Neural Networks and Computational Intelligence

Assignment II: Learning a rule

Jeroen Overschie, Erik Zhivkoplias

Group 01

January 31, 2020

1 Introduction

As we know from the lecture material the version space (given the set of examples) is a region in a weight space that is limited by that set of examples. In this study an aim is set to learn an *optimal* rule in a version space that linearly separates a dataset into two classes, using example data. For this, a perceptron can be utilized to help solve the problem, which is extended using the *Minover algorithm*. Outputs $S^\mu = \pm 1$ are considered, which are defined by a teacher perceptron. The resulting dataset is used thus guaranteed to be linearly separable.

1.1 Problem

A set of P random input vectors are considered, all with the same number of features N. The input vectors are drawn randomly from a standard Gaussian distribution $\xi_j^\mu \sim \mathcal{N}(0,1)$, with labels set as a random number of either -1 or 1. Training labels are defined by a teacher perceptron: $S^\mu = sign(w^* \cdot \xi^\mu)$. Initial w^* vector is defined as a randomly drawn w^* with $|w^*|^2 = N$.

The goal of this work is to learn a linearly separable rule in minimized version space (see Figure 1) by maximizing the stabilities of every example in dataset. Stability of every example ξ is determined by distance-based variable κ that will be explained below.

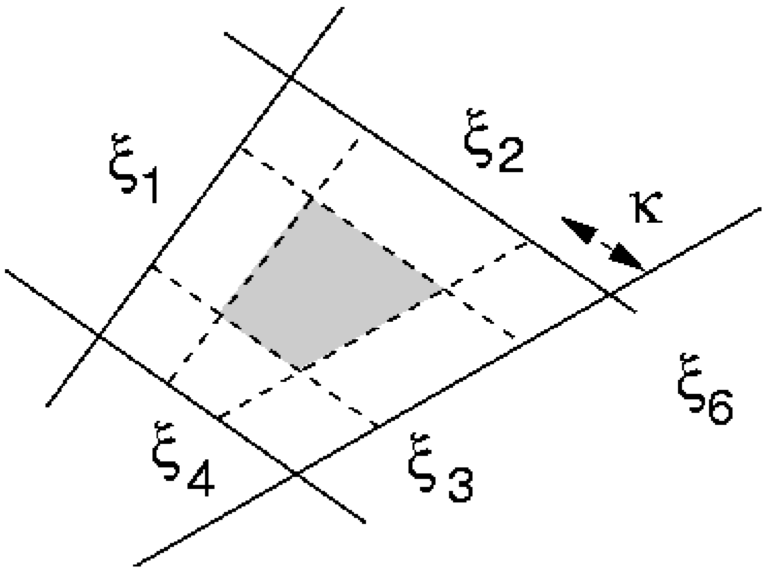

Figure 1: Learning from examples in a version space. Grey shaded area of possible w vectors gets smaller with every new example ξ.

We will cover two scenarios: first with the absence, and then with the presence of noise where the training labels S^μ defined by a teacher perceptron replaced by "noisy labels" (see corresponding section below).

2 Method

An explanation of provided below on how this study was set up, in order to observe perceptron behavior and the learning curve for an Rosenblatt algorithm extended with Minover. Accordingly, a method on how to extend Rosenblatt's algorithm is first explained. Secondly, a way of computing the generalization error is explained.

We expect that the perceptron of optimal stability will display optimal generalization behavior (see Generalization error section below) when trained from a linearly separable dataset.

2.1 Rosenblatt perceptron Minover extension

One way to find w with optimal weights as described above is to implement the perceptron of optimal stability with Minover algorithm.

The *Minover* algorithm is the sequential algorithm where at every time step t the stabilities are determined as:

$\kappa^{\nu} = \frac{w(t) \cdot \xi^{\nu} S^{\nu}}{|w(t)|}$ for all examples ν

Then, the example with the minimal stability among all is identified. Let example $\mu(t)$ be the example with minimal stability, defined as: $\kappa^{\mu(t)} = min_{\nu}\{\kappa^{\nu}(t)\}$. Denote that stability values can be negative. In case of several negative stabilities, the stability with the largest absolute value is chosen. In the case where there exist a *tie* in the scores, the choice between those ties is arbitrary. Then, using this example, the Hebbian update step is performed:

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

Note that the sequence of iterating examples is not fixed, which is, in contrast, the case with the Rosenblatt's perceptron algorithm. We used the following stopping criterion for the algorithm: the weight vector does not change anymore over a number of P epochs.

Given the final weights and the stability, $w(t_{max})$ and $\kappa(t_{max})$ respectively, a perceptron with optimal stability will yield the optimal-stability weight vector w_{max} where the stabilities of every example in dataset are maximized.

2.1.1 Generalization error

At the end of the training process, generalization error is computed. A training process ends when either of the two conditions described above are triggered, e.g. the algorithm converged. The generalization error can then be computed as such:

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

2.2 Noisy examples (bonus)

An extra experiment was conduct in order to test Minover's resilience against noisy examples. Some of the examples are distorted by changing their label values, e.g. changing their positive- or negative sign. Labels that are specified as 'true', or in fact have the value +1.0, are flipped by some degree of probability. This flip can be formulated as such:

$$S^{\mu} = \begin{cases} +sign(w^{*} \cdot \xi^{\mu}) & \text{with probability } 1 - \lambda \\ -sign(w^{*} \cdot \xi^{\mu}) & \text{with probability } \lambda \end{cases}$$

Using this formulation, the level of distortion can be controlled by the variable λ. Let its range be defined as $0 < \lambda < 0.5$. In the study, now one is able to observe what effect various levels of noise will have on the algorithm results.

2.3 Implementation

The Minover and Generalization error modifications were applied as explained in the Method. Some notes do apply, though.

Consider one execution of the algorithm, the point from where the dataset is randomly initialized to the point where the dataset either converged of the epoch limit was reached, and the generalization error is computed. In order to obtain more robust results, this execution sequence is performed at least n_D times, set to at least 10; $n_D \geq 10$.

By running the algorithm for various values of P, the learning curve is determined. For testing various P values, a range of α values are considered. In this study, a range of $\alpha \in [0.1, ..., 10]$ with linearly distributed increments, using some specified amount of increments in total (specified as parameter *a_incr*).

During the selection of minimal-stability examples, there might be a case where there exist a *tie* in the scoring computation. In this study, the built-in python SORTED function is used, after which simply the first element is chosen from this sorted array. Because the choice of the min-stability sample in case of a tie is allowed to be arbitrary, this implementation is correct.

2.3.1 Minover pseudocode

The base pseudo-algorithm upon which we based our implementation is shown in Algorithm 1. (Krauth and Mézard, 1989)

The NOTCONVERGED() function will do a test whether the stabilities have not changed, over a period of P epochs, using the generalization error as a metric.

A realization of this pseudo-code was constructed in Python. Its learning step code fragment was included in the Appendix, Chapter 5

Algorithm 1: Minover Algorithm

```
1 function Minover ($\mathbb{D}, n_{max}, w$);
  Input  : $\mathbb{D} = \{\xi^\mu, S^\mu\}_\mu^P$
           $n_{max} \in \mathbb{N}$: maximum number of epochs
  Output: $w$ student perceptron weights
2 $t_{max} := n_{max} \cdot P$
3 $t := 0$
4 $w := [0, ..., 0]^T$ where $w \in \mathbb{R}^N$
5 while $t < t_{max} \wedge$ notConverged() do
6     find $\mu(t)$ such that $k^{\mu(t)} = \underset{\nu}{min}\{k^\nu(t)\}$
7     $w(t+1) = w(t) + \frac{1}{N} \cdot \xi^{\mu(t)} \cdot S^{\mu(t)}$
8     $t := t + 1$
9 end
```

3 Results and Discussion

Using exactly the implementation discussed in the previous section, some number of experiments were performed to achieve our research goal.

3.1 Experiment A

In this experiment, a first observation of the behavior of the generalization error as α grows is made. Parameters were set as follows:

- $tmax = 100$; (max number of epochs)
- $n_D = 10$; (number of trials)
- $N = 20$;
- $\alpha \in [0.1, 10]$ with 10 equidistant increments.

Measuring generalization error for each individual training step, the generalization error can be plotted against its given α $(= \frac{P}{N})$ value.

In Figure 2 we see that as α increases the generalization error first decreases and then gets stabilized thus the perceptron of optimal stability with Minover algorithm yields reliable generalized l.s. rule if the number of examples P is large enough w.r.t. given N. It indicates that with the larger P the algorithm has more examples to learn from, therefore the larger probability that the predictions given by the teacher perceptron and the student perceptron will match each other. Here we see that the real learning starts where the storage ends (we first need to reach the storage capacity of the perceptron which is $\alpha \leq 2$). Therefore if $\alpha \to \infty$ the version space will shrink to a very small area, and the generalization error $\to \epsilon$.

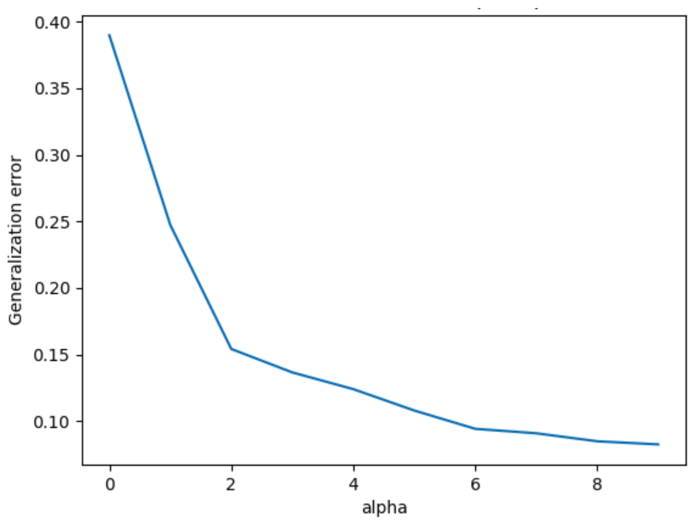

Figure 2: Average sum of error for various α values in Experiment A.

3.2 Experiment B

In this second experiment, parameters are cranked up, resulting in a higher-resolution and more reliable result. Parameters were set as follows:

- $tmax = 20, 100, 5000$;
- $n_D = 100$;
- $N = 20$;
- $\alpha \in [0.1, 10]$ with 100 equidistant increments.

Measuring generalization error for each individual training step, the generalization error can be plotted against its given α value.

In Figure 3 we demonstrate that $tmax$ has to be large enough for the stabilities to get close to optimal stability, when $tmax$ is 20 the generalization error never drops below 20% however as we increase $tmax$ up to 5000 it levels out at 5%, therefore if $\alpha \to \infty$ and $tmax \to \infty$ the version space will shrinks to a point, and the generalization error $\to 0$. From that we can conclude that the number of epochs is important parameter for the minover algorithm to converge and yield the w vector of optimal stability. $tmax = 100$ looks like an optimal trade-off between the generalization error being low and the computational cost if we need to think og the practical applications of the algorithm.

3.3 Experiment C (bonus)

In the third experiment, the dataset is distorted by noisy samples.

- $tmax = 100$
- $n_D = 10$
- $N = 20$

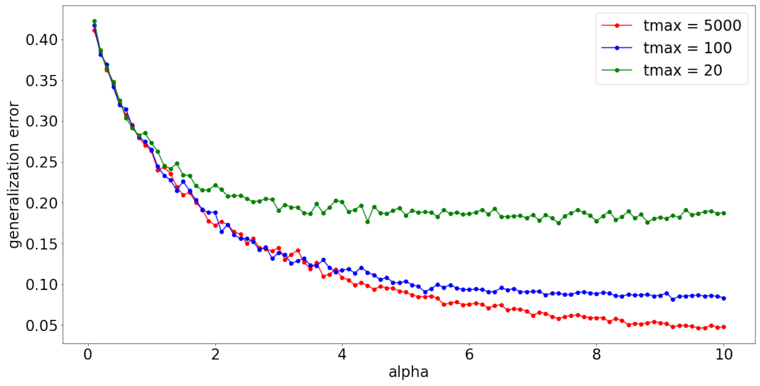

Figure 3: Average sum of error for various α values in Experiment B.

- $\alpha \in [0.1, 10]$ with 50 equidistant increments.
- $noise\_levels \in [0.0, 0.5]$ with 6 equidistant increments.

As we can see in Figure 4 if the dataset labels provided by teacher perceptron has at least 10% of noise the problem of finding the w vector of optimal stability cannot be solved with Minover algorithm as one of the reasonable assumptions about the dataset does not hold anymore. The reason for such behaviour is that in this case the labels were not generated w.r.t. the linearly separable rule that was found by teacher perceptron, but just in a random manner. Therefore the predictions for new examples is mostly just a random guess.

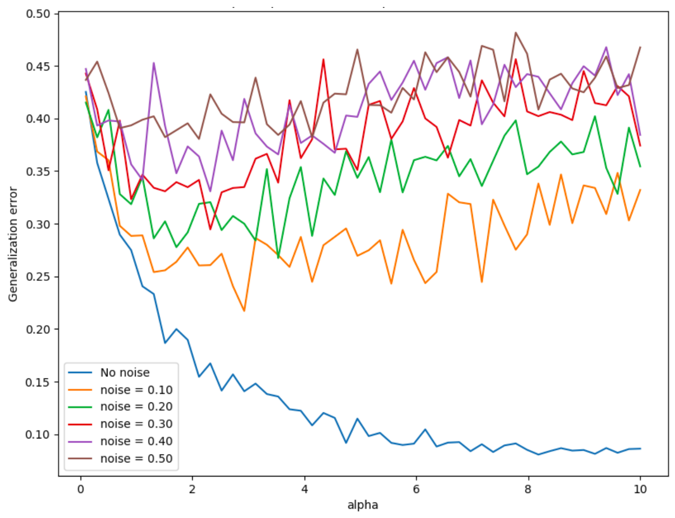

Figure 4: Average sum of error for various α values in Experiment C.

4 Conclusion

From this set of experiments we conclude that maximum stability is indeed achieved by weights of the form:

$w(t+1) = w(t) + \frac{1}{N}\xi^{\mu(t)}S^{\mu(t)}$; where $\xi^{\mu(t)}$ is an example with the lowest stability as the time t.

Moreover, the framework of iterative Hebbian learning is sufficient to find the solution for the problem of finding the perceptron of optimal stability.

4.1 Individual workload

Erik Zhivkoplias:

- writing the implementation of the Minover algorithm in Python - 65%
- writing the report - 35%:
  - introduction
  - results and discussion (partially)
  - conclusion

Jeroen Overschie:

- writing the implementation of the Minover algorithm in Python - 35%
- writing the report - 65%:
  - method
  - results and discussion (partially)
  - appendix

References

Werner Krauth and Marc Mézard. Storage capacity of memory networks with binary couplings. *Journal de Physique*, 50(20):3057–3066, 1989.

5 Appendix

5.1 Minover algorithm implementation in Python.

```python
data = []

#calculate kv(t)
for mu_t in range(P):
    current_example = xi[mu_t]
    e_mu_t = np.dot(np.dot(current_example, w), S[mu_t])
    data.append([e_mu_t, xi[mu_t], S[mu_t]])

#determine the minimal stability
data = sorted(data, key=itemgetter(0), reverse=False)
lowest_stab = data[0][0]

#determine the sample with minimal stability and its label
min_data = data[0][1]
min_label = data[0][2]

#hebbian update
w = w+(min_data*min_label)/N

if lowest_stab >= initial_lowest_stab: #if the algorithm found the
    larger distance
    initial_lowest_stab = lowest_stab
    converge = 0 #then the algorithm converges
else: #if the distance with the new example that the algorithm
    found is larger the one that we already have
    converge = converge+1 #then increase counter by 1
    if converge >= P: #once it reaches threshold (larger than P as
    it is stated in the assignment)
        break #stop the algorithm
```

Listing 1: Minover learning step python implementation.