Gradient Sparsification for Communication-Efficient Distributed Optimization

Jianqiao Wangni
University of Pennsylvania
Tencent AI Lab
wnjq@seas.upenn.edu

Jialei Wang
Two Sigma Investments
jialei.wang@twosigma.com

Ji Liu
University of Rochester
jliu@cs.rochester.edu

Tong Zhang
Tencent AI Lab
tongzhang@tongzhang-ml.org

Abstract

Modern large-scale machine learning applications require stochastic optimization algorithms to be implemented on distributed computational architectures. A key bottleneck is the communication overhead for exchanging information such as stochastic gradients among different workers. In this paper, to reduce the communication cost, we propose a convex optimization formulation to minimize the coding length of stochastic gradients. The key idea is to randomly drop out coordinates of the stochastic gradient vectors and amplify the remaining coordinates appropriately to ensure the sparsified gradient to be unbiased. To solve the optimal sparsification efficiently, several simple and fast algorithms are proposed for an approximate solution, with a theoretical guarantee for sparseness. Experiments on ℓ_2 regularized logistic regression, support vector machines, and convolutional neural networks validate our sparsification approaches.

1 Introduction

Modern large-scale machine learning applications require scaling stochastic optimization algorithms [30, 28, 15, 11] to distributed computational architectures [10, 39, 12, 20, 19, 38] or multicore systems [26, 9, 22, 25]. This problem has drawn significant attention from theoretical perspectives about its communication complexity [33, 43, 3]. In the synchronized stochastic gradient method, each worker processes a random minibatch of its training data, and then the local updates are synchronized by making an *All-Reduce* step, which aggregates stochastic gradients from all workers, and taking a *Broadcast* step that transmits the updated parameter vector back to all workers. The process is repeated until a certain convergence criterion is met. An important factor that may significantly slow down any optimization algorithm is the communication cost among workers. Even for the single machine multi-core setting, where the cores communicate with each other by reading and writing to a chunk of shared memory, conflicts of (memory access) resources may significantly degrade the efficiency. The existing works on distributed machine learning mainly focus on two directions: 1) how to design communication efficient algorithms to reduce the round of communications among workers [43, 31, 13, 27, 42], and 2) about how to use large mini-batches without compromising the convergence speed [21, 35, 36]. There are solutions to specific problems like mean estimation [17, 32], component analysis [23], clustering [6], sparse regression [18] and boosting [7]. Several papers considered the problem of reducing the precision of gradient by using fewer bits to represent floating pointing numbers [29, 44, 37, 2, 40, 8] or only transmitting coordinates [1, 24].

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

In this paper, we propose a novel approach to complement these methods above. Specifically, we sparsify stochastic gradients appropriately to reduce the communication cost, with minor sacrifice on the number of iterations. The key idea behind of our sparsification technology is to drop some coordinates of the stochastic gradient and appropriately amplify the remaining coordinates to ensure the unbiasness of the sparsified stochastic gradient. The sparsification approach can significantly reduce the coding length of the stochastic gradient and only slightly increase the variance of the stochastic gradient. This paper proposes a convex formulation to achieve the best tradeoff of variance and sparsity: the optimal probabilities to sample coordinates can be obtained given any fixed variance budget. To solve this optimization within a linear time, several efficient algorithms are proposed to find approximate optimal solutions with sparsity guarantees. The proposed sparsification approach can be encapsulated seamlessly to many bench-mark stochastic optimization algorithms in machine learning, such as SGD [4], SVRG [15, 41], SAGA [11], and ADAM [16]. Empirical study is provided to validate the proposed approach on ℓ_2 regularized logistic regression, support vector machines, and convolutional neural networks on both synthetic and real-world data sets.

2 Algorithms

We consider the problem of sparsifying a stochastic gradient vector, and formulate it as a linear planning problem. The following notations will be used throughout the paper. Consider a training data set $\{x_n\}_{n=1}^N \subset \Omega$, and each training data point x_n is associating with a loss function $f_n : \Omega \to \mathbb{R}$, that is associating with the nth data point x_n. We use $w \in \mathbb{R}^d$ to denote the model parameter vector, and consider solving the following optimization problem using stochastic optimization methods:

$$\min_w \quad f(w) := \frac{1}{N}\sum_{n=1}^N f_n(w), \quad w_{t+1} = w_t - \eta_t g_t(w_t), \tag{1}$$

where t indicates the iterations and $\mathbb{E}\left[g_t(w)\right] = \nabla f(w)$. serves as an unbiased estimate for the true gradient $\nabla f(w_t)$. The following are two ways to choose g_t, like SGD [41, 4, 34] and SVRG [15]

$$\text{SGD}: \quad g_t(w_t) = \nabla f_{n_t}(w_t), \qquad \text{SVRG}: \quad g_t(w_t) = \nabla f_{n_t}(w_t) - \nabla f_{n_t}(\widetilde{w}) + \nabla f(\widetilde{w}) \tag{2}$$

where n_t is uniformly sampled from the data set and $\widetilde{w}$ is a reference point. *The above derivation implies that the convergence of SGD is significantly dominated by $\mathbb{E}\|g_t(w_t)\|^2$ or equivalently the variance of $g_t(w_t)$.* It can be seen from the following simple derivation. Assume that the loss function $f(w)$ is L-smooth with respect to w, which means that for $\forall x, y \in \mathbb{R}^d$, $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$ (where $\|\cdot\|$ is the ℓ_2-norm). Then the expected loss function is given by

$$\mathbb{E}\left[f(w_{t+1})\right] \leq \mathbb{E}\left[f(w_t) + \nabla f(w_t)^\top (x_{t+1} - x_t) + \frac{L}{2}\|x_{t+1} - x_t\|^2\right]$$

$$= \mathbb{E}\left[f(w_t) - \eta_t \nabla f(w_t)^T g_t(w_t) + \frac{L}{2}\eta_t^2 \|g_t(w_t)\|^2\right] = f(w_t) - \eta_t \|\nabla f(w_t)\|^2 + \frac{L}{2}\eta_t^2 \underbrace{\mathbb{E}\,\|g_t(w_t)\|^2}_{\text{variance}},$$

where the inequality is due to the Lipschitzian property, and the second equality is due to the unbiased nature of the gradient $\mathbb{E}\left[g_t(w)\right] = \nabla f(w)$. So the magnitude of $\mathbb{E}(\|g_t(w_t)\|^2)$ or equivalently the variance of $g_t(w_t)$ will significantly affect the convergence efficiency.

Next we consider how to reduce the communication cost in distributed machine learning by using a sparsification of stochastic gradient $g_t(w_t)$, denoted by $Q(g(w_t))$, such that $Q(g_t(w_t))$ is unbiased, and has a relatively small variance. In the following, to simplify notation, we denote the current stochastic gradient $g_t(w_t)$ by g for short, in which we drop the subscript t and w_t. Note that g can be obtained either by SGD or SVRG. We also let g_i be the i-th component of vector $g \in \mathbb{R}^d$: $g = [g_1, \ldots, g_d]$. We propose to randomly drop out the i-th coordinate by a probability of $1 - p_i$, which means that the coordinates remains non-zero with a probability of p_i. Let $Z_i \in \{0, 1\}$ be a binary-valued random variable indicating whether the i-th coordinate is selected: $Z_i = 1$ with probability p_i and $Z_i = 0$ with probability $1 - p_i$. Then, to make the resulting sparsified gradient vector $Q(g)$ unbiased, we amplify the non-zero coordinates, from g_i to g_i/p_i. So the final sparsified vector is $Q(g)_i = Z_i(g_i/p_i)$. The whole protocol can be summarized as follows:

$$\text{Gradients } g = [g_1, g_2, \cdots, g_d], \text{Probabilities } p = [p_1, p_2, \cdots, p_d], \text{Selectors } Z = \quad [Z_1, Z_2, \cdots, Z_d],$$

$$\text{where } P(Z_i = 1) = p_i, \qquad \Longrightarrow \quad \text{Results } Q(g) = \quad \left[Z_1\frac{g_1}{p_1}, Z_2\frac{g_2}{p_2}, \cdots, Z_d\frac{g_d}{p_d}\right]$$

We note that if g is an unbiased estimate of the gradient, then $Q(g)$ is also an unbiased estimate of the gradient since $\mathbb{E}\left[Q(g)_i\right] = p_i \times \frac{g_i}{p_i} + (1 - p_i) \times 0 = g_i$.

In distributed machine learning, each worker calculates gradient g and transmits it to the master node or the parameter server for update. We use an index m to indicate a node, and assume there are totally M nodes. The gradient sparsification method can be used with a synchronous distributed stochastic optimization algorithm in Algorithm 1. Asynchronous algorithms can also be used with our technique in a similar fashion.

Algorithm 1 A synchronous distributed optimization algorithm

1: Initialize the clock $t = 0$ and initialize the weight w_0.
2: **repeat**
3: Each worker m calculates local gradient $g^m(w_t)$ and the probability vector p^m.
4: Sparsify the gradients $Q(g^m(w_t))$ and take an *All-Reduce* step $v_t = \frac{1}{M}\sum_{m=1}^{M} Q(g^m(w_t))$.
5: Broadcast the average gradient v_t and take a descent step $w_{t+1} = w_t - \eta_t v_t$ on all workers.
6: **until** convergence or the number of iteration reaches the maximum setting.

Our method could be combined with other methods which are orthogonal to us, like only transmitting large coordinates and accumulating the gradient residual which might be transmitted in the next step [1, 24]. Advanced quantization and coding strategy from [2] can be used for transmitting valid coordinates of our method. In addition, a similar objective was also formulated in [17] for studying the mean estimation problem on distributed data, with a statistical guarantee under skewness, comparably, we studied a more generalized problem, with a specific algorithm proposed to actually determine the sparsification probability vectors.

2.1 Mathematical formulation

Although the gradient sparsification technique can reduce communication cost, it increases the variance of the gradient vector, which might slow down the convergence rate. In the following section we will investigate how to find the best tradeoff between sparsity and variance for the sparsification technique. In particular, we consider how to find out the optimal sparsification strategy, given a budget of maximal variance. First note that the variance of $Q(g)$ can be bounded by

$$\mathbb{E}\sum_{i=1}^{d}[Q(g)_i^2] = \sum_{i=1}^{d}\left[\frac{g_i^2}{p_i^2} \times p_i + 0 \times (1 - p_i)\right] = \sum_{i=1}^{d}\frac{g_i^2}{p_i}.$$

In addition, the expected sparsity of $Q(g_i)$ is given by $\mathbb{E}\left[\|Q(g)\|_0\right] = \sum_{i=1}^{d} p_i$. In this section, we try to balance these two factors (sparsity and variance) by formulating it as a linear planning problem as follows:

$$\min_{p} \quad \sum_{i=1}^{d} p_i \quad \text{s.t.} \quad \sum_{i=1}^{d}\frac{g_i^2}{p_i} \leq (1+\epsilon)\sum_{i=1}^{d} g_i^2, \tag{3}$$

where $0 < p_i \leq 1, \forall i \in [d]$, and ϵ is a factor that controls the variance increase of the stochastic gradient g. This leads to an optimal strategy for sparsification given an upper bound on the variance. The following proposition provides a closed-form solution for problem (3).

Proposition 1. *The solution to the optimal sparsification problem* (3) *is a probability vector* p *such that* $p_i = \min(\lambda|g_i|, 1), \forall i \in [d]$*, where* $\lambda > 0$ *is a certain constant only depending on* g *and* ϵ*.*

2.2 Sparsification algorithms

In this section we propose two algorithms for efficiently calculating the optimal probability vector p in Proposition 1. Since $\lambda > 0$, by the complementary slackness condition we have

$$\sum_{i=1}^{d}\frac{g_i^2}{p_i} - (1+\epsilon)\sum_{i=1}^{d} g_i^2 = \sum_{i=1}^{k} g_{(k)}^2 + \sum_{i=k+1}^{d}\frac{|g_{(i)}|}{\lambda} - (1+\epsilon)\sum_{i=1}^{d} g_i^2 = 0,$$

where $\{g_{(i)}\}_{i=1}^d$ is a sorted version of $\{g_i\}_{i=1}^d$ in a descending order. This further implies

$$\lambda = \frac{\sum_{i=k+1}^d |g_{(i)}|}{\epsilon \sum_{i=1}^d g_i^2 + \sum_{i=k+1}^d g_{(i)}^2}, \quad |g_{(k+1)}| \left(\sum_{i=k+1}^d |g_{(i)}| \right) \leq \epsilon \sum_{i=1}^d g_i^2 + \sum_{i=k+1}^d g_{(i)}^2. \tag{4}$$

where we used the constraint $\lambda |g_{(k+1)}| \leq 1$. It follows that we should find the smallest k which satisfies the above inequality. Based on above reasoning, we get the following closed-form solution for p_i in Algorithm 2.

Algorithm 2 Closed Form Solution

1: Find the smallest k such that the second inequality of (4) is true, and let S_k be the set of coordinates with top k largest magnitude of $|g_i|$.
2: Set the probability vector p by

$$p_i = \begin{cases} 1, & \text{if } i \in S_k \\ \frac{|g_i| \left(\sum_{j=k+1}^d |g_{(j)}| \right)}{\epsilon \sum_{j=1}^d g_{(j)}^2 + \sum_{j=k+1}^d g_{(j)}^2}, & \text{if } i \notin S_k. \end{cases}$$

In practice, using Algorithm 2 to find S_k requires partial sorting of the gradient magnitude values, which could be computationally expensive. Therefore we developed a greedy algorithm for approximately solving the problem. We pre-define a sparsity parameter $\rho \in [0, 1]$, which implies that we aim to find p_i that satisfies $\sum_i p_i / d \approx \rho$. Loosely speaking, we want to initially set $\widetilde{p}_i = \rho d |g_i| / \sum_i |g_i|$, which sums to $\sum_i \widetilde{p}_i = \rho d$, meeting our requirement on ρ. However, by the truncation operation $p_i = \min(\widetilde{p}_i, 1)$, the expected nonzero density will be less than ρ. Now, we can use an iterative procedure, where in the next iteration, we fix the set of $\{p_i : p_i = 1\}$ and scale the remaining values, as summarized in Algorithm 3. The algorithm is much easier to implement, and computationally more efficient on parallel computing architecture.

Algorithm 3 Greedy Algorithm

1: **Input** $g \in \mathbb{R}^d$, $\rho \in [0, 1]$. Initialize $p^0 \in \mathbb{R}^d$, $j = 0$. Set $p_i^0 = \min(\rho d |g_i| / \sum_i |g_i|, 1)$ for all i.
2: **repeat**
3: Identify an active set $\mathcal{I} = \{1 \leq i \leq D | p_i^j \neq 1\}$ and compute $c = (\rho d - d + |\mathcal{I}|) / \sum_{i \in \mathcal{I}} p_i^j$.
4: Recalibrate the values by $p_i^{j+1} = \min(c p_i^j, 1)$. $j = j + 1$.
5: **until** If $c \leq 1$ or j reaches the maximum iterations. Return $p = p^j$.

2.3 Coding strategy

Once we have computed a sparsified gradient vector $Q(g)$, we need to pack the resulting vector into a message for transmission. Here we apply a hybrid strategy for encoding $Q(g)$. Suppose that computers represent a floating point scalar using b bits, which is enough for a precise representation of any variables with negligible loss in precision. We use two vectors for representing non-zero coordinates, one for coordinates $i \in S_k$, and the other for coordinates $i \notin S_k$. The vector $Q_A(g)$ represents $\{g_i : i \in S_k\}$, where each item of $Q_A(g)$ needs $\log d$ bits to represent the coordinates and b bits for the value g_i / p_i. The vector $Q_B(g)$ represents $\{g_i : i \notin S_k\}$, since in this case, we have $p_i = \lambda |g_i|$, we have for all $i \notin S_k$ the quantized value $Q(g_i) = g_i / p_i = \text{sign}(g_i) / \lambda$. Therefore to represent $Q_B(g)$, we only need one floating point $1/\lambda$, plus the non-zero coordinates i and its sign $\text{sign}(g_i)$. Here we give an example about the format,

$$\text{sparsified vector}: \quad \left[\frac{g_1}{p_1}, 0, 0, \frac{g_4}{p_4}, \frac{g_5}{p_5}, \frac{g_6}{p_6}, \cdots, 0 \right], \text{Vector } Q_A(g): \quad \left[1, \frac{g_1}{p_1}, 5, \frac{g_5}{p_5} \cdots, 0 \right],$$
$$\text{Vector } Q_B(g): \quad [4, -1/\lambda, 6, 1/\lambda, \cdots].$$

where $i = 1, 5 \in S_k, i = 4, 6 \notin S_k, g_4 < 0, g_6 > 0$. Moreover, we can also represent the indices of A and vector $Q_B(g)$ using a dense vector of $\widetilde{q} \in \{0, \pm 1, 2\}^d$, where each component $\widetilde{q}_i$ is defined as $Q(g_i) = \lambda Q(g_i)$ when $i \notin S_k$ and $\widetilde{q}_i = 2$ if $i \in S_k$. Using the standard entropy coding, we know that $\widetilde{q}$ requires at most $\sum_{\ell=-1}^2 d_\ell \log_2(d/d_\ell) \leq 2d$ bits to represent.

3 Theoretical guarantees on sparsity

In this section we analyze the expected sparsity of $Q(g)$, which equals to $\sum_{i=1}^{d} p_i$. In particular we show when the distribution of gradient magnitude values is highly skewed, there is a significant gain in applying the proposed sparsification strategy. First, we define the following notion of approximate sparsity on the magnitude at each coordinate of g:

Definition 2. *A vector $g \in \mathbb{R}^d$ is (ρ, s)-approximately sparse if there exists a subset $S \subset [d]$ such that $|S| = s$ and $\|g_{S^c}\|_1 \leq \rho \|g_S\|_1$, where S^c is the complement of S.*

The notion of (ρ, s)-approximately sparsity is inspired by the restricted eigenvalue condition used in high-dimensional statistics [5]. (ρ, s)-approximately sparsity measures how well the signal of a vector is concentrated on a small subset of the coordinates of size s. As we will see later, the quantity $(1+\rho)s$ plays an important role in establish the expected sparsity bound. Note that we can always take $s = d$ and $\rho = 0$ so that (ρ, s) satisfies the above definition with $(1+\rho)s \leq d$. If the distribution of magnitude values in g is highly skewed, we would expect the existence of (ρ, s) such that $(1+\rho)s \ll d$. For example when g is exactly s-sparse, we can choose $\rho = 0$ and the quantity $(1+\rho)s$ reduces to s which can be significantly smaller than d.

Lemma 3. *If the gradient $g \in \mathbb{R}^d$ of the loss function is (ρ, s)-approximately sparse as in Definition 2. Then we can find a sparsification $Q(g)$ with $\epsilon = \rho$ in* (3) *(that is, the variance of $Q(g)$ is increased by a factor of no more than $1+\rho$), and the expected sparsity of $Q(g)$ can be upper bounded by $\mathbb{E}\left[\|Q(g)\|_0\right] \leq (1+\rho)s$.*

Remark 1. *Lemma 3 indicates that the variance after sparsification only increase by a factor of $(1+\rho)$, while in expectation we only need to communicate a $(1+\rho)s$-sparse vector after sparsified. In order to achieve the same optimization accuracy, we may need to increase the number of iterations by a factor up to $(1+\rho)$, and the overall number of floating point numbers communicated is reduced by a factor of up to $(1+\rho)^2 s/d$.*

Above lemma shows the number of floating point numbers needed to communicate per iteration is reduced by the proposed sparsification strategy. As shown in Section 2.3, we only need to use one floating point number to encoding the gradient values in S_k^c, so there is a further reduction in communication when considering the total number of bits transmitted, this is characterized by the Theorem below.

Theorem 4. *If the gradient $g \in \mathbb{R}^d$ of the loss function is (ρ, s)-approximately sparse as in Definition 2, and a floating point scalar costs b bits, then the coding length of $Q(g)$ in Lemma 3 can be bounded by $s(b + \log_2 d) + \min(\rho s \log_2 d, d) + b$.*

Remark 2. *The coding length of the original gradient vector g is db, by considering the slightly increased number of iterations to reach the same optimization accuracy, the total communication cost is reduced by a factor of at least $(1+\rho)((s+1)b + \log_2 d)/db$.*

4 Experiments

In this section we conduct experiments to validate the effectiveness and efficiency of the proposed sparsification technique. We use ℓ_2 regularized logistic regression as an example for convex problems, and take convolutional neural networks as an example for non-convex problems. The sparsification technique shows strong improvement over a baseline of uniform sampling approach, the iteration complexity is relatively less increased comparing to the communication costs we saved. Moreover, we also conduct asynchronous parallel experiments on the shared memory architecture. In particular, our experiments show that the proposed sparsification technique significantly reduces the conflicts among multiple threads and dramatically improves the performance. In all experiments, the probability vector p is calculated by Algorithm 3 and set the maximum iterations to be 2, which generates high quality approximation of the optimal p vector.

We first validate the sparsification technique on the ℓ_2 regularized logistic regression problem using SGD and SVRG respectively: $f(w) = \frac{1}{N}\sum_n \log_2\left(1 + \exp(-a_n^\top w b_n)\right) + \lambda_2 ||w||_2^2$, where $a_n \in \mathbb{R}^d$, $b_n \in \{-1, 1\}$. The experiments are conducted on synthetic data for the convenience to control the data sparsity. The mini-batch size is set to be 8 by default unless otherwise specified. We simulated with $M = 4$ machines, where one machine is both a worker and the master that aggregates stochastic

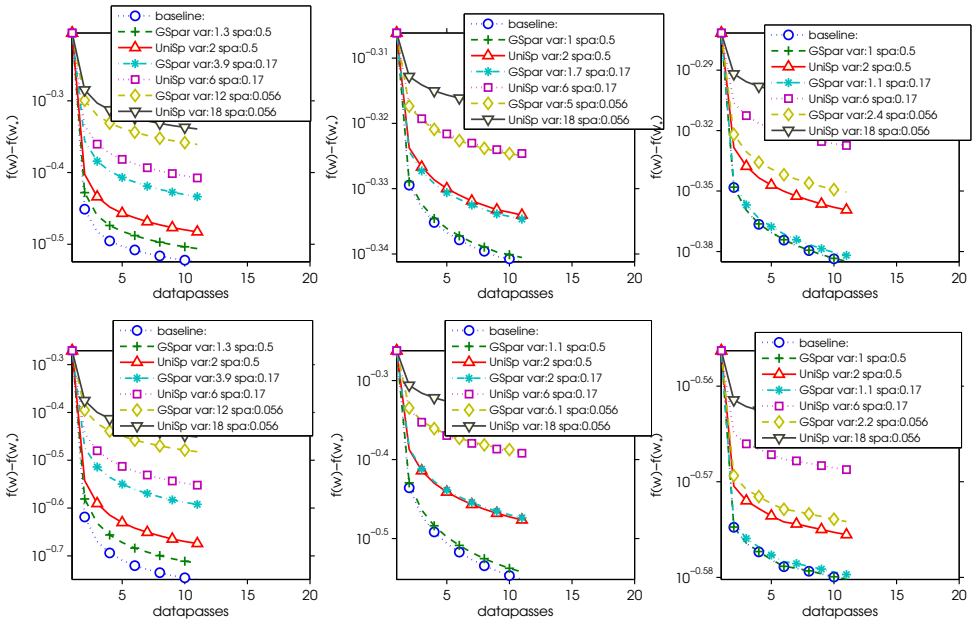

Figure 1: SGD type comparison between gradient sparsification (GSpar) with random sparsification with uniform sampling (UniSp).

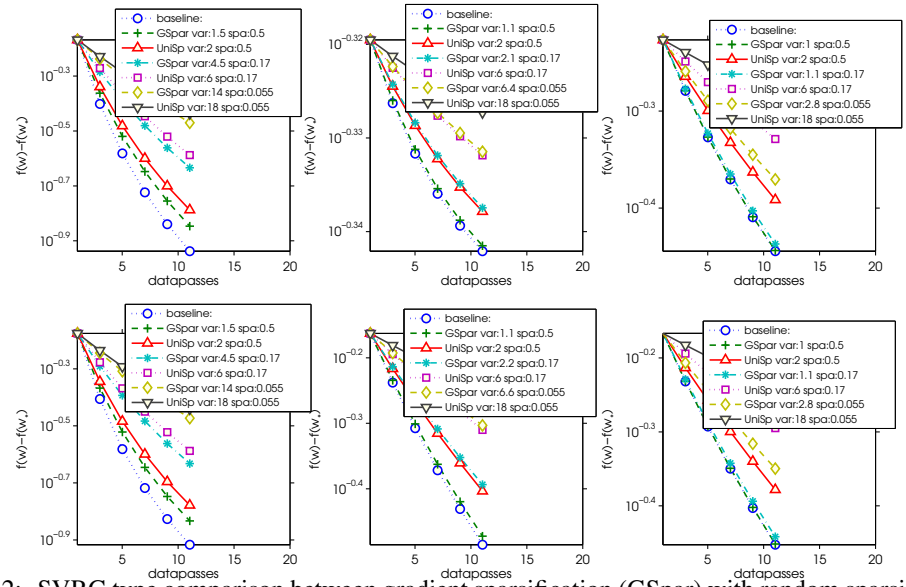

Figure 2: SVRG type comparison between gradient sparsification (GSpar) with random sparsification with uniform sampling (UniSp)

gradients received from other workers. We compare our algorithm with a uniform sampling method as baseline, where each element of the probability vector is set to be $p_i = \rho$. In this method, the sparsified vector is of ρ-sparse in expectation. The data set $\{a_n\}_{n=1}^N$ is generated as follows

$$\text{dense data:} \quad \bar{a}_{ni} \sim \mathcal{N}(0,1), \quad \forall i \in [d], n \in [N], \text{sparsify:} \quad \bar{B} \sim \text{Uniform}[0,1]^d, \quad \bar{B}_i \leftarrow C_1 \bar{B}_i,$$
$$\text{if:} \quad \bar{B}_i \leq C_2, \quad \forall i \in [d], \quad a_n \leftarrow \bar{a}_n \odot \bar{B}, \text{label:} \quad \bar{w} \sim \mathcal{N}(0, I), \quad b_n \leftarrow \text{sign}(\bar{a}_n^\top \bar{w})$$

where $\odot$ is the element-wise multiplication. We put the explanation for this process in the appendix due to limited space. We should note that by the aforementioned data generation process, the parameters C_1 and C_2 control the sparsity of data points and the gradients: the smaller these two constants are, the sparser the gradients are; and the gradient of linear models on the dataset should be expected to be $\left((1-C_2)d, C_2 \frac{C_1}{C_1+2}\right)$-approximately sparse. We set the dataset of size $N = 1024$, dimension

$d = 2048$. The step sizes are fine-tuned on each case, and in our findings, the empirically optimal step size is inversely related to the gradient variance as the theoretical analysis. In Figures 1 and 2, from the top row to the bottom row, the ℓ_2 regularization parameter λ is set to $1/(10N)$, $1/N$. And in each row, from the first column to the last column, C_2 is set to 4^{-1}, 4^{-2}, 4^{-3}. In these figures, our algorithm is denoted by **GSpar**, and the uniform sampling method is denoted by **UniSp**, and the SGD/SVRG algorithm with non-sparsified communication is denoted by **baseline**, indicating the original distributed optimization algorithm. The x-axis shows the number of data passes, and the y-axis draws the suboptimality of the objective function $(f(w_t) - \min_w f(w))$. For the experiments, we report the sparsified-gradient SGD variance as the notation '*var*' in Figure 1. And '*spa*' in all figures represents the sparsity parameter ρ in Algorithm 3. We observe that the theoretical complexity reduction against the baseline in terms of the communication rounds, which can be inferred by *var*$\times$*spa*, from the labels in Figures 1 to 2, where $C_1 = 0.9$, and the rest of figures are put in appendix due to the limited space.

By comparing the results in Figure 1, we observe that results on sparser data yields smaller gradient variance than results on denser data. Compared to uniform sampling, our algorithm generates gradients with less variance, and it converges much faster. This observation is consistent with the objective of our algorithm, which is to minimize gradient variance given a certain sparsity. The convergence slowed down linearly with respect to the increase of variance. The results on SVRG show better speed up — although our algorithm increases the variance of gradients, the convergence rate degrades only slightly.

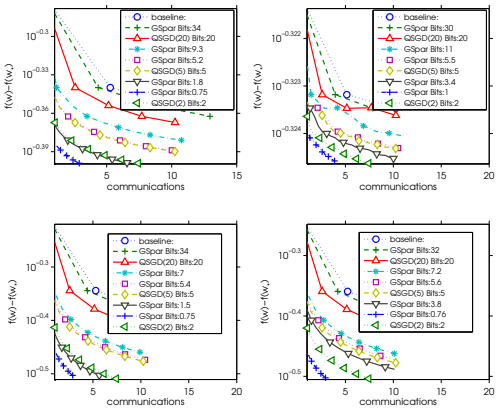

Figure 3: Comparison of the sparisified-SGD with QSGD.

We compared the gradient sparsification method with the quantized stochastic gradient descent (QSGD) algorithm in [2]. The results are shown in Figures 4. The data are generated as previous, with both strong and weak sparsity settings. From the top row to the bottom row, the ℓ_2 regularization parameter λ is set to $1/(10N)$, $1/N$. And in each row, from the first column to the last column, C_2 is set to 4^{-1}, 4^{-2}. The step sizes are set to be the same for both methods for a fair comparison after fine-tuning. In this comparison, we use the overall communication coding length of each algorithm, and note the length in x-axis. For QSGD, the communication cost per element is linearly related to b, which refers to the bits of floating point number. QSGD(b) denotes QSGD algorithm with bit number b in these figures, and the average bits required to represent per element is on the labels. We also tried to compare with the gradient residual accumulation approaches [1, 24], which unfortunately failed on our experiments, where the gradient is sparse so that lots of coordinates could be delayed infinitely, resulting in a large gradient bias to sabotage the convergence on convex problems. From Figures 4, we observe that the proposed sparsification approach is at least comparable to QSGD, and significantly outperforms QSGD when the gradient sparsity is stronger; and this concords with our analysis on the gradient approximate sparsity encouraging faster speed up.

4.1 Experiments on deep learning

This section conducts experiments on non-convex problems. We consider the convolutional neural networks on the CIFAR10dataset. We experiment with neural networks using different settings. Generally, the networks consist of three convolutional layers (3×3), two pooling layers (2×2), and one 256 dimensional fully connected layer. Each convolution layer is followed by a batch-normalization layer. The channels of each convolutional layer is set to $\{24, 32, 48, 64\}$. We use the ADAM optimization algorithm [16], and the initial step size is set to 0.02. In Figure 4.1, we plot the objective loss against the computational complexity measured by the number of epochs (1 epoch is equal to 1 pass of all training samples). We also plot the convergence with respect to the communication cost, which is the product of computations and the sparsification parameter ρ. The experiments on each setting are repeated 4 times and we report the average objective function values. The results show that for this non-convex problem, the gradient sparsification slows down the training

efficiency only slightly. In particular, the optimization algorithm converges even when the sparsity ratio is about $\rho = 0.004$, and the communication cost is significantly reduced in this setting. This experiments also shows that the optimization of neural networks is less sensitive to gradient noises, and the noises within a certain range may even help the algorithm to avoid bad local minimums [14].

4.2 Experiments on asynchronous parallel SGD

In this section, we study parallel implementations of SGD on the multi-core architecture. We employ the support vector machine for binary classification, where the loss function is $f(w) = \frac{1}{N}\sum_n \max(1 - a_n^\top w b_n, 0) + \lambda_2 ||w||_2^2$, $a_n \in \mathbb{R}^d$, $b_n \in \{-1, 1\}$. We implemented shared memory multi-thread SGD, where each thread employs a locked read, which may block other threads' writing to the same coordinate. We use atomic instructions for updating coordinates. To improve the speed of the algorithm, we also employ several engineering tricks. First, we observe that $\forall p_i < 1, \quad g_i/p_i = \text{sign}(g_i)/\lambda$ from Proposition 1, therefore we only need to assign constant values to these variables, without applying float-point division operations. Another costly operation is the pseudo-random number generation in the sampling procedure; therefore we generate a large array of pseudo-random numbers in $[0, 1]$, and iteratively read the numbers during training without calling a random number generating function. The data are generated by first generating dense data, sparsifying them and generating the corresponding labels:

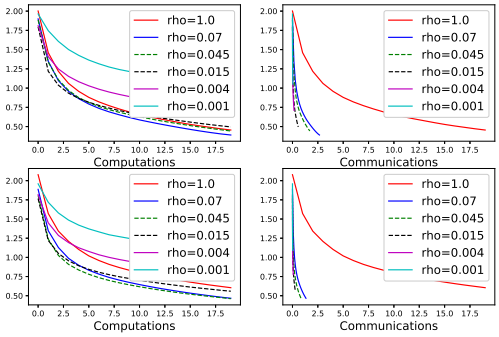

Figure 4: Comparison of convolutional neural networks of 3 layers of channels of 64 (top) and 48 (bottom) on CIFAR10. (Y-axis: loss function $f(w_t)$.)

$$\bar{a}_{ni} \sim \mathcal{N}(0,1), \forall i \in [d], n \in [N], \bar{w} \sim \text{Uniform}[-0.5, 0.5]^d, \bar{B} \sim \text{Uniform}[0,1]^d,$$
$$\bar{B}_i \leftarrow C_1 \bar{B}_i, \text{if:} \bar{B}_i \leq C_2, \forall i \in [d], a_n \leftarrow \bar{a}_n \odot \bar{B}, \quad b_n \leftarrow \text{sign}(x_n^\top \bar{w} + \sigma), \text{where}\, \sigma \sim \mathcal{N}(0,1).$$

The details of data generation are put in the appendix. We set the dataset of size $N = 51200$, dimension $d = 256$, also set $C_1 = 0.01$ and $C_2 = 0.9$. The regularization parameter λ_2 is denoted by reg, the number of threads is denoted by W(workers), and the learning rate is denoted by lrt. The number of workers is set to 16 or 32, the regularization parameter is set to $\{0.5, 0.1, 0.05\}$, and the learning rate is chosen from $\{0.5, 0.25, 0.05, 0.25\}$. The convergence of objective value against running time (milliseconds) is plotted in Figure 4.1, and the rest of figures are put in appendix due to the limited space.

From Figure 4.1, we can observe that using gradient sparsification, the conflicts of multiple threads for reading and writing the same coordinate are significantly reduced. Therefore the training speed is significantly faster. By comparing with other settings, we also observe that the sparsification technique works better at the case when more threads are available, since the more threads, the more frequently the lock conflicts occur.

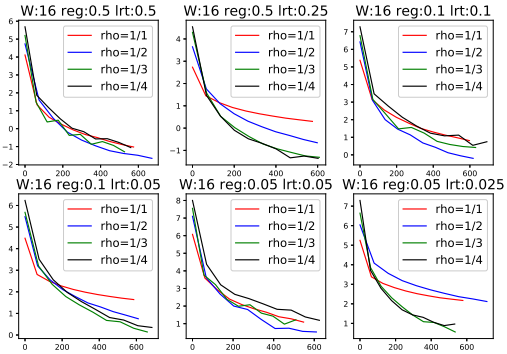

Figure 5: Loss functions by a multi-thread SVM. X-axis: time in milliseconds, Y-axis: $\log_2(f(w_t))$.

5 Conclusions

In this paper, we propose a gradient sparsification technique to reduce the communication cost for large scale distributed machine learning. We propose a convex optimization formulation to minimize the coding length of stochastic gradients given the variance budget that monotonically depends on the computational complexity, with efficient algorithms and a theoretical guarantee. Comprehensive experiments on distributed and parallel optimization of multiple models proved our

algorithm can effectively reduce the communication cost during training or reduce conflicts among multiple threads.

References

[1] Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 440–445, 2017.

[2] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pages 1707–1718, 2017.

[3] Yossi Arjevani and Ohad Shamir. Communication complexity of distributed convex learning and optimization. In *Advances in neural information processing systems*, pages 1756–1764, 2015.

[4] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[5] Peter Bühlmann and Sara Van De Geer. *Statistics for high-dimensional data: methods, theory and applications*. Springer Science & Business Media, 2011.

[6] Jiecao Chen, He Sun, David Woodruff, and Qin Zhang. Communication-optimal distributed clustering. In *Advances in Neural Information Processing Systems*, pages 3727–3735, 2016.

[7] Shang-Tse Chen, Maria-Florina Balcan, and Duen Horng Chau. Communication efficient distributed agnostic boosting. In *Artificial Intelligence and Statistics*, pages 1299–1307, 2016.

[8] Christopher De Sa, Matthew Feldman, Christopher Ré, and Kunle Olukotun. Understanding and optimizing asynchronous low-precision stochastic gradient descent. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 561–574. ACM, 2017.

[9] Christopher De Sa, Ce Zhang, Kunle Olukotun, and Christopher Ré. Taming the wild: A unified analysis of hogwild-style algorithms. In *Advances in neural information processing systems*, pages 2674–2682, 2015.

[10] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[11] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, pages 1646–1654, 2014.

[12] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in neural information processing systems*, pages 1223–1231, 2013.

[13] Martin Jaggi, Virginia Smith, Martin Takác, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 3068–3076, 2014.

[14] Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M Kakade, and Michael I Jordan. How to escape saddle points efficiently. In *International Conference on Machine Learning*, pages 1724–1732, 2017.

[15] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014.

[17] Jakub Konečnỳ and Peter Richtárik. Randomized distributed mean estimation: Accuracy vs communication. *arXiv preprint arXiv:1611.07555*, 2016.

[18] Jason D Lee, Qiang Liu, Yuekai Sun, and Jonathan E Taylor. Communication-efficient sparse regression. *Journal of Machine Learning Research*, 18(5):1–30, 2017.

[19] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 1, page 3, 2014.

[20] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27, 2014.

[21] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670. ACM, 2014.

[22] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[23] Yingyu Liang, Maria-Florina F Balcan, Vandana Kanchanapally, and David Woodruff. Improved distributed principal component analysis. In *Advances in Neural Information Processing Systems*, pages 3113–3121, 2014.

[24] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

[25] Ji Liu, Stephen J Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *The Journal of Machine Learning Research*, 16(1):285–322, 2015.

[26] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

[27] Sashank J Reddi, Jakub Konečnỳ, Peter Richtárik, Barnabás Póczós, and Alex Smola. Aide: Fast and communication efficient distributed optimization. *arXiv preprint arXiv:1608.06879*, 2016.

[28] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming: Series A and B*, 162(1-2):83–112, 2017.

[29] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[30] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(Feb):567–599, 2013.

[31] Ohad Shamir, Nati Srebro, and Tong Zhang. Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, pages 1000–1008, 2014.

[32] Ananda Theertha Suresh, X Yu Felix, Sanjiv Kumar, and H Brendan McMahan. Distributed mean estimation with limited communication. In *International Conference on Machine Learning*, pages 3329–3337, 2017.

[33] John N Tsitsiklis and Zhi-Quan Luo. Communication complexity of convex optimization. *Journal of Complexity*, 3(3):231–243, 1987.

[34] Chong Wang, Xi Chen, Alexander J Smola, and Eric P Xing. Variance reduction for stochastic gradient optimization. In *Advances in Neural Information Processing Systems*, pages 181–189, 2013.

[35] Jialei Wang, Weiran Wang, and Nathan Srebro. Memory and communication efficient distributed stochastic optimization with minibatch prox. *arXiv preprint arXiv:1702.06269*, 2017.

[36] Jialei Wang and Tong Zhang. Improved optimization of finite sums with minibatch stochastic variance reduced proximal iterations. *arXiv preprint arXiv:1706.07001*, 2017.

[37] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. *arXiv preprint arXiv:1705.07878*, 2017.

[38] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data*, 1(2):49–67, 2015.

[39] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[40] Hantian Zhang, Jerry Li, Kaan Kara, Dan Alistarh, Ji Liu, and Ce Zhang. The zipml framework for training models with end-to-end low precision: The cans, the cannots, and a little bit of deep learning. *ICML*, 2017.

[41] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM, 2004.

[42] Yuchen Zhang and Xiao Lin. Disco: Distributed optimization for self-concordant empirical loss. In *International conference on machine learning*, pages 362–370, 2015.

[43] Yuchen Zhang, Martin J Wainwright, and John C Duchi. Communication-efficient algorithms for statistical optimization. In *Advances in Neural Information Processing Systems*, pages 1502–1510, 2012.

[44] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.