DRACO: Robust Distributed Training against Adversaries

Lingjiao Chen, Hongyi Wang, Dimitris Papailiopoulos
University of Wisconsin-Madison

1 INTRODUCTION

In recent years, the computational paradigm for large-scale machine learning (ML) has started to shift towards massively large distributed systems, sometimes comprising individually small, low-cost compute nodes [1, 7, 9, 10, 12]. Moreover, recently some production models are trained from user interaction with mobile devices under the guise of Federated Learning (FL) [3, 8]. A nontrivial challenge in distributed and federated learning is protecting against adversarial compute nodes that may affect the training process in order to bias and corrupt the end model that will be used in production.

A recent line of work [2, 6] studies this problem under a synchronous training setup, where a parameter server (PS) stores the model, and compute nodes evaluate gradient updates that are shipped to the PS then. Existing work suggests that instead of averaging the gradients, the PS can use a "robust" version of averaging, *e.g.*, the geometric median. Although this approach can be robust to up to roughly half the compute nodes being adversaries, we find that it is computationally inefficient, as the cost to compute the geometric median can dwarf the cost of computing a batch of gradients.

Our Contributions: In this work, we present DRACO, a framework that uses algorithmic redundancy to robustify synchronous training against adversarial compute nodes. The high level idea of DRACO is to allow each compute node to evaluate redundant gradients. Due to the redundant computation, the PS can (i) detect the adversarial nodes first, and then (ii) recover the correct gradient average from the gradient updates shipped by the non-adversarial nodes. We show that our capacity to tolerate adversaries, is proportional to the level of gradient redundancy, giving rise to a fundamental trade-off.

Comparing with the work of [2, 6], DRACO is significantly faster in both theory and experiments on PyTorch deployed on AWS EC2, the end model is identical to one trained under no adversaries (*e.g.*, algorithmic equivalence), and finally DRACO comes with black-box model- and problem-independent robustness guarantees irrespective of the non-convexity of the problem.

2 PROBLEM STATEMENT

The process of training a model from data can be cast as an optimization known as *empirical risk minimization* (ERM):

$$\min_{w} \frac{1}{n} \sum_{i=1}^{n} \ell(w; z_i) \tag{2.1}$$

SysML 18, Feb 2018, Stanford, CA, USA©2018

where $z_i \in \mathbb{R}^m$ represents the i-th data point, n is the total number of data points, $w \in \mathbb{R}^d$ is a parameter vector, and $\ell(w; z_i)$ is a loss function that measures the accuracy, or fidelity of the model parameterized by w with respect to the data point z_i.

One way to approximately solve the above ERM is through stochastic gradient descent (SGD), which operates as follows:

$$w_k = w_{k-1} - \gamma \cdot \nabla \ell(w_{k-1}; z_{i_k}), \tag{2.2}$$

where i_k is a random data-point index sampled from $\{1, \ldots, n\}$, and $\gamma > 0$ is the learning rate. In distributed setups, variants of SGD operate in a similar way as (2.2).

Here we consider mini-batch SGD [5] , one of the most widely implemented variants, which operates as follows: The PS stores a global model, while the set of n data points is distributed among the P compute nodes, and each compute node has access to a subset (or all) of the data. During the computation phase of mini-batch SGD, each of the P compute nodes samples B/P data points from its local cached subset of the data, and computes the gradients of these sampled data points with respect to the current model (where B is commonly referred to as the batch-size). When each compute node completes its local computation, it ships its gradient updates back to the PS. The PS, upon receiving the gradient updates, applies them to the global model (via averaging), and sends the updated model back to the workers. The algorithm then continues on to its next distributed iteration.

Adversarial Nodes. Here we consider the setup where a subset of nodes can act adversarially against the training process. The goal of an adversary can either be to completely mislead the end model, or bias it towards specific areas of the parameter space. Formally, an adversarial node is defined as follows.

DEFINITION 2.1. *A compute node is considered to be an adversarial node, if it does not return the prescribed gradient update given its allocated samples. Such a node can ship back to the PS any arbitrary update of dimension equal to that of the true gradient.*

It is easy to prove that mini-batch SGD will fail to converge even if there is only a single adversarial node. One may ask if there is a way to robustify distributed training algorithms like SGD in the presence of such adversaries. In the following, we show that it is possible to introduce robustness by algorithmic redundancy while guaranteeing an identical model to that of mini-batch SGD in the adversary-free case. We would like to note that —in contrast to previous works— our framework is applicable to any gradient-based algorithms beyond SGD, such as (accelerated/coordinate/conjugate) gradient descent and LBFGS [4, 11]. We skip the discussion of other training algorithms due to space constraints.

2.1 Related Work

A series of recent works has very recently initiated the research in adversary-tolerant distributed ML. Previous works [2, 6] studies how to tolerate adversaries by the use of the geometric median