptron algorithm is shown as follows

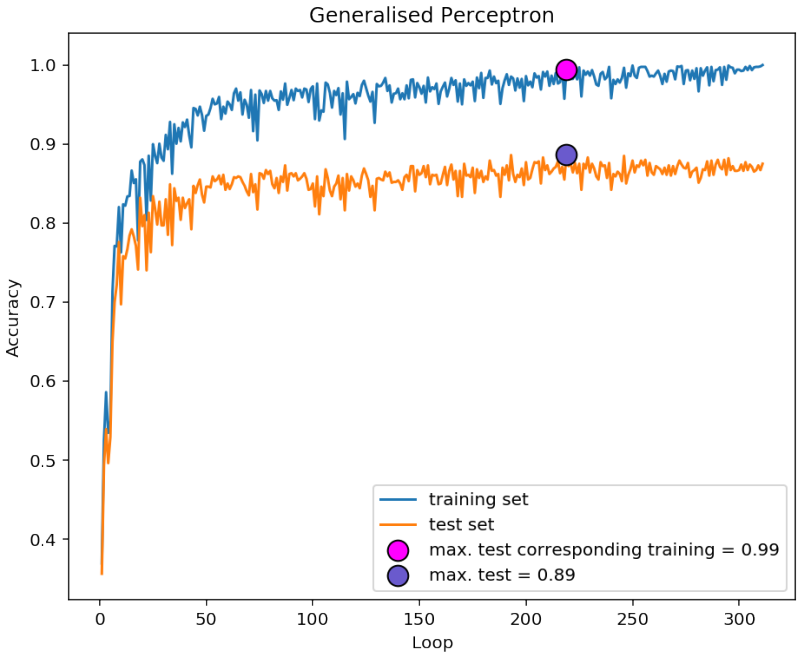

Figure 8: The accuracies on the training set and test set, implementing the generalised perceptron algorithm.

For a late-stopping (when the accuracy on the training set has already become 1), the total run time is 0.36 s, computing both the training set and test set. It can be seen from the above graph, it is sufficient to stop after some 220 iterations, and at this stopping point, the accuracy on the training set and test set is 99% and 89%

Now for the algorithm described in Geron's book, the updating method is slightly, different: instead of updating the weights according to both mis-classified labels of the false examples and the actual labels of those false examples, the update rule is now simpler (in expression). We just add an expected value of our output, and calculate the loss, which is the difference between the true output and desired output, where the idealised output should be a delta function localised at the true label index. For this method, there is no guarantee of a 100% accuracy on the training set, by experiment we found that the result is already converged when accuracy $> 96\%$. To find the optimum time to stop we again find the diverge point. And the result is shown as follows.

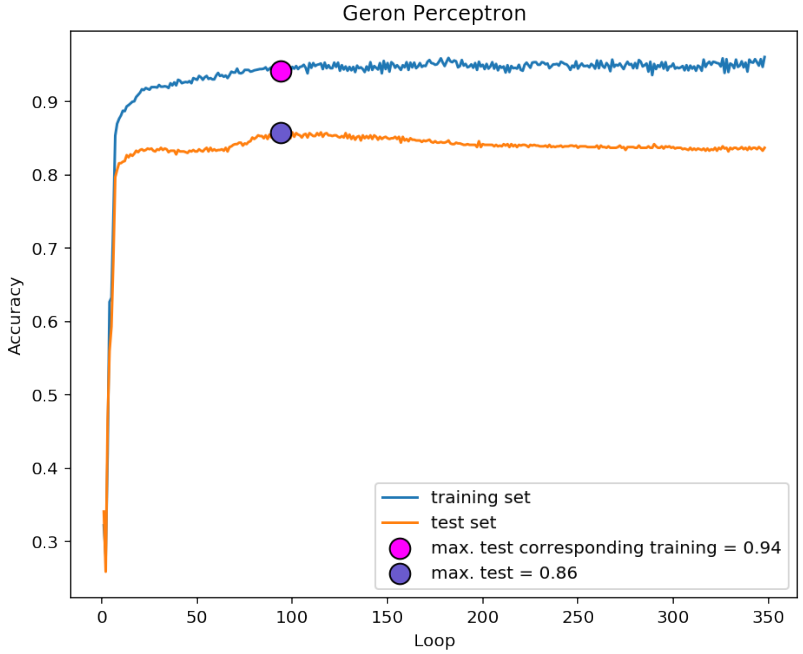

Figure 9: The accuracies on the training set and test set, implementing the perceptron algorithm described in Geron's book.

The total run time for this algorithm is 0.94 s. It can be seen from the graph above, that the optimum stopping point appears after som 100 iterations, yielding an accuracy of 94% on the training set and 86% on the test set.

5 Linear Separability

From the Cover's theorem, we can expect that for a dataset with length n larger than twice its number of features n_feature, so that $n >$ n_feature2, the points in the set are not separable. To get an idea of what this means, we compute the linear separability of a set C_i from another set C_j, in a very similar way we used to compute the confusion matrix, except for now we adopt a somewhat more advanced classifier. Set the iteraction upper limit to 300, so that the function stops after 300 iterations, the separability of the sets can be found as below

Table 2: The separability of all the clouds C_d from the remaining sets.

Cloud	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
Separability	0.8401	0.996	0.7475	0.5115	0.8361	0.0341	0.7483	1.	1.	0.9924

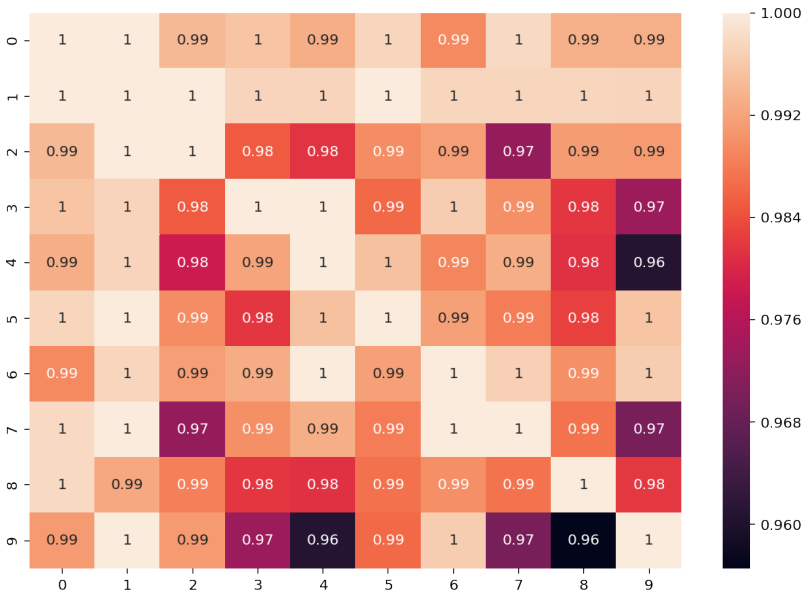

Figure 10: The separability of two sets, with 1 giving a 100% possibility to separate. So the lower the value, the less likely it is to separate one set from another.

It can be seen that, within limited time, it is most difficult to separate points in cloud C_9 from C_4 (also for C_9 and C_4). The separability of one set C_i from the remaining sets can be calculated as follows. Now set the upper limit of the iteration to 4000, and compute for each set the number of correctly classified, over the total number of points in this set. The separability of each cloud is listed in the table below. begintable From the table above it can be seen, the least separable cloud from the remainings is C_5, afterwards C_3.

6 Implement the XOR network

To understand why and how the multi-layer perceptron can solve the XOR problem, we first have to understand where the XOR problem originates from. The input of the XOR function should be $(0, 0), (0, 1), (1, 0)$ and $(1, 1)$, with corresponding outputs $0, 1, 1, 0$. Since the outputs of zeros and ones are all on the diagonal end of a square of radius one, it is not possible to draw a line to distinguish the two sets. In order to solve this problem we have to implement multi-layer perceptron. To solve this problem we started with a simplest case where the first layer has two nodes. In this setup it is very difficult to get converged results, which is because of the properties of the Sigmoid function. When x is very close to 0.5, the loss became very small and hence taking increasingly longer time to compute. To circumvent that we increased the number of nodes in the first layer to four, and replaced the activation function of the first layer to an *relu* function. The result is as follows

```
Accuracy:  75.0 % loss:  0.25 output:  [[0.13085595 0.61938893 0.88176492 0.6328128 ]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12917529 0.60855077 0.90881347 0.62193356]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12846181 0.60616875 0.91827217 0.61489675]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12799112 0.60594127 0.92252057 0.60868397]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12759844 0.60656382 0.92467688 0.60277783]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12722911 0.60760674 0.92587246 0.59703434]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12686122 0.6088912  0.92659786 0.59140162]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12648524 0.61033187 0.92708669 0.58585788]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.126097   0.61188409 0.92745582 0.58039304]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12569492 0.61352253 0.92776532 0.57500214]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12527866 0.61523161 0.92804644 0.5696826 ]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12484852 0.61700084 0.92831551 0.56443299]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.1244051  0.61882253 0.928581   0.55925244]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12394917 0.62069055 0.92884724 0.55414034]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12348155 0.62259973 0.92911634 0.54909617]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12300305 0.6245455  0.92938927 0.54411947]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12251449 0.62652373 0.92966635 0.53920976]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12201668 0.6285306  0.92994759 0.53436658]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12151038 0.63056258 0.93023281 0.52958941]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12099633 0.63261637 0.93052179 0.52487772]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.12047522 0.6346889  0.93081421 0.52023095]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.11994773 0.63677728 0.9311098  0.51564853]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.1194145  0.6388788  0.93140824 0.51112985]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.11887613 0.64099096 0.93170925 0.50667429]] estimate: [[0. 1. 1. 1.]]
Accuracy:  75.0 % loss:  0.25 output:  [[0.11833321 0.64311138 0.93201253 0.50228121]] estimate: [[0. 1. 1. 1.]]
Accuracy:  100.0 % loss:  0.0 output:  [[0.11804598 0.6442293  0.93217288 0.4999953 ]] estimate: [[0. 1. 1. 0.]]
```

Figure 11: The output of the XOR network, where the first column gives the accuracy on the four input sets, and second column gives the loss where $loss = y - d$, the third and fourth column correspond to the values after the second activation function, and the rounded value where we take > 0.5 as 1 and < 0.5 as 0.

The output is printed for every 1000 iterations, taking roughly 0.6 s. Throughout this implementation we have adopted random initialisation, which could be somewhat problematic since a random initialisation of weights does not guarantee the convergence Additionally we also found that by increasing the number of nodes we get a higher chance of convergence.

7 Conclusion

In this assignment,we have explored using the perceptron algorithm on classifying the images. We have implemented the generalised perceptron algorithm which consists of a single layer of neurons, and have found a perfect separation between the cloud sets, so that every input training image can be classified without mistake. However, we have also found that despite its accuracy on the training set, the weights predicted using the training set could not classify the test set with 100% accuracy, the optimum stopping point for achieving high accuracies for both sets lies somewhere in between. Despite the predicted most difficult to separate pairs being 7 and 9, the hardest pairs to separate is actually 0 and 6, found by using the single layer perceptron algorithm. In general, when the algorithm described in Geron's text book wa applied, the accuracy on both sets were reduced, due to an inability to adjust the value for η when the loss gets smaller. And when tested against the generalised algorithm, the performance was in general not as comparable.

When implementing the neural network from scratch, it was found that, by increasing the number of neurons in the hidden layer, and replacing the activation function of the fist layer to *relu*, it is more efficient to obtain converged results.