Communication-Efficient Distributed Optimization

Aleksei Volkov, Mikhail Shishatskiy

December 2021

Abstract

With the growing number of distributed computations, the need for optimal distributed algorithms has increased. The benefit from distributed computations is clear, as we increase the total computing power, thus reducing the computation time, and allowing processing of data extreme data volumes that are not feasible to handle using a single machine. However, there is a significant overhead caused by network communication. In this work, we implement some of the known algorithms for distributed optimization on top of the `torch.distributed` package, as it seems that there is currently no decent implementation of even the most common algorithms. We also compare the convergence of the algorithms as well as their efficiency concerning interconnect.

1 Introduction

When solving practical optimization problems, especially in the area of Machine Learning, it is often necessary to process large sets of data. When working with Big Data, enormous amounts of data might make it impossible to perform optimization on a single machine. Hence arises the need for distributed optimization algorithms that would be able to break a task down into a set of smaller problems each feasible to solve on a single machine, as demonstrated in Figure 1. However, the task is not straightforward, since in the presence of distributed communications a large overhead is associated with interconnect, raising the need for communication-efficiency of such algorithms.

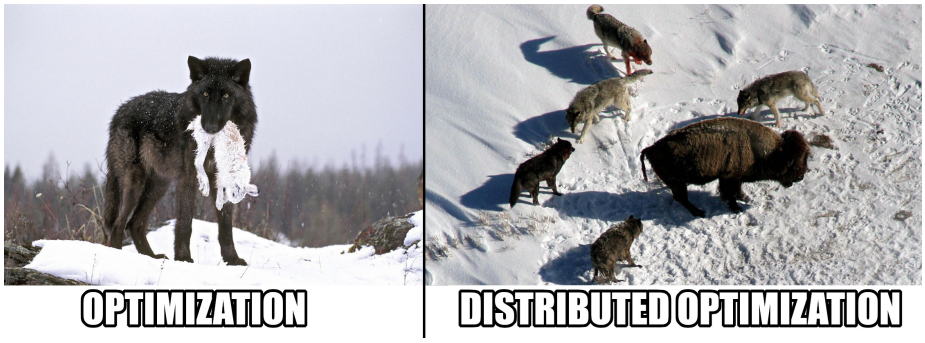

Figure 1: A visual demonstration of the advantages of distributed algorithms in tackling large problems using comparison with a pack of wolves on a hunt.

2 Consensus optimization

Suppose we are posed with an unconstrained convex optimization problem of form

$$\underset{x}{\text{minimize}} \quad \sum_{i=1}^{N} f_i(x). \tag{1}$$

We can reformulate it as

$$\begin{aligned} \underset{(\mathbf{x},z)}{\text{minimize}} \quad & \sum_{i=1}^{N} f_i(x_i) \\ \text{subject to} \quad & x_i = z,\ i = 1, \ldots, N. \end{aligned}$$

The idea is that each worker in a distributed system has its local part of the problem f_i along with the local solution x_i, there is a global solution z, and the constraint introduced enforces consensus of workers on the global optimum of the function. From now on, we will work with problems that consider decomposition in the form of (1). We shall note that most machine learning problems admit

such decomposition, as function f_i is simply a loss function on some part of the training dataset, and the whole sum is the total loss on the dataset.

3 Distributed optimization algorithms

In this section we give an overview of the algorithms we've implemented and compared. We do not provide proofs of correctness and/or convergence speed, and anyone interested might refer to the original articles, where such topics are discussed in detail.

3.1 Distributed Stochastic Gradient Descent

One possible approach to consensus optimization is distributed implementation of stochastic gradient descent: at each iteration, each machine calculates local gradient $\nabla f_i(z_i^{(t)})$, and then these are averaged to obtain the overall gradient $\nabla f(z^{(t)})$. After that, the parameters are updated accordingly to the classic implementation of stochastic gradient descent. This solution, however, is not the most efficient, as it uses parallelism purely to calculate gradients.

3.2 Alternating Direction Method of Multipliers (ADMM)

3.2.1 Introduction

Originally it is formulated for a slightly different formula, however it is trivially adapted for consensus optimization via induction. Originally, it is described for the problem of the following form:

$$\begin{aligned} \underset{(x,z)}{\text{minimize}} \quad & f(x) + g(z) \\ \text{subject to} \quad & Ax + Bz = c. \end{aligned}$$

The method is based on Dual Decomposition as well as Method of Multipliers. It starts with the formulation of the augmented Lagrangian

$$L_p(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + (\rho/2)\|Ax + Bz - c\|_2^2.$$

The method itself consists of iterations

$$x^{k+1} := \underset{x}{\operatorname{argmin}}\, L_p(x, z^k, y^k) \tag{2}$$

$$z^{k+1} := \underset{z}{\operatorname{argmin}}\, L_p(x^{k+1}, z, y^k) \tag{3}$$

$$y^{k+1} := y^k + \rho\left(Ax^{k+1} + Bz^{k+1} - c\right) \tag{4}$$

3.2.2 Assumptions

The proof of convergence relies on two assumptions

1. f, g are closed, proper and convex, which is satisfied iff its epigraph is a closed nonempty convex set;
2. For an unagmented Lagrangian $L_0(x, z, y)$ there exists a saddle point, i. e.

$$\exists x^*, y^*, z^* : \forall x, y, z \; L_0(x^*, z^*, y) \leqslant L_0(x^*, z^*, y^*) \leqslant L_0(x, z, y^*).$$

Under these assumptions, the ADMM algorithm satisfies *Residual, Objective and Dual value convergence*.

3.2.3 Practice notes

According to the paper by Boyd et al. [2011], the algorithm is slow to converge on real-world problems. However, it can achieve modest accuracy within tenth of iterations, which is sufficient in many applications.

3.3 Distributed Approximate Newton-type Method (DANE)

The DANE algorithm was originally introduced by Shamir et al. [2013]. The main idea of the method is to reduce the global consensus problem to a local Newton-type optimization

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x^{(t-1)}) + \langle \nabla f_i(x^{(t-1)}),\, x - x^{(t-1)} \rangle \\ & + \frac{1}{\eta} \left(D_{f_i}(x;\, x^{(t-1)}) + \frac{\mu}{2} \|x - x^{(t-1)}\|^2 \right) \end{aligned}$$

where $D_{f_i}(x;\, x^{(t-1)})$ is the Bregman distance associated with the local objective f_i. If we discard the terms that do not depend on the local state, this problem can be simplified to

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_i(x^{(t-1)}) + \langle \nabla f_i(x^{(t-1)}) - \eta \nabla f(x^{(t-1)}),\, x \rangle \\ & + \frac{\mu}{2} \|x - x^{(t-1)}\|^2. \end{aligned}$$

which is in-fact a mirror descent update (Beck and Teboulle [2003]).

The algorithm is multi-rounded, which means that local parameters and global gradient values are distributed across workers on each optimizer step. In the paper, authors prove, that if the problem consists of m objectives

$$f_i = \frac{1}{n} \sum_{1}^{n} f_{i,j}(x, z_j)$$

which are quadratic, L-smooth and λ-strongly convex, and $x \in \mathbb{R}^d$,

$$O\left(\frac{L^2}{n\lambda^2}\log(dm)\log\left(\frac{1}{\epsilon}\right)\right)$$

iterations are enough for the method to reach ϵ-suboptimal solution.

However, the method can be applied to any L-smooth and λ-strongly convex problem, but no rigorous proof has been produced. Our experimental observations confirm, that for machine learning applications with a big number of training samples, a couple of iterations are enough to reach an optimal solution.

4 Experimental observations

To evaluate the behavior of the algorithms described, we have taken a Fashion-MNIST classification problem, as it may be capable of demonstrating DANE's advantages over other algorithms. We've split the dataset across worker nodes, each worker solved the local optimization problem posed by the aforementioned local dataset. The Figure 2 shows the training history for each of the algorithms.

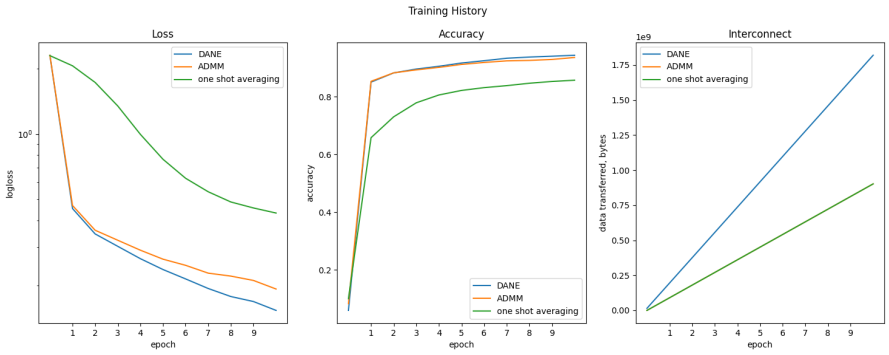

Figure 2: Training history for the `ADMM`, `DANE` and the simple `Gradient-averaging` method. Losses and accuracies are averaged between workers.

The plot shows that in equal conditions, the "smart" algorithms perform better than naïve gradient averaging. The optimizers reduce the loss significantly during the first epoch and reach an acceptable value in three epochs. After that, the model begins to overfit, and in a real-life problem, we would stop training. That is extremely useful when the number of parameters and associated data exchange expenses is significant.

Concerning data transfer, DANE has optimization steps with two rounds of communication, and ADMM has data transfer equal to such of gradient averaging. So we see twice as many bytes transferred as in gradient averaging.

Comparing DANE and ADMM methods, both perform quite similarly, but DANE shows a slightly faster convergence rate on the problem.

5 Conclusion

Methods designed for distributed optimization, like ADMM and DANE, seem to perform significantly better compared to distributed Gradient Descent but DANE is associated with greater interconnect expenses. However, it seems that additional expenses are negligible compared to faster convergence, as stochastic SGD would require much more total data transmitted to achieve the same loss value.

We have additionally discovered that the PyTorch library lacks implementation of the aforementioned methods, so we produced our implementation of DANE and ADMM as `torch.optim.Optimizer` subclasses. You may find them along with the experiment code and implementation of naïve gradient averaging in the following GitHub repository:

`https://github.com/Shishqa/distributed-optimization`.

References

A. Beck and M. Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 2003. doi: 10.1016/s0167-6377(02)00231-6.

S. Boyd, N. Parikh, E. Chu, E. K. wah Chu, E. K. wah Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 2011. doi: 10.1561/2200000016.

O. Shamir, N. Srebro, and T. Zhang. Communication efficient distributed optimization using an approximate newton-type method. *arXiv: Learning*, 2013. doi: null.