Neural Networks - Assignment 2

Manvi Agarwal - s2817047
Daniel Miedema - s2478579

February 22, 2019

1 Problem Statement

This report presents an implementation of the Minover algorithm. An artificial dataset with N dimensions and P examples is created, and a perceptron is trained with this algorithm in order to generate the weight vector with optimal stability. By doing this several times, we obtain the learning curve. This captures the relationship between the average generalization error, and the nature of the dataset, quantified by $\alpha = \frac{P}{N}$.
In Section 2, the method used to generate a randomized dataset using a teacher perceptron is presented. In Section 3, the Minover algorithm is introduced, and the procedure of updating the weight vector is presented. In Section 4, the simulation setup is explained and results are presented. In Section 5, the results are discussed. In Section 6, we extend the findings from Section 5 by first, motivating our choice of the teacher perceptron as a vector of ones, and then examining the relationship between the stability of the weight vector obtained at the end of the training process, $\kappa(t_{max})$, and α.

2 Dataset Generation

The dataset $\mathbb{D} \in \mathbb{R}^{P\times(N+2)}$ is generated by drawing P N-dimensional feature vectors, adding a clamped input to each feature vector, and specifying binary labels ± 1. Each element of the feature vectors $\mathbb{D}_{ij}$, where $1 \leq i \leq P$ and $1 \leq j \leq N$, is a uniform pseudo-random number, selected by using the `randn` command. A dataset, thus, can be represented as $\mathbb{D} = \{\boldsymbol{\xi}^\mu, 1^\mu, S^\mu\}_{\mu=1}^{P}$, where $\boldsymbol{\xi}^\mu$ refers to a feature vector, S^μ refers to its corresponding label, and 1^μ is the clamped input. Instead of selecting the labels randomly, we select a teacher perceptron $\mathbf{w}^*$ such that $|\mathbf{w}^*|^2 = N$. In this assignment, we always take the teacher perceptron to be a vector of ones. We motivate this choice in Section 6.1. Then, we obtain the labels S^μ with:

$$S^\mu = sign(\mathbf{w}^* \cdot \boldsymbol{\xi}^\mu)$$

This ensures that the dataset so obtained is guaranteed to be linearly separable, since we do not define the labels randomly, but set them according to the 'actual solution'. This way, there is no noise in the data and we can aim for a solution vector with good generalization ability.

3 Perceptron Training with the Minover Algorithm

In each epoch n_i, where $1 \leq i \leq n_{max}$, there are P time steps. At each time step $t = 1, 2, ..., P$, we calculate the stability of each example $\boldsymbol{\xi}^\nu$, defined as:

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

We select the example $\mu(t)$ which has the minimal stability $\kappa^{\mu(t)} = min_\nu\{\kappa^\nu(t)\}$. The weight vector is updated with a Hebbian step, defined as:

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

If the angle between $\mathbf{w}(t+1)$ and $\mathbf{w}(t)$ is below a certain threshold (2 degrees in our code), or if the maximum number of training steps $t_{max} = n_{max} \cdot P$ has been reached, we stop the training. Now, we calculate the generalization error on the weight vector so obtained as:

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

We repeat this multiple times for different values of α. For each α, we use different values of n_d, which represents the number of datasets we generate and run the Minover algorithm on. Thus, for each combination of α and n_d, we can obtain an average generalization error ϵ_{avg}. Using this, we determine the learning curve, which is a plot showing the the value of ϵ_{avg} as a function of $\alpha = P/N$ for each n_d.

This procedure is outlined in the code in Listing 1.

Listing 1: Generating random datasets

```
N = 20; % dimensions of input
alphastart = 0.25;
alpha_growth = 0.25;
alpha_steps = 19;
P = N*alpha; % number of examples
n_max = 300; % max number of epochs to complete training
w_teacher = ones(N+1,1); % teacher perceptron N+1 by 1
% test for 10, 50, 100, 200 datasets
n_d_list=[10 50 100 500]

for num=1:4
    alpha = alphastart;
    n_d = n_d_list(1,num)
    % obtain an average value of the generalization error for different alpha
    avrErPerAlpha =[];
    % repeat for different values of alpha
    for i=1:alpha_steps
        alpha= alpha+alpha_growth;
        P = N*alpha;
        genErrVector=[];
        % repeat for n_d number of datasets
        for num=1:n_d
            % Initialization
            D = data4(N,P, w_teacher); % P by N+2 matrix
            w = zeros(1,N+1);
            % train the perceptron for one dataset
            for epoch = 1:n_max
                stabilities = [];
                % calculate stabilities for all examples
                for t = 1:P
                    data_point = D(t,:);
                    stability = dot(w, data_point(1:N+1))*data_point(length(
                        data_point))/norm(w);
                    stabilities = [stabilities stability];
                end
                % select example with minimum stability
                [min_value,min_index] = min(stabilities);
                min_stability_data_point = D(min_index,:);
                % update weight vector using this example
                vector_1 = w;
                w = w + ((1/N+1)*min_stability_data_point(1:N+1)*
                    min_stability_data_point(length(min_stability_data_point)));
                vector_2 = w;
                % check if the angle is small enough and training should stop
```

```
                angle = rad2deg(acos(dot(vector_1,vector_2)/dot(norm(vector_1),
                    norm(vector_2))));
                if angle < 2
                    break
                end
            end
            % calculate generalization error for this dataset and store it
            genErr = 1/pi*acos(dot(vector_2, w_teacher)/(norm(vector_2)*norm(
                w_teacher)));
            genErrVector= [genErrVector genErr];
        end
        % calculate the average generalization error for n_d datasets
        avrErPerAlpha = [avrErPerAlpha transpose([alpha, mean(genErrVector)])]
    end
    % plot average generalization error as a function of alpha for different
        values of n_d
end
```

4 Simulation & Results

The algorithm was evaluated by computing ϵ_{avg}, the average generalization error, and plotting it as a function of $\alpha = P/N$. This was done for a varying number of random datasets, n_d, which was set as 10, 50, 100, and 500. The number of dimensions, N, was fixed at 20, and whereas the maximum number of epochs, n_{max}, was fixed at 300. α varied between 0.25 and 5.0, with steps sizes of 0.25. This means that for a fixed N, the number of examples, P, changed with the change in α. Fig 1 shows the results.

5 Discussion

As expected, the average generalization error, ϵ_{avg}, becomes smaller as α goes higher and more examples are given for the algorithm to train on. Interestingly, this holds for all four values of n_d. This means that providing more datasets for the same value of α does not give a significant improvement in ϵ_{avg}, although higher values of n_d leads to smoother line as the average is calculated over more number of datasets. Furthermore, as α increases, ϵ_{avg} approaches the limit of 0, meaning that the final weight vector will be in the same direction as the teacher vector and will, in fact, be a multiple of the teacher vector. This means it can classify all the examples correctly.

Our intuition is that these results will generalize over other parameter settings for the Minover algorithm. Thus, for a low generalization error, for instance if we want $\epsilon_{avg} \leq 10\%$, we now know to make sure that we set the value of P, the number of examples in the dataset, to be at least 5 times the value of N, the number of dimensions for training data. In our opinion this is surprisingly low. However, given that most classification problems come with noisy data, unlike our implementation, this will probably be a higher number in practice.

6 Bonus Exercises

6.1 Choice of teacher perceptron

The components of the feature vectors $\boldsymbol{\xi}^\mu \in \mathbb{R}^N$ are selected in an independent, random manner such that $\boldsymbol{\xi}_j^\mu \sim \mathcal{N}(0,1)$. This means that if we plot the data in $\mathbb{R}^N$, with a high probability, it will be strongly clustered around the origin. This indicates that any datasets like this will be rotationally invariant under coordinate transformation, meaning that if the axes are rotated around to change the coordinate system, in such a way that they are still orthogonal to each other, the nature of the dataset will not change. Further, suppose that, in the 'original' coordinate system, we select a teacher perceptron $\boldsymbol{w}^*$ to compute the labels such that $|\boldsymbol{w}^*|^2 = N$, but $\boldsymbol{w}^* \neq (1, 1, ..., 1)^T$. Then after the coordinate transformation, this particular $\boldsymbol{w}^*$ becomes $(1, 1, ..., 1)^T$ in the new coordinate system. Hence, the choice of $\boldsymbol{w}^* = (1, 1, ..., 1)^T$ is not a restriction in the context of this practical, but only corresponds to our choice of the coordinate system.

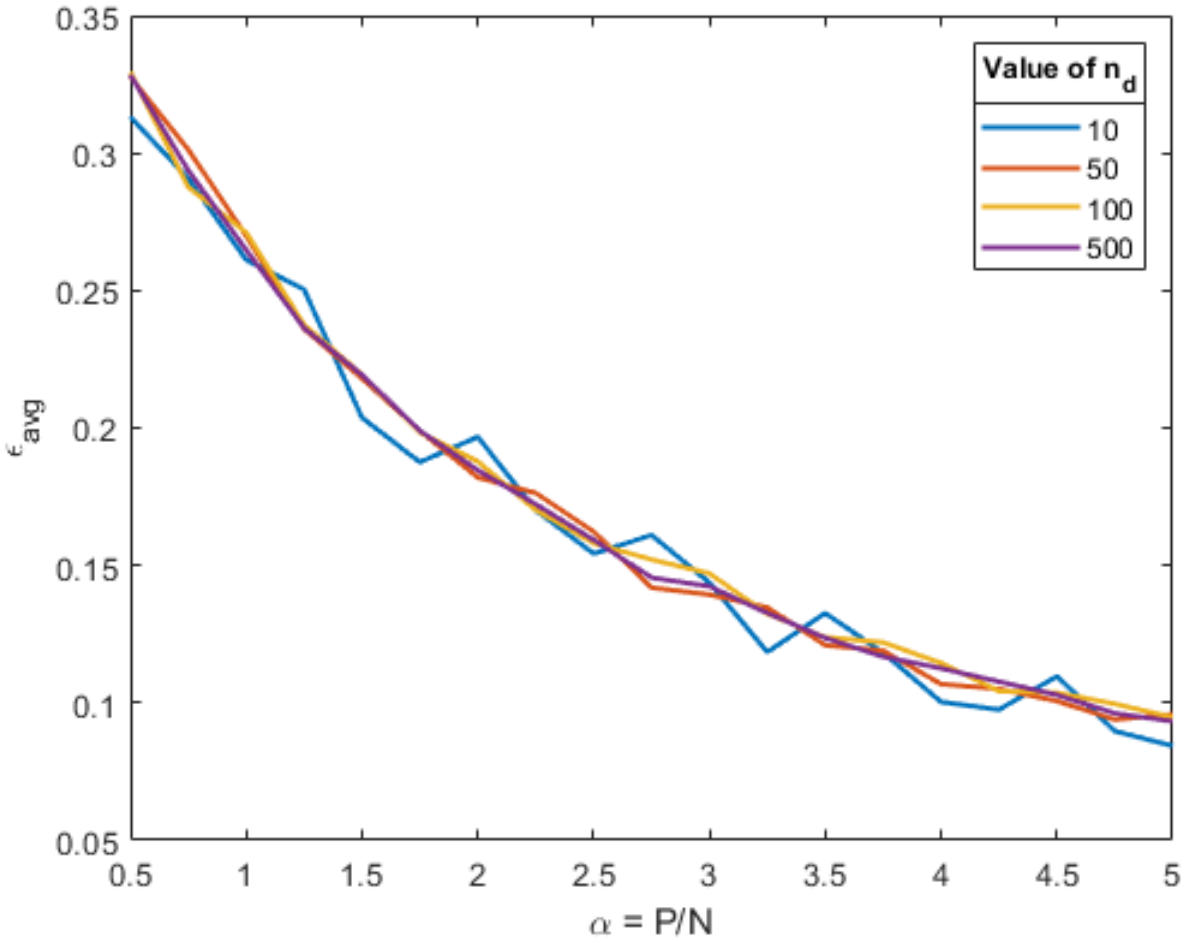

Figure 1: Investigating the average generalization error ϵ_{avg} as a function of α: $N = 20$, $n_{max} = 300$

6.2 Comparing optimal stability $\kappa(t_{max})$ with random labels and teacher-assigned labels

The Minover algorithm identifies the example with the minimum stability and updates the weight vector in such a way that this value is maximized.

When a teacher perceptron $\boldsymbol{w}^*$ is used to define the labels, the dataset will always be linearly separable. As a consequence, the stability will never go below 0. However, as the number of examples increases, the chances of a worse stability are also higher, as more space around the separating hyperplane gets filled with examples, and the maximum possible value of the minimum stability decreases. This explains the value of $\kappa(t_{max})$ approaching 0 as the value of α increases, in Fig. 2.

When labels are set randomly, the dataset is noisy, and there is no guarantee that it will be linearly separable. Hence, it is possible that the example selected as having the minimum stability is, in fact, misclassified. This means that maximizing this minimum stability value will lead to $\kappa(t_{max})$ being negative. As α goes up and more examples are added, the chances of having an example that has a worse minimum stability in a dataset that is not linearly separable becomes higher. This leads to the decreasing value of $\kappa(t_{max})$, even going into negative, as shown in Fig. 3.

Even when the labels are assigned randomly, if the value of α is small, the dataset will be linearly separable with a high probability. This was also seen in the Rosenblatt perceptron from Assignment 1. This explains why $\kappa(t_{max})$ starts off positive in Fig. 3. We also observe that the range of the values of $\kappa(t_{max})$ is dependent on the value of the weights. When the dataset is not linearly separable, the weight vector can end up oscillating between different maxima of minimum stabilities, that do not necessarily occur closer together. This allows the values of the weight vector to get more extreme as the number of oscillations increase.

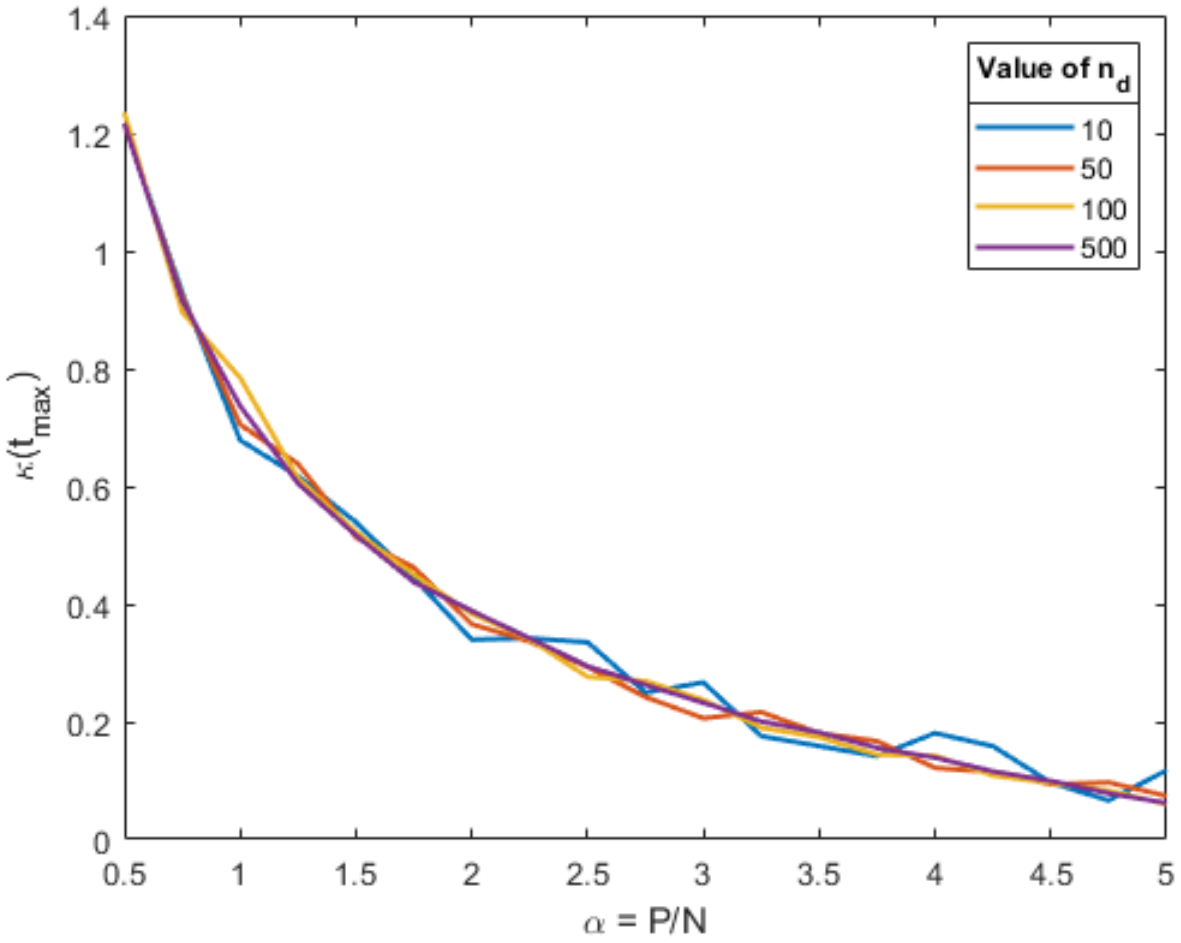

Figure 2: Investigating the average stability of the final weights vector $\kappa(t_{max})$ as a function of α for teacher generated target labels: $N = 20$, $n_{max} = 300$

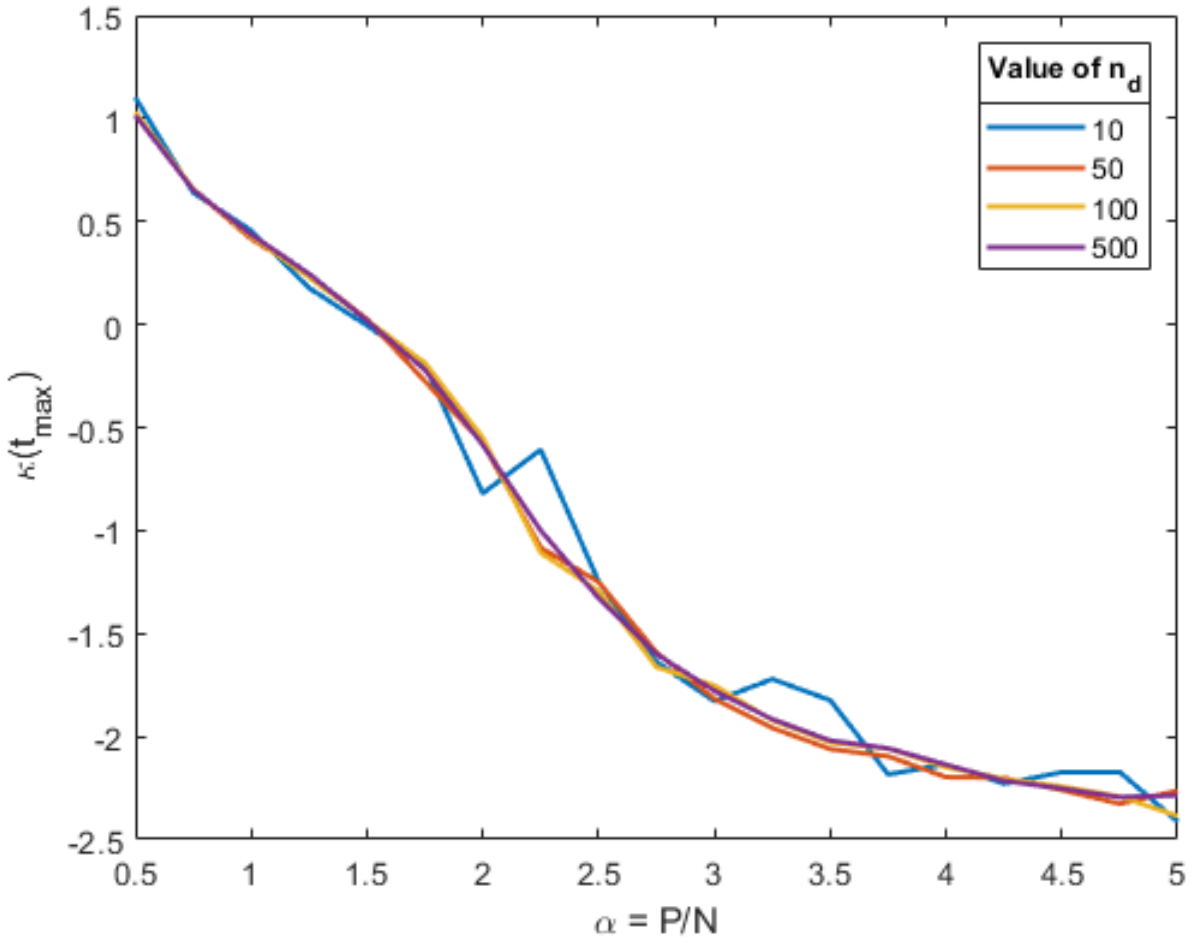

Figure 3: Investigating the average stability of the final weights vector $\kappa(t_{max})$ as a function of α for random target labels: $N = 20$, $n_{max} = 300$