Neural Networks
Assignment I - Perceptron Training
Group 10

Satyanarayan Nayak, Swastik
s.nayak.1@student.rug.nl
S4151968

Nivin Pradeep Kumar
n.pradeep.kumar@student.rug.nl
S4035593

January 31, 2020

Contents

Abstract

Perceptron is a powerful intelligent unit that functions as the building blocks a simple and/or complex Neural network(s). Perceptrons can be interpreted as a single McCulloch Pitts unit which is connected to N input neurons[1]. Rosenblatt perceptrons algorithm is a training strategy that can employ over the perceptrons to perform binary classifications. We will train the perceptron with randomly generated data with zero mean and unit variance and observe the behavior of our model under different environments.

1 Introduction

The perceptron is the simplest form of a neural network used for classification of patterns said to be linearly separable. It is an algorithm for supervised learning of binary classifiers. [1] Binary classification is the task of classifying the elements of a given set into two groups based on a prescribed rule. Rosenblatt's perceptron can handle classification tasks for linearly separable classes. In this assignment, we generate artificial data containing N-dimensional feature vector and binary labels. [2] This artificial data is used to implement sequential perceptron training. Rosenblatt's algorithm is used on the created artificial data set. The training and the algorithm are run repeatedly several times and the value of successful runs is calculated. Each of these implementations is explained briefly in the following sections.

2 Rosenblatt Perceptron

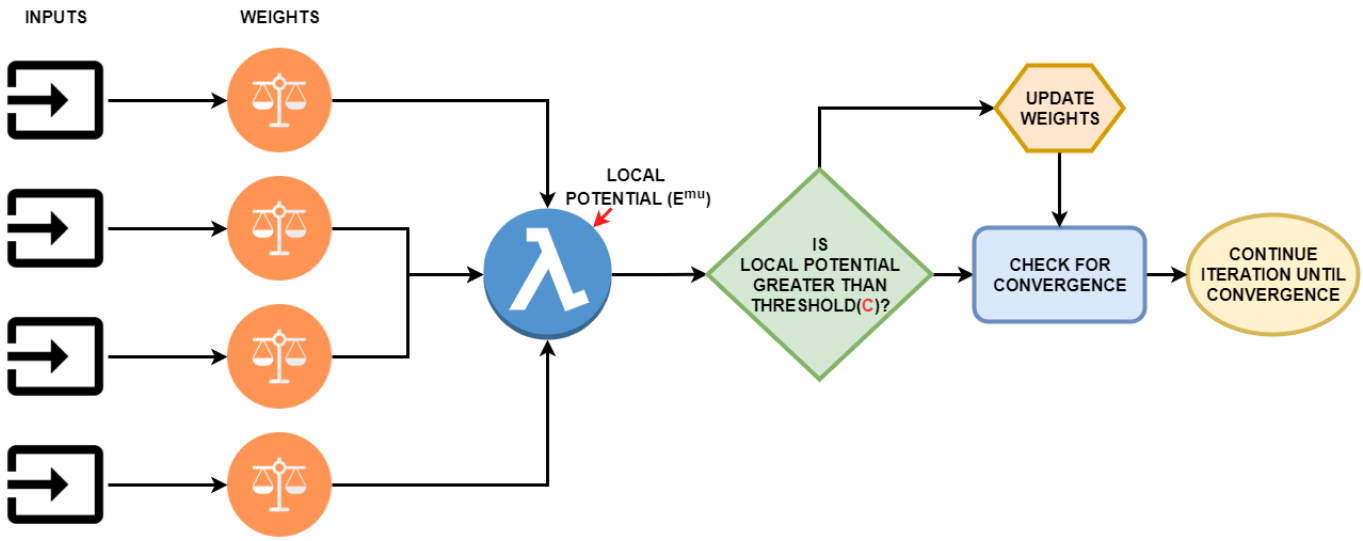

Figure 1: Rosenblatt Perceptron Algorithm representation

The Rosenblatt perceptron algorithm, in a nutshell, can be represented as shown in figure 1. An important mention is the weights of the perceptron that are responsible for providing the intelligence to the perceptron network. The weights are tweaked constantly by incoming inputs or training example data (supervised learning). Some common notations that will be used across this document are as follows,

ξ	Input or Example Data
$\mathbf{S}$	Label or Classification made by the perceptron
S_T	Target or Expected Classification
$\mathbf{W}$	Weights
E^{μ}	Local Potential

We need to shed some light over some important topics before getting into the technicalities of the Rosenblatt perceptron algorithm,

2.1 Initialization

At time t=0, the weight vector can be initialized to any required state before the start of the training, a popular initialization state is called the Tabula Rasa's where the weight vector is set to 0 (W(0) = 0) [1].

2.2 The perceptron storage problem

The perceptron storage problem state that we are interested in training our weights such that the model can reproduce the original labels as accurately as possible. N-dimensional input or example data ξ is consumed sequentially by the perceptron algorithm and is classified into either one of the labels $S = sign(W.\xi) = \pm 1$ (sign(..) simulates an activation function) based on the current weights of the model. It is highly unlikely that the output S if compared with the expected labels or classifications S_T would result in a perfect match. This is because the weight vectors are not trained to reproduce the expected classification outputs.
Consider the example data are fed to the system sequentially with time interval t = 1,2,3....., At time t, for an input ξ, the activation function will classify it to a label ± 1. If the classification made 'S' matches with the expected label S_T, the weights remain unchanged. On a misclassification $S \neq S_T$, the weights will be modified as follows,

$$W(t+1) = W(t) + \frac{1}{N} f(E^\mu).\xi^\mu S_T^\mu \tag{1}$$

where $f(E^\mu)$ is the actual algorithm that will used to train the weights (earlier mentioned $S = sign(W.\xi)$ activation function) and the E^μ is called the local potential that can be deduced as follows,

$$E^\mu = W.\xi^\mu S_T^\mu \tag{2}$$

The product $\xi^\mu S_T^\mu$ is the Hebbian term in the system which utilizes the input ξ^μ and output S_T^μ to provide learning capabilities. We introduce a constant $c > 0$ that functions as a marginal threshold with respect to E^μ. If $E^\mu > c$ then the weights remain unchanged, otherwise we will modify the weights as a means to counteract the error [1]. The value of 'c' is irrelevant because that if a weight W_1 achieves linearly separable boundary then its scalar products $W_2 = \lambda W_1 \ \ where \ \ \lambda > 0$ will also produce linearly separable boundaries, due to the linearity of the scalar product[1]. By using an appropriate activation function we hope to accurately reproduce the original labels of the example data that was used to train the perceptron model, thus solving the perceptron storage problem.

2.3 Algorithm

We will consider the data $D = \xi^\mu, S^\mu \ \ where \ \ \mu = 1, 2, 3...P$, will be fed to the system in a cyclic presentation i.e $v(t)$ = 1,2,3...P,1,2,3...[1]

Algorithm 1: Rosenblatt perceptron algorithm [1]

at discrete time step t
- Determine the index $v(t)$ of the current example according to the cyclic presentation $v(t)$ = 1,2,3...P,1,2,3...
- Compute the local potential, $E^{v(t)} = W(t).\xi^{v(t)} S_T^{v(t)}$
- Update the weight vectors according to the modulation function

$$W(t+1) = W(t) + \frac{1}{N}\Theta[c - E^{v(t)}]\xi^{v(t)} S_T^{v(t)} \tag{3}$$

- Check for convergence by checking if the perceptron storage problem is resolved.

Equivalently, the system can be realised by using the embedding strengths rather than weights

$$X^{v(t+1)} = X^{v(t)} + \Theta[c - E^{v(t)}] \tag{4}$$

All other embedding strengths other than v(t) should remain unchanged at time t.

We can formally address the Rosenblatt perceptron algorithm with equation 3 as it is the core logic behind the training. The modification of the weights is regulated by the $\Theta(...)$ function, that functions as follows.

$$\Theta[c - E^\mu] = \begin{cases} 1 & if \quad E^\mu \leq c \\ 0 & if \quad E^\mu > c \end{cases}, \tag{5}$$

A relationship can be derived between a data of size P to its dimension N, this is especially useful in defining the probability that the data is linearly separable. We will forego the technicalities in deriving the equations, for more details kindly refer to *Learning from Example-An Introduction to Neural Networks and Computational Intelligence* [1].

$$P_{l.s.}(P,N) = \frac{C(P,N)}{2^P} = \begin{cases} 1 & \text{for } P \leq N \\ 2^{1-P}\sum_{i=0}^{N-1}\binom{P-1}{i} & \text{for } P > N \end{cases} \tag{6}$$

The critical storage value is denoted as α, where

$$\alpha = \frac{P}{N} \tag{7}$$

The critical storage value for a perceptron $\alpha = 2$ [1]. As $N- > \infty$ the equation 6 becomes,

$$P_{l.s.}^{\infty}(P,N) = \begin{cases} 1 & \text{for } \alpha \leq 2 \\ 0 & \text{for } \alpha > 2 \end{cases} \tag{8}$$

We will use the equation 7 relationship while generating the example data in our implementation.

3 Implementation

3.1 Data Generation

The first step is to generate artificial data sets containing P randomly generated N-dimensional feature vectors and binary labels $D = \{\xi^{\mu}, S^{\mu}\}_{\mu=1}^{P}$. Here, the $\xi^{\mu} \in R^{N}$ are vectors of independent random components $\xi_j^{\mu} \sim N(0,1)$ with mean zero and unit variance. The example data is generated using the matlab's **`randn(P, N)`** function, and the labels $S = \{+1, -1\}$ are generated with a equal probability 1/2.
The size of the example data P is determined using the equation 7. For a series of alpha values ($\alpha = 0.75, 1.0, 1.25, ...3.0$) the fixed dimensions of the data N is used to determine the value of P.

3.2 Training

Before the training process is started the weights of the perceptron are set to Tabula Rasa's as discussed in section 2.1. The example data is fed to the system in the form of cyclic representation as discussed in section 2.3. The algorithm is a sequential procedure where at time t = 1,2,3..... (epochs), the example data at index $\mu(t)$ is fed to the system based on which it learns the weights of the perceptron. The weights of the perceptron will remain unchanged if the classification made by the perceptron is correct i.e $S = S^{\mu(t)}$, the weights are updated based on the equation 3 and equation 5 when $S \neq S^{\mu(t)}$. The learning strategy of the perceptron weights during training can be defined as follows,

$$\mathsf{W}(t+1) = \begin{cases} \mathsf{W}(t) + \frac{1}{N}\xi^{\mu(t)}S^{\mu(t)} & \text{if } E^{\mu(t)} \leq 0 \\ \mathsf{W}(t) & \text{otherwise} \end{cases}, \tag{9}$$

Where the local potential $E^{\mu(t)} = W.\xi^{\mu(t)}S^{\mu(t)}$, is consumed by the heaviside function $\Theta(c - E^{\mu(t)})$ to determine if weights of the perceptron needs to be updated or not. The $\mu(t)$ is used to uniquely identify the example data during the training process. The algorithm stops on reaching the stopping criteria where the local potential for all the example data are greater than 0 i.e $E^{\mu(t)} > 0 \ \forall \ \xi^{\mu}, \ where \ \mu = 1, 2, 3.....P$ or if the maximum number of allowed iterations n_{max} is reached.

4 Simulations and Evaluations

4.1 Perceptron storage success rate

For a fixed value of P and N, the example data and the labels are generated as discussed in section 3.1. The perceptron training continued for n_{max} iterations, and the fraction $Q_{l.s}$ is determined for the given P and N values. The alpha value is varied to get different combinations of P and N values for which we will determine the $Q_{l.s}$ value, the simulation helps us study or understand the probability $P_{l.s}$ of the perceptron to find the linear separation and to reproduce the correct labels for the example data over which the perceptron was trained.
The figure 2 shows the storage success rate for various alpha values, from our observation we notice that as the alpha value increases the error in reproducing the correct labels for the example decreases significantly. This is due to the fact that as the number of example data increases the perceptron weights are trained to attain decision boundary where the two types of data are linearly separated. On the downside, as the alpha increases the computational cycles required to run the perceptron algorithm also increases.

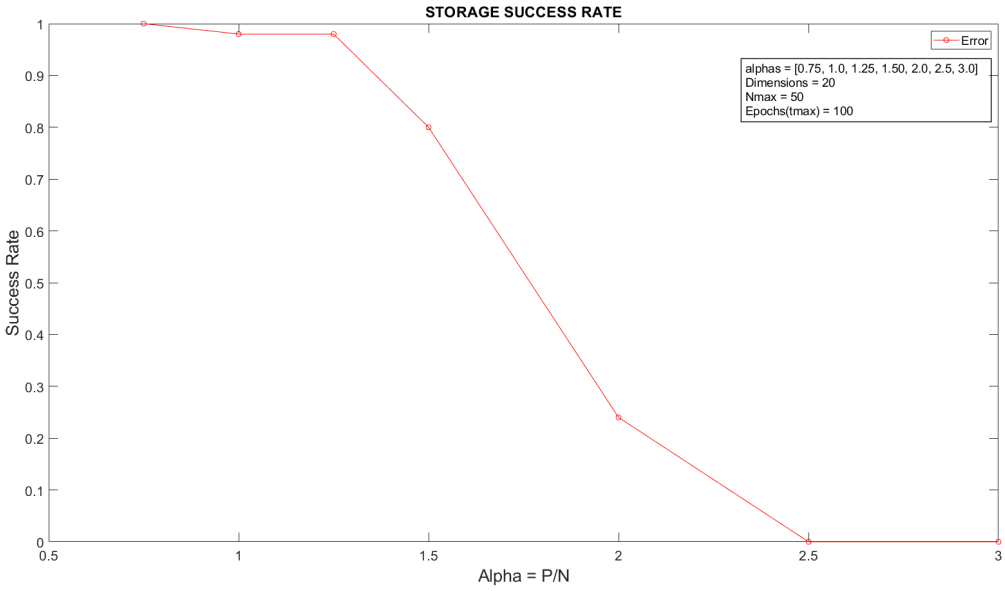

Figure 2: Perceptron storage success rate

4.2 Storage success rate by dimensions

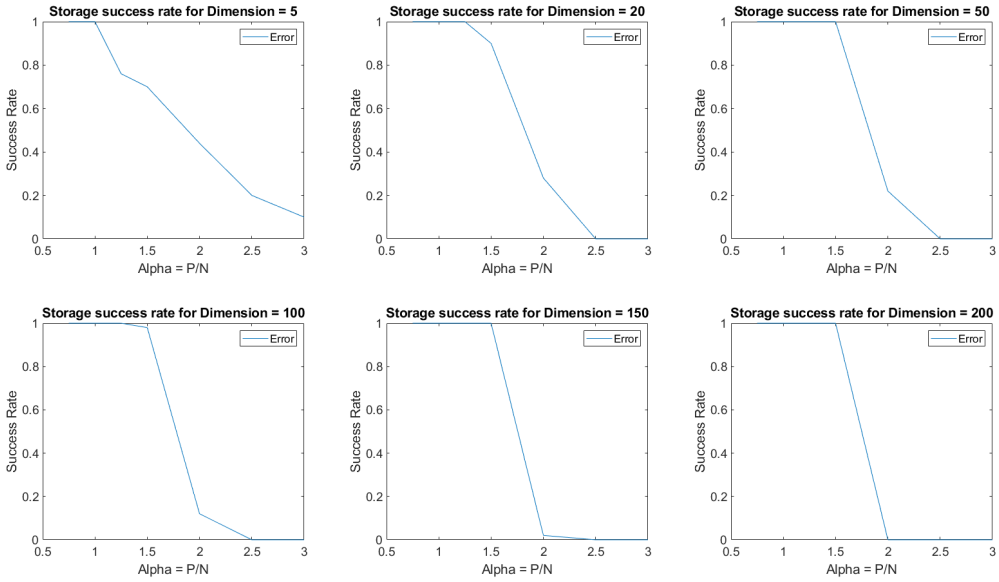

Figure 3: Perceptron storage success rate

The goal of this simulation was to determine if the storage success rate achieves a step function with the increase of dimensions N. To test this theoretical scenario we simulated the perceptron storage success on different dimensions N = 5, 20, 50, 100, 150, 200. Figure 3 shows the results from the simulation. We can observe that at N = 200, the storage success rate plot approximately appears to resemble step functions. And for a small N value (N=5) we can see that the plot resembles an inverted exponential curve rather than a step function. We limited our dimensions to 200 in the interest of CPU time. If N was increased further the storage rate should ideally match with the theoretical step function.

4.3 Embedding strengths counts by dimensions

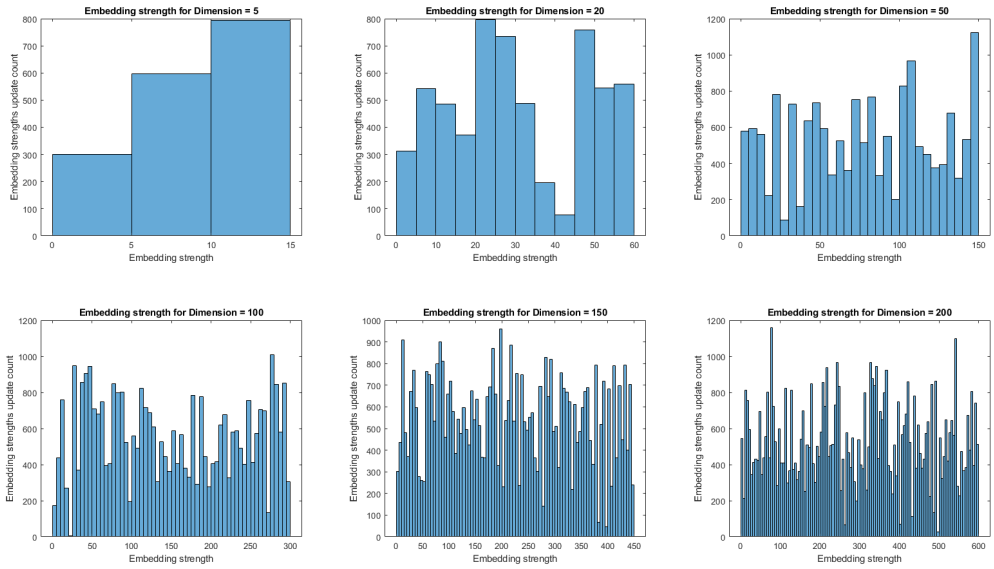

Figure 4: Embedding strengths counts by dimensions

The goal of this simulation was to observe the embedding strengths update counts to gain some valuable insights. Figure 4 shows the result of this simulation. We can see that during the perceptron training, every weight or embedding strength units are trained in varied frequencies. We observe that at any given dimensions the maximum hebbian updates performed on a single embedding strength unit is approximately around 800-1000 updates, but Unfortunately we could not derive any significant insight based on this simulation.

4.4 Significance of threshold (c) over storage success rate

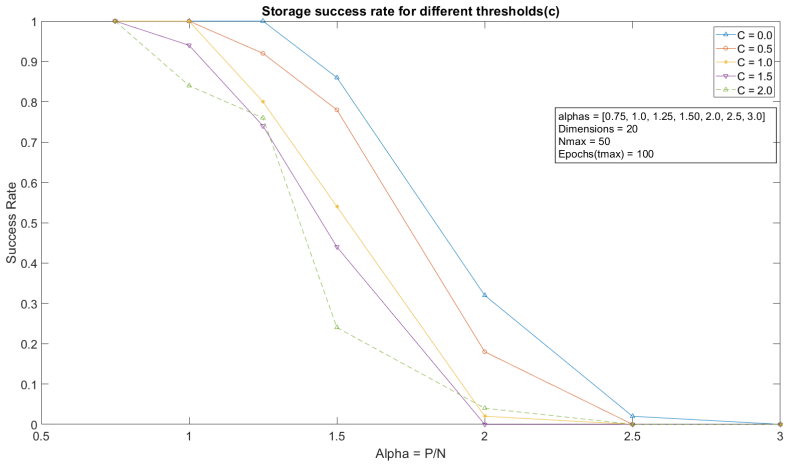

Figure 5: Storage success rate for different thresholds(c)

In this simulation, we will be observing the effects of different learning rates or threshold(c) (discussed in equation 5) over the storage success rates. A lower dimension data (N=20) was intentionally selected to observe distinct deviations between different thresholds, this eliminates the bias of dimensions over the success of storing the example data and allows us to test the scenario-based only on the thresholds. From figure 5 we can observe that at a higher threshold the error rate drops quicker at lower alpha values in comparison to a lower threshold. At $c = 1.5$ we see that convergence was achieved at $alpha = 2$, but with higher threshold $c = 2.0$ the convergence is achieved at $alpha = 2.5$, which implies that a faster convergence is not guaranteed at higher threshold or learning rate.

4.5 Significance of epochs(t_{max}) over storage success rate

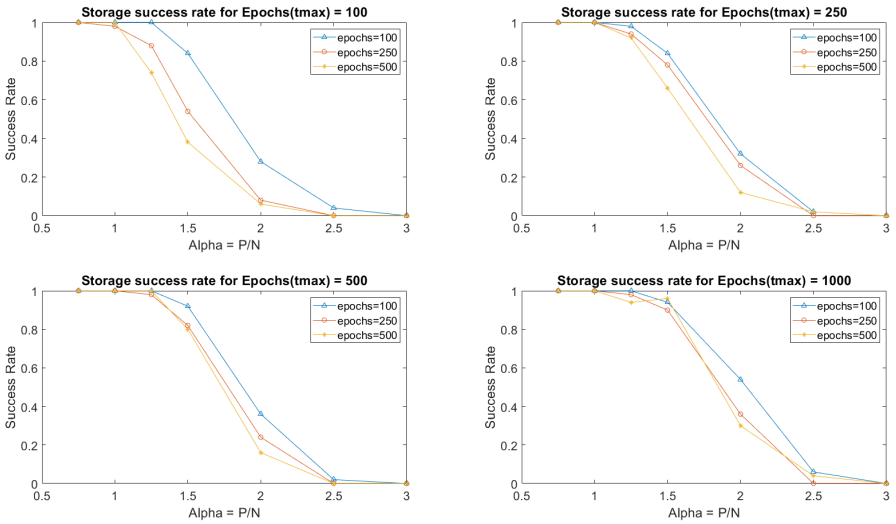

Figure 6: Storage success rate for different epochs(t_{max})

The simulation will be to test the contribution of t_{max} or epochs towards solving the perceptron storage problem for different learning rates, we have limited the learning rates to c = 0.0, 1.0, 2.0 for this study. The t_{max} controls the number of cyclic representations of data (discussed in section 2.3) is being used to train the perceptron weights. A perceptron cannot be infinitely trained over the example data due to the convergence criteria. From the figure 6 we can observe the storage success plots ($Q_{l.s}$) for different epochs, we can observe some minor improvements in epochs 50 and 200 at the threshold c = 0.0. But the gain observed is negligible. This is solely due to the convergence property of the Rosenblatt perceptron algorithm. Assume the system converges at t = 40, regardless of the value of t_{max} = 50, 100, 150, 200 the system will converge and stop the training at t = 40. We can infer that increasing the t_{max} is not a guaranteed method to improve the training accuracy but a sufficiently large epoch is required to approximately converge the model to a favorable state.

4.6 Significance of dimensions over storage success rate

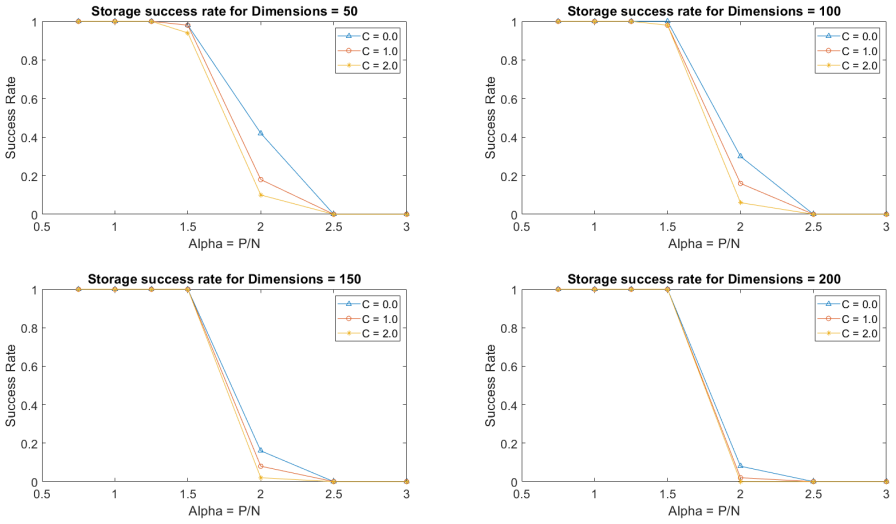

Figure 7: Storage success rate for different dimensions

From the simulations performed in the previous scenarios, we saw no significant changes in most of the scenarios, but we did see a significant improvement of $Q_{l.s}$ to resemble the step function in section 4.2. In this simulation we are

testing the significance of different dimensions $N = 50, 100, 150, 200$ over the three thresholds $c = 0.0, 1.0, 2.0$. From the figure 7 we can see that at higher dimensions regardless of the threshold (c), the $Q_{l.s}$ plots are strikingly similar, with a similar drop in the error value and convergence. This can be explained by the nature of how dimension affects the Rosenblatt algorithm. The size of the example data is determined using $P = \alpha N$ which implies that $P \propto N$, for a larger N value we will have larger example data to train the perceptron. And the perceptron weights will be tweaked on a large number of example data when compared to the scenarios with a lower dimension, which in turn helps the perceptron weights to converge approximately at the ideal decision boundary that perfectly separates the two types of data, thus contributing towards solving the perceptron storage problem. We gain valuable insight that the Rosenblatt perceptron algorithm will perform significantly better as the number of example data or dimensions increases.

4.7 Effects of inhomogenous data over the storage success rate

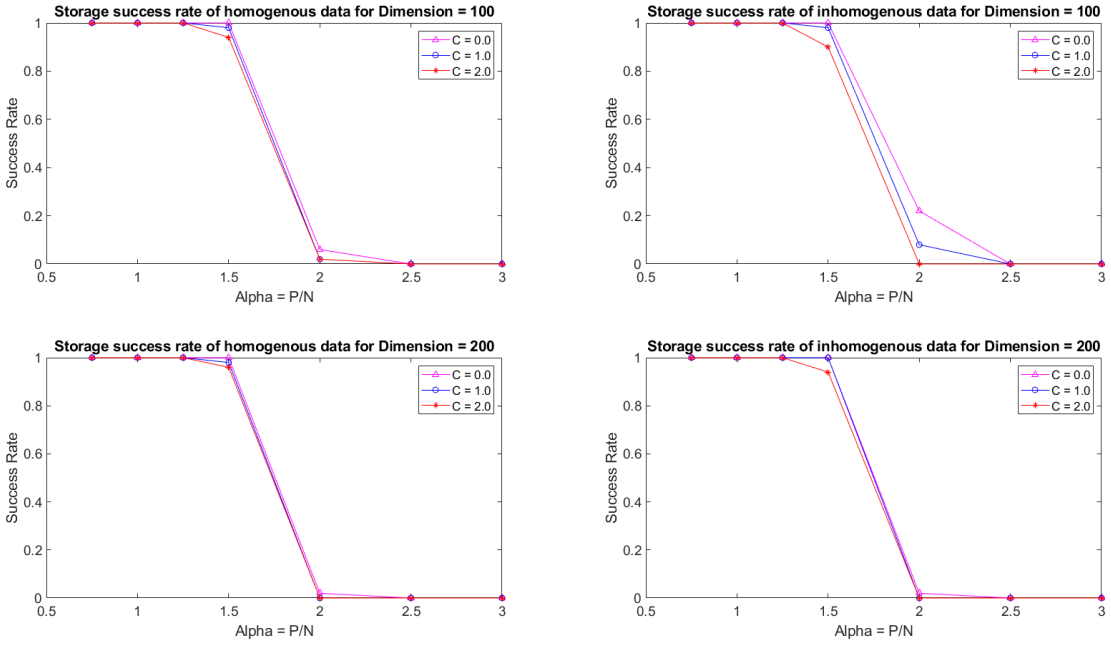

Figure 8: Storage success rate for homogenous and inhomogenous data

In this simulation we will be comparing the Rosenblatt perceptron by training them on homogenous data and inhomogenous data for different dimensions $N = 100, 200$ and thresholds $c = 0.0, 1.0, 2.0$ with a significantly large $t_{max} = 500$. To create inhomogenous example data, we generated a data $D \in R^N$ and added feature (-1) entry at N+1 dimension effectively creating the data $D' \in R^{N+1}$. From the figure 8 at C=2.0 and N=100, the inhomogenous data converged faster in comparison to the homogenous variant. Overall there is no significant difference between the convergence positions of the homogenous and inhomogenous data types.

5 Conclusion

In the assignment, we have tried to implement the Rosenblatt perceptron algorithm and simulated different scenarios to simulate and test our theoretical knowledge and hypothesis. In section 4.1 we observed the perceptron storage problem. In the following sections we tried to find the parameter(s) that minimize this error.
We observed that the epochs(t_{max}) does not play a significant role in minimizing the storage error or to maximize the probability of find the linear separation (section 4.5).
Simulations were performed on different values of threshold (c) in section 4.4, as expected the learning rate did not influence towards the probability of finding the solution, but a higher value of c was increasing the computation cycles required to achieve convergence. This is due to the linearity of the scalar products of weight vectors ($W_2 = \lambda W_1$), for a large threshold(c), we would converge at a larger W value (λW) even though a solution exists for the smaller and possible efficient weight vector 'W'.
In section 4.7 we trained the perceptron under the influence of homogenous example data and inhomogenous example data, where we found that the probability of finding the linear separation is slightly higher in inhomogenous data

when compared to homogenous data.
In section 4.2 and section 4.6 we simulated the perceptron purely based on the tweaking the dimensions of the input or example data. We even tested the scenarios for different learning rates or threshold(c). We gain some valuable insight that the probability of finding the linear separation to minimize the perceptron storage error is proportional to the dimension of the data, a higher dimension data results in more example data that in-turn trains the weight vectors of the perceptron to the ideal weights that perfectly separates the example data into two distinct classifications.

6 Individual contributions

Satyanarayan Nayak, Swastik (**S4151968**):

- Implementation of basic rosenblatt perceptron algorithm.
- Significance of threshold (c) and dimensions over storage success.
- Effects of inhmogenous data over the storage success rate.
- Realisation of the algorithm in embedding strengths.

Nivin, Pradeep Kumar (**S4035593**):

- Generation of the example data and labels
- Analysis of the perceptron storage success rate.
- Storage success rate by dimensions.
- Significance of epochs (t_{max}) over storage success.

We have worked on the report together, and have focused on the topics that we had worked on. But we have cross verified each others work and have refined this report together.

References

[1] *Learning from Examples - An Introduction to Neural Networks and Computational Intelligence.* 1994.

[2] J.-C. B. Loiseau, "Rosenblatt's perceptron, the first modern neural network." Accessed: 28-01-2020. Available: https://towardsdatascience.com/rosenblatts-perceptron-the-very-first-neural-network-37a3ec09038a.

7 Appendix

8 Readme

GitHub link : https://github.com/Swastik-RUG/NeuralNetworks/tree/master/matlab/perceptron

Simulation scenario	Matlab file to execute
Perceptron storage success rate	base.m
Storage success rate by dimensions	bonus1.m
Embedding strengths counts by dimensions	bonus2.m
Significance of threshold (c), epochs(t_max), dimensions over storage success rate	bonus3.m
Effects of inhomogenous data over the storage success rate	bonus4.m