The Minover Algorithm
Neural Networks and Computational Intelligence
Practical 2

Rogier de Weert (s1985779)
Tim Oosterhuis (s2234831)

December 2017

1 Introduction

In contrast to the Rosenblatt perceptron algorithm, which is only guaranteed to find a weight vector to linearly separate a data set, the Minover perceptron algorithm is guaranteed to find the linear separation weight vector of optimal stability. Just like with the Rosenblatt algorithm, the data needs to be linearly separable for this, and the amount of time steps needs to be large enough, and theoretically infinite.
At each iteration of the Minover perceptron the stabilities are calculated for all examples ξ^v with the following formula:

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

The example ξ^μ with the minimal stability $\kappa^\mu = \min(\kappa)$ is ~~updated~~ with a Hebbian learning step:

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

For each example ξ^v, the training label $S(\xi^v)$ is determined by a teacher perceptron with weight vector w^*, with the equation: $S(\xi^v) = \text{sign}(w^* \cdot \xi^v)$. The Minover algorithm learns in version space: we assume that the teacher perceptron is perfect and there is no noise in the training examples. To calculate how well the Minover perceptron is able to learn the rule given by the teacher perceptron in the examples, we can calculate the generalization error, which is analogous to the angle the weight vector w of the perceptron of optimal stability has with the weight vector w^* of the teacher perceptron in version space. The formula used to calculate the generalization error is given below:

$$\epsilon_g(t_{max}) = \frac{1}{\pi}\arccos(\frac{w(t_{max} \cdot w^*)}{|w(t_{max}||w^*|})$$

2 Implementation

2.1 Data generation

The input data is randomly generated, but the labels are created as a function of the input and the teacher perceptron:

$$S^\mu = sign(w^* \cdot \xi^\mu)$$

where the teacher perceptron $w^* = (1, 1, ..., 1)^\top$.
Because the squared norm is $|x|^2 = x_1^2 + x_2^2 + ...x_n^2$ any vector $w^* = (1, 1, ..., 1)^\top$ is a vector where $|w^*|^2 = 1_1^2 + 1_2^2 + ... + 1_n^2 = N$, so $|w^*|^2 = N$.

2.2 Minover algorithm

The main calculation of the minimum stability is implemented in the *calc_k_min* function:

```
function [ K_min, dat_min, lab_min ] = calc_k_min( w, data, labels )
    % Minimum stability calculation
K_min = dot(w,data(1,:)*labels(1));
dat_min = data(1,:);
lab_min = labels(1);
for t = 2:size(data,1)
    K_v_t = dot(w,data(t,:)*labels(t)); % Stability calculation
    if K_v_t < K_min % Smallest stability search
        K_min = K_v_t;
        dat_min = data(t,:);
        lab_min = labels(t);
    end
end
end
```

In the *Minover* function the dataset is shuffled after each update, and the update is applied with the datapoint with the lowest stability:

```
x = randperm(size(data,1)); % For random picking of data
data = data(x,:);
labels = labels(x);
[~, dat_min, lab_min] = calc_k_min(w, data, labels);
w_old = w; % placeholder for stopping mechanism
w = w + (1/N)*dat_min*lab_min; % Actual update
```

Training stops after a set steps input by the user, or when the absolute change in w is smaller than a certain threshold for contiguous P steps.

3 Results

Figure 1 shows the generalization error of the Minover algorithm and the Rosenblatt perceptron for different α, where N was set to 50, α ranges from 0.1 to

5.0 with a step size of 0.1, the total amount of training steps t_{max} was 200, and each α was trained 10 times for both algorithms.

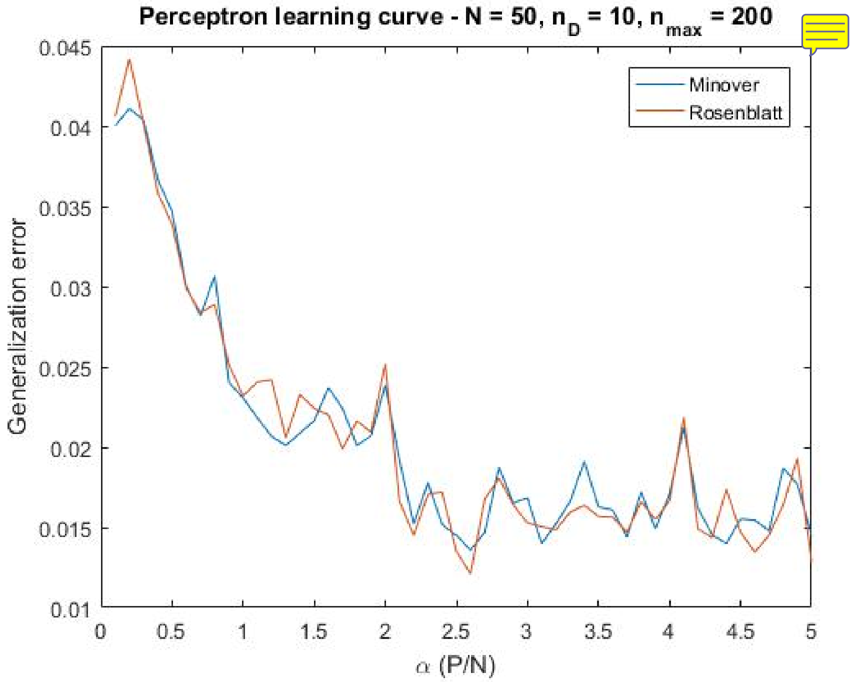

Figure 1: Generalization error comparison of Minover and Rosenblatt algorithms

4 Discussion

As can be seen in Figure 1 the generalization error for both the Minover and Rosenblatt algorithm decrease with increasing α. Surprisingly, the Minover algorithm does not outperform the Rosenblatt algorithm. We hypothesize this could be due to the small maximum training steps. If the Minover algorithm were to train longer, it could reach optimal stability more often. On the other hand, the Rosenblatt perceptron always finds a solution to a linearly separable data set. It could simply be the case, that version space for a data set with a high α (large number of examples, relative to the number of dimensions), which is defined by the examples is relatively narrow, and there is little difference between the teacher perceptron and a random perceptron in version space or the perceptron of optimal stability.

5 Bonus: Optimal stability comparison with and without teacher perceptron

The smallest stability κ_{min} is returned by the *calc_k_min* function, which is used in training phase of the Minover algorithm. After training is done, the κ_{min} is calculated once again to obtain the (presumably optimal) stability of the final weight vector. ~~Figure 2 shows a comparison of this final optimal stability for a range of α when the Minover algorithm is trained with and without a teacher perceptron. As expected, the maximal stability is a lot lower for random output. This is due to the fact that without a teacher perceptron, there is no coherent version space, and the data isn't guaranteed to be linearly separable~~.

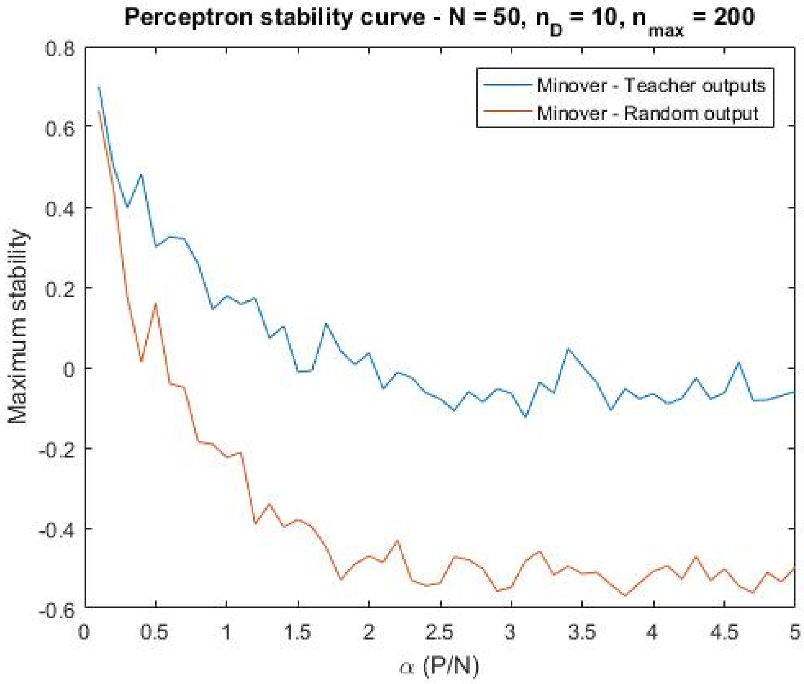

Figure 2: Optimal stability comparison

6 Bonus: Noisy data

With increasing noise in the data, the generalization error of the Minover algorithm rises as well, as can clearly be seen in Figure 3. This shows that when the data is completely random ($\lambda = 0.5$) The generalization error is at approximately 0.5 for all alpha. For lower λ we observe that the generalization error is lower, where for low α even the descending slope of the generalization error can be seen for $\lambda = 0.1$. This shows that the Minover algorithm can handle some degree of noise, but that its performance will decrease in proportion with the amount of noise present. Note that the line for $\lambda = 0.1$ is much smoother than the corresponding line found in Figure 1. This is due to the difference in the number of dimensions N. A larger number of dimensions leads to more fluctuations in the generalization error of the Minover perceptron.

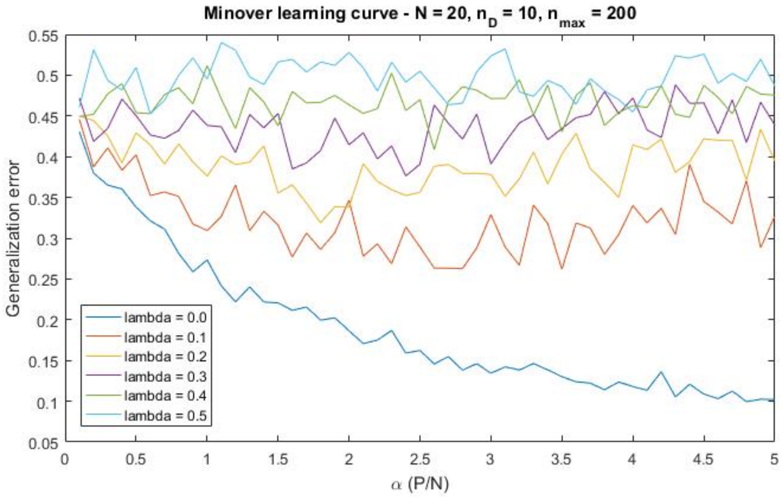

Figure 3: Generalization error for different noise levels

7 Bonus: Iris classification (inhomogeneous perceptron)

Our implementation of the Minover algorithm can only succeed in finding optimal stability in version space for a homogeneously linearly separable set of P N-dimensional examples. That is to say, only when there exists a weight vector $w \in \mathbb{R}^N$ for which $\text{sign}(w \cdot \xi^v) = S(\xi^v)$ for all examples $\xi^v \in \xi$ When our Minover implementation is applied naively to a set of examples which is inhomogeneously linearly separable in N dimensions, it will fail to find a solution. However, it is trivial to transform a set of inhomogeneously linearly separable points in N dimensions into a set of homogeneously linearly separable points in $N+1$ dimensions. To do this, we add a clamped extra input dimension to each example, and an extra weight θ to the weight vector. This is formally described by:

$$\tilde{\xi^v} = (\xi_i^v, \dots, \xi_N^v, \xi_{N+1}^v = -1) \in \mathbb{R}^{N+1}$$
$$\tilde{w} = (w_i, \dots, w_N, w_{N+1} = \theta) \in \mathbb{R}^{N+1}$$

where:

$$\tilde{w} \cdot \tilde{\xi^v} = w \cdot \xi^v - \theta$$

Adding a clamped extra input dimension is achieved with the following line of matlab code, for a data variable called ‘data’:

```
data = [data, (-1*ones(length(data), 1))];
```

Because our implementation of the Minover algorithm works for any number of dimensions N, no further adjustments were necessary in the code.
For this experiment, the *fisheriris* data set in MATLAB was preprocessed for binary classification, by taking only the third and fourth data feature of the data as a predictor data set, and by grouping the second and third classes together. The minover algorithm was applied on the preprocessed iris classification data set, after extending the iris classification data with an extra input dimension, clamped to 1.
After 10 iterations of the Minover algorithm with $t_{max} = 200$, using Monte Carlo cross-validation with a 80/20 split between the train and test data, an average prediction error of 0 was obtained, indicating that perfect linear separation was achieved each time. As a control, the same experiment was done naively (without first clamping the data in a higher dimension), which resulted in an average prediction error of 0.3867. When the experiment would be repeated often enough, we'd expect the average prediction error of the control experiment to converge toward the percentage of examples belonging to the smaller of the two classes, which is $\frac{1}{3}$.
The comparative prediction error rates of the two experiments show, that the preprocessed iris classification data is only inhomogeneously linearly separable.