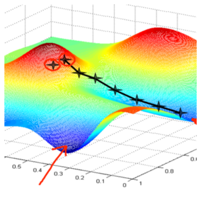
COMS 4995 Parallel Functional Programming
Final Project Proposal

Max Helman (mhh2148) and Riya Chakraborty (rc3242)

Fall 2020



1 Introduction

Gradient Descent is an optimization algorithm commonly used in regression analysis. The general idea is that you are given some sort of "loss function" of your data that you are looking to minimize, but you cannot necessarily use analytical methods such as differentiation to do so. Rather, you pick any arbitrary point on the loss function, compute the gradient at that point, and take a step in the direction of maximum negative slope. The size of the step is a function both of the gradient and the predetermined learning rate; you take either a predetermined finite number of steps or terminate once the numerical solution converges. However, since one or more sums (the value(s) of the the derivative(s), depending on the dimension of the data) must be computed at each step, there is slowdown for large datasets, which one will certainly encounter in the real world. There are a few tricks to speed this up, most of which involve extrapolating the value of the derivative(s) from a subset of the data points, but these all involve a loss of accuracy. We seek to parallelize this process to prevent compromise between runtime and accuracy.

2 Toy Example

Consider the case where, given data of the form (x, y) (a singular data point), we seek to find a regression line of the form $y = \beta_0 + \beta_1 x$ that minimizes the sum of the squared residuals. We do so by defining a sample loss function, which is itself a function of the parameters we are seeking to estimate, β_0 and β_1:

$$L(\beta_0, \beta_1) = \Sigma_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2$$

We start by entering a guess for β_0 and β_1, and compute the gradient of the loss function at this guess:

$$\nabla L(\beta_0, \beta_1) = \left(\frac{\partial L}{\partial \beta_0}, \frac{\partial L}{\partial \beta_1}\right) = (\Sigma_{i=1}^{n} - 2(y_i - \beta_0 - \beta_1 x_i), \Sigma_{i=1}^{n} - 2(y_i - \beta_0 - \beta_1 x_i)(-x_i))$$

Now, we adjust the parameters (β_0, β_1) by adding the partial derivative with respect to each parameter times a predetermined learning rate from the parameter, and repeat (updating the values of the parameters at each epoch) until we either reach a predetermined number of steps or the solution converges numerically, which can be measured via many different metrics. If we are estimating d parameters, have n data points, and have k epochs, this algorithm is $O(dnk)$, since at each epoch, a sum of length n must be computed for each of d partial derivatives. In the real world, datasets can be absolutely massive and we can be asked to estimate more than two parameters, so we are mostly concerned with the n and d terms with regards to performance (and often times, the k term is completely out of our control and depends on mathematics well beyond the scope of this project).

3 Potential for Parallelism

As the above analysis describes, the complexity of computing gradient descent begins to increase with bigger and more complicated data - namely, the key factors that we would like to consider are the number of data points (i.e. the sheer size of our data set) and the number of dimensions which we are optimizing/fitting along. Thus, these present themselves as **two areas** that could be parallelized. As the number of data points that are being "fitted" increases, the sum of squared residuals (or more generally, whichever loss function is being used in the gradient descent computation) becomes more and more computationally intensive and lends itself to a slower runtime. Furthermore, as we add more and more dimensions to the picture, we calculate more partial derivatives at each step, for each data point.

We would like to parallelize the calculation of the gradient (and illustrate the benefits of doing so via application of both sequential and parallel gradient descents on large datasets). The general strategy for this would be to parallelize the calculation of the partial derivatives, either by calculating each partial derivative in parallel (i.e. by the number of dimensions) but with sequential summation or calculating each partial derivative sequentially but performing the summations in parallel (i.e. by the number of data points). We also consider the possibility of performing both in parallel (the exact implementation details are to be determined).

4 Flexibility

As mentioned briefly above, there are two areas that we have currently identified where we would like to employ some parallelism - one is that of the data points, another is that of the dimensions. The ultimate goal is to successfully parallelize both in a variety of data contexts (and we think that this should be fairly doable), but in order to be flexible and consider time constraints, we consider the possibility that only one may be implemented. Moreover, we aim to generalize the algorithm such that we are able to run it on a wide range of real-world examples (various problems and their respective datasets) and examine how effective our newly parallel algorithm is. In particular, we want to study if it truly provides any improvement in performance of gradient descent computations on very large datasets with many variables to be optimized.