Rosenblatt Perceptron

Neural Networks and Computational Intelligence - Practical Assignment I

Samuel Giacomelli
Student Number: S3546330
s.giacomelli@student.rug.nl

Davide Pedranz
Student Number: S3543757
d.pedranz@student.rug.nl

***Abstract*—Rosenblatt perceptrons are powerful devices able to learn a linear boundary in a binary classification problem. In this assignment, we implement and train and a Rosenblatt perceptron on random generated data. We use the results of multiple simulations to estimated the capacity of the hyperplane defined by the learned weights.**

I. Introduction

The Rosenblatt Perceptron [1] is a device designed to solve a binary classification problem, i.e. the problem of choosing the correct class for given examples, represented as feature vectors. The perceptron needs to be trained on some training examples in order to determine the best hyperplane to separate them: hopefully, the learned hyperplane will be able to correctly classify also new points.

In this work, we implement and train a Rosenblatt Perceptron and run some computer simulations to verify its theoretical properties. In particular, we try to estimate the capacity of the separation hyperplane learned by the perceptron.

II. Theory

The perceptron is the first mathematical model of a biological neuron. Like a neuron, it has N incoming connections, which represents the input, and 1 outgoing connection, which represents the output (i.e. the class of the given example). The output S is a function of the input ξ, defined as the sign of the weighted sum of the input vector:

$$S = sign(\mathsf{w} \cdot \xi) = \pm 1, \tag{1}$$

where w is an N-dimensional vector. The $sign$ function simulates the activation function of the biological neuron and forces the possible output values to ± 1.

The training process aims at learning the optimal weights vector w to classify correctly all examples. At time t, an example ξ^t is presented to the perceptron: if the output equals to the correct class S^t, w is not changed; if the output is wrong, a learning step is performed to correct w. The procedure is repeated until the perceptron correctly classifies all the training dataset or until some maximum number of iterations is reached.

The Rosenblatt Perceptron training procedure is formally defined as:

$$\mathsf{w}(t+1) = \mathsf{w}(t) + \frac{1}{N}\Theta\left[c - E^{v(t)}\right]\xi^{v(t)}S^{v(t)}, \tag{2}$$

$$\Theta[c - E^{\mu}] = \begin{cases} 1 & E^{\mu} \leq c \\ 0 & E^{\mu} > c \end{cases}, \tag{3}$$

where $E^{v(t)} = \mathsf{w}(t) \cdot \xi^{v(t)}S^{v(t)}$. The weights are initialized as $\mathsf{w}(0) = 0$. The product $\xi^{v(t)}S^{v(t)}$ is called Hebbian term.

The performances of the perceptron are related to the storage capacity of the hyperplane defined by its weights. For a random dataset with P examples and N dimensions (where $S^{\mu} = 1$ with probability $1/2$), the theoretical probability of linear separability $P_{l.s.}(P, N)$ is given by:

$$P_{l.s.}(P,N) = \begin{cases} 1 & \text{for } P \leq N \\ 2^{1-P}\sum_{i=0}^{N-1}\binom{P-1}{i} & \text{for } P > N \end{cases} \tag{4}$$

For $N \to \infty$, $P_{l.s.}(P, N)$ has a well-defined behavior:

$$P_{l.s.}(P,N) = \begin{cases} 1 & \text{for } \alpha \leq 2 \\ 0 & \text{for } \alpha > 2 \end{cases}, \tag{5}$$

where $\alpha = P/N$.

III. Implementation

A. Dataset Generation

The first step of the experiment is to generate an artificial dataset with P random N-dimensional feature vectors and binary labels $\mathcal{D} = \{\xi^{\mu}, S^{\mu} \,|\, \xi^{\mu} \in \mathbb{R}^N, S^{\mu} \in \{+1, -1\}\}_{\mu=1}^{P}$. The feature vectors ξ^{μ} have independent random components $\xi_j^{\mu} \sim \mathcal{N}(0, 1)$ and are generated using the `randn` command in Matlab. The labels S^{μ} are assuming the values $\{+1, -1\}$ with equal probability of $1/2$.

```
function [X, y] = generate_dataset(P, N)
    X = randn(P, N);
    y = iff(rand(P, 1) < 0.5, -1, 1);
end
```

B. Perceptron Training

The next step is to use the Rosenblatt algorithm to implement and train a perceptron on the generated dataset. The Rosenblatt algorithm is a sequential procedure that learns the weights of the perceptron from the examples. The weights are initialized to zero, then the training examples are presented one by one to the perceptron for a given number of iterations. If the example is correctly classified according to the current weights, no update is performed; if the example is wrongly classified, the weights are updated according to the following rule:

$$\mathsf{w}(0) = 0, \tag{6}$$

$$\mathsf{w}(t+1) = \begin{cases} \mathsf{w}(t) + \frac{1}{N}\xi^{\mu(t)}S^{\mu(t)} & \text{if } E^{\mu(t)} \leq 0 \\ \mathsf{w}(t) & \text{else} \end{cases}, \tag{7}$$

where $E^{\mu(t)} = \mathsf{w}(t) \cdot \xi^{\mu(t)} S^{\mu(t)}$, $t = 1, 2, ...$ represents the current time step and $\mu(t) = 1, 2, ..., P, 1, 2, ...$ identifies the current example. The algorithm is stopped either when $E^{\mu} > 0$ for all examples in the dataset or the maximum number of iterations n_{max} is reached.

C. Experiments

For a fixed value of P and N, n_D independent datasets are generated. A new perceptron is trained on each dataset for at most n_{max} iterations. For each dataset, we compute the rate fraction $Q_{l.s.}$ of successful runs, where a run is successful if the perceptron correctly classifies each example in the dataset at the end of the training phase. $Q_{l.s.}$ gives an estimate of the probability that the perceptron finds a linear separation in a random dataset of P independent points in N dimension.

Multiple experiments are run for different values of P and N in order to compute $Q_{l.s.}$ as a function of $\alpha = P/N$. In other words, we study the probability of the perceptron to find a linear separation as a function of the rate between the number of examples and the dimension of the input.

D. Bonus

1) Weight Update Criterion: We modified the function to train the perceptron to update the weights vector w if $E^{\mu} < c$ instead of $E^{\mu} < 0$ (see Equation (7)). Also, we changed the algorithm to stop the training only if $E^{\mu} > c$ for all examples in the dataset or the maximum number of iterations n_{max} is reached.

2) Inhomogeneous Perceptrons: Rosenblatt perceptrons only learns separation hyperplanes that goes through the origin (homogeneous). In general, a solution to the classification problem may be a hyperplane that does not go through the origin (inhomogeneous), but be in the form:

$$S = sign(\mathsf{w} \cdot \xi + \theta) = \pm 1 \tag{8}$$

It is possible to generalize the Rosenblatt perceptron to learn also inhomogeneous separation hyperplanes by adding an artificial dimension to the dataset and forcing it to a non-zero constant (e.g. -1) and increasing the size of the weights vector $\mathbf{w}$ by 1 (the last element of $\mathbf{w}$ corresponds to the intercept θ).

IV. Evaluation

If not differently specified, all experiments discussed in this section are run with the following parameters: $N = 500$, $n_{max} = 200$, $n_D = 100$ and $c = 0$.

A. Storage Capacity

Figure 1 shows the results of the base experiment. The x-axis represent different values of $\alpha = P/N$, while the y-axis the success rate $Q_{l.s.}$. As expected, the function looks like a step function from 1 to 0. For $\alpha \approx 1.7$, the success rate $Q_{l.s.}$ drops from 1 to 0 very quickly.

The value of α for which the function drops is called storage capacity of the perceptron. For $N \to \infty$ (very large number of examples) and $n_{max} \to \infty$ (no limit on the maximum number of training iterations), the theoretical storage capacity of the Rosenblatt perceptron is $\alpha = 2$.

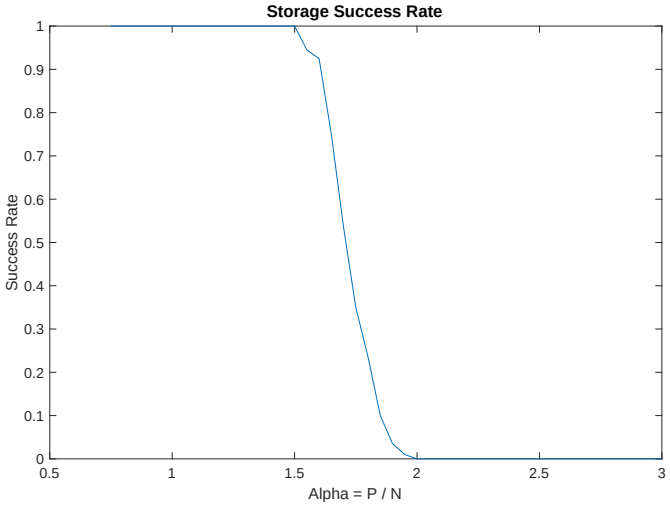

Figure 1. Storage success rate of a Rosenblatt perceptron as a function of $\alpha = P/N$. The experiments use $N = 500$, $n_{max} = 200$ and $n_D = 100$.

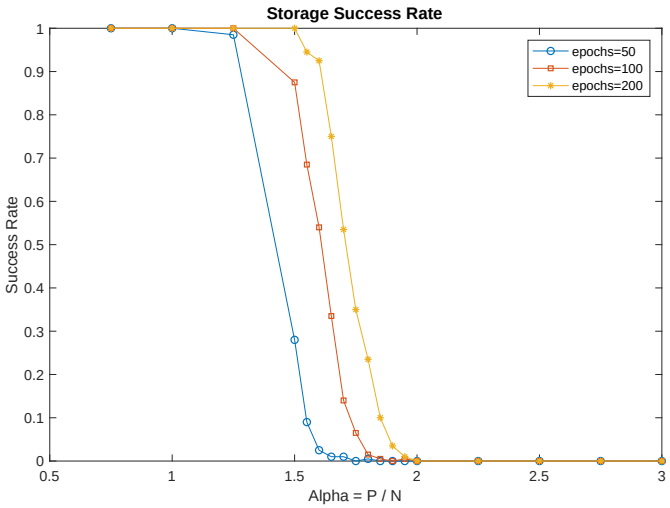

Figure 2. Storage success rate of a Rosenblatt perceptron as a function of $\alpha = P/N$ for different values of n_{max}.

B. Number of Iterations

The difference between the theoretical value and the experimental one are mainly due to the limited number of training iterations. Figure 2 gives an experimental proof of this statement: for a very small number of iterations (eg. $n_{max} = 10$), the step is close to $\alpha = 1$, while for higher values of iterations the step moves closer and closer to the theoretical value $\alpha = 2$ found with Equation (5). The theoretical result still remains far from the practical one because of the limited size of input's dimension N.

C. Number of Dimensions

The theoretical results are valid for $N \to \infty$. However, real datasets have a limited number of features. Figure 3 shows the behaviour of the perceptron for different values of N. For high values of N, the shape of the success rate $Q_{l.s.}$ as a function

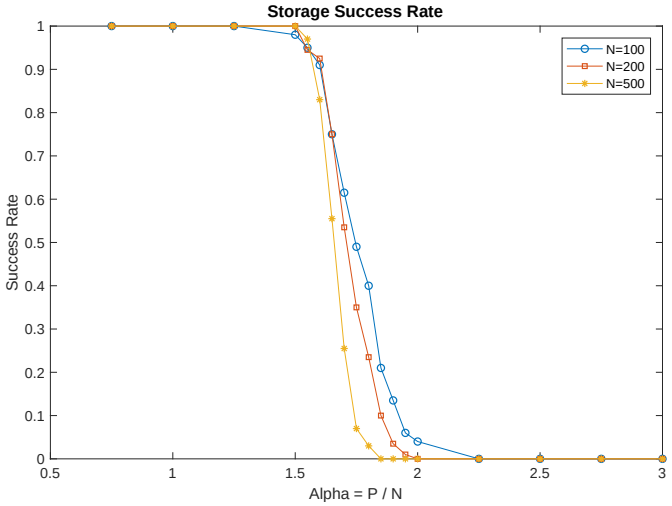

Figure 3. Storage success rate of a Rosenblatt perceptron as a function of $\alpha = P/N$ for different values of N.

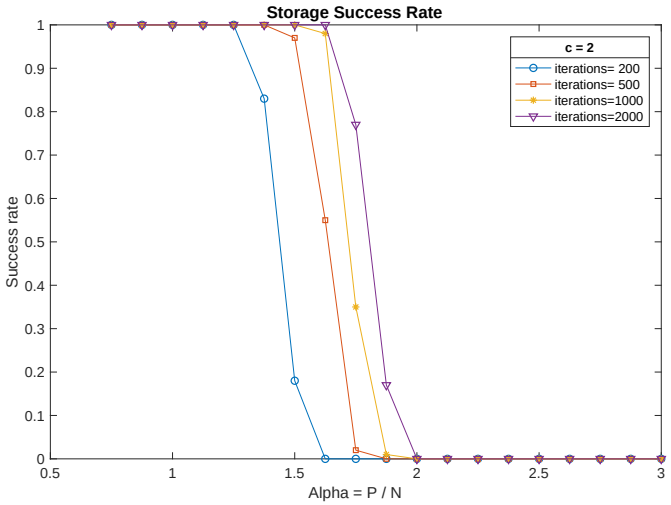

Figure 5. Storage success rate of a Rosenblatt perceptron as a function of $\alpha = P/N$ for different numbers of iterations with fixed value of $c = 2$.

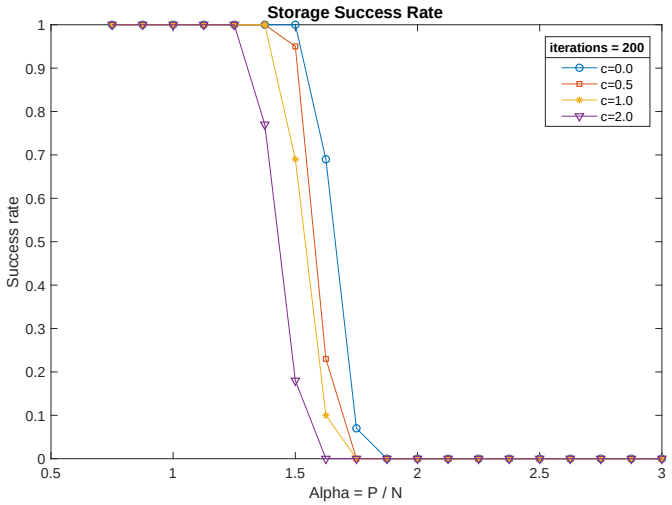

Figure 4. Storage success rate of a Rosenblatt perceptron as a function of $\alpha = P/N$ for different values of c.

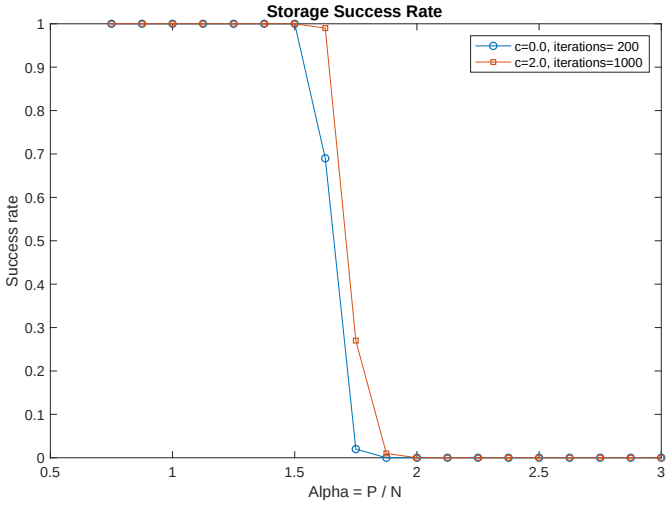

Figure 6. Storage success rate of a Rosenblatt perceptron as a function of $\alpha = P/N$ for different numbers of iterations and c.

of α is similar to a step function. For small values of N, the function looks like a smoothed step function: the smaller N is, the higher is the smoothing.

D. Weight Update Criterion

Figure 4 shows the effect of changing the values of c in the training procedure of the perceptron. For higher values of c, the curve is shifted to the left. Since an example ξ^μ is considered correctly classified only when its local potential is greater than c ($E = \mathsf{w} \cdot \xi^\mu S^\mu > c$), the potential only depends on w for a fixed ξ^μ and the update of w is fixed for a given wrong classified example, the perceptron will need a higher number of updates to increase the norm of w and make the local potentials higher than a threshold $c > 0$. Formally, we can show that the value of $c > 0$ is irrelevant (provided the perceptron is trained long enough):

$$\mathsf{w}_1 : \{E_1^\mu \geq c\}_{\mu=1}^P \Leftrightarrow \mathsf{w}_2 = \lambda \mathsf{w}_1 : \{E_2^\mu \geq c\}_{\mu=1}^P, \lambda > 0$$

To give an empirical proof of this, we fix the value of c and train the perceptron for different number of iterations n_{max}. We expect to see the curve shifted to the left for small n_{max}, and approximate a step function centered in $\alpha = 2$ for big n_{max}. Figure 5 shows that the results of the experiment confirm our hypothesis.

Figure 6 compares the curves for $c = 0$ and $c = 2.0$ for different values of n_{max}: by increasing n_{max}, the curve is "pushed" towards the right, contrasting the effect of the increased value of c. We can somehow see c as a simple version of the learning rate that is used in more complex neural networks: it "regulates" the speed of the training.

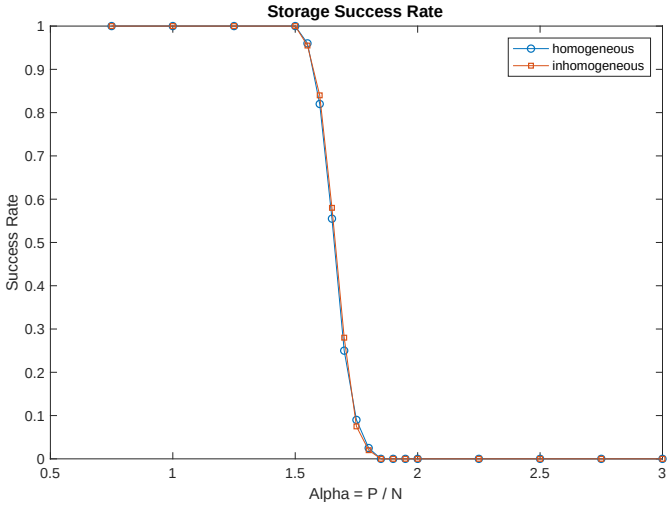

Figure 7. Storage success rate of a Rosenblatt perceptron and its inhomogeneous version for $N = 500$ as a function of $\alpha = P/N$.

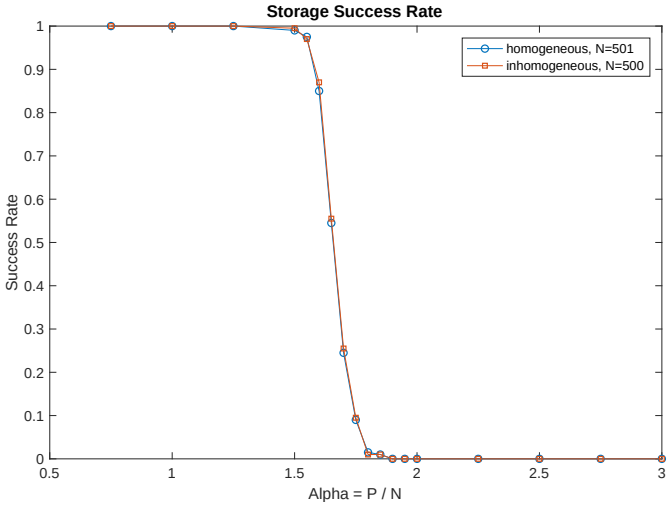

Figure 8. Storage success rate of a homogeneous Rosenblatt perceptron for $N = 501$ and its inhomogeneous version for $N = 500$ as a function of $\alpha = P/N$.

E. Inhomogeneous Hyperplanes

We run an experiment to verify the behaviour of $Q_{l.s.}$ by allowing inhomogeneous hyperplanes: we train both a normal and modified perceptron for $N = 500$. Figure 7 shows the results of the experiment. As expected, the success rate $Q_{l.s.}$ of the inhomogeneous perceptron is slightly higher than homogeneous one. However, the difference is not significant, since data points follow a normal distribution $\xi_j^\mu \sim \mathcal{N}(0,\, 1)$ and are therefore distributed around the origin.

The problem of finding an inhomogeneously solution in R^N can be solved by finding a homogeneously solution in R^{N+1}. In a second experiment, we compare the success rate $Q_{l.s.}$ of an inhomogeneous perceptron for $N = 500$ with an homogeneous perceptron for $N = 501$. Figure 8 shows the result of this experiment. As expected, the success rates for the 2 perceptrons are very close to each other.

V. Conclusion

In this assignment we implemented and trained a Rosenblatt perceptron on a randomly generated datasets. In our experiments, we estimated the probability of success of the perceptron to find a linear separation in a set of random datapoints. For a limited number of training iterations n_{max} and features N, we obtained that the capacity of the Rosenblatt perceptron is about $\alpha = 1.75$.

The difference from the theoretical results seems to be mainly caused by the limited number of training iterations: by allowing longer trainings, i.e. greater number of epochs, the plot of the success rate $Q_{l.s.}$ as a function of $alpha$ moves to the right and the capacity tends to the theoretical result $\alpha = 2$.

The parameter N influences the shape of the function: for small values of N, the function is smooth in the regions before and after the step; for higher values of N, the function gets closer to the theoretical step function.

The parameter c in the training algorithm does not influence the performances of the perceptron: indeed, it only changes the magnitude of the weights w learned during the training. However, since each training step in the algorithm can change the weights by at most $\frac{1}{N}\xi^{\mu(t)}S^{\mu(t)}$ (where $\mu(t)$ denotes the current training example), the algorithm will take a longer time to converge for higher values of c.

Rosenblatt perceptrons only learns separation hyperplanes that goes through the origin. In general, it may happen that the hyperplane that linearly separates the dataset does not pass through the origin. It is possible to generalize the Rosenblatt perceptron to learn also inhomogeneous separation hyperplanes by adding an artificial dimension to the dataset and forcing it to a non-zero constant. The success rate $Q_{l.s.}$ of inhomogeneous perceptrons seems to be slightly higher than homogeneous ones, even for random datasets distributed around the origin of the axes.

VI. Individual Workload

The workload of this assignment was divided as follows.
Davide Pedranz:

- Generation of the datasets.
- Implementation of the basic Rosenblatt algorithm.
- Experiments for different number of iterations and different values of N.

Samuel Giacomelli:

- Extension of Rosenblatt perceptron training algorithm for different values of c and inhomogeneous hyperplanes.
- Experiments for different values of c and inhomogeneous hyperplanes.

We worked together on the report: each of us focused more about the part that he implemented, but then we improved and refined the whole report together.

References

[1] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.