6.867: Homework 1

1 Gradient Descent

We started by implementing gradient descent, a numerical technique that is commonly used to optimize differentiable functions. Although gradient descent is vulnerable to being trapped in local minima, the two objective functions chosen for our experiment are quasiconvex and therefore the algorithm is guaranteed to converge on the global minimum regardless of initialization.

To better understand how the starting guess, step size, and convergence criterion hyperparameters affect the performance of the gradient descent algorithm, we conducted several experiments with different hyperparameter values, recording the value of the objective function over time as well as the time to convergence.

1.1 Starting guess

Since the optimal values according to the closed form solution for the negative Gaussian and quadratic bowl are (10, 10) and (400/15, 400/15) respectively and both functions are perfectly symmetric, we chose a series of starting guesses whose Euclidean distance from the optimal value increased linearly.

1.1.1 Gaussian

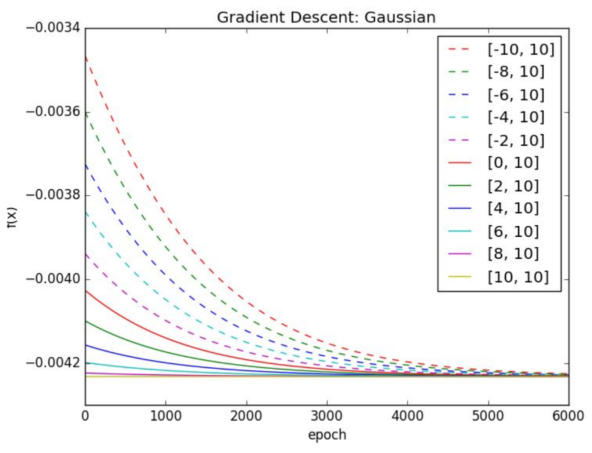

Figure 1. The value of the objective function (negative Gaussian) vs time for a set of starting guesses.

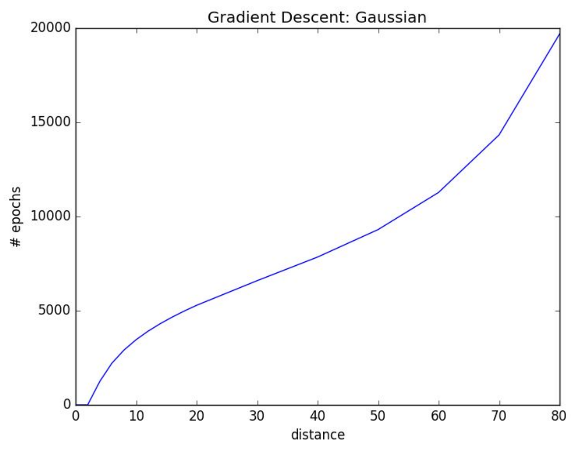

Figure 2. The number of cycles to converge vs the distance between the guess and the optimal value.

The plot on the left demonstrates that the value of our Gaussian objective function decreases monotonically regardless of the starting position. The plot on the right shows that as the distance between the starting guess and the optimal value increases, the number of cycles needed to reach converges also increases.

The training time appears to increase logarithmically at first but then accelerates - this can be explained by the fact that the gradient of the Gaussian approaches zero as the distance from the mean increases, making our adjustments smaller. This problem can be solved by using adaptive learning rates such as Adagrad and RMSProp which incorporate ideas such as momentum to adjust for small gradients.

By looking at how the norm of the gradient changes over time, we can get a better understanding of how the gradient affects the training time for simple gradient descent. The below graph shows how the norm changes for various starting points.

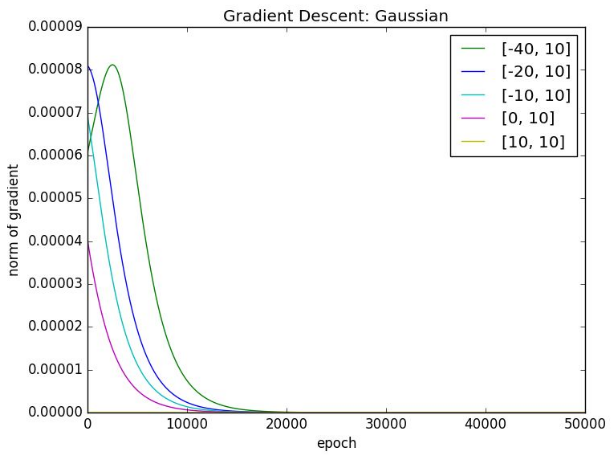

Figure 3. The norm of the gradient over time for a set of starting guesses.

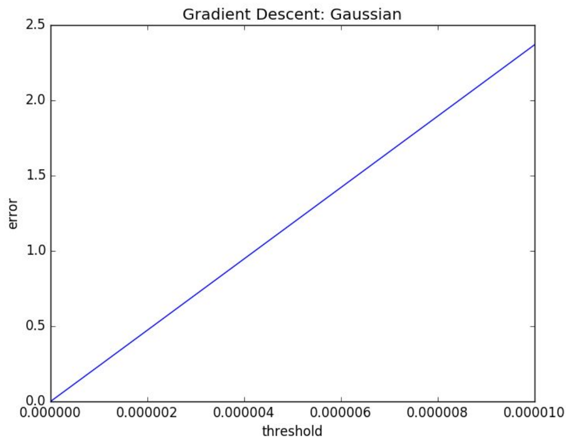

Figure 3b. The distance between the learned result and the optimal value vs the threshold.

When the starting guess is far away, the norm initially increases as we move from the near-zero regions of the Gaussian into the active region; this accounts for the drastically increased training times when the starting guess is especially bad. The convergence threshold is directly related to the distance between the final learned value and the optimal value; in other words, a larger threshold produces less accurate results. The tradeoff here is that a smaller convergence threshold will also require more time to converge, as shown in the graph on the left.

1.1.2 Quadratic

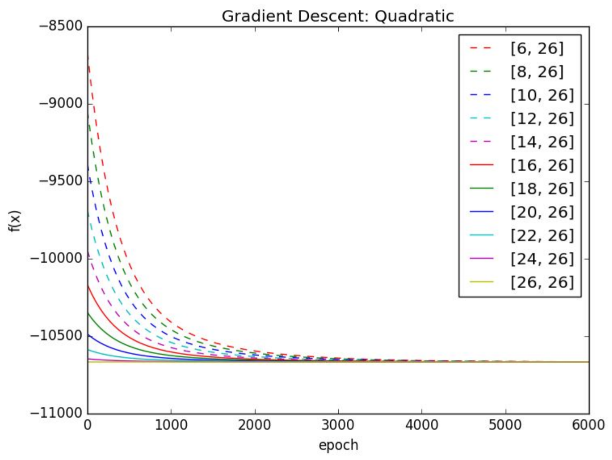

Figure 4. The value of the objective function (quadratic bowl) vs time for a set of starting guesses.

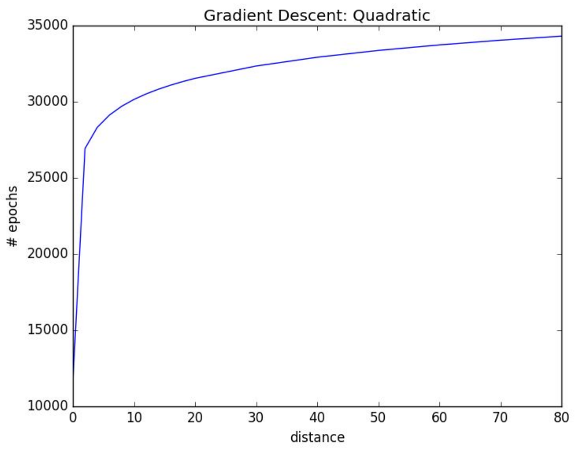

Figure 5. The number of cycles to converge vs the distance between the guess and the optimal value.

Unlike the negative Gaussian, the gradient for the quadratic bowl does not approach 0 as the distance between the starting guess and the optimal value increases. This is reflected in the figure on the right where the number of cycles required to converge appears to increase logarithmically.

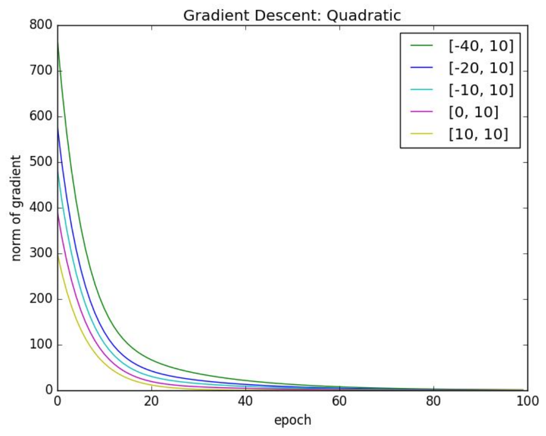

Figure 6. The norm of the gradient over time for a set of starting guesses.

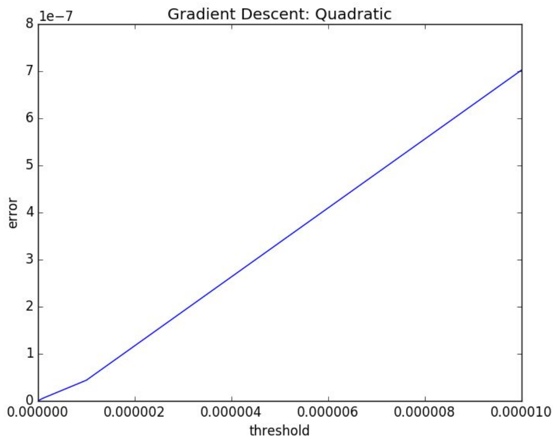

Figure 6b. The distance between the learned result and the optimal value vs the threshold.

The norm of the gradient decreases monotonically for the quadratic bowl - this explains why gradient descent converges faster. Like the Gaussian function, the threshold for the convergence criterion is directly proportional to the distance between the final value and the optimal value and inversely related to the time to convergence.

1.2 Step size

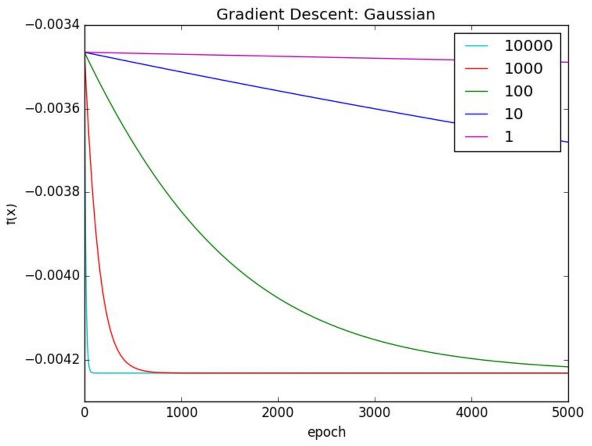

Figure 7. The value of the Gaussian objective function over time for various step sizes.

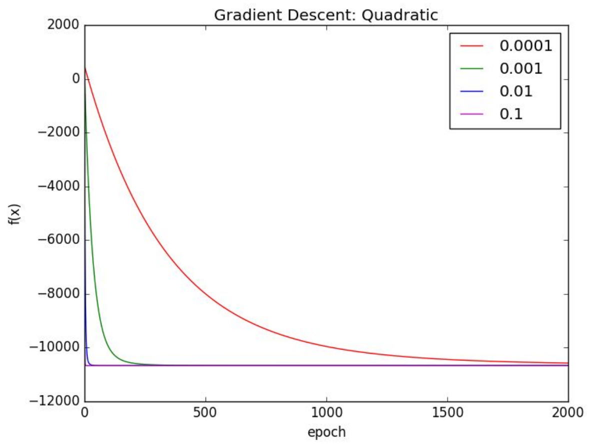

Figure 8. The value of the quadratic objective function over time for various step sizes.

For both objective functions, a larger step size generally resulted in faster convergence. A notable exception is that when the step size is too large, gradient descent fails to converge on the quadratic function and instead oscillates around the optimal value before diverging and overflowing.

1.3 Derivative approximation

Regardless of the objective function and barring numerical problems due to underflow, a smaller δ value always results in a better approximation of the gradient. However, even when the approximation was wildly off, the gradient descent procedure was robust enough to converge on the optimal solution.

δ	X	Gaussian actual	Gaussian estimate	Quadratic	Quadratic estimate
100	[16,5]	2.463e-5, -2.052e-5	1.761e-7, -1.441e-7	-215, -270	-215, -270
10	[16,5]	2.463e-5, -2.052e-5	2.344e-5, -1.953e-5	-215, -270	-215, -270
1	[16,5]	2.463e-5, -2.052e-5	2.462e-5, -2.051e-5	-215, -270	-215, -270
.1	[16,5]	2.463e-5, -2.052e-5	2.463e-5, -2.052e-5	-215, -270	-215, -270

Figure 9. The actual and estimated values for both the Gaussian and quadratic functions for various δ values.

1.4 Batch, Stochastic, Mini-batches

To evaluate the performance of batch, stochastic, and mini-batch gradient descent, we implemented a mini-batch gradient descent algorithm and then set the mini-batch size to 100, 50, 20, and 1. Since the dataset contains 100 entries, using mini-batch size 100 is equivalent to performing batch gradient descent. When the mini-batch size is 1, the result is stochastic gradient descent.

To ensure convergence, we used a dynamic step size set by (100000 + epoch) ** -0.5, where epoch (cycle) is incremented after iterating over the entire dataset; for batch gradient descent, epoch increases once every time the gradient is computed, for stochastic gradient descent, epoch only increases after the gradient is computed for all 100 entries in the dataset.

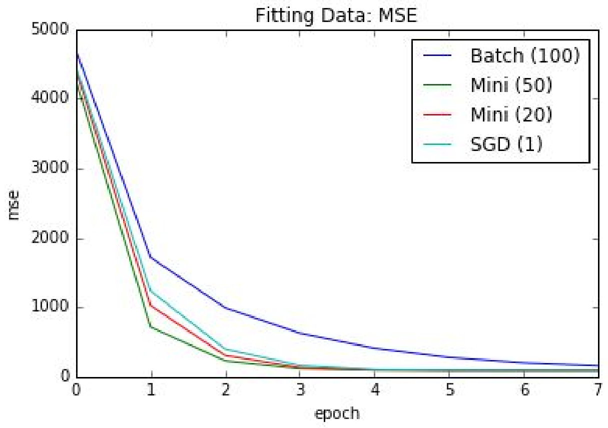

Figure 10a. The mean squared error over time for various mini-batch sizes.

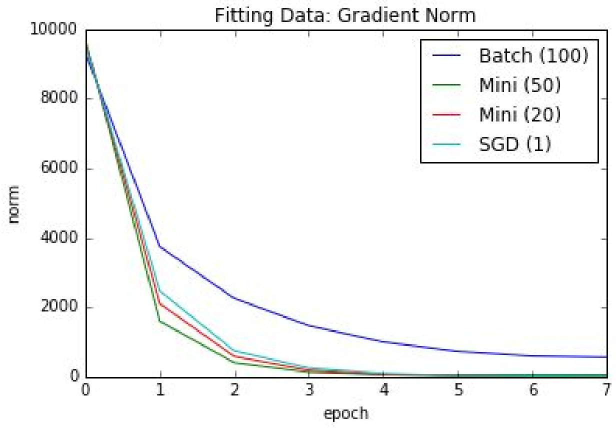

Figure 10b. The norm of the gradient over time for various mini-batch sizes.

Based on the above plots, we conclude that stochastic and mini-batch gradient descent significantly outperform the batch gradient descent algorithm, converging in fewer epochs. These three techniques all converged to the same final weights but the stochastic gradient descent algorithm required 100 times as many gradient computations as the batch gradient descent algorithm.

This difference in speed was exasperated by our use of NumPy, a numerical computing library written in C with Python bindings. In our stochastic gradient descent code, we constantly had to shuffle small pieces of data between Python and NumPy memory, significantly degrading performance; with batch gradient descent, we moved the data in bulk and allowed the highly-optimized C++ library to do the heavy lifting.

2 Linear Basis Function Regression

Figure #.

Figure #.

3 Ridge Regression

Figure #.

Figure #.

4 LASSO Regression

Figure #.

Figure #.