Neural Networks and Computational Intelligence
Assignment 2: Learning a linearly separable rule

Ayça Avcı (S4505972)
Ethelbert Uzodinma (S3886026)

January 24, 2020

1 Introduction

In this report, we applied the Minover algorithm on a dataset to achieve the optimal stability of the perceptron [1]. This algorithm was implemented while learning of a linearly separable rule from the example data.

The randomly generated dataset is constructed using the student perceptron w and the teacher perceptron $w*$ that ensures correct stability, as a result the data set is guaranteed to be linearly separable, and learning in version space is possible given the absence of noise in the dataset.

To learn a linearly separable rule in a version space, we assumed that:

- An unknown, linearly separable rule exist which assigns any possible input vector $\xi \in \mathbb{R}^N$ to the binary output $S_R^\mu = \pm 1$ where the subscript stands for "rule".
- We assume that the set of training data $\mathbb{D} = \{\xi^\mu, S_R^\mu\}_{\mu=1}^P$ comprises of perfectly reliable examples for the implementation of the rule.
- Also, that it does not contain mislabeled example data that is corrupted by any form of noise in the input or output channel.

We first run our experiment on the dataset and observe the stabilities of the weights. We then sweep through the entire dataset n times, while determining and updating the weights with the least stability (i.e the student perceptron with the lowest distance from the hyperplane). The algorithm converges on or before the maximum epoch n_{max} is reached when it find the perceptron of optimal stability, at this point the student weight vector becomes stationary.

2 Methods and Implementation

2.1 Dataset

The randomized input data that is generated consists of feature vectors of length N. In total, P number of data points are generated, each with a corresponding label $S^u \in \{-1, 1\}$. The outputs are determined by a random teacher perceptron w^* with $|w^*|^2 = N$. Every feature vector in the data set is generated by a Gaussian normal distribution with mean of zero and a variance of one.

2.2 Minover Algorithm

Learning a linearly separable rule by the Minover algorithm is achieved by implementing Hebbian learning on the least stable example or the given weight vector $w(t)$ with minimum local potential $E^{v(t)}$. We implemented the Minover algorithm as follows:

- We assume an initial *tabula rasa* state $w(0) = 0$
- For each time step $t = 1, 2, 3...$ we go through the entire set of examples and determine the stabilites of all ν examples (note: this is in contrast to the Rosenblatt algorithm in which the sequence of examples is fixed).
- Identify the index of ξ^μ with minimal overlap i.e the example that currently has the least stability:

$$\kappa^{v(t)} = \min_{\nu}\{\kappa^{\nu(t)} = \frac{w(t)\xi^\nu S_R^\nu}{|w(t)|}\} \tag{1}$$

- Update the weight vector according to:

$$w(t+1) = w(t) + \frac{1}{N}\xi^{\mu(t)}S^{\mu(t)} \tag{2}$$

- We repeat this process until we reach the maximum number of iterations $t_{max} = n_{max} = 100$.
- We then determine the generalization error (at the end of the training process) according to

$$\epsilon_g(t_{max}) = \frac{1}{\pi}\arccos(\frac{\mathbf{w}(t_{max})\cdot\mathbf{w}^*}{|\mathbf{w}(t_{max})||\mathbf{w}^*|}) \tag{3}$$

  By repeating the training for different P, determine the so-called learning curve, i.e. the final generalization error $\epsilon_g(t_{max})$ as a function of the scaled number of examples $\alpha = P/N$.

3 Results

We repeated the training process using the Minover algorithm for different values of P to determine the learning curve i.e $\epsilon_g(t_{max})$ as a function of $P = \alpha * N$. We determined the number of data points P as follows:

$$P = \alpha * N \tag{4}$$

where α = 0.25, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0. At the end of the training process, we computed the generalization error $\epsilon_g(t_{max})$ (see equation 3). Training is repeated on $n_D = 30$ randomly generated dataset where each data point has $N = 20$ features. The training process is repeated until the maximum number of iterations $t\_max = n\_max*P$ is reached.

Following generalization error graph is obtained under above conditions:

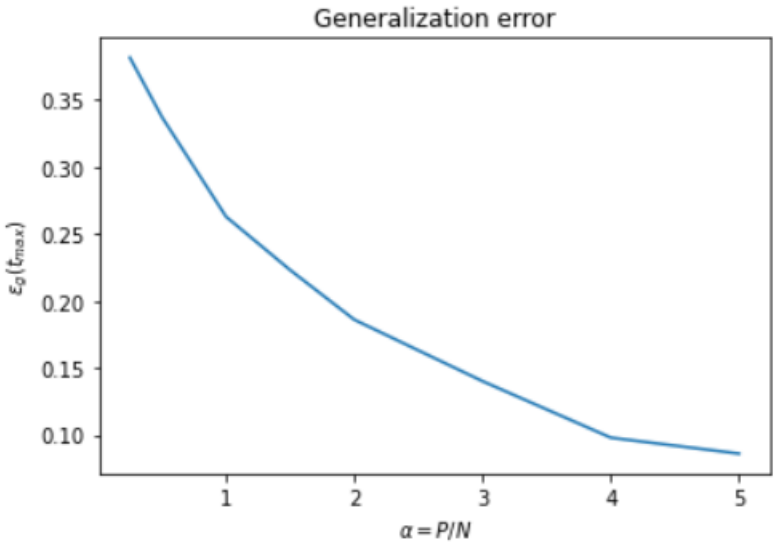

Figure 1: Generalization error for different α values for $N = 20$.

3.1 Bonus Extension 1

As an extension, we considered learning from noisy examples by replacing the true labels in the dataset by:

$$S^{\mu} = \begin{cases} +\text{sign}\left(\mathbf{w}^* \cdot \boldsymbol{\xi}^{\mu}\right) & \text{with probability } 1-\lambda \\ -\text{sign}\left(\mathbf{w}^* \cdot \boldsymbol{\xi}^{\mu}\right) & \text{with probability } \lambda\ . \end{cases} \tag{5}$$

where $0 < \lambda < 0.5$ controls the noise level in the training data (λ is determined randomly on given range below in each dataset generation). We obtained the following generalization error graph for both noise-free and noisy dataset:

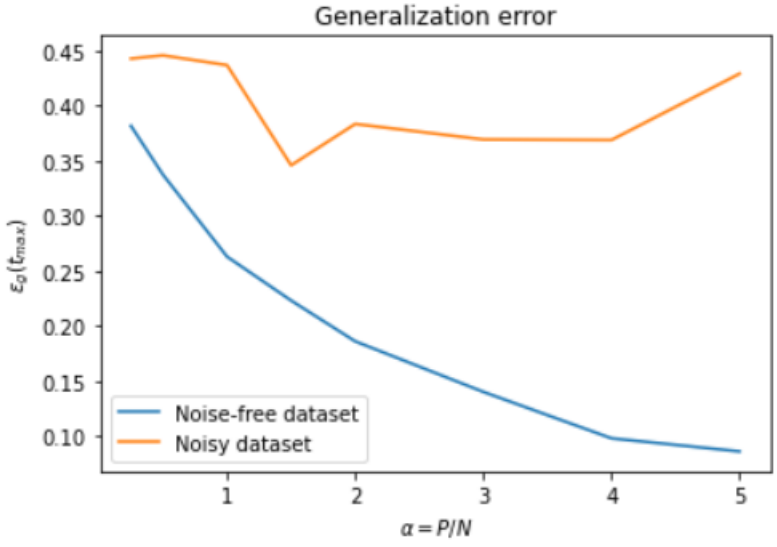

Figure 2: Generalization error with noise-free dataset vs. noisy dataset for different α values for $N = 20$.

3.2 Bonus Extension 2

As an additional extension, we determined the individual stabilities $k^v(t_{max})$ at the end of training and generated histograms of the frequency of observed stabilities as below:

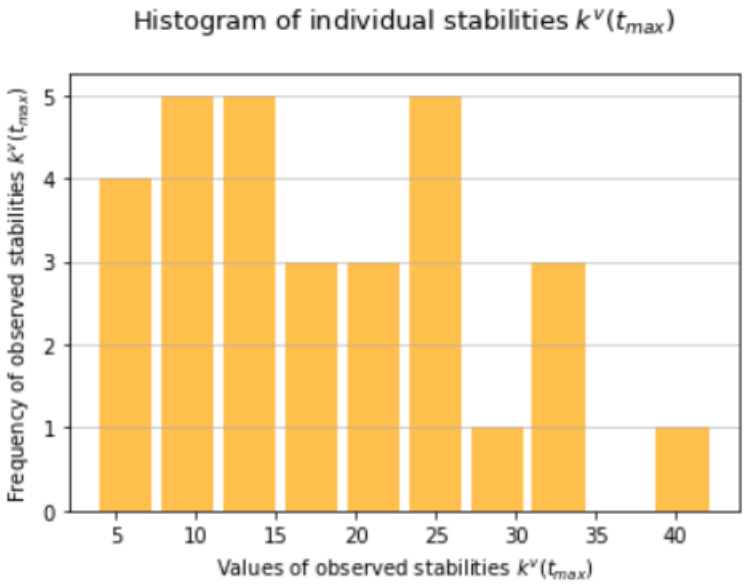

Figure 3: Histogram of individual stabilities $k^v(t_{max})$

4 Discussion

For every value of α, the training process is repeated $n_D = 30$ times, for randomly generated dataset in each iteration, and the average generalization error is computed over these 30 iterations. We plotted the average generalization error as a function of α as displayed in Figure 1.

The average generalization error seems to form a kind of a linear trend with the value for α. As the number of data points are increased, the generalization error decreases. An exact linear line might have been obtained with more and small increment of values of α, as well as, with a larger number of training runs. This indicates that the Minover algorithm is able to approximate the final weight vector $w(t_{max})$ with stability $k(t_{max})$ when more data is available along with the increased number of training runs where $t\_max \leq n\_max*P$.

4.1 Bonus Extension 1

To see the effect of noisy dataset on a learning curve i.e. the generalization error $\epsilon_g(t_{max})$, we replaced the true labels in the dataset by applying the equation 5. To compare generalization errors that are obtained from noise-free dataset and noisy dataset, we plotted Figure 2. While generalization error $\epsilon_g(t_{max})$ decreases almost linearly with increasing α values for noise-free dataset, it is not the case for noisy dataset. Generalization error falls dramatically for α values between 1.1 and 1.6, then get stable for α values between 2.1 and 4, then shows linear increase. Consequently, we concluded that by applying Minover algorithm on a given noisy dataset, the final weight vector $w(t_{max})$ with stability $k(t_{max})$ for a given set of data fails to approximate the perceptron of optimal stability w_{max}.

4.2 Bonus Extension 2

After each training process is completed, we determined the individual stabilities $k^v(t_{max})$ and generate histograms of the frequency of observed stabilities in Figure 3. We concluded that most frequently obtained

stability values are in range between 8 and 15, and between 24 and 27 with $frequency = 5$.

5 Conclusion

Minover algorithm is applied on a dataset to achieve the optimal stability of the perceptron [1]. This algorithm was implemented while learning of a linearly separable rule from the example data. We determined the generalization error $\epsilon_g(t_{max})$ for different values of P as a function of $P = \alpha * N$. We used the values of $\alpha = 0.25, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0$, and the training is repeated on $n_D = 30$ randomly generated dataset, where each data point has $N = 20$ features. The training process is repeated until the maximum number of iterations $t\_max = n\_max*P$ is reached.

We observed that the generalization error $\epsilon_g(t_{max})$ seems to form a kind of a linear relationship with the number of data points that we had. We concluded that the Minover algorithm will force the w to tend to w* with increasing training runs and availability of more data.

In Bonus Extension 1, we considered learning from noisy examples by replacing the true labels in the dataset by implementing equation 5, where $0 < \lambda < 0.5$ controls the noise level in the training data and obtained the Figure 2. We observed that Minover algorithm applied on a noisy dataset results fluctuations on a learning curve, which means the final weight vector $w(t_{max})$ with stability $k(t_{max})$ for a given set of data fails to approximate the perceptron of optimal stability w_{max}.

In Bonus Extension 2, we determined the individual stabilities $k^v(t_{max})$ at the end of training and generated histograms of the frequency of observed stabilities in Figure 3. Most frequently obtained stability values are in range between 8 and 15, and between 24 and 27 with $frequency = 5$.

References

[1] Michael Biehl, "D2-Perceptron: capacity of a hyperplane, learning a rule in version space, optimal stability ", *"An Introduction to Neural Networks and Computational Intelligence, Materials from MSc level Course"*, 2020.

Appendix A Minover Algorithm

```python
import numpy as np
import matplotlib.pyplot as plt
import math
import random

# Implementation of Minover algorithm
# Returns generalization error and the stability at the end of the training

def min_over(P, N, n_max):

    # generate data:

    labels, feature_vectors, teacher_weights = generate_data(P, N)

    weight_vector = np.zeros(N)
    min_index = 0
    t_max = 0

    while t_max <= n_max * P:
        min_stability = float("inf")

        for i in range(P):
            data_point = feature_vectors[i]
            label = labels[i]

            #calculate stability:

            stability = np.dot(weight_vector, data_point) * label

            # determine minimum stability:

            if stability <= min_stability:
                min_stability = stability
                min_index = i

        # Hebbian update:

        min_data_point = feature_vectors[min_index]
        min_label = labels[min_index]

        weight_vector += (1/N * min_data_point * min_label)

        # Increase number of epoch:

        t_max += 1

    # generalization error:

    numerator = np.dot(weight_vector, teacher_weights)
    denominator = np.linalg.norm(weight_vector) * np.linalg.norm(teacher_weights)

    error = (1 / math.pi) * math.acos(numerator / denominator)

    return error, min_stability
```

Appendix B Distribution of the workload

The main part of the project is completed with the equal contribution by each group member.

Ayça Avcı implemented the bonus extensions and added them to the report.