STATS 606 - Project Report

A Distributed Optimization Package for R

Benjamin Osafo Agyare Eduardo Ochoa Victor Verma

April 22, 2022

Abstract

In distributed optimization, there is a global objective function that is expressed as a sum of local objective functions, each of which is assigned to an agent. An example of an agent is a node in a computer network. Each agent attempts to minimize its local objective function using information on its function and information from the other agents. The aim of our project was to create an R package that implements two distributed optimization algorithms. We describe the algorithms and our package, which implements one of the algorithms. We also discuss the results of experiments in which we used our code to solve distributed versions of statistical problems.

1 Background

Because of the growth in computers' storage capacities, massive datasets are now ubiquitous. The processing and analysis of massive datasets can be accelerated through the use of a computer cluster. One problem that this approach faces is that usually statistical procedures aim to minimize a loss function that uses all of the data. Therefore, distributed solution methods are either necessary or at least highly desirable. For our project, we explored the implementation and use of two popular algorithms for distributed optimization: distributed gradient descent (DGD) [AN09] and the exact first-order algorithm (EXTRA) [WS15].

2 Introduction

Given a convex optimization problem, we say that it is a distributed optimization problem if we can express the global objective function as a sum of local objective functions belonging to different agents situated in a network [TY19]; the goal is to compute

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^{n} f_i(x)$$

Every agent minimizes its own objective function while exchanging information with other agents in the network.

Many machine learning and data analytics problems are of a scale that requires distributed optimization. However, most of the research papers about distributed optimization that we have seen focus on applications in electrical and digital systems. Given the algorithms' nature, we believe that they are well-suited to statistical problems like classification problems, regression analyses, etc.

In the class of distributed optimization algorithms, there are discrete-time and continuous-time algorithms. The former are more developed in the literature and are more feasible to implement.

2.1 Distributed Gradient Descent (DGD)

One of the most popular distributed optimization algorithms is Distributed Gradient Descent (DGD) [AN09]. This algorithm is intended to be used in situations where each agent has a convex local cost function. DGD uses diminishing step sizes. At time instant (step) k, agent i runs the following update:

$$x_i(k+1) = \sum_{j=1}^{N} w_{ij}(k)x_j(k) - \alpha(k)s_i(k), \tag{1}$$

where $x_i(k) \in \mathbb{R}^n$ is agent i's estimate of the optimal solution at time instant k, $w_{ij}(k)$ is the weight of the edge linking agents i and j, $s_i(k)$ is the gradient of the local objective function, and $\alpha(k) > 0$ is the diminishing step size. The matrix of weights W represents the level of communication between agents, and it must be doubly stochastic. The decreasing sequence of step sizes have the following properties:

$$\sum_{j=1}^{\infty} \alpha(k) = \infty \tag{2}$$

$$\sum_{j=1}^{\infty} (\alpha(k))^2 < \infty \tag{3}$$

Under the assumption that gradients are bounded, it is possible to show that DGD converges to one of the optimal solutions [AN09].

2.2 First-Order Algorithm (EXTRA)

Another popular algorithm is the Exact First-Order Algorithm (EXTRA) [WS15]. EXTRA uses the gradients from the last two iterations, unlike DGD, which just uses the gradient from the previous iteration. EXTRA can use a large fixed step size. In the first stage, agent i performs the following update:

$$x_i(1) = \sum_{j=1}^{N} w_{ij}x_j(0) - \alpha\nabla f_i(x_i(0)). \tag{4}$$

In the second stage, agent i performs another update:

$$x_i(k+2) = x_i(k+1) + \sum_{j=1}^{N} w_{ij}x_j(k+1) - \sum_{j=1}^{N} \tilde{w}_{ij}x_j(k) - \alpha(\nabla f_i(x_i(k+1)) - \nabla f_i(x_i(k))) \tag{5}$$

where $x_i(k) \in \mathbb{R}^n$ is agent i's estimate of the optimal solution at time instant k, $w_{ij}(k)$ and $\tilde{w}_{ij}(k)$ are the weight of the edge linking agents i and j, ∇f_i is the gradient of the local objective function, and $\alpha > 0$ is the fixed step size. The weight matrices are required to be doubly stochastic.

3 Algorithms

Algorithm 1 Distributed Gradient Descent (DGD)

1: Initialize $x_i(0)$ for each agent i.
2: At the k-th step, for each agent i update $x_i(k+1)$:

$$x_i(k+1) = \sum_{j=1}^{N} w_{ij}(k)x_j(k) - \alpha(k)s_i(k),$$

3: Repeat step 2 until the convergence criterion is met (tolerance/number of iterations)
4: **return** The last iteration for each agent $x_i(k)$.

Algorithm 2 First-Order Algorithm (EXTRA)

1: Initialize $x_i(0)$ for each agent i.
2: For each agent i update $x_i(1)$:

$$x_i(1) = \sum_{j=1}^{N} w_{ij}x_j(0) - \alpha\nabla f_i(x_i(0))$$

3: At the k-th step, for each agent i update $x_i(k+2)$:

$$x_i(k+2) = x_i(k+1) + \sum_{j=1}^{N} w_{ij}x_j(k+1) - \sum_{j=1}^{N} \tilde{w}_{ij}x_j(k) - \alpha(\nabla f_i(x_i(k+1)) - \nabla f_i(x_i(k)))$$

4: Repeat step 3 until the convergence criterion is met (tolerance/number of iterations)
5: **return** The last iteration for each agent $x_i(k)$.

4 Implementation

We set out to create an R package that implements DGD and EXTRA. To ensure that there wasn't already an R package that does this, we searched for distributed optimization packages in several ways. First, we looked at the CRAN task view for optimization and mathematical programming [TSB22]. The task view described all the packages on CRAN that can be used for optimization. Searching for strings like "distrib" and "DGD" and perusing the descriptions turned up nothing. We also found nothing when we searched for R repositories on GitHub using queries like "distributed optimization" and "distributed gradient descent".

We have created a package called `DistGD`. Currently, it only implements DGD. `DistGD` is in a GitHub repository located here. The repository README explains how to install the package. `DistGD` uses Apache Spark to carry out the computations. The package code interfaces with Spark through the `sparklyr` package [LKU+22]. The sole function that is exposed to the user is called `dgd()`. Its inputs are listed in Table 1. This function takes a reference `sc` to a Spark cluster; on

Argument	Class	Description
`sc`	`spark_connection`	A connection to a Spark cluster
`f_list`	`list`	A list of local objective functions
`grad_list`	`list`	An optional list of the gradients of the functions in `f_list`
`init_xs`	`list`	A list of initial values
`init_step_size`	`double`	An initial step size
`weight_mat`	`matrix`	The matrix of weights of the connections between the agents
`num_iters`	`integer`	The number of iterations to perform
`print`	`logical`	Whether to print the current minimizer estimates on each iteration
`make_trace`	`logical`	Whether to return a list with the minimizer estimates from each iteration

Table 1: The arguments of `dgd()`

each iteration, each compute instance in the cluster carries out the DGD update in Equation 1 for one local objective function. The `f_list` argument of `dgd()` is set to a list containing these functions. The user has the option of passing into `dgd()` a list of the gradients. If these aren't provided, then `grad()` in the `numDeriv` package is used to approximate them. The user needs to set the `weight_mat` argument to a matrix whose (i, j)-entry is the weight of the connection between agents i and j. Two other arguments of note are `init_step_size` and `num_iters`; `dgd()` currently uses a fixed step size and runs for a fixed number of iterations.

5 Experiments

We used the implementation in two statistics problems: OLS and Logistic Regression. The code for the experiments can be found here. For these experiments we generated 100,000 observations and we divided the data into 3 batches, and then we defined the 3 corresponding local functions:

$$f(\beta) = \sum_{i=1}^{n} L(y_i, x_i^T \beta) = \sum_{i=1}^{n_1} L(y_i, x_i^T \beta) + \sum_{i=n_1+1}^{n_2} L(y_i, x_i^T \beta) + \sum_{i=n_2+1}^{n} L(y_i, x_i^T \beta)$$

$$f(\beta) = f_1(\beta) + f_2(\beta) + f_3(\beta)$$

The results for linear regression are shown in Fig. 1; the resuls for logistic regression are shown in Fig. 2. In each problem, for each local objective function, the sequence of iterates appears to converge to the global minimizer.

6 Conclusion

In this report, we have described two frequently used algorithms for distributed optimization. We have introduced an R package we created that implements one of the algorithms and have shown how the package performs in two experiments.

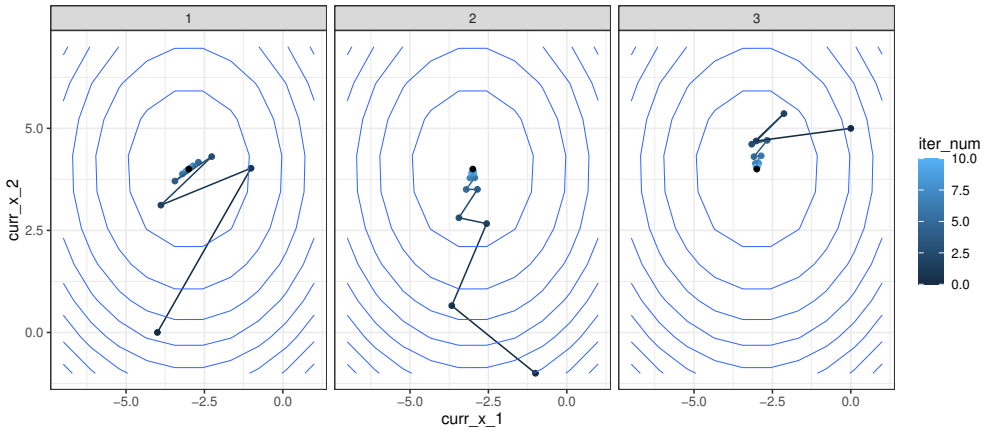

Figure 1: Ordinary Least Squares: These plots show the convergence behavior of the 3 local functions with respect to the global minimum.

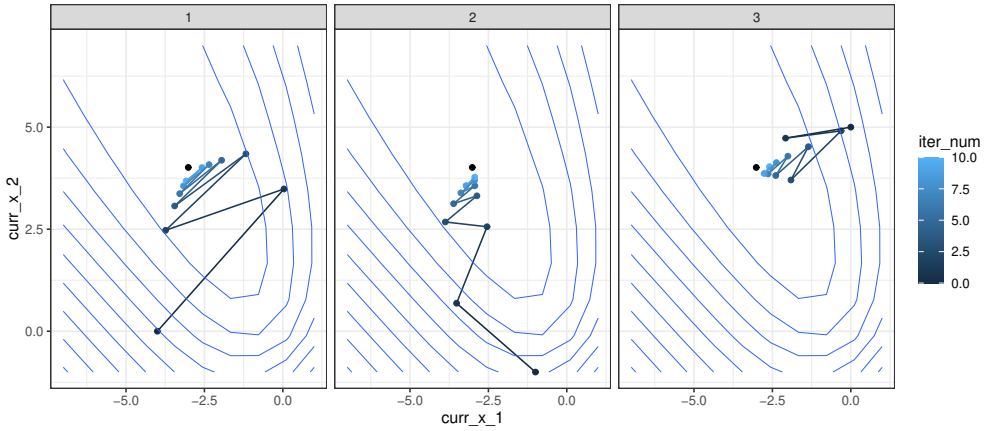

Figure 2: Logistic Regression: These plots show the convergence behavior of the 3 local functions with respect to the global minimum.

Several improvements could be made to the package. First, instead of executing DGD for a fixed number of iterations, `dgd()` should terminate the algorithm once some convergence criterion has been satisfied, like the norm of the gradient dropping below some threshold. Second, a decreasing sequence of step sizes that satisfies conditions (2) and (3) should be used instead of a fixed step size. Third, we have only tested the code in Spark's local mode, where there is just one compute instance. It would be useful to test the code on Great Lakes and on a cluster computing service like Databricks. We ran out of time before we could add a function for running EXTRA; doing this would be another way to enhance the package.

7 Contributions

Benjamin Osafo Agyare helped create the package harmonizing the scripts from Victor and experimentation from Eduardo.

Eduardo Ochoa Rivera helped with the literature review to choose the project's topic, helped in the design of the implementation, and helped with the numerical experiments.

Victor Verma helped write the code for the package and helped write the report.

References

[AN09] Asuman Ozdaglar Angelia Nedic. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions On Automatic Control*, 54, 2009.

[LKU+22] Javier Luraschi, Kevin Kuo, Kevin Ushey, JJ Allaire, Hossein Falaki, Lu Wang, Andy Zhang, Yitao Li, Edgar Ruiz, and The Apache Software Foundation. *sparklyr: R Interface to Apache Spark*, 2022. R package version 1.7.5.

[TSB22] Stefan Theussl, Florian Schwendinger, and Hans W. Borchers. CRAN task view: Optimization and mathematical programming, 2022. Version 2022-03-07.

[TY19] Xinlei Yi Tao Yang. A survey of distributed optimization. *Annual Reviews in Control*, 47:278–305, 2019.

[WS15] Qing Ling Wei Shi. Extra: An exact first-order algorithm for decentralized consensus optimization. *SIAM Journal on Optimization*, 25:44–966, 2015.