Neural Networks and Computational Intelligence

Assignment I: Perceptron Training

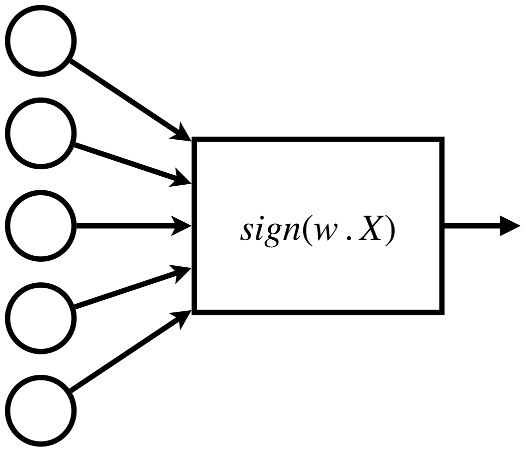
Group 12

Armand Colin
S4994582

Andrea Gasparini
S4993411

A.Y. 2021-2022

GitHub repository:
https://github.com/armand-colin/neural-networks-g12/tree/main/assignment1



Contents

1 Introduction

The Rosenblatt's Perceptron is a machine learning classifier. Given a linearly separable dichotomy - a data set which labels $Y \in \{-1, 1\}$, it is bounded to find a rule which will correctly classify all the data, after a finite number of steps.

Given P examples in a N-dimensional input space, we can write $C(P, N)$ as the capacity of a hyperplane of dimension N, i.e. the number of linearly separable dichotomies. The probability of a dichotomy with randomly assigned labels to be linearly separable is related to $C(P, N)$, and is expressed as $P_{l.s.}$:

$$P_{l.s.} = \frac{C(P, N)}{2^P}$$

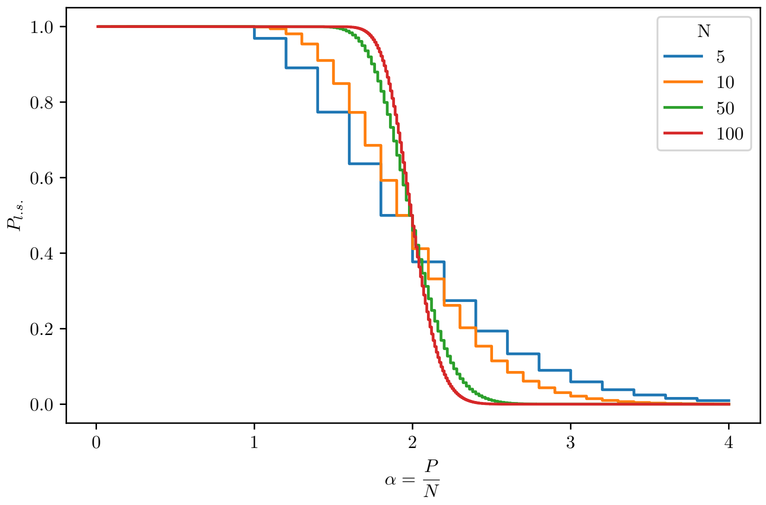


Figure 1: Plotting of $P_{l.s.}$ for various values of N

The goal of this assignment is to implement the Rosenblatt's algorithm, apply it to randomized data and to compare it's performances to the theoretical $P_{l.s.}$. Additionally, we will observe the impact of some parameters of the data and of the perceptron on the learning process.

2 Method

2.1 The Rosenblatt's algorithm

Suppose to have a data set $\mathbb{D} = \{X^\mu, Y^\mu\}_{\mu=1}^{P}$, with P N-dimensional feature vectors $X^\mu \in \mathbb{R}^N$ and labels $Y^\mu \in \{-1, 1\}$. If the data set is linearly separable, the Rosenblatt's algorithm will find a linearly separable rule, characterized by defining a hyperplane with normal $w \in \mathbb{R}^N$ such as:

$$sign(X^\mu \cdot w) = Y^\mu, \text{ for every } \mu \text{ of } P$$

To find w, or the so-called weight vector of the perceptron, the algorithm follows an iterative process:

- Initialize $w = \vec{0}$
  - We go through μ from 1 to P
  - For the current example μ, we compute E^μ, the local potential of the example as: $E^\mu = w \cdot X^\mu Y^\mu$
  - If $E^\mu > c$ with a given parameter $c \geq 0$, we can assume the perceptron successfully classified the example μ, and we proceed with the next example.
  - Otherwise, we update the weight vector as: $w = w + \frac{1}{N} X^\mu Y^\mu$

- Going through all P examples is called an epoch. If the algorithm does not update the weight vector for a given epoch, it stops here, considering that it successfully classified all the examples. Otherwise, the algorithm performs up to n_{max} epochs, at the end of which it would result in failing to find a separating hyperplane to the data set.

Note that the final weight vector can be re-written as:

$$w = \frac{1}{N} \sum_{\mu=1}^{P} x^{\mu} X^{\mu} Y^{\mu}$$

Where x^{μ} refers to the embedding strengths of the perceptron, that quantify the specific contribution of an example μ in the weight vector. Due to the characteristics of the Rosenblatt's algorithm, we could also write the iterative update of the embedding strengths as:

$$x^{\mu} = x^{\mu} + 1$$

If the current example μ is still misclassified.

2.2 Generating the data

To run the experiments, we need data to train our perceptron with, which we generated as follows:

- The labels Y are randomly affected to -1, 1 using a uniform distribution
- The feature vectors X are generated according to a normal distribution, with a given mean and variance. By default, we use mean = 0 and variance = 1. An example of a 2-dimensional data set is shown in Figure 2. A condition for the features to match the hyperplane's capacity's definition, is that none of the feature vectors are colinear. This is prevented by the random algorithm implemented.

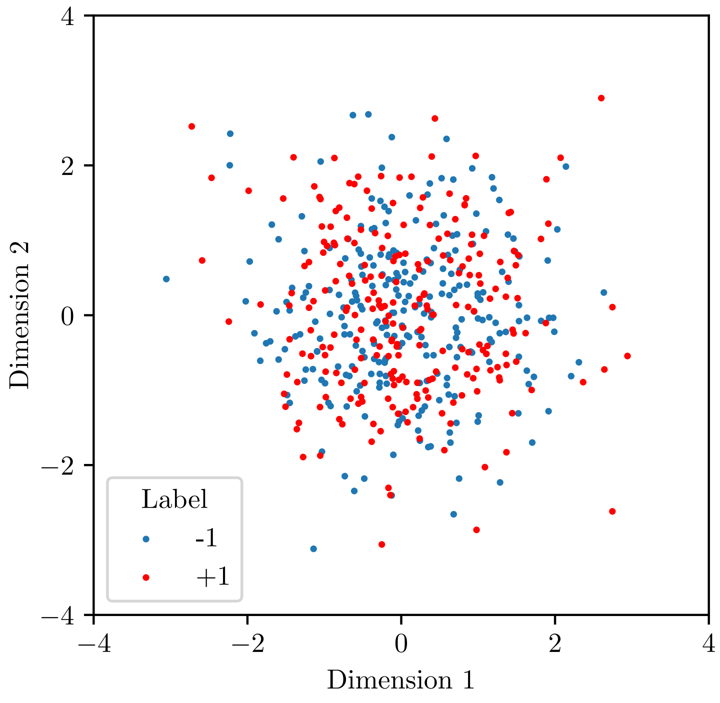


Figure 2: Generation of 500 feature vectors of dim. 2 (mean=0, variance=1)

3 Experimental setup

3.1 Parameters and metrics

In this section we define the used parameters or notations in the following experiments.

- N represents the dimension of the feature vectors.
- P represents the the size of the data set, i.e. the number of pairs of feature - label.
- α represents the relation between N and P, according to $P = \alpha N$.
- n_{max} represents the maximum count of epochs that the perceptron is allowed to perform during the training phase.
- n_D represents the number of perceptrons trained for a given N and P.
- μ refers to the index of an example in the data set. So for a data set of size P, we will have $\mu \in \{1, 2, ..., P\}$.
- $Q_{l.s.}$, or the success rate, is the ratio of perceptrons that successfully classified the data that was given to it. We consider that a perceptron successfully classifies a data set if **all** the predictions are correct.

3.2 Perceptron's success rate against input size

One of the main measure that we are going to perform is the success rate, namely $Q_{l.s.}$, of the perceptron. This measure should theoretically approach $P_{l.s.}$, that was shown in Figure 1. During this measurement, all the other parameters remain the same.

We are going to take values of α from 0.75 to 3.0, with a step $\Delta_\alpha = 0.1$, and increasing values of N.

3.3 Effect of N

In a second phase, we are going to observe the effect of N on the success rate. This experiment will be similar to the previous one, but will be ran for more values of N, from 5 to 100, and for a smaller α range of [1.5, 2.5], with a step $\Delta_\alpha = 0.1$.

3.4 Embedding strengths

We will compute the embedding strengths for big and small values of α, and see if there is any difference in the embedding strengths when the success rate is different.

3.5 Effect of the parameter c

In the perceptron's training algorithm, we previously defined the success condition as $E > c$. By default, we set $c = 0$, but different values will result in the perceptron strengthening its weights for the given features.

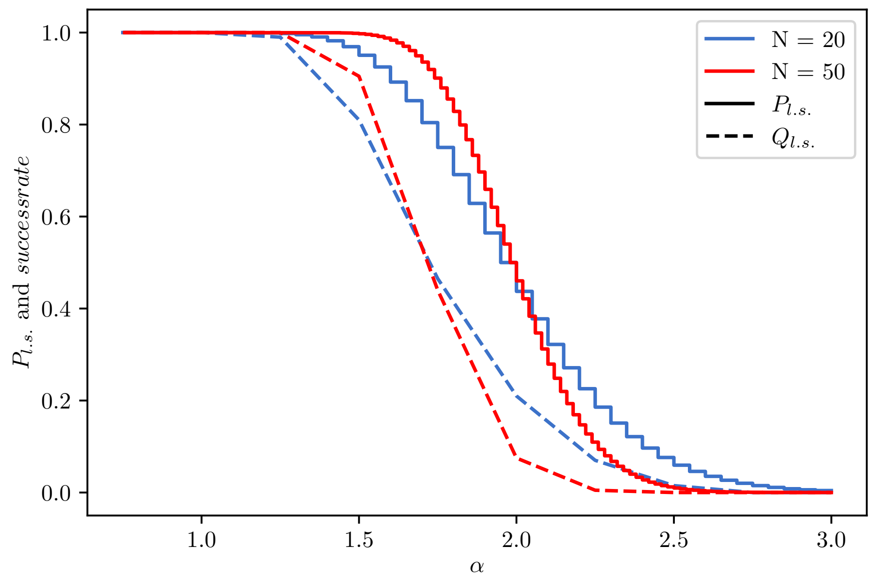
From the theory we know that the actual value of c is essentially irrelevant. For instance, considering two weight vectors $w1$ and $w2 = 1$ with $\lambda > 0$, due to the linearity of the scalar product

$$E_1^\mu = w1 \cdot \xi^\mu S^\mu \geq c > 0 \quad \text{implies} \quad E_2^\mu = w2 \cdot \xi^\mu S^\mu \geq \lambda c > 0$$

Therefore, a solution for any positive constant c can be constructed. Nonetheless, we intend to observe the eventual effect of this parameter on the success rate as it becomes bigger.

4 Results

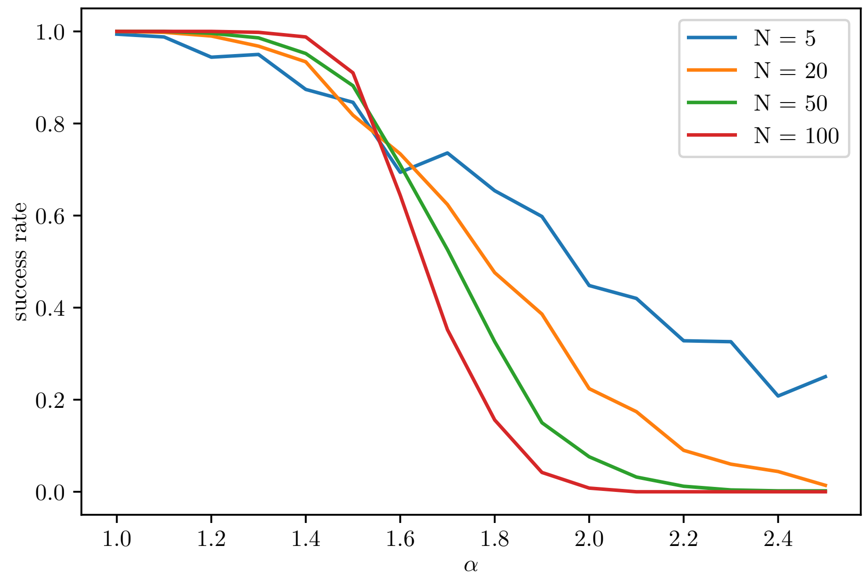
4.1 Perceptron's success rate against input size



Parameters		
n_{max}	n_D	Δ_α
100	200	0.25

Figure 3: Perceptron's $Q_{l.s.}$ (dashed lines) and $P_{l.s.}$ (solid lines) against α.

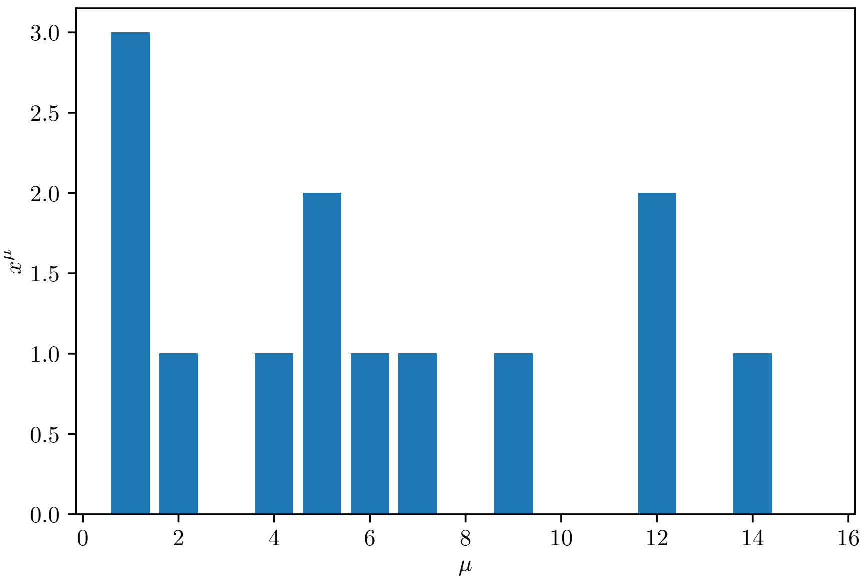
4.2 Effect of N



Parameters		
n_{max}	n_D	Δ_α
100	500	0.1

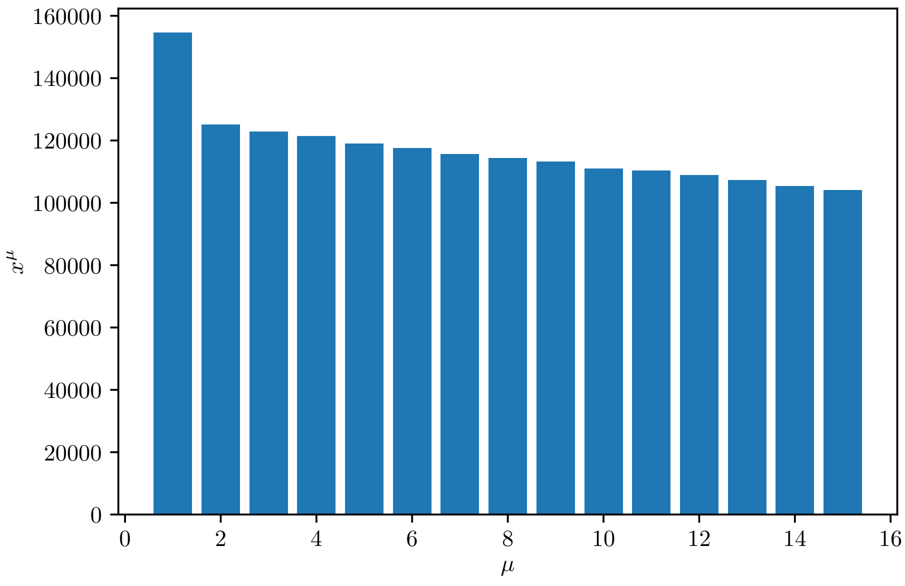
Figure 4: Perceptron's success rate against α for various N

4.3 Embedding strengths



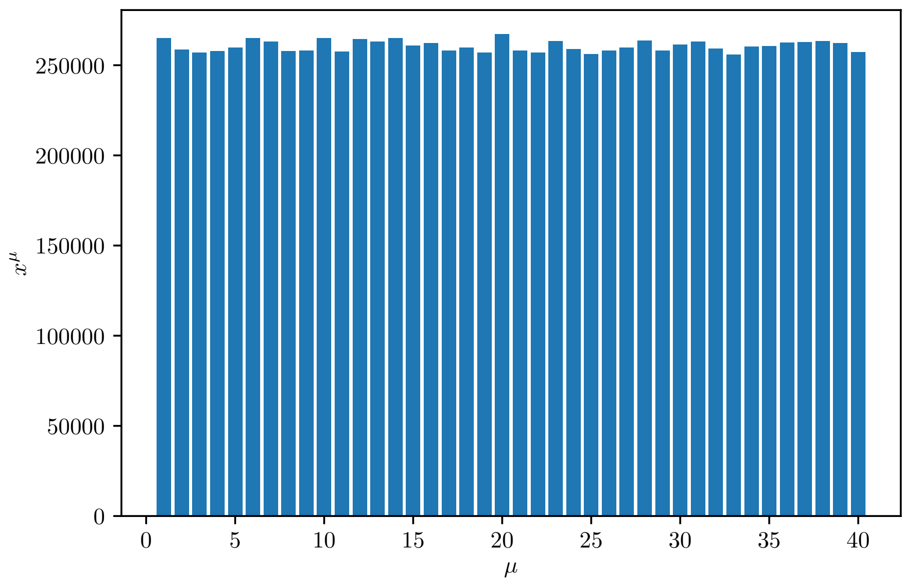
Parameters			
N	α	n_{max}	n_D
20	0.75	100	1

Figure 5: Embedding strengths for one perceptron



Parameters			
N	α	n_{max}	n_D
20	0.75	100	100000

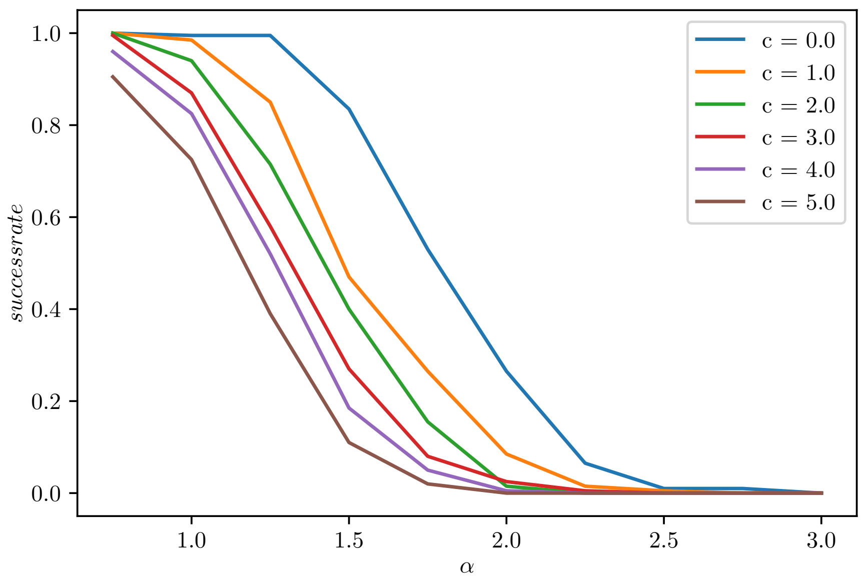
Figure 6: Embedding strengths with small α



Parameters			
N	α	n_{max}	n_D
20	2.0	100	10000

Figure 7: Embedding strengths with great α

4.4 Evaluation function with additional c parameter



Parameters			
N	n_{max}	n_D	Δ_{α}
20	100	200	0.25

Figure 8: Success rate for generated feature vectors with varying c values

5 Discussion

5.1 Perceptron's success rate in standard cases

We've observed that the perceptron's success rate $Q_{l.s.}$ follows the $P_{l.s}$ discussed in class, as we can see in Figure 3. The curves seem to have the same qualitative results:

- For $P = 0$, we can see that both $Q_{l.s.}$ and $P_{l.s.}$ tend to 1
- For $P \to +\infty$, the curves tend to 0
- The bigger N, the closer the curves approach a step function; clearer to see in Figure 4

Although, quantitatively, we can see that the crossing point of the different N curves is slightly shifted from the $P_{l.s}$. Moreover, it seems that the success rate is worse than in the theoretical model, meaning that the step function tilts before $\alpha = \dfrac{P}{N} = 2$.

We initially thought this behaviour could be explained by the nature of our data, with the features distributed according to a normal distribution, therefore we tried different ones to confirm our hypothesis. For instance, we tried with a uniform distribution, still respecting mean 0 and variance 1, that however yielded the same result.

An other explanation could be the training of the perceptron. It is possible that we are not training enough. We used an n_{max} of 100, but with bigger values, it is possible that our perceptron finds a way to classify the data, thus increasing the success rate.

5.2 Embedding strengths

The embedding strengths have two different types of shape:

- When alpha is low, we assume that some perceptrons will find a way to classify the data. This way, the first feature will have a big effect, because without any training the perceptron cannot fit the first feature vector. Then, every feature vector μ has less chance to affect the weights, because the probability of having a good vector of weights will grow. We can see the effect on Figure 6.
- When alpha is big, perceptrons will unlikely find a solution, and thus keep training until the end. This way, every μ will have the same effect at the end, because every feature vector will affect the weights at every almost every step. We can see this effect on Figure 7

5.3 Evaluation function with additional c parameter

Taking into account different non-zero values of the c parameter, as shown in Figure 8, we observe that the more we increase this threshold, the worse the success rate seems to get. The shape of the curves are similar between each other, making so that the parameter almost behave as a shifting factor, without any other huge influence. It is also interesting to notice that the main difference we have when we increase the c parameter is from $c = 0.0$ to $c = 1.0$, while the following ones all have smaller differences between each other.

As expected, since the maximum epochs n_{max} and training runs n_D are constant during the different experiments, increasing c means having a worse success rate. This is due to the fact that we can interpret c as a minimum threshold for the precision of the predictions. The bigger c is, the more the predictions should be precise, and consequently we would need more time in order to obtain the same results. Thereby this experiment confirmed our expectations.

6 Conclusion

In this assignment, we managed to implement a simple perceptron algorithm, that tries to find an N-dimensional hyperplane to separate a set of features related to binary labels {-1, 1}. This algorithm has been faced with randomly generated data, to analyse its success with different sets of data - and different distributions of the features.

We have observed that the success rate of the perceptron algorithm is similar to the $P_{l.s}$ studied in class, although it is not as good as the theoretical model. We provided some possible explanations to this phenomenon, and it would be interesting to upgrade the model according to the improvements we suggested previously.

Finally, we have observed the influence of the data distribution and the perceptron's acceptance threshold - the c parameter - on the success rate. What we can retain is that the homogeneity and the acceptance threshold can have a very big effect on the success rate. Inhomogeneous data brings to a worse success rate, because of the random nature of the data. The acceptance threshold also forces the perceptron to find a perfect fit, but since the data is randomly generated, it will try to over-fit to some samples.

Note on individual workload Since we worked on a shared repository on GitHub and the majority of the time using a VoIP software, our individual workload is approximately equal.