UNIVERSITY OF GRONINGEN

FACULTY OF MATHEMATICS AND NATURAL SCIENCES
DEPARTMENT OF COMPUTING SCIENCE

NEURAL NETWORKS AND COMPUTATIONAL INTELLIGENCE

Assignment II

Authors

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

2nd February 2018

university of groningen / faculty of science and engineering

Contents

1 Introduction

The purpose of this report is to show and analyze the results of the MinOver algorithm when applied to a randomized dataset. In section 2 we will explain the algorithm and its steps, and also discuss variable initialization and coding decisions. In section 3 we will show the results from running the algorithm and finally, in section 4 we will discuss these results.

MinOver is a perceptron algorithm used on datasets to achieve linear separability, meaning to classify the vectors of the dataset in groups, based on a linear function. Similar to the Rosenblatt perceptron, it processes the elements of the dataset one at a time. In every step t, it calculates the stability of all examples and keeps the minimum in memory. The weights are updated using the vector with minimal stability. After certain conditions have been met and all stabilities are >0, the dataset is linearly separable.

2 Problem solution

The first step of the algorithm is to generate sets of P, N-dimensional random vectors $D = \{\xi^\mu, S^\mu\}$, where $S^\mu = sign(w^*)$. First all of the stabilities are calculated using

$$\kappa^v(t) = \frac{w(t) \cdot \xi^v S^v}{|w(t)|}$$

for all v. The vector with minimal stability at time t is selected using

$$\kappa^{\mu(t)} = min_v\{\kappa^v(t)\}$$

. The last step is to update the weights using the sample with minimal stability using

$$w(t+1) = w(t) + \frac{1}{N}\xi^{\mu(t)}S^{\mu(t)}$$

.

Initially, we consider fixed dimensions for all vectors N=20, number of generated datasets maxDdatasets=30 and

$$n_{max} = 100$$

. We set the range for α at (0.25, 6) with a step of 0.25.

Foe every α we compute a number P for the vector population as $P = \alpha * N$. Then for every P we create 30 datasets. Lastly, for each dataset we performed $t_{max} = n_{max} * P$ training steps.

At each step we calculate the stability vector for each example and then perform the Hebbian update. For each step we also store the minimum stability. The inner loop for the training steps written in Matlab can be seen in Listing 1

```
        for t=1:t_max
            stab = (weightsSt * D .* S) / norm(weightsSt);
                % Calculate stabilities
            [stab_min, idx] = min(stab);
                % Find minimum stability
            vector = D(:,idx);
                % Find vector and label belonging to idx
            label = S(idx);
            old_weightsSt = weightsSt;
                % Store old weights for similarity checking
            weightsSt = weightsSt + learning_rate * vector' * label;
                % Modify the weights using Hebbian update step
            error = 1/pi*acos((weightsSt*weightsT')/(norm(weightsSt)*norm
    (weightsT)));   % Calculate generalization error
            similarity = pdist([weightsSt;old_weightsSt], 'cosine');
                % Stop if the weights do not change anymore
            if similarity < 0.001
               break;
            end
        end
```

Listing 1: MinOver Matlab code

The training concludes when we have either reached t_{max} steps or the weight vector w does not change anymore. The last check we do is to see if the stabilities have changed and stop if they have not. In the next section you can see the results of the training.

3 Evaluation

By running the Matlab program for α between 0.25 and 6, N=20 and 30 datasets, we obtain the following results. Figure 1 shows the average generalization error for the different P, and figure 2 the average minimum stability for each P.

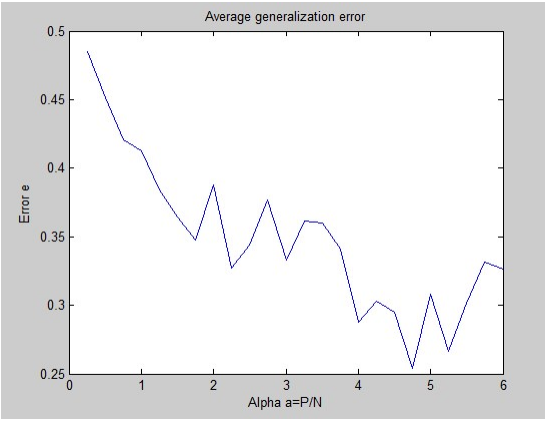

Figure 1: Average error for every P and alpha=5

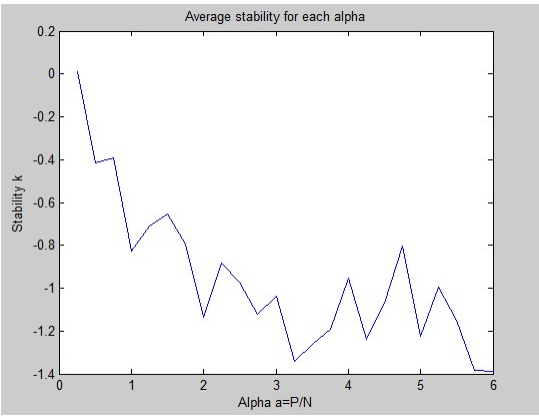

Figure 2: Average stability for every P and alpha=6

4 Result discussion

Judging from the above plots, one may assume that the examples from the datasets produced are not correctly classified since the average stability for each is not greater than zero. The stability of each example is its distance from the decision boundary, so in case of a correct classification the stability is > 0, 0 < otherwise. The generalization error is decreasing as α increases, as expected from the theoretical background.

It is not certain what causes the wrong stability, maybe the arbitrary initial values for N, n_{max} and the number of datasets produced each time, although we tried several different values for each variable. We tried $n_{max} = 50$ and a different range for α but the results were pretty much the same. The training steps were followed to the letter. It is assumed that something went wrong during the coding, or a different combination of initial values was needed.