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):

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

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 $\{1, \dots, n\}$, 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 as the update rule instead of averaging. Compared to off-the-shelf averaging update rule in adversary-free setting, the authors of the above papers show different variants of the geometric median rule can yield similar convergence results in terms of training error, while being robust to adversarial nodes. Specifically, the authors show that by using a variant of the geometric median, up to roughly half the compute nodes can be adversarial without affecting convergence rates significantly (for convex problems). Unfortunately, computing the geometric median exactly is a rather computationally inefficient procedure.

3 DRACO: ROBUST DISTRIBUTED TRAINING VIA ALGORITHMIC REDUNDANCY

In this section we present a sketch of our algorithmic redundancy approach.

In Draco instead of relying on computation from the PS side (unlike the geometric median (GM) technique of [2, 6]), we require that the compute nodes evaluate a few more gradients than what they are originally assigned with. Overall, if a node initially computes $\frac{B}{P}$ gradients, then we request that it computes a total of $r \cdot \frac{B}{P}$ gradients (selected in a systematic way that we omit to describe due to lack of space). We can show that Draco through the added redundancy of factor r can tolerate up to $(r-1)/2$ adversarial nodes (and hence up to 50% of all compute nodes) during training with absolutely no effect on the training or testing accuracy of the model trained a in an adversary-free setting.

Note that to mitigate the effect of adversarial nodes, Draco needs r times the amount of gradient computation of "vanilla" mini-batch SGD. Thus, a natural question is how small r can be to tolerate a certain number (say s) of adversarial nodes. In the long version of this paper, we manage to show that for algorithms that have a hard requirement on zero effect on the training procedure in the pressence of $(r-1)/2$ adversaries, a factor of r redundancy is information theoretically required. In addition, we seek to provide a lower bound on the computational complexity of the update rules at both the compute node and the PS.

The next question is, is there a systematic approach to design a framework that achieves this optimal behavior? The answer is positive, and its name is Draco. Specifically, Draco achieves an optimal result, by materializing tools that we borrow form Coding Theory (a repetition code and a cyclic code).

Brief summary of the techniques: Suppose $\hat{s} \triangleq 2s+1$ divides n. Our repetition code implemented by Draco works as follows. We first divide all compute nodes into $\hat{s}$ groups. In each group, all nodes compute the average of a set of exactly the same gradients, call it $\hat{g}$. Among all values returned by the compute nodes in the same group, the PS uses majority vote to select one value. This guarantees that as long as less than half of the nodes in a group are adversarial, the majority procedure negates the presence of adversaries and returns $\hat{g}$. We omit to describe the cyclic code due to space constraints, but its main utility is that it discards the "$\hat{s} \triangleq 2s+1$ divides n" condition required by the repetition code.

4 PRELIMINARY EXPERIMENTS

We compare our algorithm against the technique in [2, 6] that uses the geometric median for averaging, instead of mean. We implemented both the geometric median based mini-batch SGD and Draco in PyTorch. Here we show preliminary experiments on a simple fully connected layer neural network on MNIST, running on m4.2xlarge instances in AWS EC2. Our cluster contains 45 compute nodes.

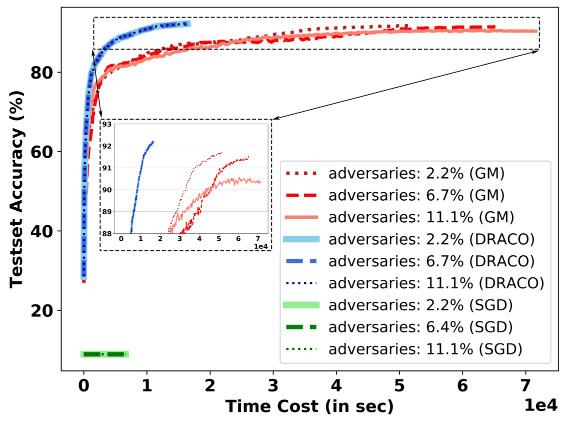

Figure 1: Convergence performance.

In our experiments, we set the number of adversarial nodes to $s = 1, 3, 5$ out of a total of 45 compute node. At each iteration, we randomly have 1, 3, and 5 adversarial nodes, each of which always returns a constant vector with magnitude -100 to the PS. As shown in Figure 1, while the ordinary SGD algorithms fails to converge, Draco converges with fewer iterations, smaller total runtime cost and higher testing accuracy than the geometric median approach. In fact, as shown from the table below our approach yields around an order of magnitude faster runtime.

% of adversarial nodes	2.2%	6.7%	11.1%
Draco	84.15	84.15	84.15
geometric median	401.49	419.65	506.53
Speedup gain	**4.77×**	**4.99×**	**6.02×**

Table 1: Time and Speedups to reach 88% Accuracy

REFERENCES

[1] M. Abadi et al. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*, 2016.
[2] P. Blanchard et al. Machine learning with adversaries: Byzantine tolerant gradient descent. In *NIPS*, 2017.
[3] K. Bonawitz et al. Practical secure aggregation for federated learning on user-held data. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.
[4] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *CoRR*, abs/1606.04838, 2016.
[5] J. Chen et al. Revisiting distributed synchronous SGD. In *CoRR*, 2016.
[6] Y. Chen et al. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. In *SIGMETRICS*, 2018.
[7] J. Duchi et al. Distributed dual averaging in networks. In *NIPS*, 2010.
[8] J. Konecny et al. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.
[9] M. Li et al. Scaling distributed machine learning with the parameter server. In *OSDI*, 2014.
[10] F. Niu et al. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2012.
[11] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2006.
[12] S. Zhang et al. Deep learning with elastic averaging sgd. In *NIPS*, 2015.