University of Groningen

Faculty of Mathematics and Natural Sciences
Department of Computing Science

Neural Networks and Computational Intelligence

Assignment I

Authors

Name	Student Number	E-mail
Sebastian Wehkamp	S2589907	s.wehkamp@student.rug.nl
Dimitris Laskaratos	S3463702	d.laskaratos@student.rug.nl

February 2, 2018

university of groningen / faculty of science and engineering

Contents

1 Problem Statement

The task for this assignment is to implement Rosenblatt's perceptron algorithm and apply it on a set of randomly generated data. First we must generate a N-dimensional dataset $\mathbb{D} = \{\xi^\mu, S^\mu\}_{\mu=1}^P$. On this randomly generated dataset we should run a Rosenblatt perceptron.

2 Solution

2.1 Step 1

In the first step we create a dataset $D = \{\xi^\mu, S^\mu\}_{\mu=1}^P$, where $\xi^\mu \in \mathcal{R}^N$ are vectors of N randomly selected independent components ξ_j^μ with mean 0 and variance 1, and S^μ is a label with value 1 or -1, also randomly selected with probability 1/2. The random generation of the vector components is done using the *randn()* function of MATLAB. The code shown in Listing 1 implements this.

```
% Function which generates vectors and labels
function [vectors, labels] = generateVector(dim, num)
    % Define possible labels
    possibleLabels = [-1, 1];
    % Initialize vectors randomly
    vectors = randn(dim,num);
    % Initialize all labels selected from possibleLabels
    labels = possibleLabels(randi(2,[1,num]));
end
```

Listing 1: Function which generates vectors and labels.

2.2 Step 2

After creating P vectors and an equal number of labels, the next step is to train the perceptron with the given dataset D. With an arbitrary number *epochs* and a weight vector w, we iteratively compute the error $E^{\mu(t)} = w(t) \cdot \xi^{\mu(t)} S^{\mu(t)}$, where t=1,2....*epochs*. The iteration ends when all errors>0 or t reaches *epochs*. In each iteration we set the value of $w(t+1) = w(t) + \frac{1}{N} \xi^{\mu(t)} S^{\mu(t)}$ if $E^\mu(t) \leq 0$, or $w(t+1) = w(t)$ otherwise.
Then we iterate again through the vectors to ascertain the number of correct labels, meaning error>0. If the number of correct labels matches the number of vectors, we break the loop because the algorithm is successful. An illustration of the above can be seen in the pseudo-code shown in Listing 2.

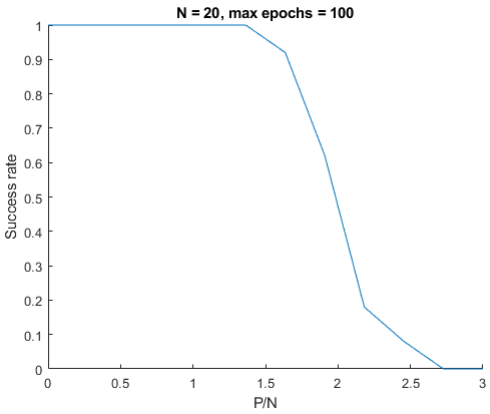
```
Create empty array "weights" with 1 column and N rows;

for every epoch do:
    for every point P
        Calculate Error
        if error<=0:
           Adjust weight;
        end
    end
    if every label is correct;
        break loop;
    end
end
```

Listing 2: Perceptron pseudo code

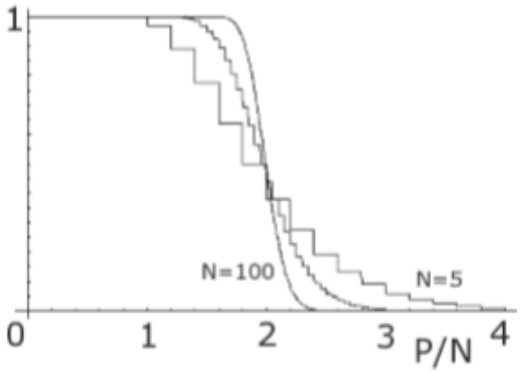
Next, we run the above 50 times to test it for $P = \alpha N$ randomly generated data, where N=20. We used the MATLAB method *drange()* for distributed (parallel) execution, since each iteration is independent of the previous. Lastly, we determine the fraction of successful runs for each $\alpha = \frac{P}{N}$ where $\alpha = 0, 0.25, 0.75....3$.

3 Evaluation

The first thing we tested was using the recommended parameters of $N = 20$, a maximum α of 3, and 100 max epoch set to 100. The resulting plot can be found in Figure 1a. We expected the plot to look like Figure 1b and our plot is very similar to the plot presented in the lectures. As expected, the accuracy drops dramatically for every N as α approaches 2.



(a) Plot of success rate with respect to α.



(b) Plot from slides

Figure 1: Image comparison between our accuracy plot versus the reference plot from the slides

When we increase N to 200 we expect a steeper curve as can be seen in Figure 1b. Our result can be seen in Figure 2 and does indeed contain a steeper curve than the image shown in Figure 1a.

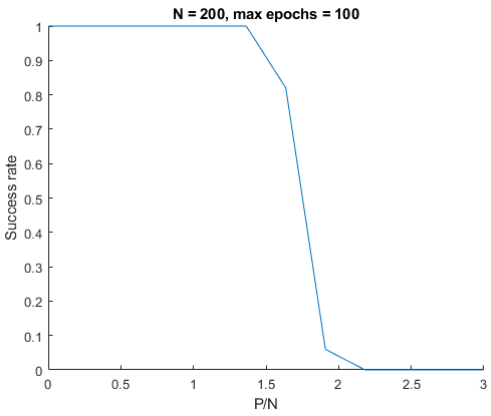


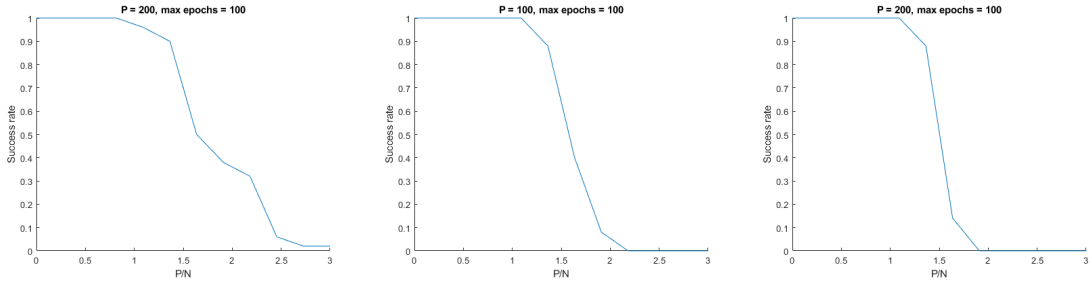
Figure 2: Plot of success rate with respect to α using $N = 200$.

Theoretically the perceptron can store

$$P(\alpha) = \begin{cases} 1 \text{ for } \alpha \leq 2 \\ 0 \text{ for } \alpha > 2 \end{cases}$$

as N grows larger. This means that there is a point $\alpha = 2$ after which the perceptron is not able to linearly separate the data. In our results, as seen in Figure 2, we can see that our perceptron's performance starts to go down at about $\alpha = 1.5$ and is almost zero when $\alpha = 2$. This is due to $\alpha = 2$ being the theoretical limit of the perceptron which will be approached when $N \to \infty$.

We also performed some experiments by varying the P instead of N. Beforehand we expect that we get about the same results as we got from varying N since the fraction α is relevant. Our results can be found in Figure 3. As you can see the curves are very similar to the ones we obtained by varying N with the same drop off point of $\alpha = 1.5$.



(a) $P = 20$ (b) $P = 100$ (c) $P = 200$

Figure 3: Results obtained by varying P.