# A Study of the Performance of Gradient Descent Algorithms
# MIT 18.065 Final Project Spring 2018

Ekin Karasan
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139
ekarasan@mit.edu

Tom Leech
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139
tleech@mit.edu

## 1. Introduction

Gradient descent is an algorithm that is widely used in various optimization problems. It is an iterative algorithm that is used to find the minimum of a function. Starting at a given point, the gradient descent algorithm takes steps proportional to the negative of the gradient of the function being minimized at that point. Gradient descent can be used for a large set of different problems and is widely used in machine learning to find weights of coefficients. In this project, we explore the performance of various versions of the gradient descent algorithm in creating a linear classifier. Particularly, we want to observe changes in convergence rate and accuracy of the classifier for stochastic gradient descent and several variations of mini-batch gradient descent.

## 2. Related Work

To build a linear classifier, the gradient descent algorithm predicts the coefficients of the linear classifier and updates these coefficients using the error on the training data. The goal is to minimize the total error on the training dataset. The gradient descent algorithm can vary depending on how the points are used to calculate the error function and how the classifier parameters are updated. Two main versions of the gradient descent algorithm are stochastic gradient descent and mini-batch gradient descent.

### 2.1. Stochastic Gradient Descent

In stochastic gradient descent (SGD), the error is calculated and the model parameters are estimated at each point in the training dataset. This version of gradient descent can be easily adapted to online machine learning algorithms since it updates the model at each sample.

Some advantages of SGD are that it is simple to implement, can result in faster results for some problems and that its noisy nature allows it to not converge at local minima. Some disadvantages are that updating the model so frequently is very computationally expensive and that the noisy learning process can make it hard for the algorithm to converge.

### 2.2. Mini-batch Gradient Descent

In the mini-batch gradient descent algorithm the training dataset is split up into smaller sized batches. The error is then calculated within each of the batches and the model parameters are updated.

Some advantages of mini-batch gradient descent are that it is less computationally intensive and does not need to store a large amount of data in memory. Some disadvantages are that it requires the addition of a new parameter and that it might lead to slower convergence if the model parameters are only updated once for each batch.

## 3. Approach

Our approach consists of three main aspects: building a model, building a dataset, and evaluating the performance of our model when using different training methods. To evaluate our model's performance we divide our dataset into a training set and a test set. The points in the training set are used to train our model and the points in the test set are used to evaluate it's performance. We also look at the rate of convergence of the training error and the time it takes to train.

### 3.1. The Model

For our linear classifier, we decided to use a very simple neural network. Neural networks are a widely used classifier model and are interesting for us to study because although our application is simple, neural networks can grow very quickly and the impact of small variations in training methods grows with the complexity of the network. Our model has two inputs, $x_1$ and $x_2$, and one output, $y$. It has no hidden layers. The general form of a neural network with this structure is:

$$y = Wx + b$$

where $x$ is the column vector of inputs, $W$ is a matrix of weights, and $b$ is a scalar. Expanding the components gives us:

$$y = W_1x_1 + W_2x_2 + b$$

Finally, to scale our output so that it ranges from 0 to 1, we pass it through a logistic sigmoid function, giving:

$$y = \frac{1}{1 + e^{-(W_1x_1+W_2x_2+b)}}$$

### 3.2. The Data

To build our dataset, we generated two two-dimensional gaussians and place their means close enough for there to be a decent amount of overlap. This ensured that while our data was linearly seperable, there was enough noise that our classifier could not converge to a perfect solution by looking at only a few points. The first gaussian distribution had the following mean and covariance matrix:

$$\mu_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \sigma_1 = \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$$

The second gaussian distribution had the following mean and covariance matrix:

$$\mu_2 = \begin{bmatrix} -4 \\ 5 \end{bmatrix} \quad \sigma_2 = \begin{bmatrix} 8 & 2 \\ 2 & 8 \end{bmatrix}$$

We generated 10,000 randomly sampled points from each gaussian. The randomly generated points can be seen in Figure 1 where green points correspond to the first gaussian distribution and red points correspond to the second gaussian distribution. Of our total 20,000 points, we allocated $10\%$ for the test set while the other 18,000 were left as the training set. Because we knew the data was linearly separable and we were not worried about overfitting, we decided to forgo a validation set.

### 3.3. Training Methods

The first step in any gradient descent algorithm is to calculate the gradient. In larger models, the gradient is normally approximated using numberical methods. But because of the simplicity of our model, it is possible to compute the gradient exactly. Let $Q_i(w)$ be the squared error between our model's calculated $y$ and the ground-truth $y^*$ associated with point $i$. Then the total error (which is what we want to minimize) is given by:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i^*)^2$$

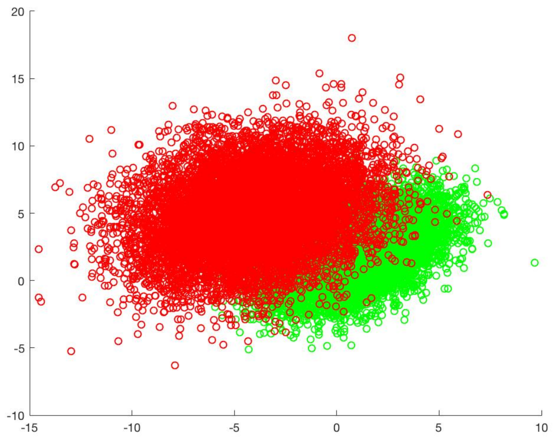

Figure 1. The data shown as a scatterplot with green points corresponding to distribution 1 and red points corresponding to distribution 2.

The gradient of $Q(w)$ is given by

$$\nabla Q(w) = \begin{bmatrix} \frac{\partial}{\partial W_1}(y_i - y_i^*)^2 \\ \frac{\partial}{\partial W_2}(y_i - y_i^*)^2 \\ \frac{\partial}{\partial b}(y_i - y_i^*)^2 \end{bmatrix} \tag{1}$$

$$\frac{\partial}{\partial W_1} = \frac{2x_1 e^{-W_1x_1 - W_2x_2 - b}\left(\frac{1}{e^{-W_1x_1 - W_2x_2 - b}+1} - y_i^*\right)}{(e^{-W_1x_1 - W_2x_2 - b} + 1)^2} \tag{2}$$

$$\frac{\partial}{\partial W_2} = \frac{2x_2 e^{-W_1x_1 - W_2x_2 - b}\left(\frac{1}{e^{-W_1x_1 - W_2x_2 - b}+1} - y_i^*\right)}{(e^{-W_1x_1 - W_2x_2 - b} + 1)^2} \tag{3}$$

$$\frac{\partial}{\partial b} = \frac{2 e^{-W_1x_1 - W_2x_2 - b}\left(\frac{1}{e^{-W_1x_1 - W_2x_2 - b}+1} - y_i^*\right)}{(e^{-W_1x_1 - W_2x_2 - b} + 1)^2} \tag{4}$$

Once we calculate the gradient at a given data point, how we choose to update our model parameters from this gradient is what distinguishes the different training algorithms we look at. We first tested the two main approaches discussed in Section 2: stochastic gradient descent and mini-batch gradient descent. Within the mini-batch gradient descent algorithm we tested the performance of the algorithm by changing how the points in each batch were chosen and how frequently the model was updated. The four main categories in which the performance is tested were:

1. Picking the points in each batch randomly from the entire training dataset and updating the model with each training sample in the batch.
2. Picking the points in each batch randomly from the points that were not used in previous batches and updating the model with each training sample in the batch.
3. Picking the points in each batch randomly from the entire training dataset and updating the model once every batch by looking at the average error over the entire batch.
4. Picking the points in each batch randomly from the points that were not used in previous batches and updating the model once every batch by looking at the average error over the entire batch.

In addition to looking at these versions of the mini-batch gradient descent algorithm, we also analyzed the performance of different batch sizes. The results of all of these experiments are presented in Section 4.

## 4. Results

The first tested algorithm was stochastic gradient descent where a total of 18000 points were trained at a random order. The progression of the training error (calculated by looking at the error across all training points) is displayed in Figure 2. The total duration of training was 0.5929s and the error in the test set (consisting of another set of 2000 points) was calculated as 7.10%. The same training and test sets were used for the rest of the algorithms.

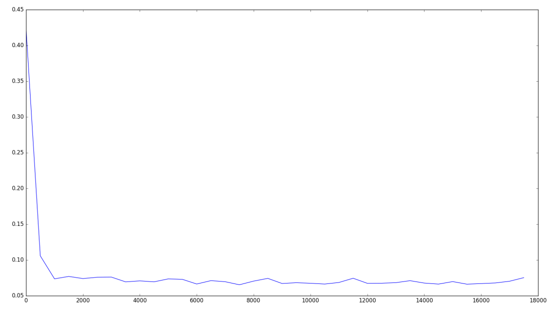

Figure 2. The progression of the training error, calculated by looking at the error across all training points, with the stochastic gradient descent algorithm.

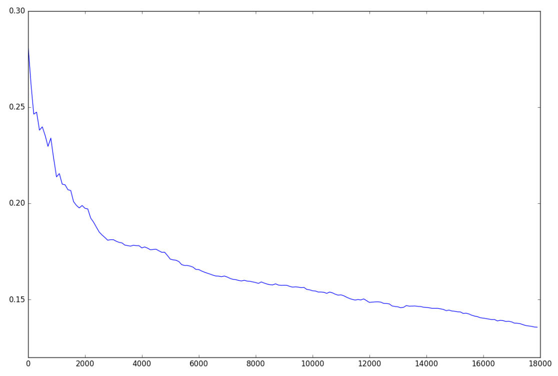

Figure 3. The progression of the training error with the mini-batch gradient descent algorithm with replacement of training points from previous batches and averaging across a batch before updating model.

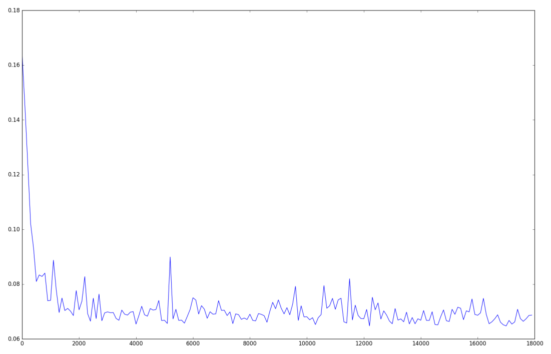

Figure 4. The progression of the training error with the mini-batch gradient descent algorithm with replacement of training points from previous batches and updating model at every sample.

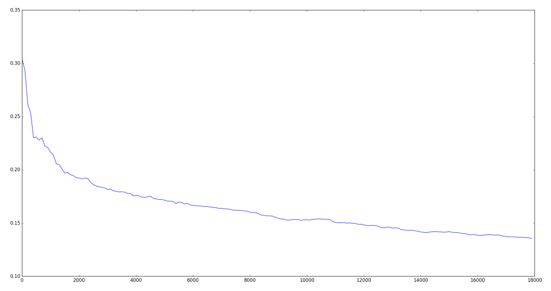

Figure 5. The progression of the training error with the mini-batch gradient descent algorithm without replacement of training points from previous batches and averaging across a batch before updating model.

The next tested algorithm was mini-batch gradient descent when points in the batch are picked randomly from the entire dataset and the model is updated once at the end of each batch with the average error over the entire batch. The progression of the training error is displayed in Figure 3. The total duration of training was 0.9997s and the error in the test set was calculated as 13.2%.

The following experiment involved testing the mini-batch gradient descent when points in the batch are picked randomly from the entire dataset and the model is updated with every training sample. The progression of the training error is displayed in Figure 4. The total duration of training was 0.9086s and the error in the test set was calculated as 6.66%.

The next experiment involved testing the mini-batch gradient descent when points in the batch are picked randomly from points that were not in previous batches and the model is updated once at the end of each batch with the average error over the entire batch. The progression of the training error is displayed in Figure 5. The total duration of training was 1.234s and the error in the test set was calculated as 13.2%.

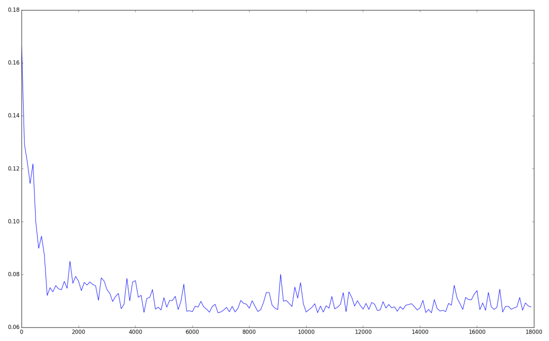

Figure 6. The progression of the training error with the mini-batch gradient descent algorithm without replacement of training points from previous batches and updating model at every sample.

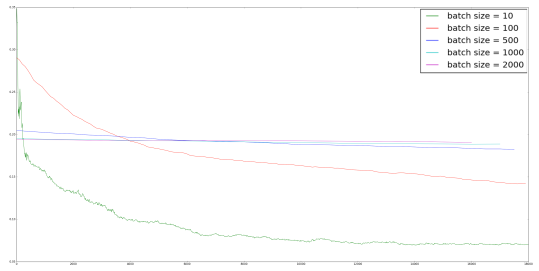

Figure 7. The progression of training error in the mini-batch gradient descent algorithm for varying batch sizes with single update for entire batch with the average error. The batch sizes tested were 10, 100, 500, 1000, and 2000.

The final experiment with mini-batch gradient descent looked at when points in the batch are picked randomly from points that were not in previous batches and the model is updated once with every training sample. The progression of the training error is displayed in Figure 6. The total duration of training was 1.28s and the error in the test set was calculated as 6.65%.

The errors in test set and training duration for the 5 experiments described above can be summarized in Table 1.

The effect of batch size on the performance of the mini-batch gradient descent algorithm was tested separately for the case where errors in the batch are averaged to make a single model update (with averaging) and the case where model is updated with every training sample (without averaging). In both cases, the batch sizes used were 10, 100, 500, 1000 and 2000 and the points in

| Algorithm | Test Set Error | Training Duration(s) |
|---|---|---|
| SGD | 7.10% | 0.5929 |
| MB 1 | 13.2% | 0.9997 |
| MB 2 | 6.66% | 0.9086 |
| MB 3 | 13.2% | 1.234 |
| MB 4 | 6.65% | 1.280 |

Table 1. Results of the 5 experiments discussed above. SGD corresponds to the stochastic gradient descent algorithm. MB 1-4 correspond to the four variations of the mini-batch gradient descent algorithm described above in order. To summarize, MB 1 corresponds to the variation with replacement and averaging, MB 2 corresponds to the variation with replacement and no averaging, MB 3 corresponds to the variation without replacement and averaging and MB 4 corresponds to the variation without replacement and without averaging.

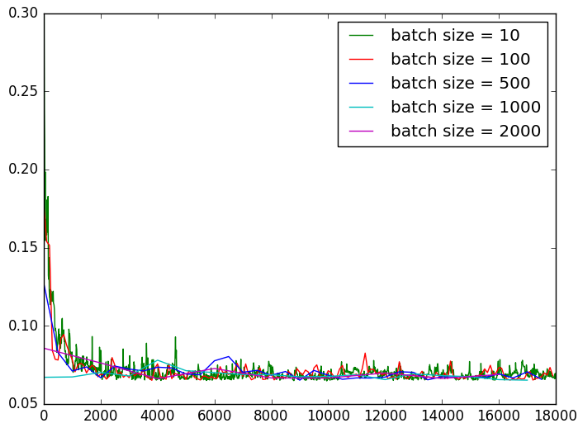

Figure 8. The progression of training error in the mini-batch gradient descent algorithm for varying batch sizes with updates at every training sample. The batch sizes tested were 10, 100, 500, 1000, and 2000.

each batch were chosen randomly from the entire training set (with replacement). The progression of the training error for the case with averaging is displayed in Figure 7 and the progression of the training error for the case without averaging is displayed in Figure 8. The results for the error rates in the test set with various batch sizes are displayed in Table 2.

| Batch Size | With Averaging | Without Averaging |
|---|---|---|
| 10 | 18.1% | 6.60% |
| 100 | 13.7% | 6.60% |
| 500 | 18.1% | 6.55% |
| 1000 | 18.75% | 6.55% |
| 2000 | 19.15% | 6.55% |

Table 2. Test set errors for various batch sizes of the mini-batch gradient descent algorithm with and without averaging.

## 5. Discussion

The results of performance for various variations of the gradient descent algorithm have been shown in Section 4. The progression of training error for the stochastic gradient descent algorithm presented in Figure 2 shows a very fast noise free convergence. When the mini-batch algorithm is introduced and updates to the model are made with each sample we expect to see a similar behavior in convergence, since updates are made in a similar fashion in both cases. We see that mini-batch gradient descent algorithm with updates in the model done at each training sample follow the same convergence pattern as seen in Figures 4 and 6 (for both the replacement and no replacement cases). However, we see that the progression contains a lot of more noise due to the batches. We see in Table 1 that the test-set errors are similar for all three cases although the error is slightly lower with mini-batch gradient descent with updates at each sample (~6.65%) compared to stochastic gradient descent (7.10%).

Figures 4-6 show that there is a clear difference in the progression of error for mini-batch gradient descent with and without averaging. For the case where there is no averaging, in other words the model is updated with each training point, the progression shows a very fast convergence and noisy behavior. On the other hand, when there is averaging, or the model is updated once at the end of each batch with the average error in that batch, the progression of error is slower and almost noise-free. This means that the training might require a longer duration to achieve the same performance when averaging is used. We also observe that there is a higher test error for algorithms in which averaging is used. However, this might be due to the fact that the model was not trained for a period long enough and was not able to find the optimal solution. In Figures 3 and 5 it can be clearly seen that the training error is still decreasing steadily at the point where training is ended. There is not much of a difference in training duration for the same number of training points processed for mini-batch gradient descent with and without averaging. However, mini-batch gradient descent with averaging might need to be trained with more batches to reach an optimum solution which might lead to a longer duration.

We also observe in Figures 4-6 that replacement of training points when choosing a new batch does not really have an effect on the training error progression. We also see in Table 1 that it also does not effect overall test set error. However, Table 1 also shows that mini-batch gradient descent without replacement of the points used in previous batches takes a much longer duration. This is due to the fact that additional computation need to be performed to not allow points to be considered in multiple batches. Although this appears as a very small difference in time, it might accumulate to be much larger when larger models with many layers are considered.

Finally, it can be seen in Figure 8 and Table 2 that with mini-batch gradient descent without averaging, the batch size does not have an impact on the training error progression or the test set error. This makes sense intuitively since updates are made at each training point separately, the size of the batch should not have much of an impact. On the other hand, for mini-batch gradient descent when averaging is used, the batch size makes a very considerable difference. We can see in Figure 7 that for batch sizes larger than 100, the training error does not converge to lower values. This is due to the fact that a single update is made for such a large batch that this does not allow the model to converge to an optimal point. Since we observe almost no change in training error, this low performance cannot be attributed to a small learning rate. We also observe in Table 2 that the test set errors are generally very high, with the exception of a batch size of 100. This could mean that a batch size of 10 is too low and batch sizes of 500, 1000 and 2000 are too high and generalize too much.

## 6. Conclusion

In this project we tested the performance of stochastic gradient descent and several variations of mini-batch gradient descent in creating a linear classifier to separate 2 gaussian distributions. Several tests are made to assess the performance including progression of the training error, error in test set and the duration of training. It is observed that various changes in the algorithm can significantly change the performance of the algorithm.

In the future it would be interesting to let mini-batch with averaging run to completion and evaluate the final test error with the other algorithms which seem to converge almost immediately. It would also be interesting to try to reproduce our results on a larger scale with a system that can not use analytically calculated derivatives. It is possible that without having exact derivatives, the benefits of averaging to reduce noise would become more apparent.

All code can be found at: `https://github.mit.edu/tleech/gradient_descent_variations`