EARIN Lab 1 Variant 4 Report

Dawid Wypych

Aayush Gupta

Implementation of Solution

The gradient descent method can be used to find a function’s minimum from some starting point. It relies on using the gradient to find the direction of the greatest rate decrease in the function and guessing the next point in that direction, some distance away from the starting point. By repeating this, the guessed points will trend towards the minimum. This algorithm is repeated until subsequent guessed points no longer change (or change very little), the guessed points diverge (or fail to converge), or the computation time becomes excessive. The most relevant line of code in our program (which determines each guess) is:

```
x = x - learning_rate * grad
```

The gradient descent method’s speed and effectiveness are governed by two hyperparameters, the learning rate and the maximum number of iterations. The learning rate, which is effectively the step size, determines the distance between guesses. Whereas the maximum number of iterations establishes a hard limit to the number of times the algorithm will run

One limitation of this method is that it can get stuck in local minima and saddle points. Both cases can happen if the learning rate is too small to escape them. Running the algorithm using multiple different starting points and hyperparameters can potentially alleviate this problem. Another limitation is that it can be difficult to know which hyperparameter values are optimal for a given function. Once again, running the algorithm multiple times with varying hyperparameter values can help fine tune the algorithm.

Analysis of Results

A number of different values for the hyperparameters were tested. For the learning rate, the values of 0.01, 0.001, 0.0001, and 0.00001 were tested, whereas the maximum number of iterations was set to 100, 1000, 10000, 100000, and 500000. The starting point for the function was also varied between tests. Two tools were used to analyze the results of each test: the value of the function after the final iteration and a graph showing the values of the function at each iteration. The value of the function after the final iteration showed how close the function was to converging for each set of hyperparameters. The graph also showed this, but it also showed the behavior of the algorithm at each iteration before the end point.

For all parameter values and starting points tested, the minimum was found at $x^* = (1, 1, 1)$, and the value of the function at that point was $f(x^*) = 0$. However, not all hyperparameter values pointed to that. The exact values obtained from sets of hyperparameters varied based on the parameters and starting point chosen. However, there were many clear trends.

Increasing the maximum number of iterations improved the results in all cases other than ones where the algorithm diverged. This came at the cost of increased computing time.

The lowest learning rate tested struggled to converge and, even after 500000 iterations, often did not produce a good result. Increasing the learning rate generally resulted in much higher quality results that could be obtained with fewer iterations. However, if the learning rate was too high, the algorithm rapidly diverged. In a few cases, a higher learning rate caused the values produced by the algorithm to oscillate, instead of diverging. This seemed to indicate that the algorithm's guesses were overshooting the minimum, but not by so much that they caused the algorithm to diverge.

Generally, the ideal combination of learning rate and maximum number of iterations is one that produces an acceptable result in as little time as possible while remaining reasonably stable.

Reflections

In general, the algorithm worked as expected and produced results there were able to be reliably analyzed. Including graphs of the values of the function at each iteration definitely helped with understanding the behavior of the algorithm at various points, particularly when the rate of convergence would suddenly change.

One possible improvement would be to add functionality that could automatically determine the highest learning rate that does not cause the algorithm to diverge. Doing so would make it easier to study the effects of the learning rate near the point of divergence, where the algorithm’s behavior is more likely to be unpredictable.