Neural Networks
Assignment II - Learning a rule
Group 10

Satyanarayan Nayak, Swastik
s.nayak.1@student.rug.nl
S4151968

Nivin Pradeep Kumar
n.pradeep.kumar@student.rug.nl
S4035593

January 31, 2020

Contents

Abstract

Perceptron is a powerful intelligent unit that functions as the building blocks a simply and/or complex Neural network(s). Perceptrons can be interpreted as a single McCulloch Pitts unit which is connected to N input neurons[1]. Minover algorithm attempts to find the linear separability of the training data by updating the least stable point. We will simulate the algorithm on varied data sizes with different levels of noise. We will also compare the minover algorithm against rosenblatt and kmeans to gain some valuable insights.

1 Introduction

The perceptron is the simplest form of a neural network used for classification of patterns said to be linearly separable. It is an algorithm for supervised learning of binary classifiers. [1] Binary classification is the task of classifying the elements of a given set into two groups based on a prescribed rule. Rosenblatt's perceptron can handle classification tasks for linearly separable classes. In this assignment, we define and implement a teacher perceptron, which implements the Minover algorithm. We also determine the generalized error at the end of the training process. In this report, we can get an understanding of learning of a linearly separable rule from example data.

2 Teacher and Student Perceptron

For data sets of reliable noise-free examples, it is not clear that a perceptron can reproduce the labels in D correctly since the target might not be linearly separable. To simplify our considerations, we restrict ourselves to cases, in which the rule is given by a function of the form $S_R(\xi) = sign(w^*.\epsilon)$ for a particular weight vector w^*. All weight vectors λw^* with $\lambda > 0$ would define the same rule. A perceptron with weights w^* is always correct. It is therefore referred to as the teacher perceptron (or teacher, for short) and can be thought of providing the example data from which to learn.[1]
The student perceptron is the optimal weights that are determined by training a perceptron by sequentially feeding the input data $D = \{\xi, S\}$ provided by the teacher perceptron. The student perceptron can be any weight vector within the vector space. The goal of the student perceptron is to learn from the example data provided by the teacher perceptron to ideally match its weight vector W with the teacher perceptron's weight vector $W*$ [1].
The convergence is attained when there are minimal changes in the weights of the student perceptron. The generalization error ϵ_g of the student is determined by finding the inverse cosine between the weight vector of the student and the weight vector of the teacher.

$$\epsilon_g = \frac{1}{\pi} arccos(\frac{W.W^*}{|W||W^*|}) \tag{1}$$

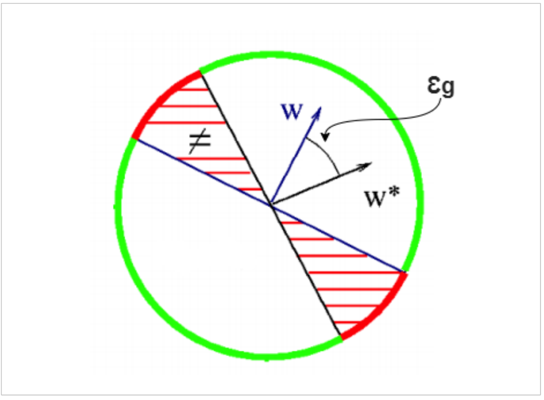

Figure 1: Generalization error of the perceptron in a student/teacher scenario [1]

2.1 The MinOver Algorithm

An intuitive algorithm that achieves optimal stability for a given linear separable data set D. The so-called MinOver algorithm performs Hebbian updates for the currently least stable example in D. We assume here tabula rasa initialization, i.e. $w(0) = 0$. The perceptron always aims at improving the least stable example [1]. As there is a Hebbian update at every learning iteration we cannot use the local potential $E^\mu > 0$ as convergence or stopping criteria. The convergence criteria of the Minover algorithm will be to stop the algorithm when there are no significant differences in the weight vectors, formally we can denote this condition as $(W(t-1) - W(t)) < Threshold.$

Algorithm 1: MinOver Algorithm [1]

```
At discrete time step t with current weight W(t)
- Compute the local potent $E^\mu(t) = w(t).\xi^\mu S_T^\mu$ for all examples in $D$
- Determine the index $\hat{\mu}$ of the training example with minimal overlap, i.e. with the currently lowest local
  potential.
      $E^\mu = min\{E^\mu(t)\}_{\mu=1}^{P}$          (2)
  - Update the weight vector according based on the index $\hat{\mu}$ as follows,
      $W(t+1) = W(t) + \frac{1}{N}\xi^\mu S_T^\mu$          (3)
or, equivalently, increment the corresponding embedding strength
      $x^\mu(t+1) = x^\mu(t) + 1$          (4)
```

3 Implementation

3.1 Data Generation

The first step is to generate artificial data sets containing P randomly generated N-dimensional feature vectors and binary labels $D = \{\xi^\mu, S^\mu\}_{\mu=1}^{P}$. Here, the $\xi^\mu \in R^N$ are vectors of independent random components $\xi_j^\mu \sim N(0,1)$ with mean zero and unit variance. The example data is generated using the matlab's **`randn(P, N)`** function, and the labels $S = \{+1, -1\}$ are generated by using the weights of the teacher weight vector $S_R = sign(\xi * W^*)$, which can also be extended to incorporate noisy labels.
The size of the example data P is determined using the relationship $P = \alpha * N$, where α is the storage capacity of the perceptron. For a series of alpha values ($\alpha = 0.75, 1.0, 1.25, ...3.0$) the fixed dimensions of the data N is used to determine the value of P.

3.2 Data Generation with noise

The steps involved in generating the noisy data is similar to as discussed in section 3.1, with an addition of the below, mentioned conditions to mutate the labels to simulate noisy data.

$$S^\mu = \begin{cases} +sign(W^*.\xi^\mu) & \text{with probability } \lambda - 1 \\ -sign(W^*.\xi^\mu) & \text{with probability } \lambda \end{cases} \tag{5}$$

We achieved this by using `rand(P, 1)` to generate a series of P number of values in the range of [0,1], if the value in the generated list is less than the probability λ, we will mutate the label in that index position to simulate the noise in the data.
The noisy data is used to study the scenarios where the teacher perceptron fails to provide the correct labels, as one of the assumptions made in the teacher-student perceptron model is that the teacher is always correct. But in a real-world scenario, it is highly unlikely that the teacher will be the absolute truth.

3.3 Training

Before the training process is started the weights of the perceptron are set to Tabula Rasa's $W(0) = 0$. The example data is fed to the system in the form of cyclic representation $\mu = 1, 2, 3, ...P, 1, 2, 3....$, where μ is the index position of the example data.
The algorithm is a sequentially executed for a maximum epochs $t_m ax = n_{max} * P$ times, where at any given time $t = 1, 2, 3.....t_{max}$ where an example of data ξ^μ is presented to the perceptron for learning. The minimum local potential is determined and a Hebbian update is performed on the weights of the perceptron as discussed in section 2.1. The generalization error $\epsilon_g(t_{max})$ is determined using the equation 1, the error value measures the deviation of the student weight or perceptron from the teacher weight or perceptron. Multiple simulations are run by tweaking independent parameters to understand the effects of the parameter over the Minover algorithm.

4 Simulations and Evaluations

4.1 MinOver general scenario

In this scenario we keep the the dimension (N) of the example data fixed and determined the generalization error for different values of $alpha = 0.1, 0.2, 0.3, ..., 5.0$, the noise in the system $\lambda = 0$. From figure 1 we can see that for a higher value of alpha which implies more number of example data used for training ($P = \alpha * N$), the generalization error drops significantly. This is expected since every example data ξ^{μ} in a teacher-student model is treated as decision boundary that might cut the version space thereby contributing towards the learning of the student perceptron resemble to that of its teacher perceptron.
We can say that as the alpha tends to infinity the angle between the student and the teacher perceptron will be zero, i.e the student weight vector and the teacher weight vector will be parallel to each other and converge at the same point, mathematically represented as.

$$\lim_{\alpha \to \infty} W = W*$$

In our simulations, we assume that the teacher perceptron weight $W* = (1, 1, 1, ...1)^T$ as this does not violate the loss of generality. The labels for a given data are generated by using the weights of the teacher weights and the labels will be utilized by the student perceptron for learning, so any required state of the teacher perceptron is acceptable. Also for any weight W, that can linearly separate the example data than its scalar multiple $W_2 = \lambda * W$ will also produce linearly separable boundaries, due to the linearity of the scalar product [1].

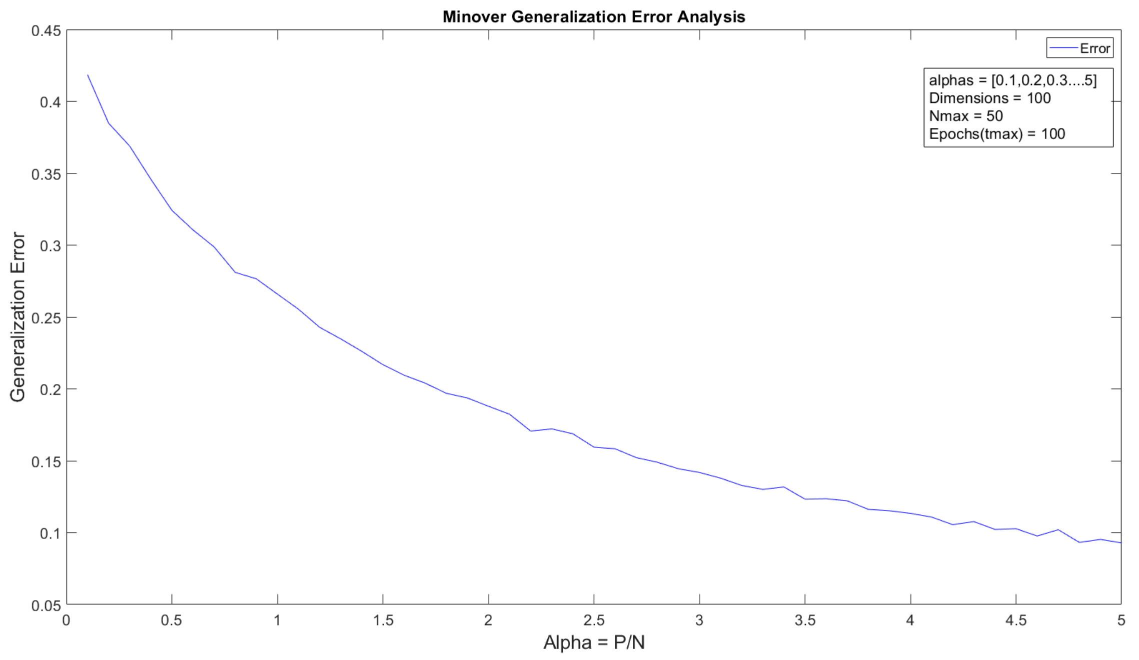

Figure 2: Generalization error of MinOver algorithm

4.2 MinOver vs Rosenblatt perceptron algorithm

In this simulation, we will compare the generalization errors produced by training the example data over the MinOver algorithm and Rosenblatt perceptron algorithm.
Figure 3 shows the results of these simulations, we observe that there is negligible difference between the Minover and Rosenblatt algorithm in the absence of noise (λ). The minover algorithm performs relatively better than rosenblatt algorithm at lower alpha values ($alpha = [0.1, 0.2, ...1.5]$) but this small gain comes at a price as the minover algorithm is relatively computationally expensive in comparison to the rosenblatt perceptron algorithm. At $\alpha = 5.0$, the minover algorithm takes 12 seconds and the rosenblatt algorithm takes 0.5 seconds to finish its execution, which is a significant difference in the performance of 1:24.

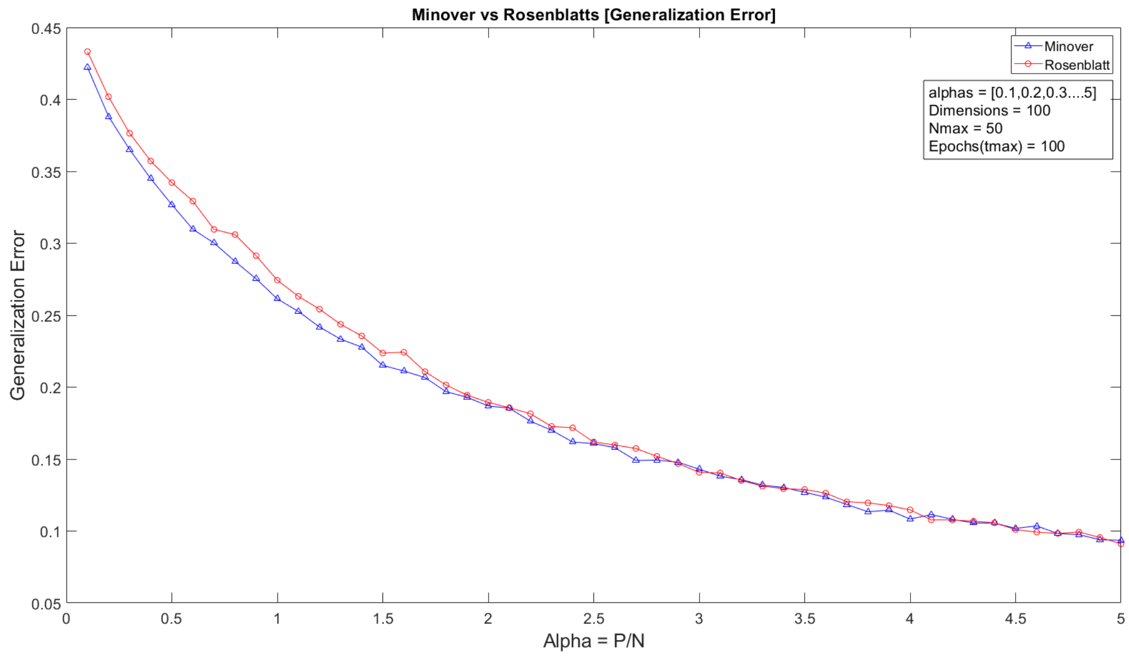

Figure 3: Generalization error for minover and rosenblatt algorithm

4.3 MinOver vs Rosenblatt perceptron algorithm with noise

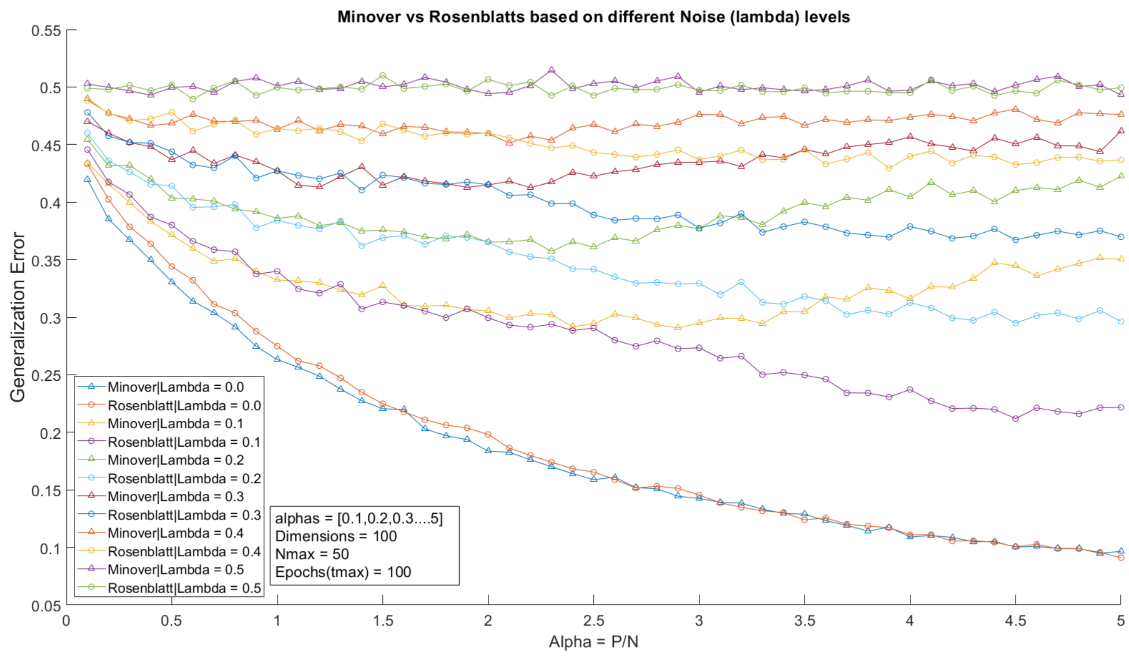

Figure 4: Generalization error for minover and rosenblatt algorithm with different noise

In this simulation, we simulated both the minover algorithm and the Rosenblatt perceptron algorithm under the influence of different levels of noise. Figure 4 shows the results of this simulation. The generalization error in the absence of the noise is added for relative comparison. For the majority of the scenarios, we can see that generalization error at lower alpha values is either decreasing slightly or constant, and at alpha values 2.5 to 3.0, the generalization error starts rising again. This rise in generalization error is due to the increase in the example data observed at alpha values 2.5 and 3.0 as $P \propto \alpha$. At noise level, $\lambda = [0.4, 0.5]$ we can see that the generalization error is approximately equal to $0.5 = 50\%$, the probability to classify the example data correctly is equivalent to a coin flip, which indicates that it is as good as generating the labels randomly with a probability of 1/2. We can infer that minover and Rosenblatt algorithms are very sensitive to noise, which is expected as they adopt the learn from mistakes concept of learning.

4.4 MinOver vs Rosenblatt perceptron vs kmeans algorithm

In this simulation, we are exploring a curious question of comparing the performance of perceptron based binary classifiers against the off the shelf Kmeans algorithm. Figure 5 shows the results of this simulation, the Kmeans performs well at lower alpha values and performs badly at higher alpha values. One explanation for this behavior could be that the data generated is normally distributed and does not truly have distinct cluster formation. When the number of samples was less the Kmeans were able to maintain two different clusters, and as the size of the data increases the distinction between the two clusters started diminished and the Kmeans were unable to classify the labels correctly. We can learn from this simulation that a data set that is performing efficiently over a perceptron based system might not necessarily work for a distance-based clustering algorithm like Kmeans. Formally the statement holds for all the different types of algorithms.

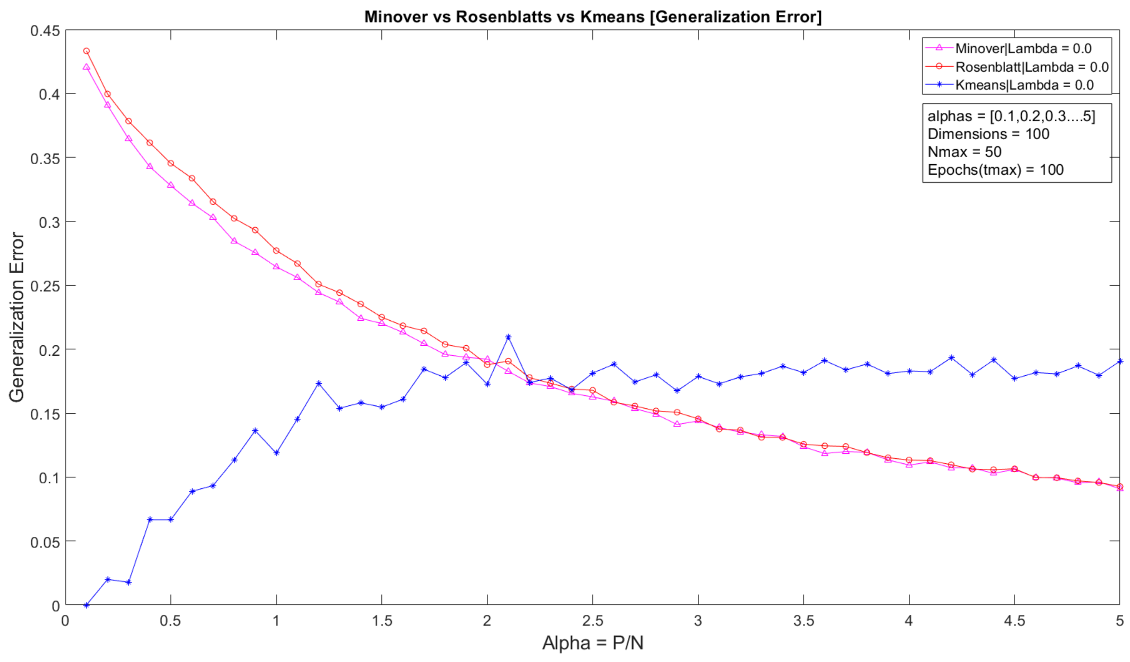

Figure 5: Generalization error for minover, rosenblatt and Kmeans algorithm with different noise

4.5 Adaline algorithm

In this simulation we will train the perceptrons using the adaline algorithm for different learning rates. As a preliminary step we detemined the maximum allowed learning rate based on our data to satisfy the convergence criteria.

$$0 < \hat{\eta} < \frac{2}{max\{C^{\mu\mu}{}_{\mu=1}^{P}\}} \tag{6}$$

$$C^{\mu\mu} = \frac{(\xi^{\mu})^2}{N} \tag{7}$$

From our analysis we determined that a learning rate within the range of 0.1 to 0.3 was feasible for our study. Figure 6 shows the results of the simulation for different learning rates, the generalization error of the adaline is similar to the generalization error produced by the rosenblatt and minover algorithms as shown in figure 3. Although we were unable to reduce the generalization error using the adaline, the computational load required to train the perceptron was drastically reduced with execution times ranging from 0.002 seconds - 0.12 seconds - 0.48 seconds (low-median-high). The learning rate allows us to fine tune the algorithm into sufficiently small steps in order to converge quickly, thus boosting the performance of the adaline algorithm.

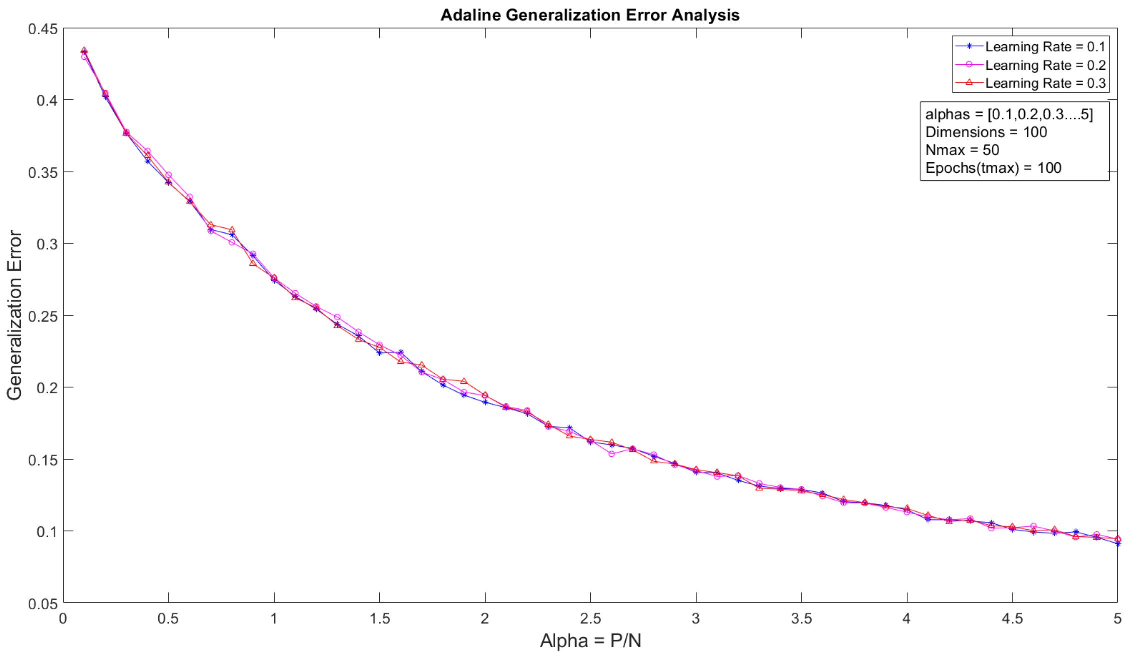

Figure 6: Generalization error of Adaline algorithm

5 Conclusion

In the assignment, we have simulated various scenarios about Minover algorithm and compared it against similar algorithms. The minover algorithm attempts to maximize the stability of the system but adding Hebbian updates to the least stable training example. The minover algorithm has a similar generalization error to that of the Rosenblatts algorithm at lower noise levels and at higher noise levels, both minover and Rosenblatts algorithms diverge significantly from the teacher perceptron. The divergence worsens as the size of the noisy training data increases.
The minover algorithm is computation extensive with a 1:24 ratio in comparison with the Rosenblatt algorithm under the same environment, and as discussed in section 4.2. There is no significant gain in minimizing the generalization error by using the minover algorithm over the Rosenblatt perceptron algorithm. The significant increase in the computational power requirement is due to the $t_{max} = n_{max} * P$ of minover algorithm being larger in comparison to the Rosenblatt algorithm, and also at each epoch step the minover algorithm computes the minimum stability for all the example data points. The rosenblatt algorithm can converge early when there is no significant weight updations, but the minover algorithm cannot converge early due to Hebbian updates being performed in every epoch.
We compared the minover and Rosenblatt algorithm with the off the shelf Kmeans algorithm and inferred that Kmeans performed worse as the number of training examples increases but on the contrary, the perceptron based models perform significantly better (discussed in section 4.4).
We also trained the example data over the adaline algorithm and compared it against the minover and rosenblatts generalization error, we gained an valuable insight that the adaline algorithm is computationally efficient without any loss of accuracy.

6 Individual contributions

Satyanarayan Nayak, Swastik (**S4151968**):

- Implementation of basic MinOver algorithm.
- MinOver vs Rosenblatt comparison with noise.
- MinOver vs Rosenblatt vs Kmeans comparision.
- Adaline algorithm implementation and evaluation.

Nivin, Pradeep Kumar (**S4035593**):

- Generation of the example data and labels

- Assisted with the implementation of basic MinOver algorithm.
- MinOver general scenario.
- MinOver vs Rosenblatt comparison without noise.

We have worked on the report together, and have focused on the topics that we had worked on. But we have cross verified each others work and have refined this report together.

References

[1] *Learning from Examples - An Introduction to Neural Networks and Computational Intelligence.* 1994.

7 Appendix

7.1 Minover algorithm implementation

```
function [student] = minover(data, labels, epochs)
    column_count = size(data,2);
    row_count = size(data,1);
    student = zeros(column_count,1);
    threshold = 1e-3;
    % The iteration is executed for a maximum of N_max * T_max times
    for t=1:epochs*row_count

            % Used later to check convergance (method commonly used in
            % kmeans to find convergance)
            prev_student_weights = student;

            % Find the least stable point
            % Ignoring |student| denominator as it doesn't affect the
            % results (scale factor)
           [~, indx] = min((data * student) .* labels);

           % Update the least stable point with the hebbian term.
           student = student + (data(indx,:)' * labels(indx))/column_count;

           % Check if the weights changes are below the threshold level for early termination.
           if all(norm(prev_student_weights - student) / norm(student) < threshold)
             break
           end
    end
end
```

7.2 Data generation

```
function [data, labels] = generate_data_with_labels(P, N, teacher, lambda)
    data = randn(P, N);
    % Generate an array of P size with 0-1 values, values that are less
    % than lambda will be chosen as the noise labels
    noise = cmp_labels(rand(P, 1) < lambda, -1, +1);
    % Multiply noise, if old label = -1 and then a -1 * -1 = flip to +1
    labels = sign(data * teacher).*noise;
end
```

7.3 Compare Labels

```
% Ref: https://stackoverflow.com/questions/14612196/is-there-a-matlab-conditional-if-operator-
    that-can-be-placed-inline-like-vbas-i#answer-14613083

function result = cmp_labels(condition, a, b)
    if isscalar(condition)
       if condition
           result = a;
       else
           result = b;
       end
    else
        result = (condition) .* a + (~condition) .* b;
    end
end
```

7.4 Adaline learning rate checker

```
% Checks if the learning rate selected is suitable to achieve convergence.
function lr = check_valid_learning_rate(lr, data, N)
    C = [];
    for j=1:size(data,1)
        xi = data(j);
        C =  norm(xi)/N;
    end
    limit = 2/max(C);
    if lr > 0 && lr < limit
        lr = lr;
    else
        fprintf("Invalid learning rate, max possible learning rate %f \n", limit-0.1);
        lr = limit - 0.1;
    end
end
```

8 Readme

GitHub link : https://github.com/Swastik-RUG/NeuralNetworks/tree/master/matlab/minover

Simulation scenario	**Matlab file to execute**
MinOver general scenario	base.m
MinOver vs Rosenblatt perceptron algorithm	minover_vs_rosenblatt.m
MinOver vs Rosenblatt perceptron algorithm with noise	minover_vs_rosenblatt_noisy.m
MinOver vs Rosenblatt perceptron vs kmeans algorithm	minover_rosenblatt_kmeans.m
Adaline algorithm	adaline_simulation.m