Neural Networks - Assignment 1

Manvi Agarwal - s2817047
Daniel Miedema - s2478579

February 22, 2019

1 Problem Statement

This report presents an implementation of the Rosenblatt perceptron algorithm. An artificial dataset with N dimensions and P examples is created, and a Rosenblatt perceptron is trained by sequential presentation of examples from the dataset. By doing this several times, we analyze the relationship between the proportion of successful perceptron training runs, $Q_{l.s.}$, and the nature of the dataset, quantified by $\alpha = \frac{P}{N}$.
In Section 2, the method used to generate a randomized dataset is presented. In Section 3, the algorithm is introduced, and the procedure of updating the weight vector is presented. In Section 4, the simulation setup is explained and results are presented. In Section 5, the results are discussed. In Section 6, we extend the findings from Section 5 by adding a clamped input θ to all feature vectors and increasing the value of N, and then observing the behaviour of $Q_{l.s.}$.

2 Dataset Generation

The dataset $\mathbb{D} \in \mathbb{R}^{P\times(N+1)}$ is generated by drawing P N-dimensional feature vectors and binary labels ± 1. Each element of the feature vectors $\mathbb{D}_{ij}$, where $1 \le i \le P$ and $1 \le j \le N$, is a uniform pseudo-random number, selected by using the `randn` command. Each label $\mathbb{D}_{ij}$, where $1 \le i \le P$ and $j = N+1$, are selected from $\{1, -1\}$ with equal probability 0.5, using the `randsample` command. Such a dataset can be represented as $\mathbb{D} = \{\xi^\mu, S^\mu\}_{\mu=1}^{P}$, where ξ^μ refers to a feature vector and S^μ refers to its corresponding label. The implementation of this has been shown in Listing 1.

Listing 1: Generating random datasets

```
function y = data(N, P)
y = randn(P,N);
output_vector = transpose(randsample([1,-1],P,true));
y = [y output_vector];
end
```

3 Sequential Perceptron Training and the Rosenblatt Algorithm

In each epoch n_i, where $1 \le i \le n_{max}$, there are P time steps. At each time step $t = 1, 2, ..., P$, feature vector $\mathbb{D}(i, 1:N)$, where $1 \le i \le P$ is presented, and the weights are updated according to the following rule:

$$\mathbf{w}(t+1) = \begin{cases} \mathbf{w}(t) + \frac{1}{N}\xi^{\mu(t)}S^{\mu(t)}, & \text{if } E^{\mu(t)} \le 0 \\ \mathbf{w}(t), & \text{otherwise} \end{cases}$$

This means that at time step t, first, the example $\mu^t = \mathbb{D}(t, :)$ is selected. Then, the error is calculated as:

$$E^{\mu(t)} = \mathbf{w}(t).\{\xi^{\mu(t)}, 1^{\mu(t)}\}S^{\mu(t)}$$

If this error is negative or 0, the weight vector **w** is updated. This updation continues until the maximum number of epochs is reached, or the algorithm converges to the right solution. This process is implemented as shown in Listing 2.

Listing 2: Rosenblatt perceptron algorithm

```
for epoch = 1:n_max
    if success == 0
        count = 0
        for t = 1:P
            data_point = D(t,:)
            error = dot(w, data_point(1:N+1))*data_point(length(data_point))
            if error <= 0
                for weight = 1:length(w)
                   w(weight) = w(weight)+((1/(N+1))*data_point(weight)*
                       data_point(length(data_point)))
                end
            else
                count = count + 1
            end
        end
        if count == P
            success = 1
        end
    end
end
```

4 Simulation & Results

The algorithm was evaluated by computing $Q_{l.s.}$, the fraction of successful runs, and plotting it as a function of $\alpha = P/N$. This was done for a varying number of random datasets, n_d, which was set as 10, 50, 100, and 500. The number of dimensions, N, was fixed at 20, and whereas the maximum number of epochs, n_{max}, was fixed at 300. α varied between 0.75 and 3.0, with steps sizes of 0.25. This means that for a fixed N, the number of examples, P, changed with the change in α. Fig 1 shows the results.

5 Discussion

For $\alpha \leq 1$ we can see that classification of every data set with 100% accuracy is reached with this algorithm. This is expected since the Vapnik-Chervonenkis dimension is equal to N. In other words, when the number of data points is less than or equal to the number of dimensions, every dataset is linearly separable. When this is the case, the Rosenblatt algorithm, given enough time, will always be able to find the solution.

By visual inspection of the graph, it can be seen that for $\alpha \leq 2$, the algorithm achieves $0.5 \leq Q_{l.s.} \leq 1$. As N approaches infinity, the theoretical value of $Q_{l.s.}$ has been shown to be:

$$Q_{l.s.} = \begin{cases} 1, & \text{if } \alpha \leq 2 \\ 0, & \textit{if } \alpha > 2 \end{cases} \tag{1}$$

This gives the property that for any value of N in the training data, the expected success rate is 0.5 for $\alpha = 2$. This counting rule assumes that the dataset presented to the algorithm is in general position.

In our implementation, this boundary indeed looks to be around $\alpha \leq 2$. However, from our experience, to obtain this result, we have to increase the maximum number of epochs, n_{max}, to a high enough number. We initially set n_{max} to 100, but then had to increase it to 300 to observe this behaviour. This means that the algorithm must be allowed to run for 'long enough' for it to converge.

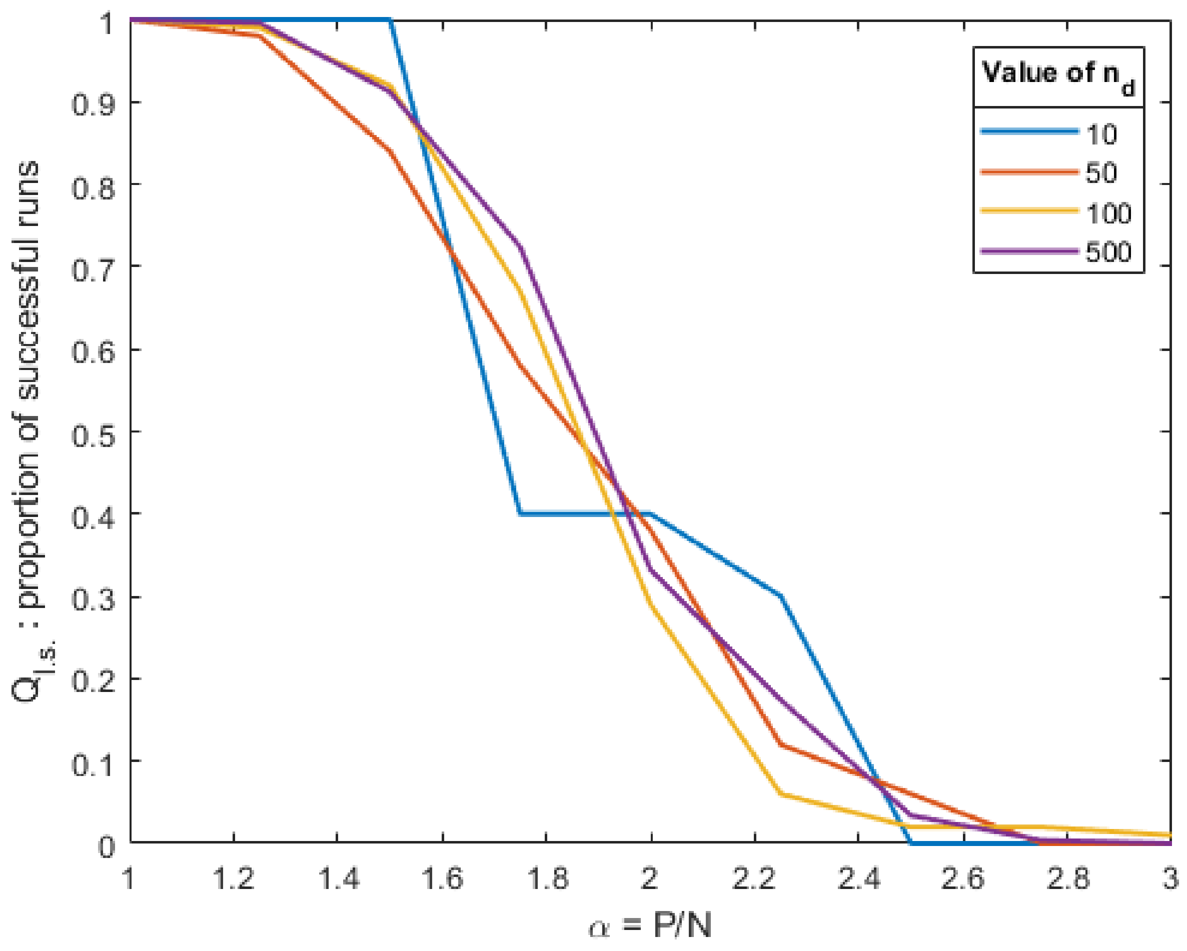


Figure 1: Investigating the success rate of Rosenblatt perceptron algorithm as a function of α: $N = 20$, $n_{max} = 300$

Finally, as per our expectations, as the value of n_d increases, the graph gets smoother as the sample mean of the success rate converges to the true mean of the distribution of success rates for α.

6 Bonus Exercises

6.1 Adding a clamped input to the feature vectors

By modifying the code presented in Listing 1, the dataset $\mathbb{D}$ is modified such that $\mathbb{D} \in \mathbb{R}^{P\times(N+2)}$. This is done by generating a bias vector. This is added as a column of ones to the dataset, which will account for the bias weight in the weight vector $\mathbf{w}$. Hence, the dataset so generated can be represented as $\mathbb{D} = \{\xi^\mu, 1^\mu, S^\mu\}_{\mu=1}^{P}$. Again, with N fixed at 20, n_{max} fixed at 300, and α going from 0.75 to 3.0 with step sizes of 0.25, the results can be seen in Fig. 2.

In Fig. 2, it can be seen that, for $\alpha \leq 2$, $Q_{l.s.} \geq 0.5$. This is similar to the results obtained from Fig. 1. This is in line with the fact that homogenous linear separability in $\mathbb{R}^N$ is equivalent to inhomogenous linear separability in $\mathbb{R}^{N+1}$. This means that if the number of dimensions of each input vector is increased by 1 by adding a clamped input, it does not affect the linear separability of the dataset in this higher dimension.

6.2 Changing the number of dimensions

We changed the value of N, which represents the number of dimensions in the feature vectors without the clamped input. N was set at 10, 25, 50 and 100, and you can see the results for the four cases in Fig. 3.

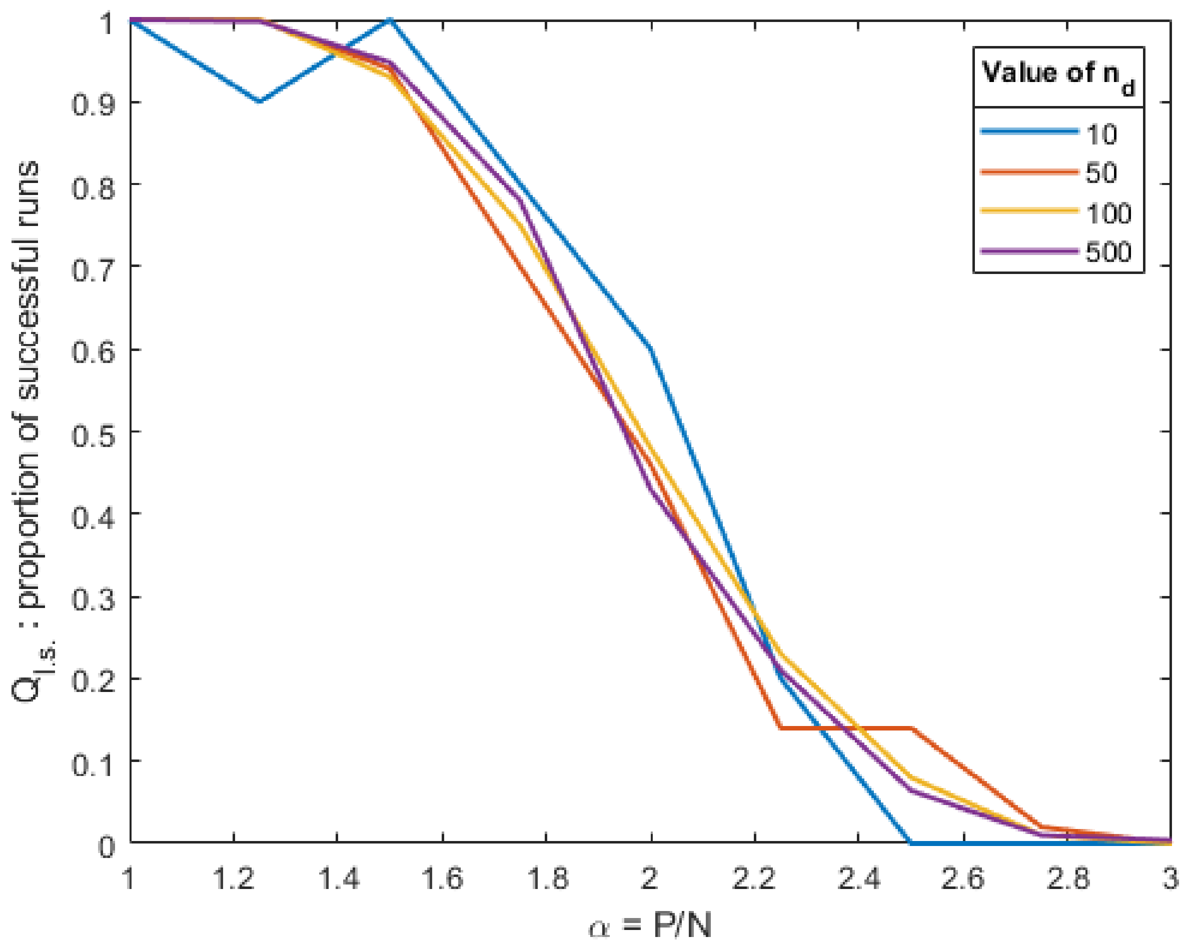


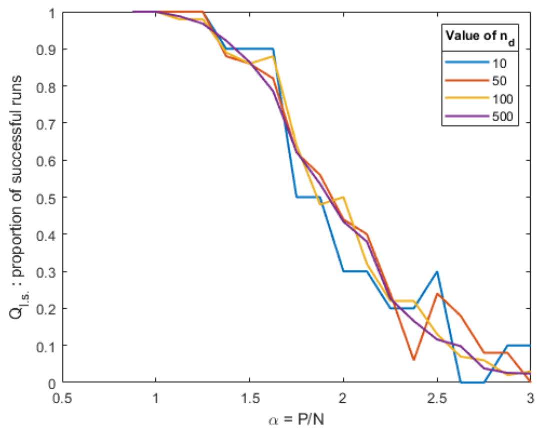
Figure 2: Effect of adding a clamped input to all feature vectors: $N = 20$, $n_{max} = 300$

We see that as n_d increases, the graph gets smoother, which confirms our findings from Section 4. Furthermore, as N increases, $f(\alpha) = Q_{l.s.}$ gets closer to a step function at $\alpha = 2$. This is in line with the findings presented in Equation 1, in Section 5.

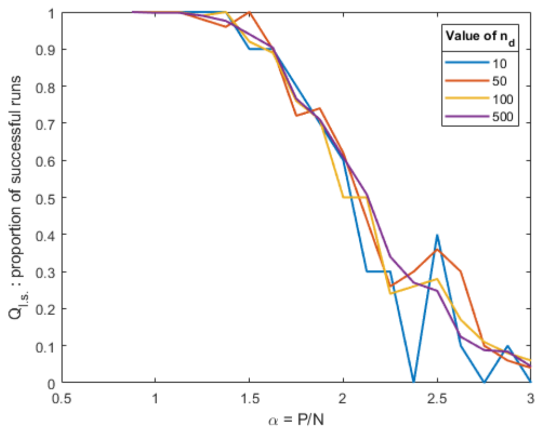
6.3 Combining different numbers of dimensions with clamped input

We combined the ideas from subsections 6.1 and 6.2 by both adding clamped inputs to the feature vectors, and changing the value of N. The results are presented in Fig. 4.

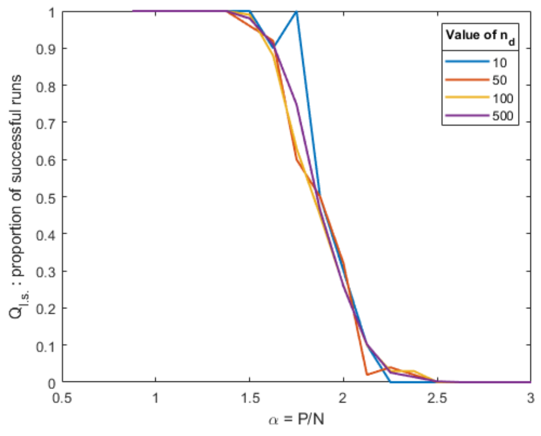
We do not see any significant difference between the findings from datasets without the clamped input (Fig. 3) and those with the clamped input (Fig. 4). This further confirms our findings from Section 6.1, where we concluded that adding a clamped input to the feature vectors does not affect the behaviour of $Q_{l.s.}$. We still see that as n_d increases, the graph gets smoother, and as N increases, $f(\alpha) = Q_{l.s.}$ gets closer to a step function at $\alpha = 2$, similar to Fig. 3.



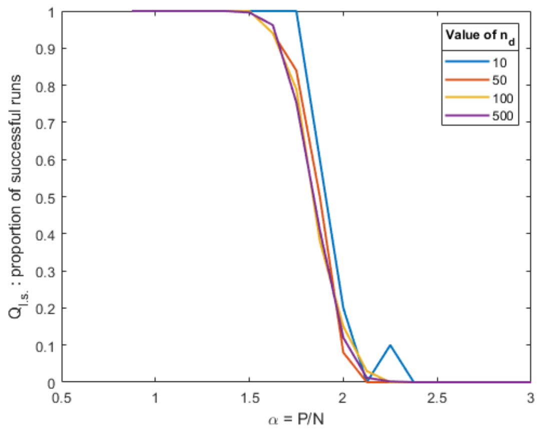
(a) $N = 10$



(b) $N = 25$

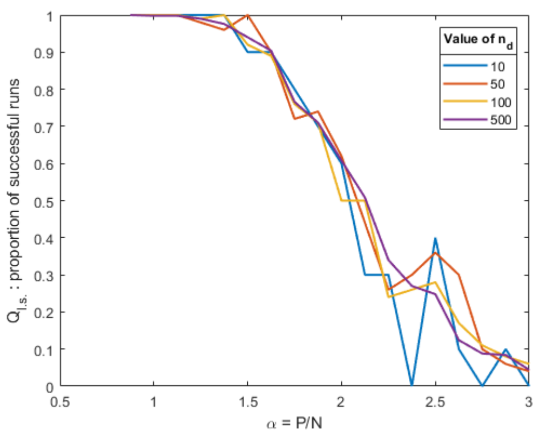


(c) $N = 50$

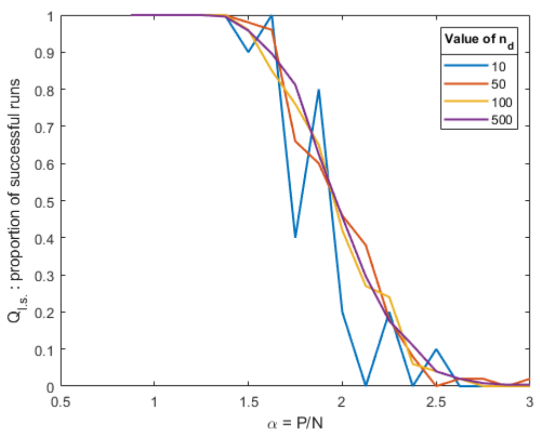


(d) $N = 100$

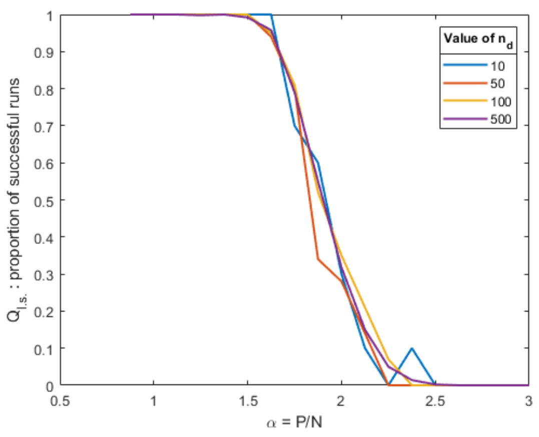
Figure 3: Effect of changing N, the number of dimensions: $n_{max} = 300$



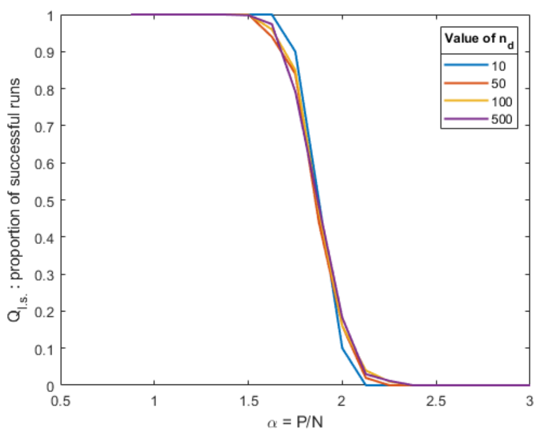
(a) $N = 10$



(b) $N = 25$



(c) $N = 50$



(d) $N = 100$

Figure 4: Effect of adding a clamped input to all feature vectors and changing N, the number of dimensions: $n_{max} = 300$