The Rosenblatt Perceptron
Neural Networks and Computational Intelligence
Practical 1

Rogier de Weerd
Tim Oosterhuis

December 2017

1 Introduction

For our first neural networks and computational intelligence assignment, we were asked to implement the Rosenblatt perceptron. The Rosenblatt perceptron is a two-class classification algorithm that is bound to find a separating hyper-plane in a finite amount of steps for any N-dimensional labeled data set of P examples, given that this data set is linearly separable. The architecture of the Rosenblatt perceptron is that of a simple feed forward network, with N-dimensional inputs, an N-dimensional vector of adaptive weights, and a single threshold output. In the training phase, the weights of the network are only adapted if the output doesn't correspond to the label of an input example. That is to say, it learns from mistakes. Training with the Rosenblatt perceptron occurs sequentially for all examples in a given order, and then repeats iteratively until no more mistakes are made for any example, after which the solution is found.

Well written but citations missing

2 Implementation

2.1 a: generating random data

We created a function data_matrix(P,N), which creates an artificial N-dimensional labeled data set of P examples. The data is drawn from a normal distribution with a mean of 0 and a standard deviation of 1. The labels are generated by drawing from the same normal distribution, and taking the sign. This ensures that the labels are randomly and (on average) evenly distributed between -1 and 1. The source code for the data matrix function can be found in the listing below:

```
function [ data_matrix, label_vector ] = data_matrix( P, N )
%Function to create artificial dataset
```

```
%input args:
%P, nr of data points in the dataset
%N, nr of dimensions of the data
%output args:
%data_matrix, a P by N datamatrix
%label_vector, a corresponding labelvector of length P
    data_matrix = randn(P,N); % data creation
    label_vector = sign(randn(P,1)); % label creation, labels are -1 or 1
    label_vector(label_vector==0)=1; % 0 labels are converted to 1
end
```

2.2 b: rosenblatt perceptron algorithm

The Rosenblatt perceptron algorithm was implemented as a function, which returns for a single labeled data set whether a linear separation is achieved in a given number of iterations, and returns the last set of weights of the perceptron. For each time step, weights are updated with the function

$$w(t+1) = \begin{cases} w(t) + \frac{1}{N}\xi^{\mu(t)}S^{\mu(t)} & \text{if } w(t)\cdot\xi^{\mu(t)}S^{\mu(t)} \leq c \\ w(t) & \text{if } w(t)\cdot\xi^{\mu(t)}S^{\mu(t)} > c \end{cases}$$

where $w(t)$ is the weight vector of the perceptron in time step t, N is the total number of examples, $\xi^{\mu(t)}$ is a specific example with index $\mu(t)$ which is selected in time step t, and $S^{\mu(t)}$ is the label of example $\xi^{\mu(t)}$. c is a constant which determines the threshold for whether or not the weights will be updated after being presented with an example. For the main experiment $c = 0$, meaning the weights will be updated if the perceptron classifies an example wrongly. The weights are initialized to zero: $w(0) = \vec{0}$. If the weights haven't changed for an entire iteration after evaluating all the examples, we know that the algorithm has converged to a solution, and we stop updating the weights. The algorithm runs for the max number of iterations specified by the user, plus one extra. This way the algorithm tries to find a solution for the number of epochs specified by the user, because it needs to check whether the weights have changed compared to the previous iteration, to know if a solution has been found. If the data is two-dimensional, the function plots the data and the solution weight vector and separating plane, as feedback to the user.

```
function [ w, success ] = rosenblatt( n_max, data, labels )
%Rosenblatt perceptron function
%input args:
%n_max, integer number of iterations for which the function should maximally run
%data, a P by N datamatrix
%labels, a corresponding labelvector of length P
%output args:
%w, the last weight vector of the network (which linearly separates the data, if a solution is found
%success, boolean whether solution is found


w = zeros(1,size(data,2)); % create w(0), size of dimensions
```

```
N = size(w,2);      % placeholder

if(N==2)    % for printing in 2D
    scatter_(data, labels);
end

for n = 1:(n_max + 1)% number of epochs
    changed = 0;
    for t = 1:size(data,1) % for each data-point
        E_mu_t = dot(w,data(t,:)*labels(t));
        if (E_mu_t <= 0)
            w = w + (1/N)*data(t,:)*labels(t);
            changed = 1;
        end
    end
    if (changed==0)
        break
    end
end


% if input is two dimensional, plot it along with the separating plane of the proposed solution
if(N==2)
    hold on
    plotv(w', 'black')
    x = [0, w(1)];
    y = [0, w(2)];
    w_coeffs = polyfit(x, y, 1);
    w_orth_a = -1/w_coeffs(1);
    w_orth_b = 0;
    axis equal
    xlims = xlim(gca);
    w_orth = xlims*w_orth_a+w_orth_b
    line( xlims, w_orth);
    hold off
end


end
```

3 Evaluation

In the main experiment, the explained algorithm for the Rosenblatt perceptron was trained multiple times with different variables. The perceptron was trained with different N dimensions and a range of α, with a fixed number n_d of randomized data-sets used per N and α to correctly average, and a fixed number of epochs n_{max} per training step. To summarize the setup for the experiment was:

- range of N dimensions (5, 20, 50, 100)
- fixed n_{max} maximum training steps (200)
- fixed n_d random data sets (100)

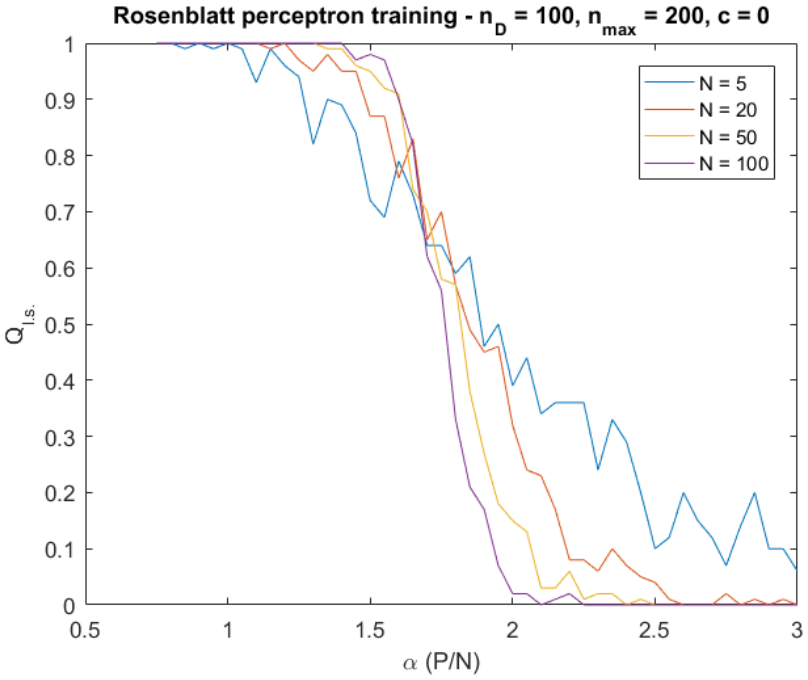

Figure 1: Success rate of Rosenblatt perceptron

- range of α (0.75 : 0.05 : 3.0)

For each setting of N and α the success rate ($Q_{l.s.}$) was stored. This success rate is an average of the number of successes of the perceptron for the n_d random data sets. Figure 1 show the results of the performed experiment. Note that the purple line (N=100) stays at $Q_{l.s.} = 1$ for all α, where all other lines descend to $Q_{l.s.}(\alpha) = 0$

The standard deviations over n_d random datasets can be better illustrated if you use error plots instead of plain plots.

3.1 Evaluating the influence of the weight adaption threshold c

A second experiment was performed to evaluate the influence of the weight adaption threshold constant c. The setup for this experiment was as follows:

- fixed N, number of dimensions (20)
- fixed n_{max} maximum training steps (200)
- fixed n_d random data sets (100)
- range of α (0.75 : 0.25 : 3.0)
- range of values for c (-0.5, -0.25, 0, 0.25, 0.5)

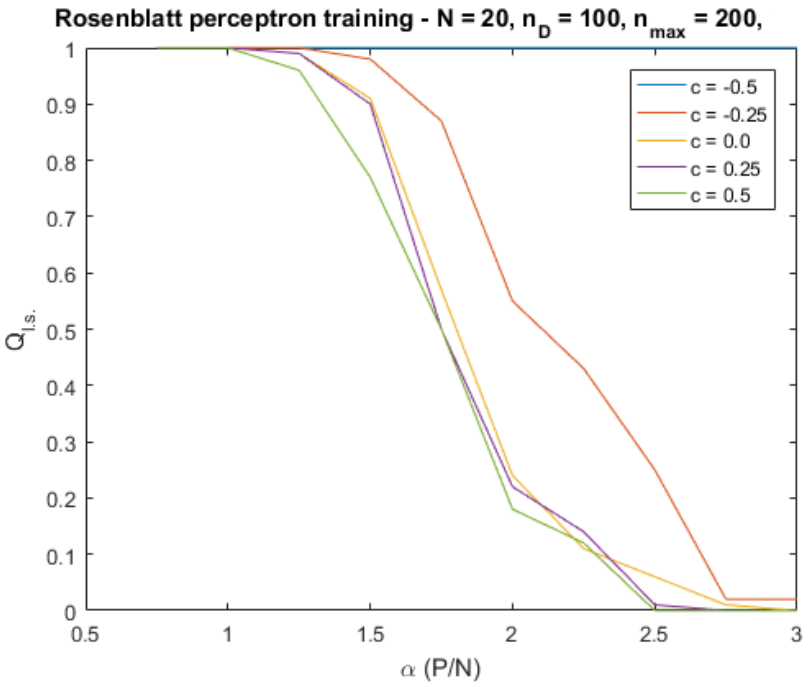

Figure 2: Success rate of the Rosenblatt perceptron for different weight adaption thresholds c.

The step size for α was increased, compared to the first experiment due to time and computational constraints. Similarly to the first experiment, a success rate (Q) was stored, for each setting of α and c. The resulting success rates have been plotted in figure 2. For negative values of c, the success rate seems to increase.

4 Discussion

For all N, $Q_{l.s.}(\alpha)$ is 1 when α is 1 or lower. This is as expected and in line with the presented probability $P_{l.s.}(\alpha)$ that was derived in class. Beyond $\alpha = 1$ the $Q_{l.s.}(\alpha)$ decreases for all N. For N = 5 this success rate immediately decreases, but slowly decreases. When N is increased this descend starts later but is far more steeper. The best example for this is seen in the line for N = 100, which is much steeper than the line for N=5, and starts to descend at $\alpha = 1.4$. All the success rates seem to reach 0.5 somewhere between $\alpha = 1.8$ and $\alpha = 2$. At $\alpha = 2$ the success rate for N=5 is approximately 0.4, decreasing to an astounding approximation of 0.1 for N=100. We hypothesize this is much lower than the expected success rate of 0.5 at $\alpha = 2.0$ due to the suppressing nature of n_{max}. The training is stopped when a certain number of training epochs have been

reached. It is possible that the perceptron did not find a solution in the given time, while there could be one found when this n_{max} is set larger. We expect that when n_{max} is increased the success rate $Q_{l.s.}(\alpha)$ at $\alpha = 2.0$ will tend more towards 0.5 for all N dimensions. The success rate for N=5 tends to reach 0 beyond $\alpha = 3$, where this tends to zero much faster for N=20 and even faster for N=50 (this already resembles zero at approximately $\alpha = 2.6$. The success rate $Q_{l.s.}(\alpha)$ for N=100 reaches 0 at approximately $\alpha = 2.1$. All this is clearly in line with the presented probability $P_{l.s.}(\alpha)$ that was derived in class, and even the plots slightly resemble each other. It can be seen that when N is increased, the success rate function more and more resembles a step function. The only difference is found in that the success rate $Q_{l.s.}(\alpha)$ of 0.5 is reached at an earlier α than 2. As noted before we hypothesize this effect will disappear when a higher n_{max} is taken. Training will however take much longer, and time issues might surface in that case.

4.1 The influence of changing the weight adaption threshold c.

As seen in figure 2, the success rate Q increases greatly for negative values of c, going to all the way up to a success rate of 1 for $c = -0.5$. However, this success rate of the Rosenblatt function with negative values of c no longer corresponds to the successful linear separation rate $Q_{l.s.}$. Classification in the Rosenblatt perceptron is deemed successful by the algorithm if the dot product of the perceptron weight vector $\vec{w}$ with a (positive) example vector $\vec{\xi}$ is larger than c. If c is zero, like in the standard Rosenblatt algorithm, having c equal to this dot product corresponds to a point lying exactly on the (hyper)plane orthogonal to the weight vector $\vec{w}$. Therefore successful classification of all points implies linear separation between the classes.
Taking a negative value for c makes the algorithm lenient with respect to errors in classification, provided the error margin is smaller than $(-)c$. This is why the success rate tends to 1 for large enough negative values of c, because the angle that the weight vector and a (positive) example vector are allowed to have for a "successful" classification, will eventually grow large enough to encompass the entire N-dimensional space.
For positive values of c the opposite happens. The algorithm becomes so stringent, that it rejects valid solution weight vectors (in terms of linear separation), for which the angle of the separating (hyper)plane with the nearest example is smaller than some margin angle, determined by c. When looking at figure 2, this does not seem to have as large as an impact on the success rate as equal negative values of c. We expect that for large enough values of c, the success rate will eventually drop towards zero, regardless of α. Setting c to a small positive value might actually be good solution in some cases where there is input noise (if the data set is still expected to be linearly separable), because then if a solution is found, it is guaranteed to to be somewhat robust to small changes of the examples.

Indeed a positive c is related to the width of separating hyperplane. Good observation and well written.

Addition of a conclusion will enrich the report.