ROB313 Assignment 3: Gradient-Based Optimization

Andrew Jairam
1006941972
andrew.jairam@mail.utoronto.ca

Introduction and Objectives

So far, analytical implementations of minimizing loss functions were discussed. In practice, analytical implementations are not straightforward: it is more common to have non-convex and non-solvable loss functions that warrant the use of numerical methods. This report aims to examine different implementations of gradient descent: one of such numerical methods to minimize loss functions.

The first objective is to explore gradient descent's applications to linear regression analysis through the use of three different methods: full-batch gradient descent, stochastic mini-batch gradient descent, and stochastic mini-batch gradient descent with momentum. These methods were compared and contrasted based on the time taken to converge to the optimal loss found analytically and the error of the predictions on the test set using the resulting weights. Additionally, batch size, learning rate, and momentum parameter selection were examined to try and observe the effects of each hyperparameter.

Secondly, gradient descent in classification problems by minimizing the log-likelihood of a sigmoid was explored. The overarching goals of this implementation are the same as the regression case, where it is desired to compare and contrast full-batch and mini-batch methods by their test error and accuracy and play with hyperparameter selections to observe their effects.

Using Gradient Descent to Learn the Weights of a Linear Regression Model

This section summarizes the analysis of using gradient-based optimization to learn the weights of a linear least-squares model trained on the first 1000 training points of the *pumadyn32* dataset. To set up this study, four instances of gradient descent were implemented: full-batch gradient descent (GD), mini-batch stochastic gradient descent (SGD) with batch sizes 1 and 10, and mini-batch SGD with batch size 1 and momentum. For each instance, six learning rates incremented in increasing powers of ten were taken to observe the effect of a range of learning rates on the model. The best-performing learning rate for each method was selected as the one that minimized the RMSE between the prediction using the gradient descent weights over 10,000 iterations of gradient descent. Convergence was measured by finding the iteration number and runtime where the loss calculated using gradient descent was within 10^{-3} of the optimal loss found using SVD. Test RMSE and the optimal hyperparameters are tabulated in Table 1 below, along with “10^{-3} convergence”: denoting the number of iterations and time taken for the gradient descent loss to get within 10^{-3} of the optimal loss. Note that recording ‘None’ means that this convergence criterion was never satisfied.

	Full-Batch	SGD Mini-Batch, size 1	SGD Mini-Batch, size 10	SGD + Momentum Mini-Batch, size 1
Learning rate, η	0.01	0.001	0.001	0.001
Momentum Parameter, □	-	-	-	0.9
Test RMSE	0.870653	0.882743	0.871906	0.869300
Iterations for 10^{-3} convergence	297	None	5060	None
Time for 10^{-3} convergence (s)	0.105507	None	1.040517	None

Table 1: Results of Gradient Descent Methods Applied to Linear Regression

From above, all methods were approximately similar in sufficiently predicting on the test set, and as such, convergence rates would be a better indicator to compare the four methods. From above, the full-batch algorithm performed the best in time and iterations to converge. This does make sense, as only 100 training points are needed to loop through, which is relatively small. Additionally, it appears that the SGD mini-batch size 1 methods did not converge to within 10^{-3} convergence at all. From Figure 2 below, which shows the plots using the optimal hyperparameters for each method, it can be seen that both SGD mini-batch size 1 methods do converge to close to the optimal loss within around 2000 iterations, but with much greater error on the order of 10^{-2}. Increasing the batch size to 10 greatly helps convergence: the takeaway here then is that in practice, using a batch size of one is in general not good practice unless a fast approximation of the weights is wanted.

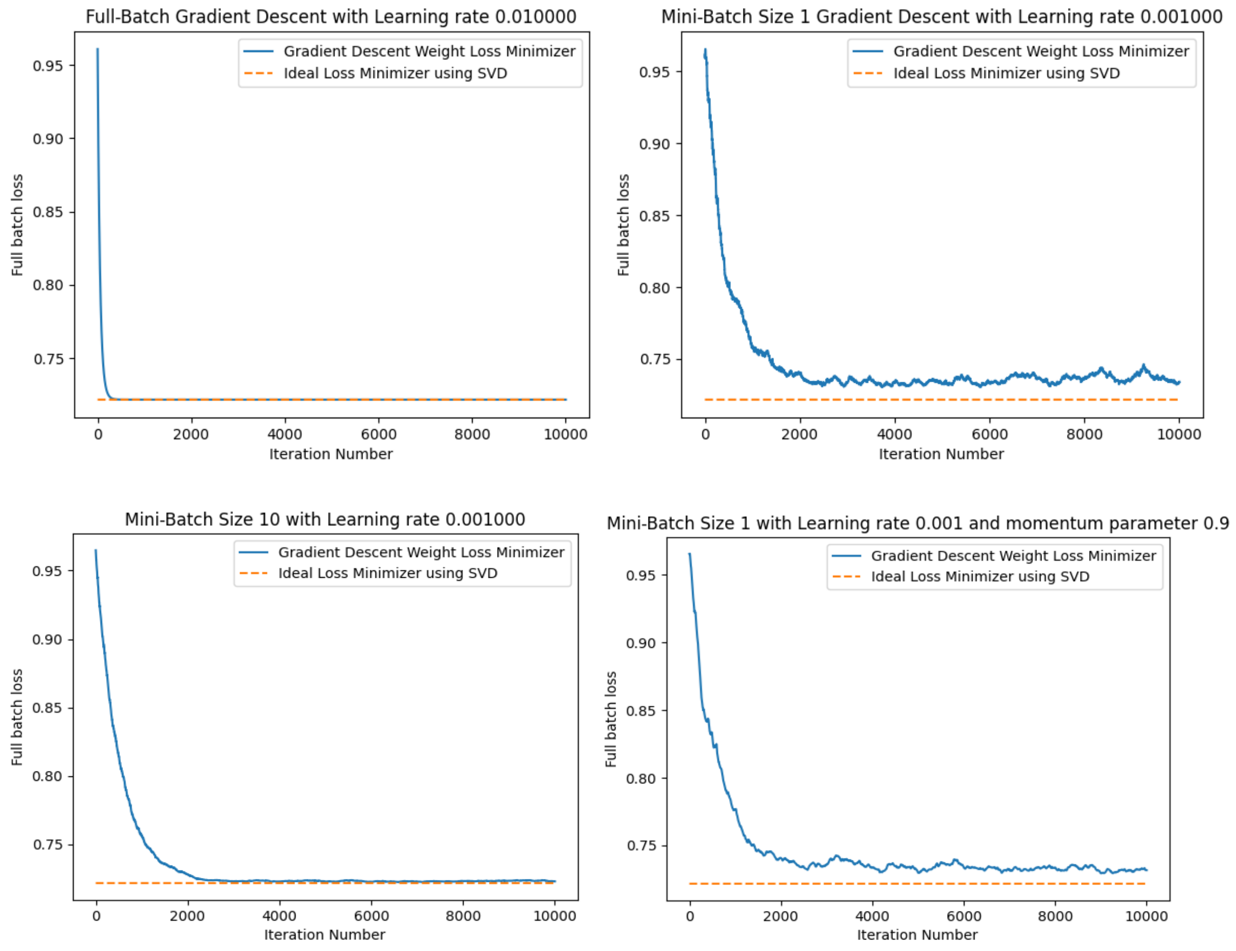

Figure 2: Plots of the Four Methods with Optimal Hyperparameters

To examine the effects of varying learning rates, Figure 3 shows the use of small and large learning rates. Having the learning rate too low causes extremely slow convergence, making it unfeasible to construct the model over 10,000 iterations, and having the learning rate too high either explodes the loss, causing the program to overflow, or significantly corrupts the loss. This does agree with literature, as large learning rates are synonymous with overfitting. In general, SGD mini-batch methods were found to require smaller learning rates than full-batch GD methods for overfitting not to occur, but keep computation time similar as fewer actions are performed per iteration.

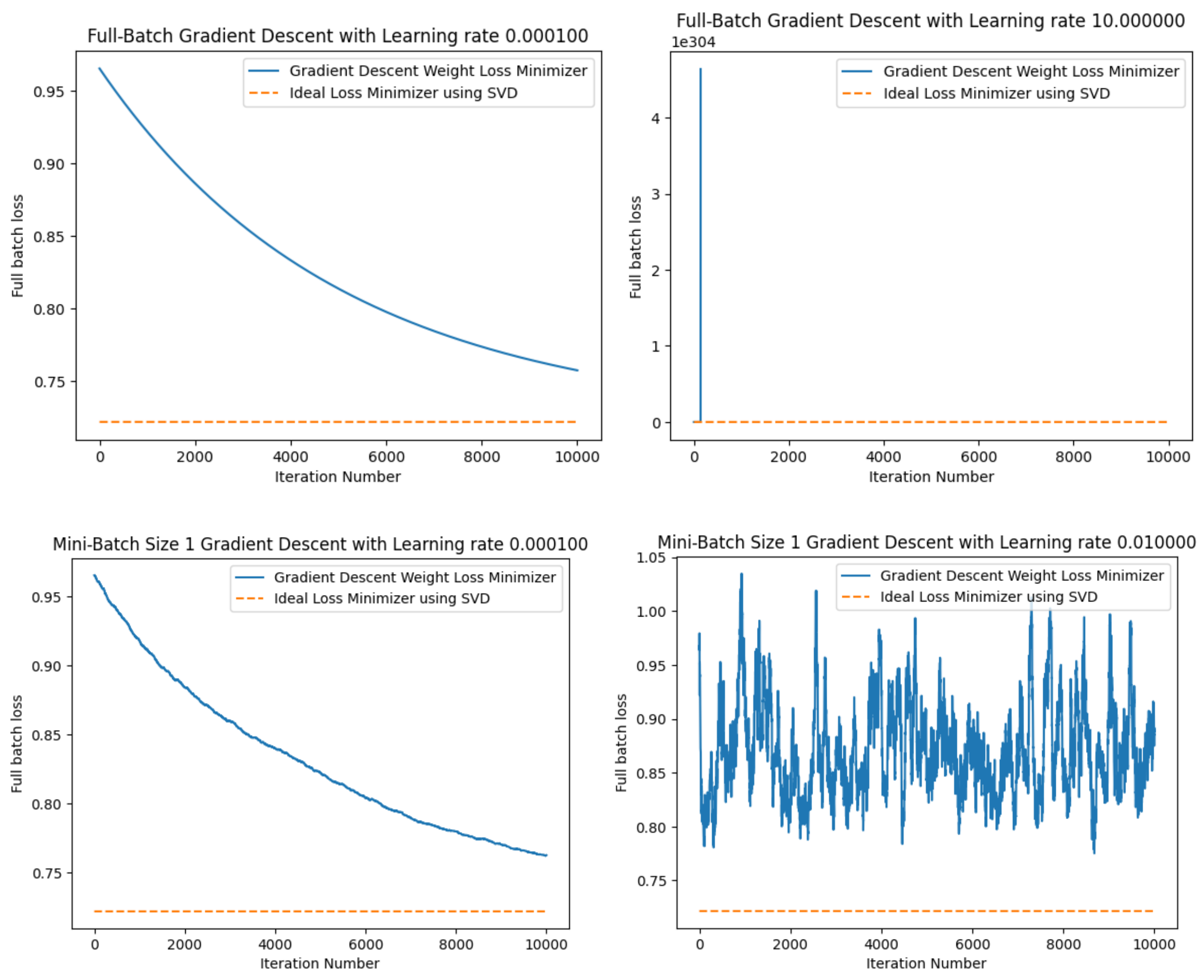

Figure 3: Effects of Low Learning Rates (left) and Large Learning Rates (right)

As a final comment, it can be noted that the impact of the momentum hyperparameter is negligible in this implementation. Momentum is introduced to reduce overfitting: which can be visually seen in the final few iterations. Comparing to the mini-batch SGD plot without momentum, it appears that the plot with momentum shows 'tighter' convergence, but this effect is not significant. This method does perform the best out of the four in terms of test RMSE: where the reduction of overfitting is likely apparent.

Using Gradient Descent to Learn the Weights of a Logistic Regression Model

The use of gradient descent to learn the weights of a logistic regression model for binary classification on the Iris dataset is explored here. This classifier uses Bernoulli likelihood with a sigmoid estimator from the scipy module on the second feature of the Iris dataset that determines if the flower is an *iris virginica* or not. Similarly to regression, both full-batch GD and SGD mini-batch methods were implemented over multiple learning rates to measure the accuracy of the classifiers, and the exact full batch negative log-likelihood over 10,000 iterations was plotted. The best-performing learning rate was taken as the rate at which the accuracy of the predictions on the test set is maximized, and the corresponding negative log-likelihood on the test set was recorded. These results are tabulated in Table 4 below.

	Full Batch GD	SGD Mini-Batch Size 1
Negative log-likelihood on the test set	7.24983	7.37058
Accuracy on the test set	73%	73%
Best Learning Rate	0.00001	0.001

Table 4: Results of the Logistic Regression Analysis

From the results, we can immediately see that negative test log-likelihood is a better performance metric than accuracy as the accuracies of prediction are the same due to the discreteness of guessing in binary classification problems: negative log-likelihood has higher precision. In addition, the negative log-likelihood is indicative of the loss: so it can be used to determine which model has higher-quality predictions. As can be seen in both the tabulated results and Figure 5 below showing the plots of both graphs, the full batch GD algorithm performed better than the mini-batch SGD algorithm with batch size 1. However, from conclusions made on the two algorithms in the last section, the batch size should be increased for future analysis as it was determined that a greater batch size yields greater convergence accuracy. Regardless, it has been shown that the mini batch SGD algorithm is a good approximation of the optimal model, and is a viable candidate especially as the size of the dataset increases.

The effects of varying the learning rates are the same as in the regression case as shown in Figure 6, which shows the effects of low and high learning rates. In practice, it is important to test many learning rates to achieve a good model: for instance, the optimal mini batch SGD learning rate was much greater in this classification problem than it was in the regression problem.

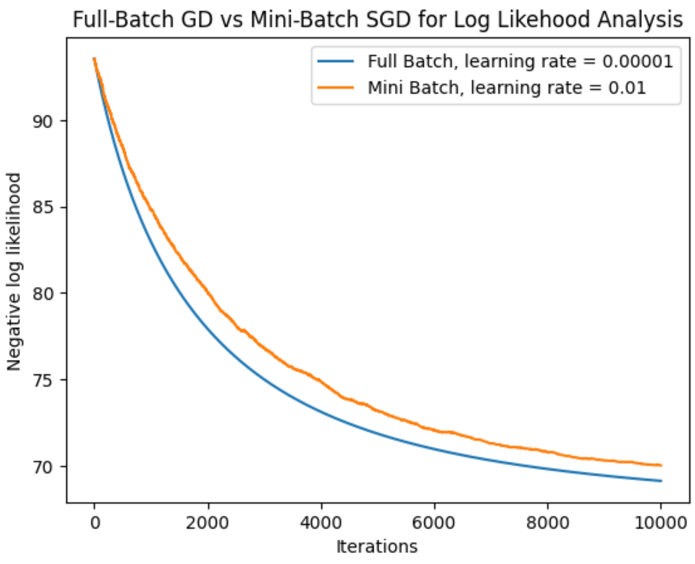

Figure 5: Mini Batch SGD vs Full Batch GD for a Logistic Regression Problem Comparison

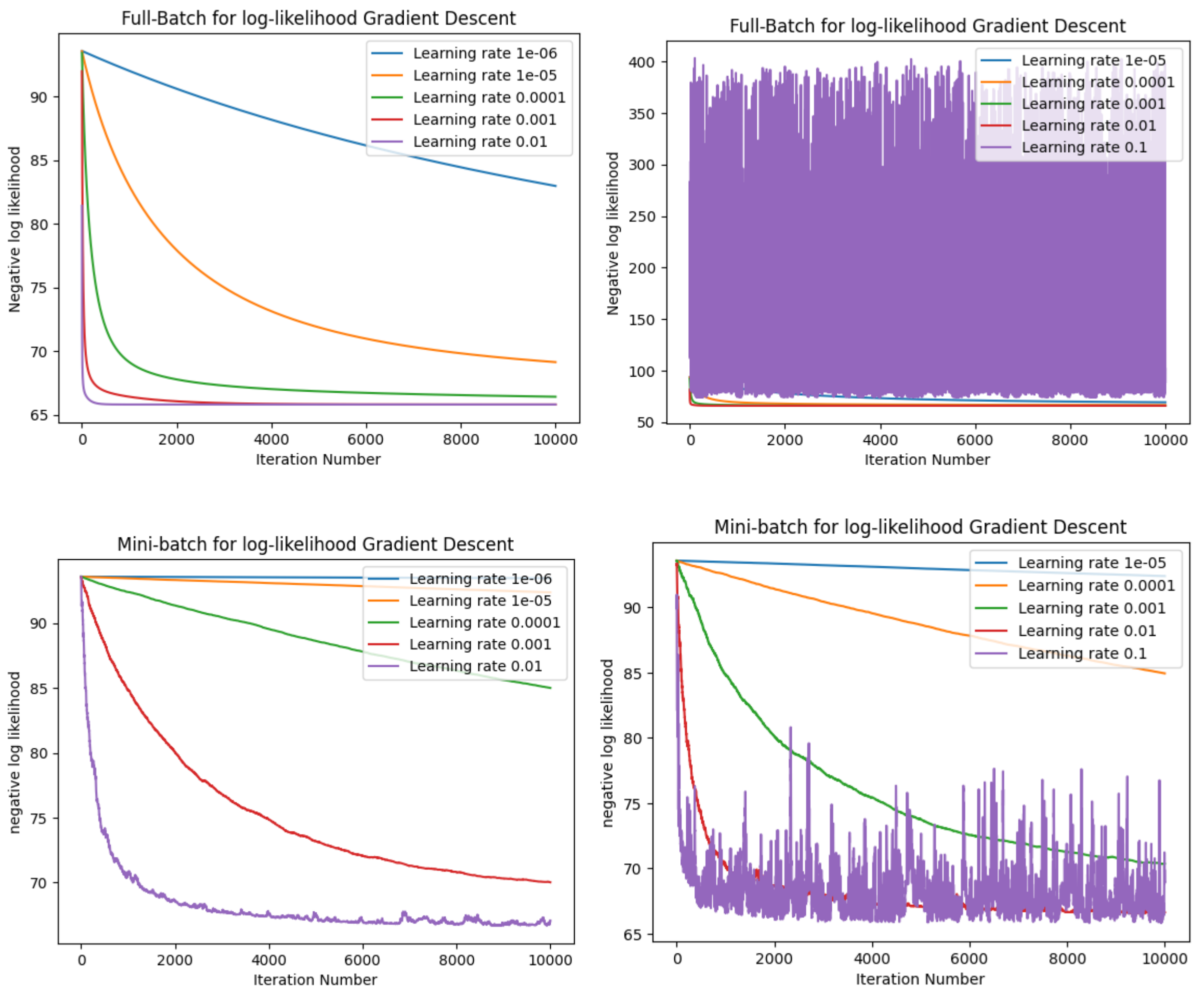

Figure 6: Effects of Different Learning Rates for Both Methods

Conclusion

The analysis of Full Batch gradient descent and Mini Batch stochastic gradient descent applied to two different problems has yielded a few key results. Unless the size of the given dataset is extremely large, full-batch gradient descent performs the best in terms of time and iterations to converge to the optimal loss, but the SGD mini batch method was proven to be a viable candidate in converging to the optimal loss for larger datasets. When implementing such a mini batch method, it is important to take a large batch size, as using smaller batch sizes was shown to yield inaccuracies in converging to the optimal loss. The momentum extension to SGD was not a significant upgrade in this implementation, but it is shown that it does reduce overfitness.

The general trend can be observed that higher learning rates correspond to faster convergence and more overfitting, and lower learning rates correspond to slower and more stable convergence, which aligns with literature. The choice of hyperparameter in practice would then of course rely on the use of the desired model, or standard validation practices to choose optimal values.

Finally, when analyzing the performance of gradient-based optimization, the best indications are convergence times and test RMSE for linear regression models, and negative log-likelihood for classification models due to the discreteness of accuracy. These metrics excel in evaluating the small differences between models by giving an indication of how accurately each method minimizes loss, and how fast it comes to do so.