Convergence and Runtime of Approximate Gradient Coded Gradient Descent

Hongyi Wang [1] **Zachary Charles** [1] **Dimitris Papailiopoulos** [1]

Abstract

Gradient coding mitigates system delays in distributed gradient descent (GD) by introducing redundancy. Prior work has shown that exact gradient coded gradient descent (EGC-GD), which recovers the true gradient at each iteration, can significantly outperform uncoded GD. Approximate gradient coded gradient descent (AGC-GD) further reduces system delays by only computing approximate gradients. In this work, we establish explicit convergence rates for AGC-GD and show that under a probabilistic delay model, its expected runtime is faster than both EGC-GD and uncoded GD. We also provide a real systems implementation of AGC-GD. We use this implementation to conduct extensive experiments on real world datasets and distributed clusters, demosntrating that AGC-GD leads to significant speedups over both EGC-GD and uncoded GD.

1. Introduction

In order to contend with the size and scale of modern data and models, many production-scale machine learning solutions employ distributed training methods. Ideally, distributed implementations of learning algorithms have speedups that scale linearly with the number of compute nodes. Unfortunately, in practice these gains fall short of this, even with a small number of compute nodes. Several studies (Dean et al., 2012; Qi et al., 2017; Grubic et al., 2018), have consistently reported a tremendous gap between ideal and realizable speedup gains. One cause of this is the presence of *straggler* nodes, compute nodes whose runtime is significantly slower than the average node in the system. Stragglers become especially problematic when running *synchronous* distributed algorithms such as GD.

Recent work has used coding theory to mitigate the effects of stragglers. (Tandon et al., 2017) used *gradient codes* to mitigate stragglers in distributed gradient-based algorithms. Their gradient coding methods are able to compute the sum of n gradients using fewer than n compute nodes by assigning more tasks per worker. Unfortunately, if the number of tasks per worker is not too large, this may require waiting for nearly all the compute nodes to finish. A reasonable alternative to this are *approximate* gradient codes (AGCs), which only recover an approximate sum of gradients. AGCs require fewer non-stragglers, and utilize the observation that first order methods are often robust to small amounts of noise (Mania et al., 2017).

Our Contributions We provide detailed convergence rate and runtime of analyses of AGC-GD under the Polyak-Łojasiewicz (PL) condition, a non-convex generalization of strong convexity. While GD achieves linear convergence rates on such functions, SGD only achieves a rate of $O(1/T)$ (Karimi et al., 2016). We show that despite its stochasticity AGC-GD achieves linear convergence down to a small noise floor. Next, we use the probabilistic runtime model in (Lee et al., 2016) to analyze the expected runtime of AGC-GD. We show that given n compute nodes, AGC-GD is roughly $\log(n)$ faster than both EGC-GD and uncoded GD. Finally, we provide extensive empirical comparisons of these methods. Our results generally show that AGC-GD leads to up to $6\times$ and $3\times$ faster distributed training over uncoded GD and EGC-GD, respectively.

Related Work Prior strategies for mitigating stragglers includes replicating jobs across nodes (Shah et al., 2016) and dropping straggler nodes (Ananthanarayanan et al., 2013). (Lee et al., 2016) proposed the use of erasure codes for speeding up the computation of linear functions in distributed learning systems. Since then, many other works have analyzed the use of coding theory for distributed tasks with linear structure (Fahim et al., 2017; Park et al., 2018; Lee et al., 2017; Dutta et al., 2016; Yu et al., 2017). For nonlinear tasks, various gradient coding methods have been proposed (Tandon et al., 2017; Raviv et al., 2017; Halbawi et al., 2018; Li et al., 2018; Ye & Abbe, 2018) that analyze various theoretial and practical concerns of gradient codes.

AGCs have been shown to need many fewer non-straggler nodes. (Raviv et al., 2017; Charles et al., 2017). While

[1]University of Wisconsin–Madison. Correspondence to: Zachary Charles <zcharles@wisc.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

(Karakus et al., 2017) developed AGCs for problems relating to least-squares, (Raviv et al., 2017; Charles et al., 2017; Charles & Papailiopoulos, 2018) use AGCs in non-linear settings. (Raviv et al., 2017) uses expander graphs to construct approximate gradient codes with small error in the worst-case straggler setting, (Charles et al., 2017) focuses on the setting where the stragglers are chosen randomly. The aforementioned work generally only analyze proxies to the convergence rates of AGCs, and lack careful convergence rate and runtime analyses.

2. Gradient Coding

Suppose we wish to minimize $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$ using k compute nodes and one master node, often referred to as the parameter server (PS). In order to apply GD, we need to compute $\nabla f_i(x)$ for all i. In uncoded GD, the PS sends x to each node and partitions the n gradient computations among the nodes. After finishing, the nodes send their output back to the PS, which averages them and updates x. To avoid having to wait for the slowest node to finish, (Tandon et al., 2017) used *gradient coding* to ensure they could compute $\nabla f(x)$ from proper subsets of the k compute nodes.

In gradient coding, each node is assigned c of n possible tasks, with some amount of redundancy in the task assignment. After waiting for some number of compute nodes to finish, the PS takes a linear combination of the outputs of the non-straggler nodes and uses this as a gradient update. A gradient code consists of a choice of task assignment (encoding) and linear combination given a set of non-stragglers (decoding). In exact gradient coding, the PS waits until enough nodes finish such that the PS can compute $\nabla f(x)$ exactly, while in approximate gradient coding it does not.

We focus on the fractional repetition code (FRC) from (Tandon et al., 2017). While first used for exact gradient coding, it was used for approximate gradient coding in (Charles et al., 2017). Let $\ell = kc/n$ and for simplicity, assume ℓ is a positive integer. The FRC works as follows. The first ℓ nodes are assigned the first c tasks, the second ℓ nodes assigned the next c tasks, and so on. After waiting for $r < n$ of the compute nodes to finish, the PS decodes by taking the sum of outputs of one non-straggler node from each of the n/c groups (if that group has a non-straggler nodes). The output of the FRC is

$$g(x) = \frac{1}{n}\sum_{i=1}^{n/c} Y_i g[i](x) \tag{1}$$

where $g[i](x) = \sum_{j=1}^{n/c} \nabla f_{c(i-1)+j}(x)$ and Y_i is an indicator variable denoting whether or not there is a non-straggler among nodes $\{\ell(i-1)+1, \ldots, \ell i\}$.

Given x_0, we iteratively update via $x_{t+1} = x_t - \gamma g(x_t)$. Note that uncoded GD is the special case when n/k. (Tandon et al., 2017) shows that if $r \geq k - c + 1$, then no matter which nodes are stragglers, $g(x) = \nabla f(x)$, so the update is the same as uncoded GD. EGC-GD is the above update method when $r = k - c + 1$, and AGC-GD is the method when $r < k - c + 1$. Note that AGC-GD is inherently stochastic, as $g(x)$ may not equal the true gradient $\nabla f(x)$.

3. Theory

Convergence Analysis We first analyze the convergence rate of AGC-GD using techniques inspired by the analysis of GD on PL functions in (Karimi et al., 2016). Let $\Delta_T = \mathbb{E}[f(x_T) - f^*]$. We then have the following theorem.

Theorem 1. *Suppose that f is β-smooth and μ-PL with $\beta \geq \mu > 0$ and that for all $i \in [n]$ and x, $\|\nabla f_i(x)\| \leq \sigma$. Further suppose that $c \geq n\ln(2)/r$. If $\gamma = \beta^{-1}$, then*

$$\Delta_T \leq \left(1 - \frac{(1 - e^{-cr/n})\mu}{\beta}\right)^T \Delta_0 + \frac{2ce^{-cr/n}\sigma^2}{\mu n}.$$

When n is large and r is a constant fraction of n, this shows that AGC-GD has similar convergence rates to EGC-GD and uncoded GD on PL functions.

Probabilistic Runtime Analysis We next analyze the expected runtime of uncoded, EGC-, and AGC-GD. We assume $n = k$ to make our results easier to parse. Analogous results can be derived when $n \geq k$. As in (Lee et al., 2016), we assume that the amount of time required to compute a full gradient update on a single node is a continuous non-negative random variable T_0 with cumulative distribution function $Q(t)$. When the algorithm is partitioned in to B subtasks, we assume that each of the B subtasks has independent runtime T_i with cumulative distribution function $Q(Bt)$. This assumes the job partitioning is symmetric and that all compute nodes have the same compute power.

(Lee et al., 2016) found that empirically, $Q(t)$ is close to the cumulative distribution of a shifted exponential distribution. We therefore assume $\mathbb{P}[T_0 \leq t] = Q(t) = 1 - e^{-\lambda(t-1)}$. Here λ is the *straggling parameter*. If λ is smaller, the straggler effect is more pronounced. We then analyze the expected amount of time uncoded GD, EGC-GD, and AGC-GD require to achieve a given accuracy ϵ.

Let $\Delta_0 = f(x_0) - f^*$. Assume f is μ-PL and β-smooth, and let $\kappa = \mu/\beta$. In AGC-GD, suppose $r = \delta n$ for $\delta \in (0, 1)$. Suppose $1/\lambda$ is a positive integer. By (Lee et al., 2016), the expected runtime of EGC-GD is minimized when $c := 1/\lambda$ tasks per worker. To obtain a fair comparison, we assume $c = 1/\lambda$ in both EGC-GD and AGC-GD. Let $\eta = 1 - e^{-cr/n} = 1 - e^{-c\delta}$. For reasonable values of c, η is close to 1. We get the following theorem.

Theorem 2. *Let $T_\epsilon^{\mathrm{Unc}}, T_\epsilon^{\mathrm{EGC}}, T_\epsilon^{\mathrm{AGC}}$ denote the amount of time required for uncoded, EGC-, and AGC-GD with*

step-size $\gamma = 1/\beta$ *to reach error* ϵ. *If* $\epsilon \geq 3ce^{-cr/n}\sigma^2/n$,

$$\mathbb{E}[T_\epsilon^{\mathrm{Unc}}] \leq \frac{\log(\Delta_0/\epsilon)}{\log(1/(1-\kappa))}\frac{c\log(n)+c+1}{n}.$$

$$\mathbb{E}[T_\epsilon^{\mathrm{EGC}}] \leq \frac{\log(\Delta_0/\epsilon)}{\log(1/(1-\kappa))}\frac{c\log(n/c)+c+1}{n}.$$

$$\mathbb{E}[T_\epsilon^{\mathrm{AGC}}] \leq \frac{\log(3\Delta_0/\epsilon)}{\log\left(\frac{1}{1-\eta\kappa}\right)}\frac{c^2\log\left(\frac{1}{1-\delta}\right)+c^2+c}{n}.$$

Therefore, AGC-GD can lead to almost a $\log(n)$ speedup over uncoded GD and EGC-GD, even up to error levels as small as e^{-c}/n. In particular, as $n \to \infty$, the speedup gain of AGC-GD increases while the error level tends to 0.

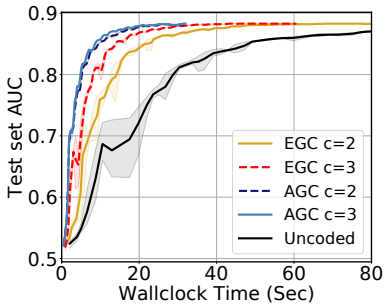
4. Experiments

We compare AGC-GD, uncoded GD, and EGC-GD on various logistic regression and least-squares tasks. While convex, such problems often have hundreds of thousands of features. Note that AGC-GD will stop earlier if some maximum fraction δ of the compute nodes have finished. For each experimental setup, we tune δ to get the best end-to-end performance.

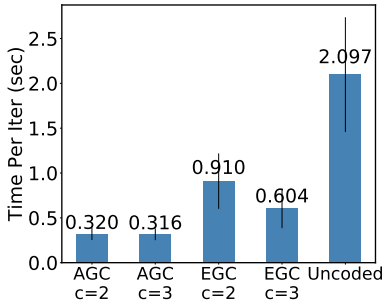
Experimental Setup We implemented all algorithms in python using MPI4py (Dalcin et al., 2011). We compared uncoded GD, EGC-GD, and AGC-GD with different redundancies across a distributed cluster consists of a PS node and 30 compute nodes. The compute nodes are `m1.small` instances on Amazon EC2. We used a larger instance `c3.8xlarge` as our PS node. Each worker is initially assigned a number of partitions of the data, with the number depending on the method used. In the t-th iteration, the PS broadcasts the latest model x_t to all workers. Each worker computes the gradient(s) of this model with respect to their data partition. Each worker then sends their gradient(s) to the PS. For uncoded GD and EGC-GD, we udpate the model once enough workers have finished such that we can compute the full gradient. In AGC-GD, we only wait for a fixed fraction δ of the workers to finish. We also performed versions of our experiments with extra artificial delays (using `time.sleep`) in the nodes to simulate practical scenarios where the communication overheads are heavy. The delay (in seconds) for each worker is drawn independently from an exponential distribution with parameter $\lambda = 1/2$.

While experimental results are shown just for logistic regression on the Amazon Employee Access dataset[1], other experiments are given in the extended version of this paper. We deployed one-hot encoding on each feature, and compared test set AUC (Bradley, 1997) of the three methods as a function of total time.

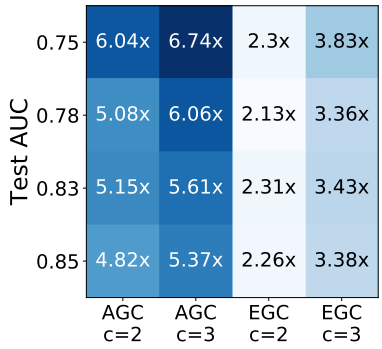
[1] `kaggle.com/c/amazon-employee-access-challenge`



(a) Convergence



(b) Per ter runtime



(c) Speedups

Figure 1. Results on Amazon dataset with artificial simulated stragglers: (a) convergence performance, (b) per iteration runtime, (c) Speedups of AGC and EGC over uncoded GD

Results The average times per iteration on the Amazon dataset are given in Figure 1(b), while the time to reach a given test AUC is given in Figure 1(a). Here, we assign $c = 2$ or $c = 3$ tasks per worker. For both c, we wait for at most $11/30 \approx 36.7\%$ of the workers to finish. Intuitively, as c increases we want to decrease δ, as otherwise AGC-GD will, with high probability be indistinguishable from EGC. Our results show that both AGC-GD and EGC-GD outperform uncoded GD. Moreover, as in our theory, AGC-GD consistently converges faster than EGC-GD. Speedups for both AGC-GD and EGC-GD were measured under simulated straggler effect. As shown in Figure 1, we observed that both AGC-GD and EGC-GD perform significantly faster than uncoded GD. Moreover, for a fixed redundancy ratio c, AGC-GD attains up to 3 times speedup gain over EGC-GD.

References

Ananthanarayanan, G., Ghodsi, A., Shenker, S., and Stoica, I. Effective straggler mitigation: Attack of the clones. In *NSDI*, volume 13, pp. 185–198, 2013.

Bradley, A. P. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

Charles, Z. and Papailiopoulos, D. Gradient coding via the stochastic block model. *arXiv preprint arXiv:1805.10378*, 2018.

Charles, Z., Papailiopoulos, D., and Ellenberg, J. Approximate gradient coding via sparse random graphs. *arXiv preprint arXiv:1711.06771*, 2017.

Dalcin, L. D., Paz, R. R., Kler, P. A., and Cosimo, A. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124–1139, 2011.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pp. 1223–1231, 2012.

Dutta, S., Cadambe, V., and Grover, P. Short-dot: Computing large linear transforms distributedly using coded short dot products. In *Advances In Neural Information Processing Systems*, pp. 2100–2108, 2016.

Fahim, M., Jeong, H., Haddadpour, F., Dutta, S., Cadambe, V., and Grover, P. On the optimal recovery threshold of coded matrix multiplication. In *Communication, Control, and Computing (Allerton), 2017 55th Annual Allerton Conference on*, pp. 1264–1270. IEEE, 2017.

Grubic, D., Tam, L., Alistarh, D., and Zhang, C. Synchronous multi-GPU deep learning with low-precision communication: An experimental study. 2018.

Halbawi, W., Azizan, N., Salehi, F., and Hassibi, B. Improving distributed gradient descent using reed-solomon codes. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 2027–2031. IEEE, 2018.

Karakus, C., Sun, Y., and Diggavi, S. Encoded distributed optimization. In *Information Theory (ISIT), 2017 IEEE International Symposium on*, pp. 2890–2894. IEEE, 2017.

Karimi, H., Nutini, J., and Schmidt, M. Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 795–811. Springer, 2016.

Lee, K., Lam, M., Pedarsani, R., Papailiopoulos, D., and Ramchandran, K. Speeding up distributed machine learning using codes. In *Information Theory (ISIT), 2016 IEEE International Symposium on*, pp. 1143–1147. IEEE, 2016.

Lee, K., Pedarsani, R., Papailiopoulos, D., and Ramchandran, K. Coded computation for multicore setups. In *Information Theory (ISIT), 2017 IEEE International Symposium on*, pp. 2413–2417. IEEE, 2017.

Li, S., Kalan, S. M. M., Avestimehr, A. S., and Soltanolkotabi, M. Near-optimal straggler mitigation for distributed gradient methods. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 857–866. IEEE, 2018.

Mania, H., Pan, X., Papailiopoulos, D., Recht, B., Ramchandran, K., and Jordan, M. I. Perturbed iterate analysis for asynchronous stochastic optimization. *SIAM Journal on Optimization*, 27(4):2202–2229, 2017.

Park, H., Lee, K., Sohn, J.-y., Suh, C., and Moon, J. Hierarchical coding for distributed computing. *arXiv preprint arXiv:1801.04686*, 2018.

Qi, H., Sparks, E. R., and Talwalkar, A. Paleo: A performance model for deep neural networks. In *Proceedings of the International Conference on Learning Representations*, 2017.

Raviv, N., Tamo, I., Tandon, R., and Dimakis, A. G. Gradient coding from cyclic mds codes and expander graphs. *arXiv preprint arXiv:1707.03858*, 2017.

Shah, N. B., Lee, K., and Ramchandran, K. When do redundant requests reduce latency? *IEEE Transactions on Communications*, 64(2):715–722, 2016.

Tandon, R., Lei, Q., Dimakis, A. G., and Karampatziakis, N. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pp. 3368–3376, 2017.

Ye, M. and Abbe, E. Communication-computation efficient gradient coding. *arXiv preprint arXiv:1802.03475*, 2018.

Yu, Q., Maddah-Ali, M., and Avestimehr, S. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication. In *Advances in Neural Information Processing Systems*, pp. 4403–4413, 2017.