EE 565 Machine Learning
Project 3 Report

Mehrdad Ghyabi

***Abstract*—This is a summarized report about the perceptron theory and its implementation in classification problems, different learning algorithms for perceptron theory, and first-order and second-order gradient algorithms and their performance on a few famous cost functions. These algorithms can be implemented in various machine learning methods as a part of learning process.**

***Index Terms*— perceptron, online learning, batch learning, learning rate, gradient descent, second-order gradient, Rosenbrock's cost function, Himmelblau's cost function, Newton's method, Levenberg-Marquardt Method**

I. Introduction

This report is consisted of eleven sections. The first three sections after the introduction focus on the performance of perceptron theory and two different learning algorithms, online learning, and batch learning. Sections V to VII are focused on gradient descent algorithm and its performance on three different cost functions, quadratic, Rosenbrock's, and Himmelblau's. Sections VIII to X are about the performance of two second-order descent optimization algorithms, Newton's method and Levenberg-Marquardt method. Their performance is evaluated on quadratic and Himmelblau's cost functions. Last section is a concise summary of the work done and the results.

II. Problem 1: Perceptron Model

In this part, the algorithm of perceptron was implemented into a function. This function was used the tasks defined in the next two problems. Since this function is going to be used many times in the next two problems, it had to be written in an efficient way, in order to save time. This function was written in a way that it can implement the perceptron algorithm in just two steps, whether the first input ($\boldsymbol{x}$) is a single data point or a data set. The matrix multiplication was used to achieve that goal. The function was formulated using the equations below, in which $\boldsymbol{x}$ is the input data node/set, $\boldsymbol{w}$ is the weight vector, φ is a nonlinear function, and y is the output/label.

$$y = \varphi(\boldsymbol{w}^T \boldsymbol{x}) \tag{1}$$

$$\varphi(v) = \begin{cases} +1, & v > 1 \\ 0, & v = 1 \\ -1, & v < 1 \end{cases} \tag{2}$$

III. Problem 2: Online Perceptron Learning

In this section, different aspects of the online perceptron learning are investigated. Datapoints are drawn from a double moon distribution with r = 1 and $w = 0.6$ and variable d_{DM}. After completing the Python code, different learning rates were used to solve the problem. To get results that are presented in this section, a learning rate η = 0.001 was selected. Also, the order of datapoints was shuffled before each epoch.

The updating at each step inside an epoch is performed using the following equation. in which, e_n is the classification error corresponding to the nth datapoint and can be either 1, 0, or -1.

$$\boldsymbol{w} := \boldsymbol{w} + \eta e_n \boldsymbol{x_n} \tag{3}$$

A. Accuracy in case of null initialization

In this part, changes of accuracy of online perceptron learning model during a single epoch is investigated. The assumption here is that the initial weight vector is null. To rule out the uncertainty from datapoints randomness and have a smoother plot, the experiment is repeated 30 times and the average of results are presented. Accuracy is simply the percentage of datapoints that are classified correctly. Cost function is calculated using the following equation in which d is true label and y is the resulting label from classifier.

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} (d_i - y_i)^2 \tag{4}$$

Figures 1 to 6 illustrate cost function and accuracy of the model versus progress of the model in a single epoch, for d_{DM} = 0.0, d_{DM} = 0.5, and d_{DM} = *-0.5.* It is notable that the patterns in cost function and accuracy look like inversion of each other.

In case of d_{DM} = 0.5, the data is classified with the least amount of error and accuracy reaches 1, because the two half-moons are completely separated. In case of d_{DM} = 0.0, the model still works well but not as good as previous case. The worst case among the three, is d_{DM} = -0.5, which more fluctuations in the plots, least accuracy and largest cost function can be observed. It is because the fact that classes cannot be separated using a

straight line.

B. Illustration of sample datasets

For each case of d_{DM} one sample classification is presented in Figures 7 through 9. In each figure, correctly classified members of the class 1 are plotted as "blue +", correctly classified members of class 2 as "green x" and incorrectly classified points are plotted as "red *". These results match with the results from last part. The more divided the two moons are, the accurate the perceptron classifications are.

C. Accuracy in case of random initialization

The procedure of part A is repeated here with one difference. Instead of initializing the weight vector with zero, here the weight vector is initialized with uniform random values between [-0.5, 0.5]. Figures 10 to 15 show cost function and accuracy for data drawn from double moon distribution with d_{DM} = 0.0, d_{DM} = 0.5, and d_{DM} = *-0.5*. The final accuracy is smaller in case of d_{DM} = -0.5, which is because the datapoints are not separate and it is not possible to draw a straight line to separate them.

IV. Problem 3: Batch Perceptron Learning

In this section, different aspects of the batch perceptron learning are investigated. Datapoints are drawn from a double moon distribution with r = 1 and $w = 0.6$ and different values for parameter d_{DM}. After completing the Python code, different learning rates were used to solve the problem. To get results that are presented in this section, a learning rate $\eta = 0.001$ was selected. In this section weights are initialized with uniform random values between [-0.5, 0.5].

In batch learning, only the misclassified data vectors are used to update the weight vector. The updating takes place by using the following equation, in which, M is the set of misclassified data vectors at each iteration. Number of iterations is limited to 100 in this problem.

$$\boldsymbol{w} := \boldsymbol{w} + \eta \sum_{n|x_n \in M} (\boldsymbol{x}_n \boldsymbol{d}_n) \tag{5}$$

A. Implementation of batch learning

A dataset was drawn from double moon distribution and was classified using the batch learning perceptron algorithm. The result is visualized in the Figure 16. In the figure, correctly classified members of the class 1 are plotted as "blue +", correctly classified members of class 2 as "green x" and incorrectly classified points are plotted as "red *". In this case the algorithm classifies the dataset without error. The final weight vector is: [-0.0015 -0.0019 0.2814]. It is notable that it results in a line very close to x axis whose equation is: $y = 0$.

B. Investigating cost function and accuracy

In this part, the general behavior of cost function and training accuracy in different iterations is studied. A dataset was drawn from the double moon distribution with parameter d_{DM} = 0.5. Then, using the exact same database the batch learning algorithm was applied 30 times, each time with a different initial weight vector. Average plots of cost function and accuracy are shown in Figure 17 and Figure 18 respectively.

The cost function average decreases from about 130 to less than 76 and accuracy increases from 0.49 to 0.61. both plots show steady behaviors.

C. Investigating cost function and accuracy part two

The same process from part B was repeated here on a data set drawn from double moon distribution with parameter d_{DM} = 0.0 and average plots of cost function and accuracy are shown in Figure 19 and Figure 20 respectively.

The cost function average decreases from about 113 to 69 and accuracy increases from 0.47 to 0.52. both plots show steady behaviors. The cost function is less than the one from part B because in this part datapoints are nearer to the decision boundary resulting a decrease in the cost function. Here the accuracy is less than last part and it was anticipated because this case should be easier to classify with a straight line. This shows that accuracy may be a better merit to judge about the classifier performance.

D. Investigating cost function and accuracy part three

The same process from part B was repeated here on a data set drawn from double moon distribution with parameter d_{DM} = -0.5 and average plots of cost function and accuracy are shown in Figure 21 and Figure 22 respectively.

The cost function average decreases from about 93 to 55 and accuracy increases from 0.45 to 0.50. both plots show steady behaviors. Here the pattern discussed in part C is repeated, we see a smaller cost function and a worse accuracy, showing that accuracy should be taken into consideration when talking about this classifying problem.

E. Batch learning perceptron for XOR dataset

The same process in part A is repeated here. The only difference here is that the training dataset is drawn from a XOR distribution. A sample result is shown in Figure 23. Because of the nature of the XOR distribution, the perceptron theory and batch learning seem to be performing poorly. The weight vector corresponding to decision boundary shown in Figure 23 is [0.0222 -0.1026 -0.0593], which is very close to the line $y = -x$. based on initialization, this line or the line $y = x$ may be the best result perceptron can give in this case.

F. Cost function and accuracy for XOR dataset

The same scheme in parts B to D of this problem was followed here to study cost function and accuracy of batch learning perceptron in case of data with XOR distribution. average plots of cost function and accuracy are shown in Figure 24 and Figure 25 respectively. The cost function decreases from 69 to 57 and accuracy is around 0.5 throughout the iterations. This shows batch learning perceptron fails to classify this dataset properly, because it tries to reduce the cost function and it does, but reducing cost function doesn't increase the accuracy, because of the specific distribution of datapoints in each class.

V. Problem 4: Gradient Descent with a Quadratic Cost Function

In this section performance of the gradient descent on a quadratic cost function is investigated. Whenever necessary, a thresholding value of 0.01 was used to terminate the iterating loop. The following equation shows the gradient descent weight update.

$$\boldsymbol{w} := \boldsymbol{w} - \eta \nabla J(\boldsymbol{w}) \tag{6}$$

A. Optimizing the learning rate

In this part required iterations for learning rate values between 0 and 0.35 were determined to find the optimized learning rate for gradient descent. The weight vector at each try was set to $[-5,5]^T$ and the error was considered to be the geometric norm (distance) between each point and E^*, the global minimum in the weight space. The results showed that the optimum learning rate is 0.25 with a minimum number of necessary iterations of 12. Figure 26 shows a plot of number of iterations versus learning rate that supports these results.

B. Optimizing the learning rate in a repetitive manner

To be more general in optimizing the learning rate, in this part initial weights were drawn from a Gaussian distribution with $\mu = 0$ and $\sigma = 3$. The rest of the process was like the part A. the algorithm was repeated 30 times at each learning rate value and the average number of iterations was plotted against learning rates in the Figure 27. Results showed that the optimum learning rate is 0.24 with average 9.5 iterations necessary for convergence.

C. Optimizing the learning rate in a repetitive manner

In this part, instead of thresholding on $E(\boldsymbol{w})$, the criteria was chosen to be $J(\boldsymbol{w})$. The reason is that in practice we do not know the location of global minimum. Also, when algorithm gets near the minima, it may be unable to converge because of flatness of the cost function in that area.

Results showed an optimum learning rate of 0.23 in this case, with average iterations of 5. A plot of results is presented in the Figure 28.

D. Line search

A line search algorithm was developed to enhance the gradient descent. After finding the direction to move in the weight space, learning rate was set to 1. This learning rate was modified iteratively by multiplying a modification parameter $\beta = 0.8$ until the updated learning rate result in a cost function smaller than the cost function at the current location. The resulting path in case of a quadratic cost function is shown in the Figure 29. The starting point was set $[-5,5]^T$ and number of iterations until convergence to the same threshold as part A was 16, but after 6 iterations it gets to the vicinity of the global minimum.

VI. Problem 5: Gradient Descent with Rosenbrock's Cost Function

In this section performance of the gradient descent on a Rosenbrock's cost function is investigated. Whenever necessary, a thresholding value of 0.01 was used to terminate the iterating loop.

A. Learning rate effects

In this part a starting point of $[-0.5,1.0]^T$ was chosen to study gradient descent on a Rosenbrock's cost function. In the first trial the learning rate was set to $\eta = 0.002$. The results in Figure 30 show that algorithm did not converge after 2000 iterations. This is because of the particular cost function in this case. Learning rates greater than 0.002 cause the algorithm to lose the path and learning rates less than or equal to 0.002 won't converge because of the flatness of the path to the global minimum.

B. Line search

The same line search algorithm from Problem4D was adopted here. The same characteristic of the cost function prevented the algorithm to converge to the global minimum within 2000 iterations. The line search was able to find the vicinity of the target but as soon as it was trapped in the flat area near the target, it became stuck. The result is illustrated in the Figure 31.

VII. Problem 6: Gradient Descent with Himmelblau's Cost Function

In this section performance of the gradient descent on a Himmelblau's cost function is investigated. Whenever necessary, a thresholding value of 0.01 was used to terminate the iterating loop.

A. Learning rate effects

In this part a starting point was drawn from a gaussian distribution with $\mu = 0$ and $\sigma = 1$ to study the gradient descent on Himmelblau's cost function. Since there are four global minima, the result differs based on the initial point. A sample result is presented in the Figure 32 showing convergence to the threshold value in 8 iterations.

B. Investigating the cost function

With an initialization similar to the previous part, the optimization algorithm was tested with 30 different initial points. Figure 33 shows the average cost function of 30 experiments as a function of iteration. The descending behavior of the plot shows the acceptable performance of the gradient descent in this case.

C. Optimum learning rate

The same scheme as problem4B was implemented in this part to find the optimum learning rate for gradient descent on Himmelblau's cost function. Since there are four possible targets in the weight space and the approached target depends on the initial point which is drawn from a Gaussian distribution, the resulting plot does not look smooth like the one from

problem4B. The resulting plot is shown in the figure 34. Based on the results, the optimum learning rate is 0.77 with 6.1 iterations before convergence.

VIII. Problem 7: Newton's Method with Quadratic Cost Function

In this problem starting point was drawn from a gaussian distribution with $\mu = 0$ and $\sigma = 1$ to study the Newton's method with a quadratic function. The result is presented in the Figure 35. This method converged in 106 Iterations and did not show a zigzag behavior. Overall its performance was good, however, this was a specific bowl-shaped quadratic cost function. In general, for this method to work, Hessian matrix $\boldsymbol{H}$ should be positive definite, a criterion that would not necessarily be met in every case. The weight updating process is based on the following equation.

$$\boldsymbol{w} := \boldsymbol{w} - \eta \boldsymbol{H}^{-1} \nabla J(\boldsymbol{w}) \tag{7}$$

IX. Problem 8: Newton's Method with Himmelblau's Cost Function

In this problem starting point was drawn from a gaussian distribution with $\mu = 0$ and $\sigma = 1$ to study the Newton's method with a Himmelblau's cost function. The result is presented in the Figure 36. Newton's method is not performing well on this cost function even after 2000 iterations. This method works with the second order derivatives (Hessian matrix $\boldsymbol{H}$), making it susceptible of getting stuck in cases like this, that there are more than one minimum in the cost function.

X. Problem 9: Levenberg-Marquardt Method with Himmelblau's Cost Function

In this problem starting point was drawn from a gaussian distribution with $\mu = 0$ and $\sigma = 1$ to study the Levenberg-Marquardt method with a Himmelblau's cost function. The result of this method is dependent to the value of λ. If $\lambda = 0$, the method reduces to the Newton's method. The result is presented in the Figure 37. With $\lambda = 30$, this method converged with 8 iterations. Here the regularization term helps the algorithm to move away from saddle points. For that to happen λ should be big enough. The weight updating process is based on the following equation.

$$\boldsymbol{w} := \boldsymbol{w} - \eta [\boldsymbol{H} + \lambda \boldsymbol{I}]^{-1} \nabla J(\boldsymbol{w}) \tag{7}$$

XI. Conclusions

In the first part of this project, the perceptron algorithm was investigated using two different learning algorithms.to evaluate performance of learning algorithms, accuracy is a more reliable measure than cost function. Using a single perceptron can not result in nonlinear decision boundaries, showing the need for development of a more complex model that can result in more complex, nonlinear, decision surfaces.

In the second part of this project different optimization methods' performance with different cost functions was studied. Various experiments were done on learning rate and its effect on gradient descent. It is concluded that performance of gradient descent is highly dependent on the properties and shape of the cost function. A scheme for line search was proposed which enhanced the performance of the gradient descent to some extent, however, it is not effective in every cost function. At last, two second-order descent methods, Newton's method and Levenberg-Marquardt method, were investigated. It was concluded that adding a regularization term to the Newton's method can help it to achieve better performance in the case of a Himmelblau's cost function.

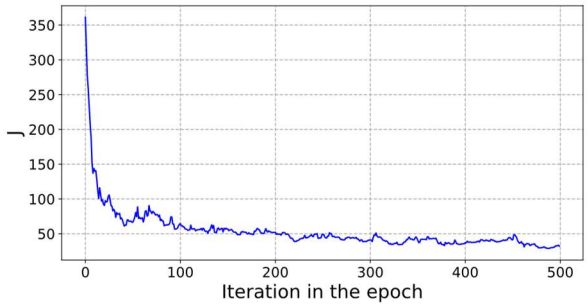

Fig. 1. Cost function in one epoch for parameter $d = 0.0$

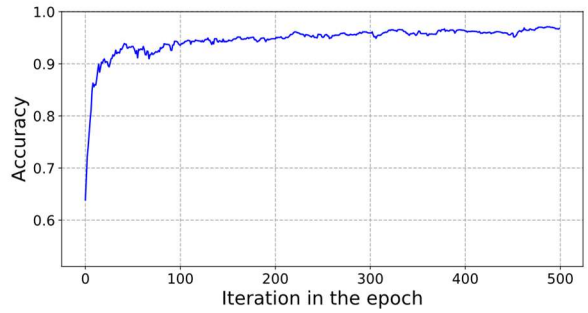

Fig. 2. Accuracy in one epoch for parameter $d = 0.0$

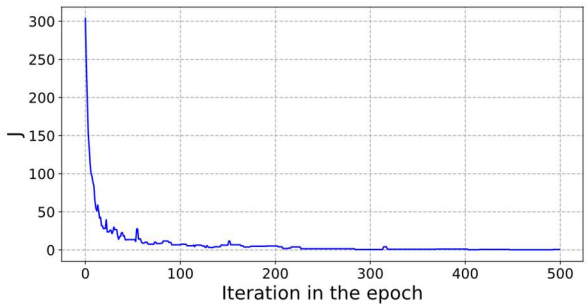

Fig. 3. Cost function in one epoch for parameter $d = 0.5$

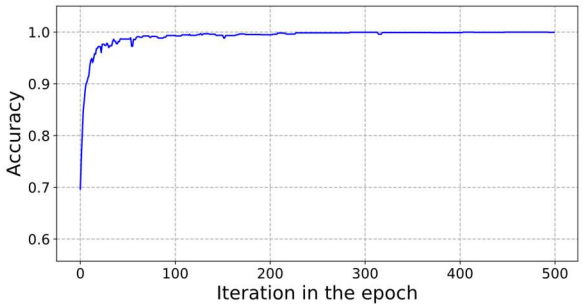

Fig. 4. Accuracy in one epoch for parameter $d = 0.5$

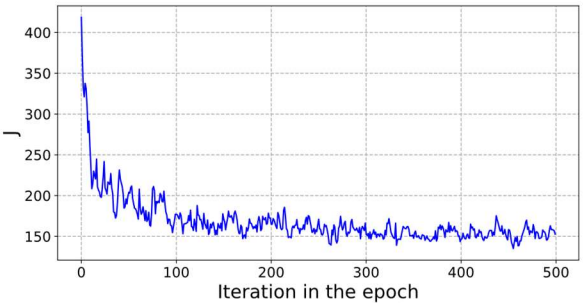

Fig. 5. Cost function in one epoch for parameter $d = -0.5$

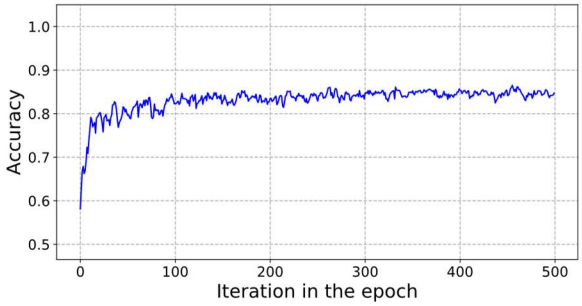

Fig. 6. Accuracy in one epoch for parameter $d = -0.5$

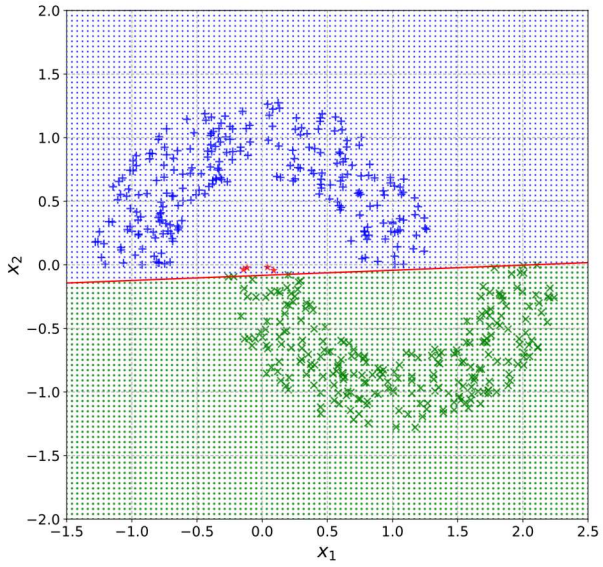

Fig. 7. Classified data drawn from double moon dist. with parameter $d = 0.0$

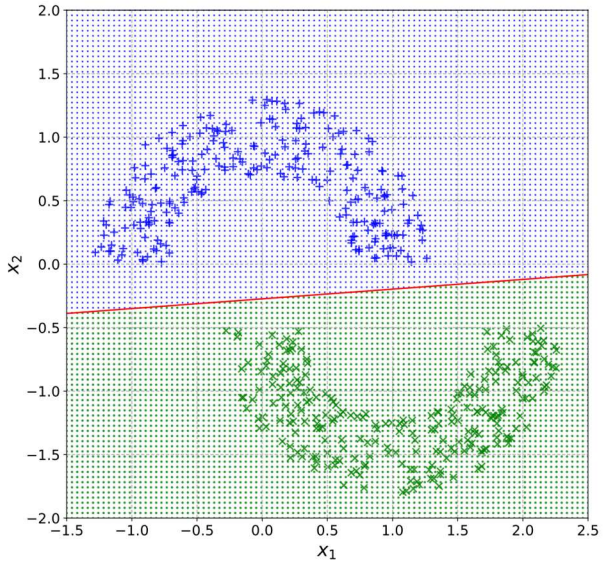

Fig. 8. Classified data drawn from double moon dist. with parameter $d = 0.5$

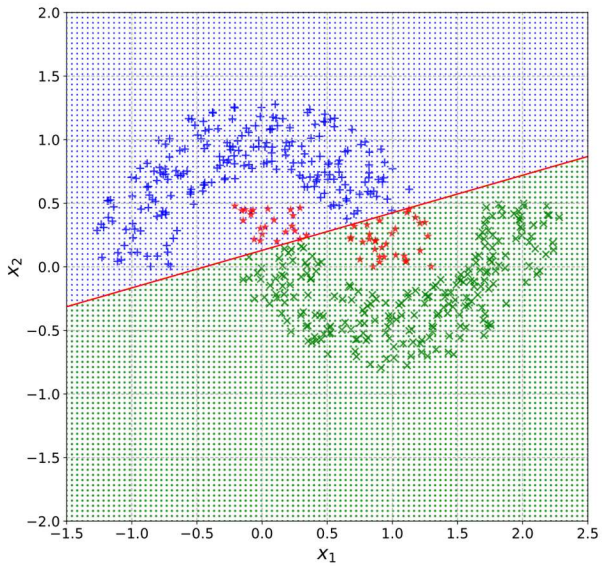

Fig. 9. Classified data drawn from double moon dist. with parameter d = -0.5

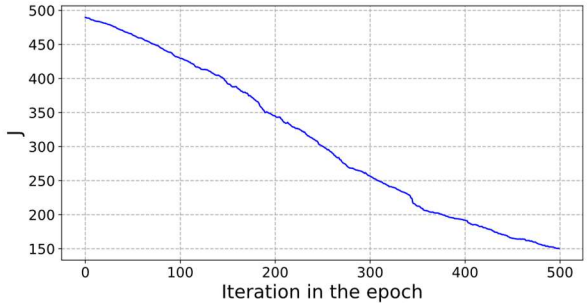

Fig. 10. Cost in one epoch with random initialization for parameter d = 0.0

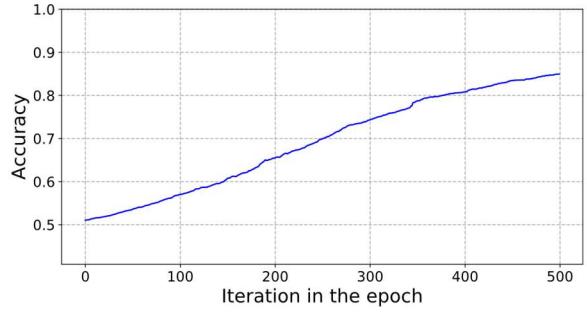

Fig. 11. Accuracy in one epoch with random initialization for parameter d = 0.0

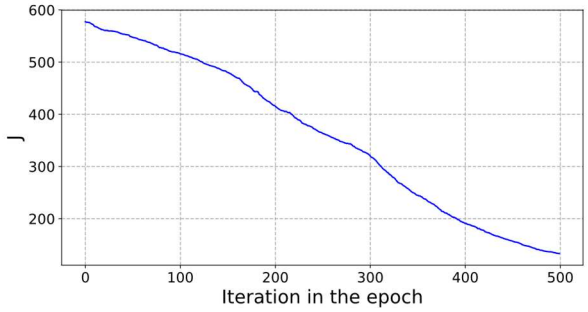

Fig. 12. Cost in one epoch with random initialization for parameter d = 0.5

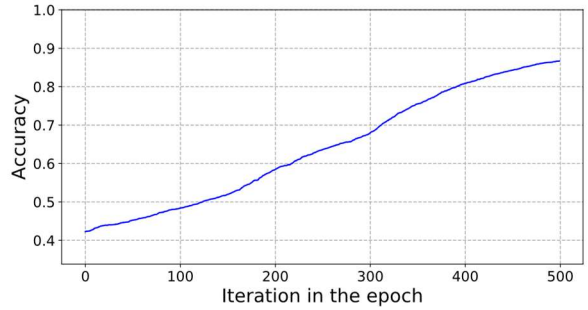

Fig. 13. Accuracy in one epoch with random initialization for parameter d = 0.5

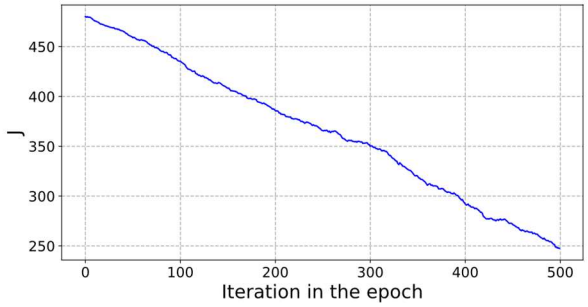

Fig. 14. Cost in one epoch with random initialization for parameter d = -0.5

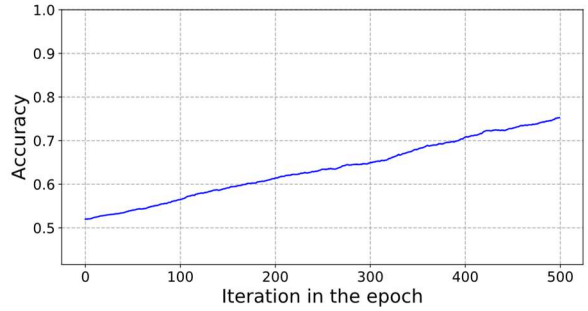

Fig. 15. Accuracy in epoch with random initialization for parameter d = -0.5

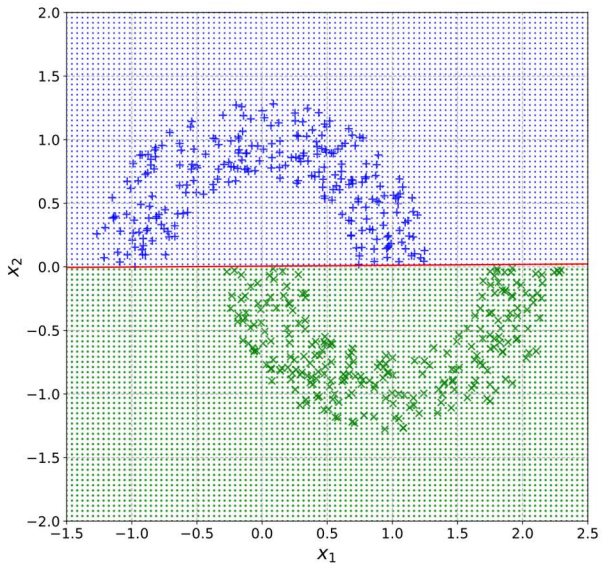

Fig. 16. Sample result of batch learning perceptron with parameter $d = 0.0$

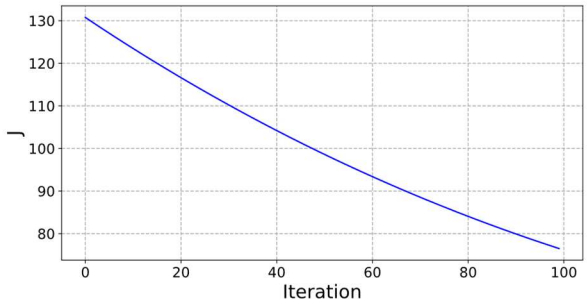

Fig. 17. Average cost function versus iteration for data with d=0.5

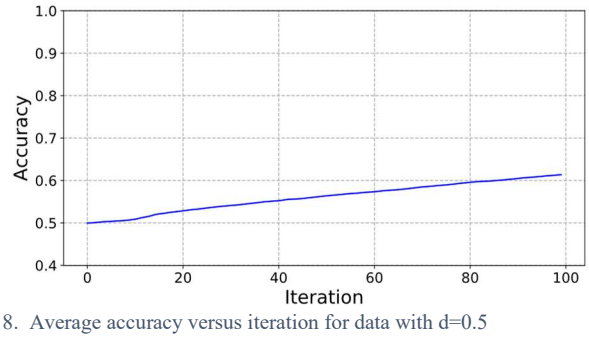

Fig. 18. Average accuracy versus iteration for data with d=0.5

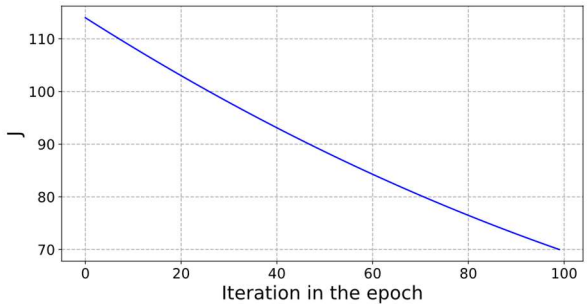

Fig. 19. Average cost function versus iteration for data with d=0.0

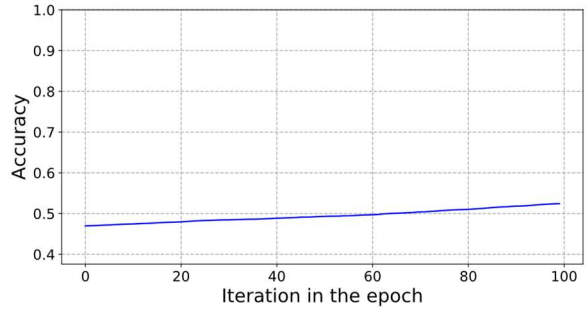

Fig. 20. Average accuracy versus iteration for data with d=0.0

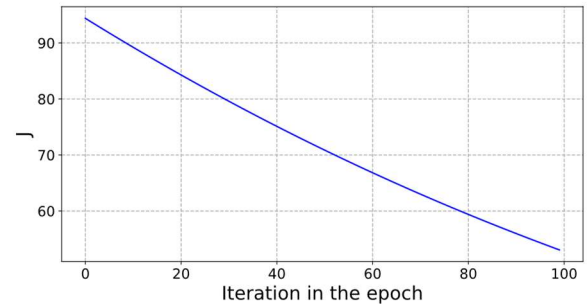

Fig. 21. Average cost function versus iteration for data with d=-0.5

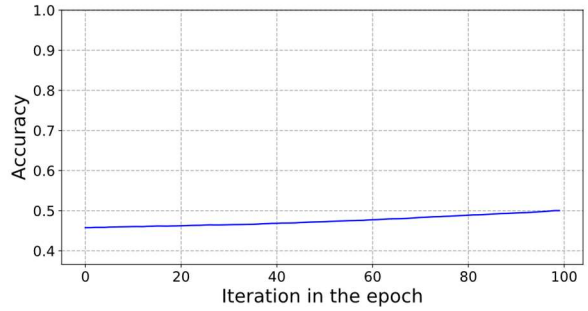

Fig. 22. Average accuracy versus iteration for data with d=-0.5

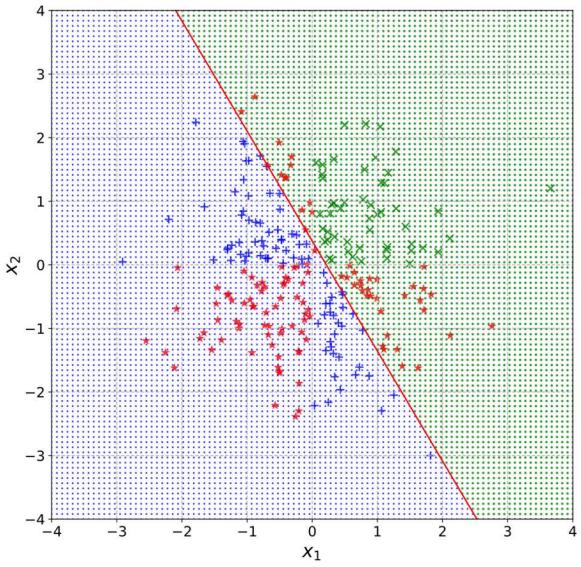

Fig. 23. Sample result of batch learning perceptron for XOR dataset

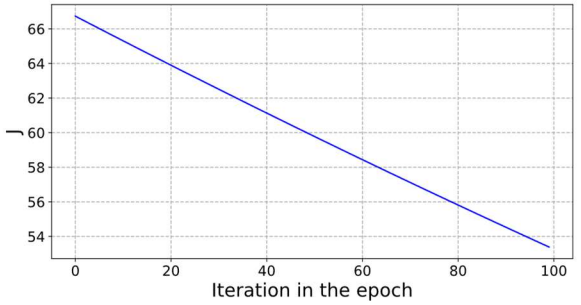

Fig. 24. Average cost function versus iteration for XOR data

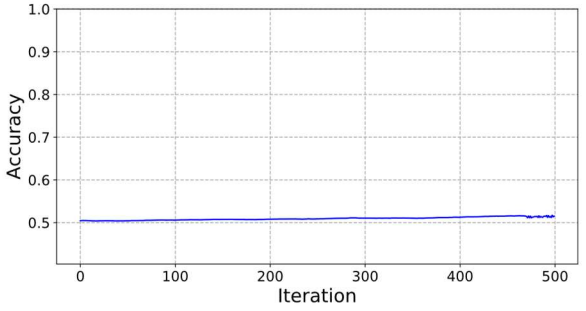

Fig. 25. Average accuracy versus iteration for XOR data

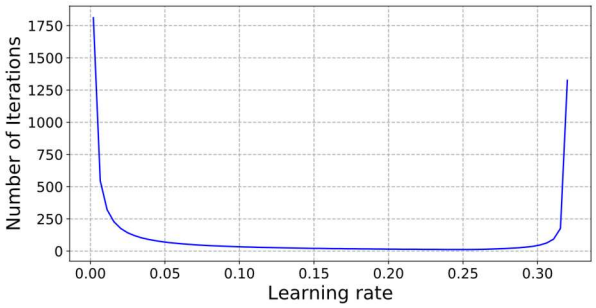

Fig. 26. Number of iterations versus learning rate

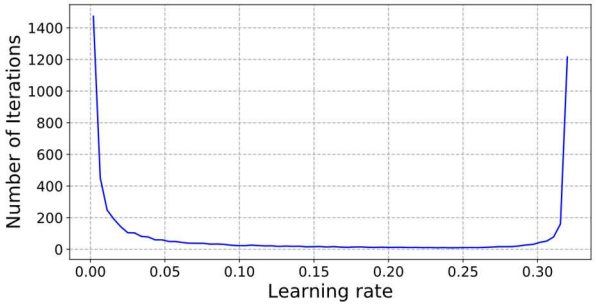

Fig. 27. Average number of iterations versus learning rate

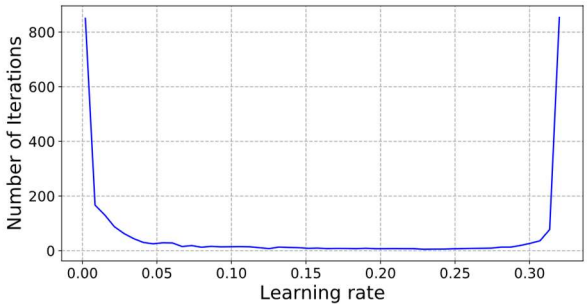

Fig. 28. Average number of iterations versus learning rate with J as cost function

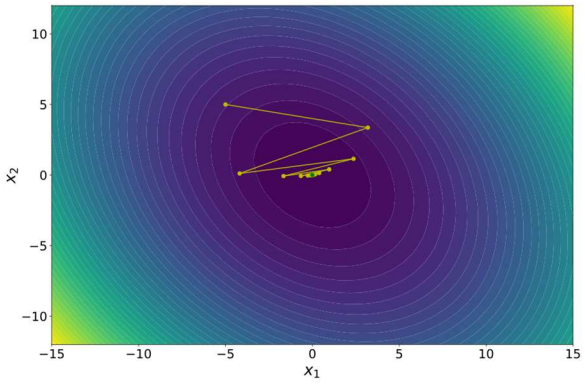

Fig. 29. Illustration of performance of the line search algorithm

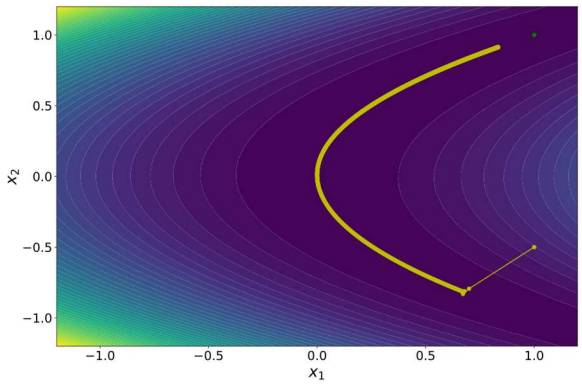

Fig. 30. Illustration of locations in weight space with learning rate=0.002

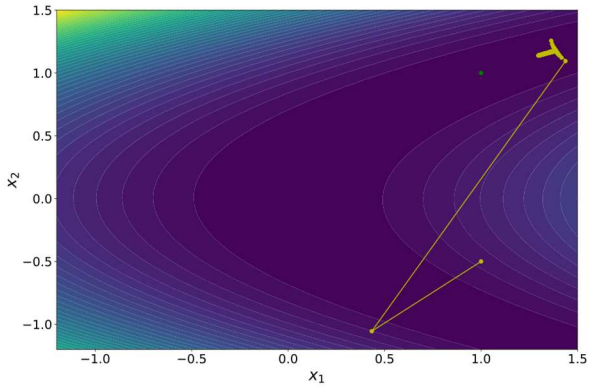

Fig. 31. Illustration of performance of the line search algorithm in case of Rosenbrock's cost function

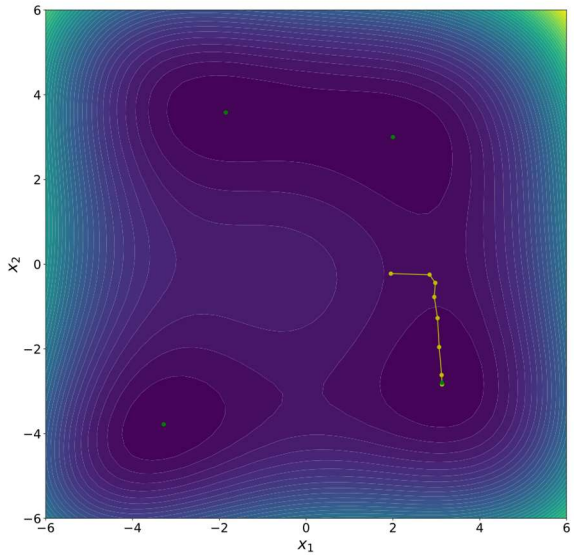

Fig. 32. Illustration of performance of the gradient descent in case of HImmelblau's cost function

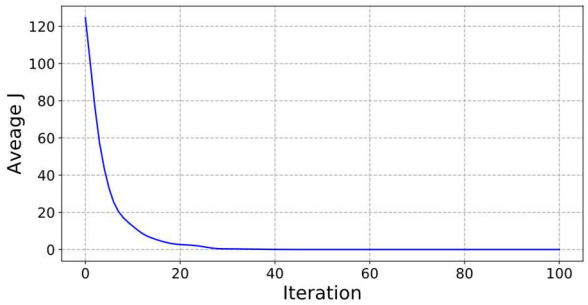

Fig. 33. Average cost function versus number of iterations

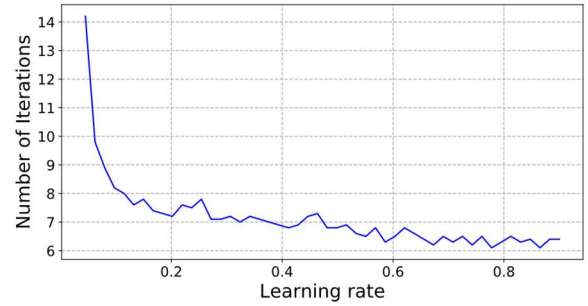

Fig. 34. Average number of iterations before convergence versus learning rate

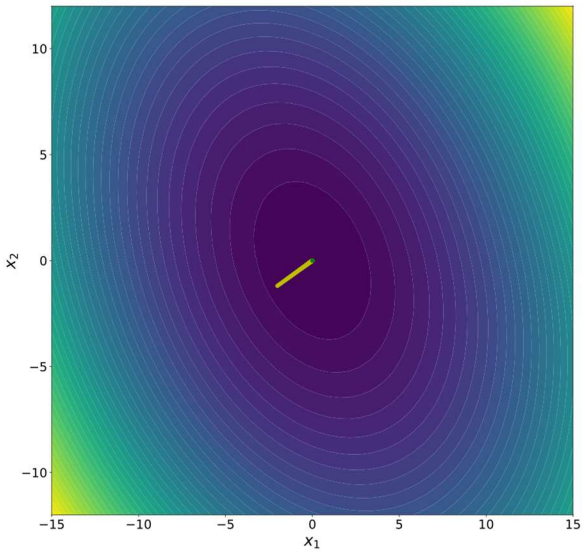

Fig. 35. Performance of Newton's method with the quadratic cost function

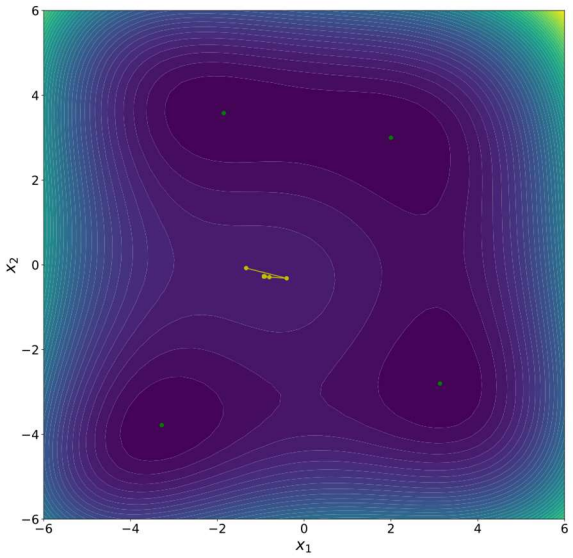

Fig. 36. Performance of Newton's method with the Himmelblau's cost function

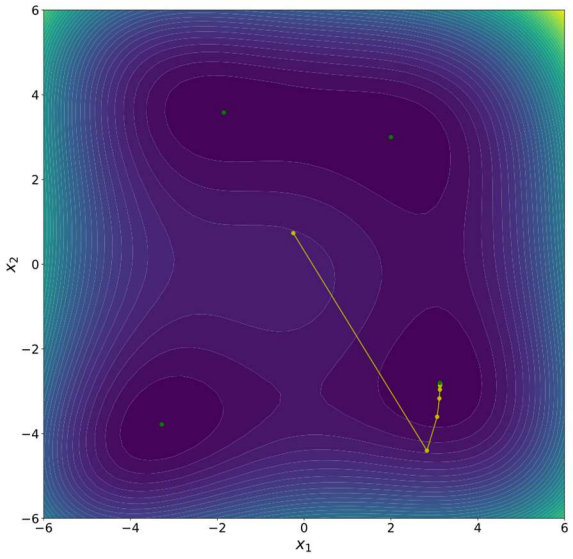

Fig. 37. Performance of Levenberg-Marquardt method with the Himmelblau's cost function