VARIANCE REDUCTION WITH SPARSE GRADIENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Variance reduction methods which use a mixture of large and small batch gradients, such as SVRG (Johnson & Zhang, 2013) and SpiderBoost (Wang et al., 2018), require significantly more computational resources per update than SGD (Robbins & Monro, 1951). We reduce the computational cost per update of variance reduction methods by introducing a sparse gradient operator blending the top-K operator (Stich et al., 2018; Aji & Heafield, 2017) and the randomized coordinate descent operator. While the computational cost of computing the derivative of a model parameter is constant, we make the observation that the gains in variance reduction are proportional to the magnitude of the derivative. In this paper, we show that a sparse gradient based on the magnitude of past gradients reduces the computational cost of model updates without a significant loss in variance reduction. Theoretically, our algorithm is at least as good as the best available algorithm (e.g. SpiderBoost) under appropriate settings of parameters and can be much more efficient if our algorithm succeeds in capturing the sparsity of the gradients. Empirically, our algorithm consistently outperforms SpiderBoost using various models to solve various image classification tasks. We also provide empirical evidence to support the intuition behind our algorithm via a simple gradient entropy computation, which serves to quantify gradient sparsity at every iteration.

1 INTRODUCTION

Optimization tools for machine learning applications seek to minimize the finite sum objective

$$\min_{x\in\mathbb{R}^d} f(x) \triangleq \frac{1}{n}\sum_{i=1}^{n} f_i(x), \tag{1}$$

where x is a vector of parameters, and $f_i : \mathbb{R}^d \to \mathbb{R}$ is the loss associated with sample i. Batch SGD serves as the prototype for modern stochastic gradient methods. It updates the iterate x with $x - \eta\nabla f_{\mathcal{I}}(x)$, where η is the learning rate and $f_{\mathcal{I}}(x)$ is the batch stochastic gradient, i.e.

$$\nabla f_{\mathcal{I}}(x) = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} \nabla f_i(x).$$

The batch size $|\mathcal{I}|$ in batch SGD directly impacts the stochastic variance and gradient query complexity of each iteration of the update rule. Lower variance improves convergence rate without any changes to learning rate, but the step-size in the convergence analysis of SGD decreases with variance (Robbins & Monro, 1951), which suggests that learning rates can be increased when stochastic variance is decreased to further improve the convergence rate of gradient-based machine learning optimization algorithms. This is generally observed behavior in practice (Smith et al., 2018; Hoffer et al., 2017).

In recent years, new variance reduction techniques have been proposed by carefully blending large and small batch gradients (e.g. Roux et al., 2012; Johnson & Zhang, 2013; Defazio et al., 2014; Xiao & Zhang, 2014; Allen-Zhu & Yuan, 2016; Allen-Zhu & Hazan, 2016; Reddi et al., 2016a;b; Allen-Zhu, 2017; Lei & Jordan, 2017; Lei et al., 2017; Allen-Zhu, 2018b; Fang et al., 2018; Zhou et al., 2018; Wang et al., 2018; Pham et al., 2019; Nguyen et al., 2019; Lei & Jordan, 2019). They are alternatives to batch SGD and are provably better than SGD in various settings. While these methods allow for greater learning rates than batch SGD and have appealing theoretical guarantees, they require a per-iteration query complexity which is more than double than that of batch SGD. This

leads to the critique by Defazio (2019), which questions the utility of variance reduction techniques in modern machine learning problems. While their position is reasonable, they only show that naive implementations of variance reduction methods do not work and do not rule out the possibility that variance reduction can be effective when combined with new ideas. In this paper, we take a step in this direction by significantly improving the computational complexity of variance reduction methods.

The central observation of this work is the following: While the cost of computing the derivative of a model's parameter is constant, the variance of the model parameter's derivative is not. We experimentally show that, toward the beginning of training, the magnitude of the derivative of model parameters are roughly equivalent and unstructured, and toward the end of training, they become progressively more structured. We measure this behavior by computing the entropy of the empirical distribution over the magnitude of the derivative of the model parameters. We exploit this observation to reduce the query complexity of variance reduction methods by applying a sparse gradient operator typically used to reduce the communication complexity of distributed optimization (Stich et al., 2018; Aji & Heafield, 2017) applications. Stich et al. (2018) addressed the communication complexity of distributed SGD by transmitting sparse gradients between computer nodes over a low bandwidth communication channel. Aji & Heafield (2017) expands on this work by using a memory vector to record dense gradient updates. This is done by adding a dense gradient to the memory vector at each iteration and transmitting to other nodes on the network the gradient coordinates which rank among the top k in terms of their magnitude. The transmitted coordinates are set to zero, which allows coordinates with smaller derivatives to accumulate enough magnitude to eventually be transmitted.

Like the memory vector in Aji & Heafield (2017), we also use a vector to rank the variance of the additional gradient computations introduced by variance reduction methods, and compute both the derivative of model parameters which rank among the top k_1 of the stored memory vector, and additionally compute the gradient of k_2 randomly selected model parameters from the remaining $d - k_1$ parameters.

The rest of the paper is organized as follows. We begin by providing a sparse variance reduction algorithm based on a combination of SCSG (Lei et al., 2017) and SpiderBoost (Wang et al., 2018). We then explain how to perform sparse back-propagation in order to realize the benefits of sparsity. We prove both that our algorithm is as good as SpiderBoost, and under reasonable assumptions, has better complexity than SpiderBoost. Finally, we present our experimental results which include an empirical analysis of the sparsity of various image classification problems, and a comparison between our algorithm and SpiderBoost.

2 Stochastic Variance Reduction with Sparse Gradients

Generally, variance reduction methods reduce the variance of stochastic gradients by taking a snapshot $\nabla f(y)$ of the gradient $\nabla f(x)$ every m steps of optimization, and use the gradient information in this snapshot to reduce the variance of subsequent smaller batch gradients $\nabla f_{\mathcal{I}}(x)$ (Johnson & Zhang, 2013; Wang et al., 2018). Methods such as SCSG (Lei & Jordan, 2017) utilize a large batch gradient, which is typically some multiple in size of the small batch gradient b, which is much more practical and is what we do in this paper. To reduce the cost of computing additional gradients, we use sparsity by only computing a subset k of the total gradients d, where $y \in \mathbb{R}^d$.

2.1 SpiderBoost with Sparse Gradients

For $d, k, k_1, k_2 \in \mathbb{Z}_+$, let $k = k_1 + k_2$, where $1 \leq k \leq d$ for a parametric model of dimension d. In what follows, we define an operator which takes vectors x, y and outputs y', where y' retains only k of the entries in y, k_1 of which are selected according to the coordinates in x which have the k_1 largest absolute values, and the remaining k_2 entries are randomly selected from y. The k_1 coordinate indices and k_2 coordinate indices are disjoint. Formally, the operator $\mathrm{rtop}_{k_1,k_2} : \mathbb{R}^d \rightarrow$

$\mathbb{R}^d$ is defined for $x, y \in \mathbb{R}^d$ as

$$\left(\mathrm{rtop}_{k_1,k_2}(x,y)\right)_\ell = \begin{cases} y_\ell & \text{if } k_1 > 0 \text{ and } |x|_\ell \geq |x|_{(k_1)} \\ \frac{(d-k_1)}{k_2} y_\ell & \text{if } \ell \in S \\ 0 & \text{otherwise,} \end{cases}$$

where $|x|$ denotes a vector of absolute values, $|x|_{(1)} \geq |x|_{(2)} \geq \ldots \geq |x|_{(d)}$ denotes the order statistics of coordinates of x in absolute values, and S denotes a random subset with size k_2 that is uniformly drawn from the set $\{\ell : |x|_\ell < |x|_{(k_1)}\}$. For instance, if $x = (11, 12, 13, -14, -15), y = (-25, -24, 13, 12, 11)$ and $k_1 = k_2 = 1$, then S is a singleton uniformly drawn from $\{1, 2, 3, 4\}$. Suppose $S = \{2\}$, then $\mathrm{rtop}_{1,1}(x, y) = (0, 4y_2, 0, 0, y_5) = (0, -96, 0, 0, 11)$. If $k_1 + k_2 = d$, $\mathrm{rtop}_{k_1,k_2}(x, y) = y$. On the other hand, if $k_1 = 0$, $\mathrm{rtop}_{0,k_2}(x, y)$ does not depend on x and returns a rescaled random subset of y. This is the operator used in coordinate descent methods. Finally, $\mathrm{rtop}_{k_1,k_2}(x, y)$ is linear in y. The following Lemma shows that $\mathrm{rtop}_{k_1,k_2}(x, y)$ is an unbiased estimator of y, which is a crucial property in our later analysis.

Lemma 1. *Given any $x, y \in \mathbb{R}^d$,*

$$\mathbb{E}\left(\mathrm{rtop}_{k_1,k_2}(x,y)\right) = y, \quad \mathrm{Var}\left(\mathrm{rtop}_{k_1,k_2}(x,y)\right) = \frac{d-k_1-k_2}{k_2} \| \mathrm{top}_{-k_1}(x,y)\|^2,$$

where $\mathbb{E}$ is taken over the random subset S involved in the rtop_{k_1,k_2} operator and

$$(\mathrm{top}_{-k_1}(x,y))_\ell = \begin{cases} y_\ell & \textit{if } k_1 > 0 \textit{ and } |x|_\ell < |x|_{(k_1)} \\ 0 & \textit{otherwise.} \end{cases}$$

Our algorithm is detailed as below.

Algorithm 1: SpiderBoost with Sparse Gradients.

Input: Learning rate η, inner loop size m, outer loop size T, large batch size B, small batch size b, initial iterate x_0, memory decay factor α, sparsity parameters k_1, k_2.

$\mathcal{I}_0 \sim \mathrm{Unif}(\{1, \ldots, n\})$ with $|\mathcal{I}_0| = B$
$M_0 := |\nabla f_{\mathcal{I}_0}(x_0)|$
for $j = 1, ..., T$ **do**
  $x_0^{(j)} := x_{j-1},\ M_0^{(j)} := M_{j-1}$
  $\mathcal{I}_j \sim \mathrm{Unif}(\{1, \ldots, n\})$ with $|\mathcal{I}_j| = B$
  $\nu_0^{(j)} := \nabla f_{\mathcal{I}_j}(x_0^{(j)})$
  $N_j := m$ (for implementation) or $N_j \sim \mathrm{Geom}(m)$ (for theory)
  for $t = 0, \ldots, N_j - 1$ **do**
    $x_{t+1}^{(j)} := x_t^{(j)} - \eta \nu_t^{(j)}$
    $\mathcal{I}_t^{(j)} \sim \mathrm{Unif}([n])$ with $|\mathcal{I}_t^{(j)}| = b$
    $\nu_{t+1}^{(j)} := \nu_t^{(j)} + \mathrm{rtop}_{k_1,k_2}\left(M_t^{(j)}, \nabla f_{\mathcal{I}_t^{(j)}}(x_{t+1}^{(j)}) - \nabla f_{\mathcal{I}_t^{(j)}}(x_t^{(j)})\right)$
    $M_{t+1}^{(j)} := \alpha|\nu_{t+1}^{(j)}| + (1-\alpha) M_t^{(j)}$
  $x_j := x_{N_j}^{(j)},\ M_j := M_{N_j}^{(j)}$

Output: $x_{\mathrm{out}} = x_T$ (for implementation) or $x_{\mathrm{out}} = x_{T'}$ where $T' \sim \mathrm{Unif}([T])$ (for theory)

The algorithm includes an outer-loop and an inner-loop. In the theoretical analysis, we generate N_j as Geometric random variables. This trick is called "geometrization", proposed by Lei & Jordan (2017) and dubbed by Lei & Jordan (2019). It greatly simplifies analysis (e.g. Lei et al., 2017; Allen-Zhu, 2018a). In practice, as observed by Lei et al. (2017), it does not make a difference if N_j is simply set to be m. For this reason, we apply "geometrization" in theory to make arguments clean and readable. On the other hand, in theory the output is taken as uniformly random elements from the set of last iterates in each outer loop. This is a generic strategy for nonconvex optimization, as an analogue of the average iterates for convex optimization, proposed by Nemirovski et al. (2009). In practice, we simply use the last iterate as convention.

Similar to Aji & Heafield (2017), we maintain a memory vector at each iteration of our algorithm. We assume the optimization procedure is taking place locally and thus do not transmit and zero out

any components. Instead, we maintain an exponential moving average $M_t^{(j)}$ of the magnitudes of each coordinate of our gradient estimate $\nu_t^{(j)}$. We then use $M_t^{(j)}$ as an approximation to the variance of each gradient coordinate in our rtop_{k_1,k_2} operator. With $M_t^{(j)}$ as input, the rtop_{k_1,k_2} operator targets k_1 high variance gradient coordinates in addition to the k_2 randomly selected coordinates.

The cost of invoking rtop_{k_1,k_2} is dominated by the algorithm for selecting the top k coordinates, which has linear worst case complexity when using the introselect algorithm.

2.2 Sparse Back-Propagation

One crucial step in using sparsity with any optimization algorithm is computing the gradient in a way that takes advantage of sparsity. Consider the forward pass of a deep neural network, where ϕ is a deep composition of parametric functions,

$$\phi(x;\theta) = \phi_L(\phi_{L-1}(\ldots \phi_0(x;\theta_0)\ldots;\theta_{L-1});\theta_L). \tag{2}$$

The unconstrained problem of minimizing over the θ_ℓ can be rewritten as a constrained optimization problem as follows:

$$\begin{aligned} \min_{\theta} \quad & \frac{1}{n}\sum_{i=1}^{n} \mathrm{loss}(z_i^{(L+1)}, y_i) \\ \text{s.t.} \quad & z_i^{(L+1)} = \phi_L(z_i^{(L)};\theta_L) \\ & \vdots \\ & z_i^{(\ell+1)} = \phi_\ell(z_i^{(\ell)};\theta_\ell) \\ & \vdots \\ & z_i^{(1)} = \phi_0(x_i;\theta_0). \end{aligned} \tag{3}$$

In this form, z_i^{L+1} is the model estimate for data point i. Consider $\phi_\ell(x;\theta_\ell) = \sigma(x^T\theta_\ell)$ for $1 \leq \ell < L$, ϕ_L be the output layer, and σ be some subdifferentiable activation function. If we apply the rtop_{k_1,k_2} operator per-layer in the forward-pass, with appropriate scaling of k_1 and k_2 to account for depth, we see that the number of multiplications in the forward pass is reduced to $k_1 + k_2$: $\sigma(\mathrm{rtop}_{k_1,k_2}(v,x)^T\,\mathrm{rtop}_{k_1,k_2}(v,\theta_\ell))$. A sparse forward-pass yields a computation graph for a $(k_1 + k_2)$-parameter model, and back-propagation will compute the gradient of the objective with respect to model parameters in linear time (Chauvin & Rumelhart, 1995).

2.3 Gradient Query Complexity

We assume that sampling an index i and accessing the pair $\nabla f_i(x)$ incur a unit of cost and accessing the truncated version $\mathrm{rtop}_{k_1,k_2}(y, \nabla f_i(x))$ incur $(k_1 + k_2)/d$ units of cost. Note that calculating $\mathrm{rtop}_{k_1,k_2}(y, \nabla f_{\mathcal{I}}(x))$ incurs $|\mathcal{I}|(k_1 + k_2)/d$ units of computational cost. Given our framework, the theoretical complexity of the algorithm is

$$C_{\mathrm{comp}}(\epsilon) \triangleq \sum_{j=1}^{T}\left(B + 2bN_j\frac{k_1+k_2}{d}\right). \tag{4}$$

3 Theoretical Complexity Analysis

3.1 Notation and assumptions

Denote by $\|\cdot\|$ the Euclidean norm and by $a \wedge b$ the minimum of a and b. For a random vector $Y \in \mathbb{R}^d$,

$$\mathrm{Var}(Y) = \sum_{i=1}^{d}\mathrm{Var}(Y_i).$$

We say a random variable N has a geometric distribution, $N \sim \mathrm{Geom}(m)$, if N is supported on the non-negative integers with

$$\mathbb{P}(N = k) = \gamma^k (1 - \gamma), \quad \forall k = 0, 1, \ldots,$$

for some γ such that $\mathbb{E}N = m$. Here we allow N to be zero to facilitate the analysis.

Assumption **A**1 on the smoothness of individual functions will be made throughout the paper.

A1 f_i is differentiable with

$$\|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x - y\|,$$

for some $L < \infty$ and for all $i \in \{1, \ldots, n\}$.

As a direct consequence of assumption **A**1, it holds for any $x, y \in \mathbb{R}^d$ that

$$-\frac{L}{2}\|x - y\|^2 \leq f_i(x) - f_i(y) - \langle \nabla f_i(y), x - y \rangle \leq \frac{L}{2}\|x - y\|^2. \tag{5}$$

To formulate our complexity bounds, we define

$$f^* = \inf_x f(x), \quad \Delta_f = f(x_0) - f^*.$$

Further we define σ^2 as an upper bound on the variance of the stochastic gradients:

$$\sigma^2 = \sup_x \frac{1}{n} \sum_{i=1}^n \|\nabla f_i(x) - \nabla f(x)\|^2. \tag{6}$$

3.2 Worst-case guarantee

Theorem 1. *Under the following setting of parameters*

$$\eta L = \sqrt{\frac{k_2}{6dm}}, \quad B = \left\lceil \frac{2\sigma^2}{\epsilon^2} \wedge n \right\rceil$$

For any $T \geq T(\epsilon) \triangleq 4\Delta_f / \eta m \epsilon^2$,

$$\mathbb{E}\|\nabla f(x_{\mathrm{out}})\| \leq \epsilon.$$

If we further set

$$m = \frac{Bd}{b(k_1 + k_2)},$$

the complexity to achieve the above condition is

$$\mathbb{E}C_{\mathrm{comp}}(\epsilon) = O\left(\left(\frac{\sigma}{\epsilon^3} \wedge \frac{\sqrt{n}}{\epsilon^2} \right) L\Delta_f \sqrt{\frac{b(k_1 + k_2)}{k_2}} \right).$$

Recall that the complexity of SpiderBoost (Wang et al., 2018) is

$$O\left(\left(\frac{\sigma}{\epsilon^3} \wedge \frac{\sqrt{n}}{\epsilon^2} \right) L\Delta_f \right).$$

Thus as long as $b = O(1), k_1 = O(k_2)$, our algorithm has the same complexity as SpiderBoost under appropriate settings. The penalty term $O(\sqrt{b(k_1 + k_2)/k_2})$ is due to the information loss by sparsification.

3.3 Data adaptive analysis

Let

$$g_t^{(j)} = \|\operatorname{top}_{-k_1}(M_t^{(j)}, \nabla f(x_{t+1}^{(j)}) - \nabla f(x_t^{(j)}))\|^2,$$

and

$$G_t^{(j)} = \frac{1}{n} \sum_{i=1}^n \|\operatorname{top}_{-k_1}(M_t^{(j)}, \nabla f_i(x_{t+1}^{(j)}) - \nabla f_i(x_t^{(j)}))\|^2.$$

By Cauchy-Schwarz inequality and the linearity of top_{-k_1}, it is easy to see that $g_t^{(j)} \leq G_t^{(j)}$. If our algorithm succeeds in capturing the sparsity, both $g_t^{(j)}$ and $G_t^{(j)}$ will be small. In this subsection we will analyze the complexity under this case. Further define R_j as

$$R_j = \mathbb{E}_j g_{N_j}^{(j)} + \frac{\mathbb{E}_j G_{N_j}^{(j)}}{b}, \tag{7}$$

where $\mathbb{E}_j$ is taken over all randomness in j-th outer loop (line 4-13 of Algorithm 1).

Theorem 2. *Under the following setting of parameters*

$$\eta L = \sqrt{\frac{b \wedge m}{3m}}, \quad B = \left\lceil \frac{3\sigma^2}{\epsilon^2} \wedge n \right\rceil$$

For any $T \geq T(\epsilon) \triangleq 6\Delta_f / \eta m \epsilon^2$,

$$\mathbb{E}\|\nabla f(x_{\mathrm{out}})\|^2 \leq \frac{2\epsilon^2}{3} + \frac{(d - k_1 - k_2)m}{k_2} \mathbb{E}\bar{R}_T,$$

where

$$\bar{R}_T = \frac{1}{T} \sum_{j=1}^{T} R_j.$$

If $\mathbb{E}\bar{R}_T \leq \epsilon^2 \frac{k_2}{3(d-k_1-k_2)m}$, then

$$\mathbb{E}\|\nabla f(x_{\mathrm{out}})\| \leq \epsilon.$$

If we further set

$$m = \frac{Bd}{b(k_1 + k_2)},$$

the complexity to achieve the above condition is

$$\mathbb{E}C_{\mathrm{comp}}(\epsilon) = O\left(\left(\frac{\sigma}{\epsilon^3} \wedge \frac{\sqrt{n}}{\epsilon^2} \right) L\Delta_f \sqrt{\frac{k_1 + k_2}{d} \frac{b}{b \wedge m}} \right).$$

In practice, m is usually much larger than b. As a result, the complexity of our algorithm is $O(\sqrt{(k_1 + k_2)/d})$ smaller than that of SpiderBoost if our algorithm captures the sparsity as desired. Although this type of data adaptive analysis is not as clean as the worst-case guarantee (Theorem 1), it can reveal the potentially superior performance of the algorithm. Similar analyses have been done for various other algorithms, including AdaGrad (Duchi et al., 2011) and Adam (Kingma & Ba, 2014).

4 EXPERIMENTS

We run two key sets of experiments to demonstrate the performance of Sparse SpiderBoost, as well as to illustrate the potential of sparsity as a way to improve the gradient query complexity of variance reduction methods. By computing the entropy of the empirical distribution over the magnitude of the derivative of the model parameters, we address one of the key assumptions of this approach: That while the computational cost of each model parameter's derivative is the same, the gains in variance reduction, as well as progress made toward optima, may be different. We also provide experiments on image classification tasks to illustrate the performance of SpiderBoost with and without sparsity.

In all experiments, unless otherwise specified, we run SpiderBoost and Sparse SpiderBoost with a learning rate $\eta = 0.1$, large-batch size $B = 1000$, small-batch size $b = 100$, inner loop length of $m = 10$, memory decay factor of $\alpha = 0.5$, and k_1 and k_2 both set to 5% of the total number of model parameters. We call the sum $k_1 + k_2 = 10\%$ the sparsity of the optimization algorithm. For models, we use a 2-layer fully connected neural network with hidden layers of width 100, a simple convolutional neural net which we describe in detail in appendix B, and Resnet-18 (He et al., 2015). All models use ReLu activations. For datasets, we use CIFAR-10 (Krizhevsky et al.), SVHN (Netzer et al., 2011), and MNIST (LeCun & Cortes, 2010). None of our experiments include Resnet-18 on MNIST as MNIST is an easier dataset; it is included primarily to provide variety for the other models we include in this work.

4.1 GRADIENT ENTROPY AS A MEASURE OF GRADIENT STRUCTURE

Our method relies partially on the assumption that the magnitude of the derivative of some model parameters are greater than others. To measure this, we compute the entropy of the empirical distribution over the magnitude of the derivative of the model parameters. This metric provides a way for us to intuit the effectiveness of our sparsity operator throughout training. In Algorithm 1, the following term updates our estimate of the variance of each coordinate's derivative:

$$M_{t+1} := \alpha|G_t| + (1-\alpha)M_t.$$

Consider the entropy of the following probability vector $p = \frac{M_t}{\|M_t\|_1}$. The entropy of p provides us with a measure of how much structure there is in our gradients. To see this, consider the hypothetical scenario where $p_i = \frac{1}{d}$. In this scenario we have no structure; the top k_1 component of our sparsity operator is providing no value and entropy is maximized. On the other hand, if a single entry $p_i = 1$ and all other entries $p_j = 0$, then the top k_1 component of our sparsity operator is effectively identifying the only relevant model parameter.

To measure the potential of our sparsity operator, we compute the entropy of p while running SpiderBoost on a variety of datasets and model architectures. The results of running this experiment are summarized in the following table.

Table 1: Entropy of Memory Vectors

	FC NN	Conv NN	Resnet-18
CIFAR-10	16.41	13.38	22.59
SVHN	15.36	13.00	22.62
MNIST	14.29	14.21	-

(a) Entropy at $t = 0$.

FC NN	Conv NN	Resnet-18
8.09	2.66	21.70
8.05	2.97	21.31
9.77	2.77	-

(b) Entropy after 150 epochs.

The entries of tables 1a and 1b correspond to the entropy of the memory vector before and after training. We believe a better metric is the symmetric KL divergence, but the ratio we provide is easier to compute and allows us to illustrate our point without ambiguity.

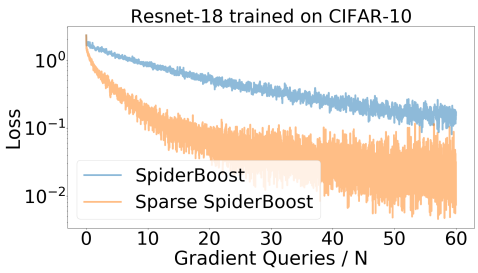
Our operator, which exploits gradient structure, reduces overall variance by targeting model parameters that have high variance derivatives. The table suggests that such gradient structure exists. For each model, the entropy at the beginning of training is almost maximal. For example, maximum entropy of the convolutional model, which consists of $62,006$ parameters, is 15.92. This is mainly due to random initialization of model parameters. After 150 epochs, the entropy of M_t for the convolutional model drops to approximately 3, which suggests a substantial amount of gradient structure.

Note that for the datasets that we tested, the gradient structure depends primarily on the model and not the dataset. In particular, for Resnet-18, the entropy appears to vary minimally after 150 epochs. We hypothesize that this may be due to our use of a constant learning rate (vs. performing exponential decay every fixed number of epochs).

There is structure in the gradients of some of the neural networks we present, and our algorithm will take advantage of that structure when it's available.

4.2 PERFORMANCE ON IMAGE CLASSIFICATION TASKS

We choose to test our algorithm on image classification tasks as these tasks are high dimensional and challenging from an optimization point of view. Figure 1 compares SpiderBoost alone to SpiderBoost with 10% sparsity (10% of parameter derivatives). All experiments in this section are run for 50 epochs. Hyperparameter tuning of sparsity values 0.1, 0.2, 0.4, and 0.8 for Sparse SpiderBoost showed that a value of 0.1 performs best. In our comparison to SpiderBoost, we measure the number of gradient queries over the size of the dataset N. A single gradient query is taken to be the cost of computing a gradient for a single data point. If i is the index of a single sample, then $\nabla f_i(x)$ is a single gradient query. Using the batch gradient to update model parameters for a dataset of size N has a gradient query cost of N. For a model with d parameters, using a single sample to update $\frac{k}{d}$ model parameters has a gradient query cost of $\frac{k}{d}$, etc.



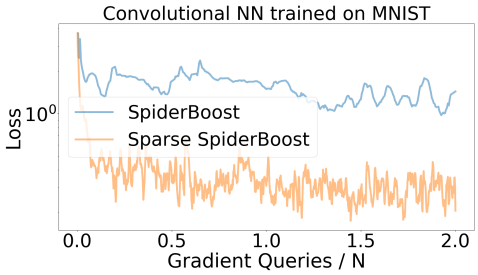


Figure 1: SpiderBoost with 10% sparsity (10% of parameter derivatives) compared to SpiderBoost without sparsity. Figure (a) compares the two algorithms using Resnet-18 and Cifar-10. Figure (b) compares the two algorithms using a convolutional neural network trained on MNIST. The x-axis measures gradient queries over N, where N is the size of the respective datasets. Plots are in log-scale.

Our results of fitting the convolutional neural network to MNIST show that sparsity provides a significant advantage compared to using SpiderBoost alone. We only show 2 epochs of this experiment since the MNIST dataset is fairly simple and convergence is rapidly achieved. The results of training Resnet-18 on CIFAR-10 suggests that our sparsity algorithm works well on large neural networks, and non-trivial datasets. Results for the rest of our experiments can be found in appendix B.

5 Discussion

In this paper, we show how sparse gradients with memory can be used to improve the gradient query complexity of SVRG-type variance reduction algorithms. While we provide a concrete sparse variance reduction algorithm for SpiderBoost, the techniques developed in this paper can be adapted to other variance reduction algorithms.

Our rigorous theoretical analysis proves multiple properties of our algorithm. We show that our algorithm provides a way to explicitly control the gradient query complexity of variance reduction methods, a problem which has thus far not been explicitly addressed. Assuming our algorithm captures the sparsity structure of the optimization problem, we also prove that the complexity of our algorithm is an improvement over SpiderBoost. The results of our direct comparison to SpiderBoost validates this assumption, and our entropy experiment empirically supports the hypothesis that gradient sparsity does exist.

The results of our entropy experiment also support the results in Aji & Heafield (2017), which show that the top k operator generally outperforms the random k operator. Not every problem we tested exhibited sparsity structure. While this is true, our analysis proves that our algorithm performs no worse than SpiderBoost in these settings. Even when there is no structure, our algorithm reduces to a random sampling of $k_1 + k_2$ coordinates.

Another potential weakness of our method is the technical difficulty of implementing a sparse backpropagation algorithm in modern machine learning libraries, such as Tensorflow (Abadi et al., 2015) and Pytorch (Paszke et al., 2017). Models implemented in these libraries generally assume dense structured parameters. The optimal implementation of our algorithm makes use of a sparse forward pass and assumes a sparse computation graph upon which backpropagation is executed. Libraries that support dynamic computation graphs, such as Pytorch, will construct the sparse computation graph in the forward pass. This makes the required sparse backpropagation trivial and suggests that our algorithm will perform best on libraries that support dynamic computation graphs.

While our algorithm makes progress toward improving the practical viability of variance reduction algorithms, we believe further improvements can be made, such as better utilization of reduced variance during training, and better control over increased variance in very high dimensional models such as dense net (Defazio, 2019). We recognize these issues and hope to make progress on them in future work. We'd also like to note the connection between our work and some of the challenges faced in distributed optimization. Our work can be used to decrease overall variance and communication complexity in the distributed setting.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. *CoRR*, abs/1704.05021, 2017. URL `http://arxiv.org/abs/1704.05021`.

Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1200–1205. ACM, 2017.

Zeyuan Allen-Zhu. Katyusha x: Practical momentum method for stochastic sum-of-nonconvex optimization. *arXiv preprint arXiv:1802.03866*, 2018a.

Zeyuan Allen-Zhu. Natasha 2: Faster non-convex optimization than sgd. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 2675–2686. Curran Associates, Inc., 2018b. URL `http://papers.nips.cc/paper/7533-natasha-2-faster-non-convex-optimization-than-sgd.pdf`.

Zeyuan Allen-Zhu and Elad Hazan. Variance reduction for faster non-convex optimization. *ArXiv e-prints abs/1603.05643*, 2016.

Zeyuan Allen-Zhu and Yang Yuan. Improved SVRG for non-strongly-convex or sum-of-non-convex objectives. In *International conference on machine learning*, pp. 1080–1089, 2016.

Yves Chauvin and David E. Rumelhart (eds.). *Backpropagation: Theory, Architectures, and Applications*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1995. ISBN 0-8058-1259-8.

Aaron Defazio. On the ineffectiveness of variance reduced optimization for deep learning, 2019. URL `https://openreview.net/forum?id=B1MIBs05F7`.

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*, pp. 1646–1654, 2014.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Cong Fang, Chris Junchi Li, Zhouchen Lin, and Tong Zhang. Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator. In *Advances in Neural Information Processing Systems*, pp. 689–699, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 1729–1739, USA, 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4. URL `http://dl.acm.org/citation.cfm?id=3294771.3294936`.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, pp. 315–323, USA, 2013. Curran Associates Inc. URL `http://dl.acm.org/citation.cfm?id=2999611.2999647`.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL http://www.cs.toronto.edu/~kriz/cifar.html.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Lihua Lei and Michael Jordan. Less than a Single Pass: Stochastically Controlled Stochastic Gradient. In Aarti Singh and Jerry Zhu (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 148–156, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR.

Lihua Lei and Michael I Jordan. On the adaptivity of stochastic gradient-based optimization. *arXiv preprint arXiv:1904.04480*, 2019.

Lihua Lei, Cheng Ju, Jianbo Chen, and Michael I Jordan. Non-convex finite-sum optimization via scsg methods. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 2348–2358. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6829-non-convex-finite-sum-optimization-via-scsg-methods.pdf.

Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011. URL http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf.

Lam M Nguyen, Marten van Dijk, Dzung T Phan, Phuong Ha Nguyen, Tsui-Wei Weng, and Jayant R Kalagnanam. Optimal finite-sum smooth non-convex optimization with sarah. *arXiv preprint arXiv:1901.07648*, 2019.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Nhan H Pham, Lam M Nguyen, Dzung T Phan, and Quoc Tran-Dinh. Proxsarah: An efficient algorithmic framework for stochastic composite nonconvex optimization. *arXiv preprint arXiv:1902.05679*, 2019.

Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alex Smola. Stochastic variance reduction for nonconvex optimization. *arXiv preprint arXiv:1603.06160*, 2016a.

Sashank J Reddi, Suvrit Sra, Barnabás Póczos, and Alex Smola. Fast incremental method for nonconvex optimization. *arXiv preprint arXiv:1603.06159*, 2016b.

H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

Nicolas Le Roux, Mark Schmidt, and Francis Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems*, pp. 2663–2671, 2012.

Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1Yy1BxCZ.

Sebastian U. Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified SGD with memory. *CoRR*, abs/1809.07599, 2018. URL http://arxiv.org/abs/1809.07599.

Zhe Wang, Kaiyi Ji, Yi Zhou, Yingbin Liang, and Vahid Tarokh. Spiderboost: A class of faster variance-reduced algorithms for nonconvex optimization. *CoRR*, abs/1810.10690, 2018. URL http://arxiv.org/abs/1810.10690.

Lin Xiao and Tong Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

Dongruo Zhou, Pan Xu, and Quanquan Gu. Stochastic nested variance reduction for nonconvex optimization. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 3925–3936. Curran Associates Inc., 2018.

A TECHNICAL PROOFS

A.1 PREPARATORY RESULTS

Lemma 2 (Lemma 3.1 of Lei & Jordan (2019))**.** *Let $N \sim \mathrm{Geom}(m)$. Then for any sequence $D_0, D_1, \ldots$ with $\mathbb{E}|D_N| < \infty$,*

$$\mathbb{E}(D_N - D_{N+1}) = \frac{1}{m}\left(D_0 - \mathbb{E}D_N\right).$$

Remark 1. *The requirement $\mathbb{E}|D_N| < \infty$ is essential. A useful sufficient condition if $|D_t| = O(\mathrm{Poly}(t))$ because a geometric random variable has finite moments of any order.*

Lemma 3 (Lemma B.2 of Lei & Jordan (2019))**.** *Let $z_1, \ldots, z_M \in \mathbb{R}^d$ be an arbitrary population and $\mathcal{J}$ be a uniform random subset of $[M]$ with size m. Then*

$$\mathrm{Var}\left(\frac{1}{m}\sum_{j\in\mathcal{J}} z_j\right) \le \frac{I(m<M)}{m}\cdot\frac{1}{M}\sum_{j=1}^{M}\|z_j\|_2^2.$$

Proof of Lemma 1. WLOG, assume that $|x_1| \ge |x_2| \ge \ldots \ge |x_d|$. Let S be a random subset of $\{k_1+1, \ldots, d\}$ with size k_2. Then

$$\left(\mathrm{rtop}_{k_1,k_2}(x,y)\right)_\ell = y_\ell\left(I(\ell \le k_1) + \frac{d-k_1}{k_2}I(\ell \in S)\right).$$

As a result,

$$\mathbb{E}\left[\left(\mathrm{rtop}_{k_1,k_2}(x,y)\right)_\ell\right] = y_\ell\left(I(\ell \le k_1) + \frac{d-k_1}{k_2}I(\ell > k_1)P(\ell \in S)\right) = y_\ell,$$

and

$$\begin{aligned}\mathrm{Var}\left[\left(\mathrm{rtop}_{k_1,k_2}(x,y)\right)_\ell\right] &= \left(\frac{d-k_1}{k_2}\right)^2 y_\ell^2 I(\ell > k_1)P(\ell \in S)(1-P(\ell \in S)) \\ &= \frac{d-k_1-k_2}{k_2}y_\ell^2 I(\ell > k_1).\end{aligned}$$

Therefore,

$$\mathrm{Var}\left(\mathrm{rtop}_{k_1,k_2}(x,y)\right) = \frac{d-k_1-k_2}{k_2}\sum_{\ell>k_1} y_\ell^2 = \frac{d-k_1-k_2}{k_2}\|\mathrm{top}_{-k_1}(x,y)\|^2.$$

□

A.2 ANALYSIS OF A SINGLE INNER LOOP

Lemma 4. *For any j, t,*

$$\mathbb{E}_{j,t}(\nu_{t+1}^{(j)} - \nu_t^{(j)}) = \nabla f(x_{t+1}^{(j)}) - \nabla f(x_t^{(j)})$$

and

$$\mathrm{Var}_{j,t}(\nu_{t+1}^{(j)} - \nu_t^{(j)}) \leq \frac{\eta^2 L^2}{b}\|\nu_t^{(j)}\|^2 + \frac{d-k_1-k_2}{k_2}\left(g_t^{(j)} + \frac{G_t^{(j)}}{b}\right),$$

where $\mathbb{E}_{j,t}$ *and* $\mathrm{Var}_{j,t}$ *are taken over the randomness of* $\mathcal{I}_t^{(j)}$ *and the random subset* S *involved in the* rtop_{k_1,k_2} *operator.*

Proof. By definition,

$$\nu_{t+1}^{(j)} - \nu_t^{(j)} = \mathrm{rtop}_{k_1,k_2}\left(M_t^{(j)}, \nabla f_{\mathcal{I}_t^{(j)}}(x_{t+1}^{(j)}) - \nabla f_{\mathcal{I}_t^{(j)}}(x_t^{(j)})\right).$$

Let S be the random subset involved in rtop_{k_1,k_2}. Then S is independent of $(\mathcal{I}_t^{(j)}, M_t^{(j)}, x_{t+1}^{(j)}, x_t^{(j)})$. By Lemma 1,

$$\mathbb{E}_S\left(\nu_{t+1}^{(j)} - \nu_t^{(j)}\right) = \nabla f_{\mathcal{I}_t^{(j)}}(x_{t+1}^{(j)}) - \nabla f_{\mathcal{I}_t^{(j)}}(x_t^{(j)})$$

and

$$\mathrm{Var}_S\left(\nu_{t+1}^{(j)} - \nu_t^{(j)}\right) = \frac{d-k_1-k_2}{k_2}\left\|\mathrm{top}_{-k_1}\left(M_t^{(j)}, \nabla f_{\mathcal{I}_t^{(j)}}(x_{t+1}^{(j)}) - \nabla f_{\mathcal{I}_t^{(j)}}(x_t^{(j)})\right)\right\|^2.$$

Since $\mathcal{I}_t^{(j)}$ is independent of $(M_t^{(j)}, x_{t+1}^{(j)}, x_t^{(j)})$, the tower property of conditional expectation and variance implies that

$$\mathbb{E}_{j,t}\left(\nu_{t+1}^{(j)} - \nu_t^{(j)}\right) = \mathbb{E}_{\mathcal{I}_t^{(j)}}\left(\nabla f_{\mathcal{I}_t^{(j)}}(x_{t+1}^{(j)}) - \nabla f_{\mathcal{I}_t^{(j)}}(x_t^{(j)})\right) = \nabla f(x_{t+1}^{(j)}) - \nabla f(x_t^{(j)}),$$

and

$$\mathrm{Var}_{j,t}\left(\nu_{t+1}^{(j)} - \nu_t^{(j)}\right) = \mathbb{E}_{\mathcal{I}_t^{(j)}}\left(\mathrm{Var}_S\left(\nu_{t+1}^{(j)} - \nu_t^{(j)}\right)\right) + \mathrm{Var}_{\mathcal{I}_t^{(j)}}\left(\mathbb{E}_S\left(\nu_{t+1}^{(j)} - \nu_t^{(j)}\right)\right). \quad (8)$$

To bound the first term, we note that top_{-k_1} is linear in y and thus

$$\begin{aligned}
&\mathbb{E}_{\mathcal{I}_t^{(j)}}\left\|\mathrm{top}_{-k_1}\left(M_t^{(j)}, \nabla f_{\mathcal{I}_t^{(j)}}(x_{t+1}^{(j)}) - \nabla f_{\mathcal{I}_t^{(j)}}(x_t^{(j)})\right)\right\|^2 \\
&= \left\|\mathbb{E}_{\mathcal{I}_t^{(j)}}\,\mathrm{top}_{-k_1}\left(M_t^{(j)}, \nabla f_{\mathcal{I}_t^{(j)}}(x_{t+1}^{(j)}) - \nabla f_{\mathcal{I}_t^{(j)}}(x_t^{(j)})\right)\right\|^2 \\
&\quad + \mathrm{Var}_{\mathcal{I}_t^{(j)}}\left[\mathrm{top}_{-k_1}\left(M_t^{(j)}, \nabla f_{\mathcal{I}_t^{(j)}}(x_{t+1}^{(j)}) - \nabla f_{\mathcal{I}_t^{(j)}}(x_t^{(j)})\right)\right] \\
&= g_t^{(j)} + \mathrm{Var}_{\mathcal{I}_t^{(j)}}\left[\frac{1}{b}\sum_{i\in\mathcal{I}_t^{(j)}}\mathrm{top}_{-k_1}(M_t^{(j)}, \nabla f_i(x_{t+1}^{(j)}) - \nabla f_i(x_t^{(j)}))\right] \\
&\leq g_t^{(j)} + \frac{G_t^{(j)}}{b},
\end{aligned} \quad (9)$$

where the last inequality uses Lemma 3. To bound the second term of equation 8, by Lemma 3,

$$\begin{aligned}
&\mathrm{Var}_{\mathcal{I}_t^{(j)}}\left(\mathbb{E}_S\left(\nu_{t+1}^{(j)} - \nu_t^{(j)}\right)\right) = \mathrm{Var}_{\mathcal{I}_t^{(j)}}\left(\nabla f_{\mathcal{I}_t^{(j)}}(x_{t+1}^{(j)}) - \nabla f_{\mathcal{I}_t^{(j)}}(x_t^{(j)})\right) \\
&\leq \frac{1}{b}\frac{1}{n}\sum_{i=1}^n \|\nabla f_i(x_{t+1}^{(j)}) - \nabla f_i(x_t^{(j)})\|^2 \overset{(i)}{\leq} \frac{L^2}{b}\|x_{t+1}^{(j)} - x_t^{(j)}\|^2 \overset{(ii)}{=} \frac{\eta^2 L^2}{b}\|\nu_t^{(j)}\|^2,
\end{aligned}$$

where (i) uses assumption **A**1 and (ii) uses the definition that $x_{t+1}^{(j)} = x_t^{(j)} - \eta\nu_t^{(j)}$. □

Lemma 5. *For any* j, t*,*

$$\mathbb{E}_{j,t}\|\nu_{t+1}^{(j)} - \nabla f(x_{t+1}^{(j)})\|^2 \leq \|\nu_t^{(j)} - \nabla f(x_t^{(j)})\|^2 + \frac{\eta^2 L^2}{b}\|\nu_t^{(j)}\|^2 + \frac{d-k_1-k_2}{k_2}\left(g_t^{(j)} + \frac{G_t^{(j)}}{b}\right),$$

where $\mathbb{E}_{j,t}$ *and* $\mathrm{Var}_{j,t}$ *are taken over the randomness of* $\mathcal{I}_t^{(j)}$ *and the random subset* S *involved in the* rtop_{k_1,k_2} *operator.*

Proof. By Lemma 4, we have

$$\nu_{t+1}^{(j)} - \nabla f(x_{t+1}^{(j)}) = \nu_t^{(j)} - \nabla f(x_t^{(j)}) + \left(\nu_{t+1}^{(j)} - \nu_t^{(j)} - \mathbb{E}_{j,t}(\nu_{t+1}^{(j)} - \nu_t^{(j)})\right).$$

Since $\mathcal{I}_t^{(j)}$ is independent of $(\nu_t^{(j)}, x_t^{(j)})$,

$$\mathrm{Cov}_{j,t}\left(\nu_t^{(j)} - \nabla f(x_t^{(j)}), \nu_{t+1}^{(j)} - \nu_t^{(j)}\right) = 0.$$

As a result,

$$\mathbb{E}_{j,t}\|\nu_{t+1}^{(j)} - \nabla f(x_{t+1}^{(j)})\|^2 = \|\nu_t^{(j)} - \nabla f(x_t^{(j)})\|^2 + \mathrm{Var}_{j,t}(\nu_{t+1}^{(j)} - \nu_t^{(j)}).$$

The proof is then completed by Lemma 4. □

Lemma 6. *For any j,*

$$\mathbb{E}_j\|\nu_{N_j}^{(j)} - \nabla f(x_{N_j}^{(j)})\|^2 \le \frac{m\eta^2 L^2}{b}\mathbb{E}_j\|\nu_{N_j}^{(j)}\|^2 + \frac{\sigma^2}{B} + \frac{(d-k_1-k_2)m}{k_2}R_j,$$

where $\mathbb{E}_j$ is taken over all randomness in j-th outer loop (line 4-13 of Algorithm 1). 4.

Proof. By definition,

$$\begin{aligned}
\|\nu_{t+1}^{(j)}\| &\le \|\nu_t^{(j)}\| + \left\|\mathrm{rtop}_{k_1,k_2}\left(M_t^{(j)}, \nabla f_{\mathcal{I}_t^{(j)}}(x_{t+1}^{(j)}) - \nabla f_{\mathcal{I}_t^{(j)}}(x_t^{(j)})\right)\right\| \\
&\le \|\nu_t^{(j)}\| + \left\|\nabla f_{\mathcal{I}_t^{(j)}}(x_{t+1}^{(j)}) - \nabla f_{\mathcal{I}_t^{(j)}}(x_t^{(j)})\right\| \\
&\le \|\nu_t^{(j)}\| + \sum_{i\in\mathcal{I}_t^{(j)}}\left\|\nabla f_i(x_{t+1}^{(j)}) - \nabla f_i(x_t^{(j)})\right\| \\
&\le \|\nu_t^{(j)}\| + b\sqrt{n}\sigma.
\end{aligned}$$

As a result,

$$\|\nu_t^{(j)}\| \le \|\nu_0^{(j)}\| + tb\sqrt{n}\sigma, \tag{10}$$

Thus,

$$\|\nu_t^{(j)} - \nabla f(x_t^{(j)})\|^2 \le 2\|\nu_t^{(j)}\|^2 + 2\|\nabla f(x_t^{(j)})\|^2 = \mathrm{Poly}(t).$$

This implies that we can apply Lemma 2 on the sequence $D_t = \|\nu_t^{(j)} - \nabla f(x_t^{(j)})\|^2$.

Letting $j = N_j$ in Lemma 5 and taking expectation over all randomness in $\mathbb{E}_j$, we have

$$\begin{aligned}
&\mathbb{E}_j\|\nu_{N_j+1}^{(j)} - \nabla f(x_{N_j+1}^{(j)})\|^2 \\
&\le \mathbb{E}_j\|\nu_{N_j}^{(j)} - \nabla f(x_{N_j}^{(j)})\|^2 + \frac{\eta^2 L^2}{b}\mathbb{E}_j\|\nu_{N_j}^{(j)}\|^2 + \frac{d-k_1-k_2}{k_2}\mathbb{E}_j\left(g_{N_j}^{(j)} + \frac{G_{N_j}^{(j)}}{b}\right) \\
&= \mathbb{E}_j\|\nu_{N_j}^{(j)} - \nabla f(x_{N_j}^{(j)})\|^2 + \frac{\eta^2 L^2}{b}\mathbb{E}_j\|\nu_{N_j}^{(j)}\|^2 + \frac{d-k_1-k_2}{k_2}R_j.
\end{aligned} \tag{11}$$

By Lemma 2,

$$\begin{aligned}
&\mathbb{E}_j\|\nu_{N_j}^{(j)} - \nabla f(x_{N_j}^{(j)})\|^2 - \mathbb{E}_j\|\nu_{N_j+1}^{(j)} - \nabla f(x_{N_j+1}^{(j)})\|^2 \\
&= \frac{1}{m}\left(\|\nu_0^{(j)} - \nabla f(x_0^{(j)})\|^2 - \mathbb{E}_j\|\nu_{N_j}^{(j)} - \nabla f(x_{N_j}^{(j)})\|^2\right) \\
&= \frac{1}{m}\left(\mathbb{E}_j\|\nu_0^{(j)} - \nabla f(x_{j-1})\|^2 - \mathbb{E}_j\|\nu_{N_j}^{(j)} - \nabla f(x_j)\|^2\right),,
\end{aligned} \tag{12}$$

where the last line uses the definition that $x_{j-1} = x_0^{(j)}, x_j = x_{N_j}^{(j)}$. By Lemma 3,

$$\mathbb{E}_j\|\nu_0^{(j)} - \nabla f(x_{j-1})\|^2 \le \frac{\sigma^2 I(B<n)}{B}. \tag{13}$$

The proof is completed by putting equation 11, equation 12 and equation 13 together. □

Lemma 7. *For any j, t,*

$$f(x_{t+1}^{(j)}) \leq f(x_t^{(j)}) + \frac{\eta}{2}\|\nu_t^{(j)} - \nabla f(x_t^{(j)})\|^2 - \frac{\eta}{2}\|\nabla f(x_t^{(j)})\|^2 - \frac{\eta}{2}(1-\eta L)\|\nu_t^{(j)}\|^2.$$

Proof. By equation 5,

$$\begin{aligned} f(x_{t+1}^{(j)}) &\leq f(x_t^{(j)}) + \left\langle \nabla f(x_t^{(j)}), x_{t+1}^{(j)} - x_t^{(j)} \right\rangle + \frac{L}{2}\|x_t^{(j)} - x_{t+1}^{(j)}\|^2 \\ &= f(x_t^{(j)}) - \eta \left\langle \nabla f(x_t^{(j)}), \nu_t^{(j)} \right\rangle + \frac{\eta^2 L}{2}\|\nu_t^{(j)}\|^2 \\ &= f(x_t^{(j)}) + \frac{\eta}{2}\|\nu_t^{(j)} - \nabla f(x_t^{(j)})\|^2 - \frac{\eta}{2}\|\nabla f(x_t^{(j)})\|^2 - \frac{\eta}{2}\|\nu_t^{(j)}\|^2 + \frac{\eta^2 L}{2}\|\nu_t^{(j)}\|^2. \end{aligned}$$

The proof is then completed. □

Lemma 8. *For any j,*

$$\mathbb{E}_j\|\nabla f(x_j)\|^2 \leq \frac{2}{\eta m}\mathbb{E}_j(f(x_{j-1}) - f(x_j)) + \mathbb{E}_j\|\nu_{N_j}^{(j)} - \nabla f(x_j)\|^2 - (1-\eta L)\mathbb{E}_j\|\nu_{N_j}^{(j)}\|^2,$$

where $\mathbb{E}_j$ is taken over all randomness in j-th outer loop (line 4-13 of Algorithm 1).

Proof. Since $\|\nabla f(x)\| \leq \sigma$ for any x,

$$|f(x_{t+1}^{(j)}) - f(x_t^{(j)})| \leq \sigma\|\nu_t^{(j)}\|.$$

This implies that

$$|f(x_t^{(j)})| \leq \sigma \sum_{k=0}^{t} \|\nu_t^{(j)}\| + |f(x_0^{(j)})|.$$

As shown in equation 10, $\|\nu_t^{(j)}\| = \mathrm{Poly}(t)$ and thus $|f(x_t^{(j)})| = \mathrm{Poly(t)}$. This implies that we can apply Lemma 2 on the sequence $D_t = f(x_t^{(j)})$.

Letting $j = N_j$ in Lemma 7 and taking expectation over all randomness in $\mathbb{E}_j$, we have

$$\mathbb{E}_j f(x_{N_j+1}^{(j)}) \leq \mathbb{E}_j f(x_{N_j}^{(j)}) + \frac{\eta}{2}\|\nu_{N_j}^{(j)} - \nabla f(x_{N_j}^{(j)})\|^2 - \frac{\eta}{2}\|\nabla f(x_{N_j}^{(j)})\|^2 - \frac{\eta}{2}(1-\eta L)\|\nu_{N_j}^{(j)}\|^2.$$

By Lemma 2,

$$\mathbb{E}_j f(x_{N_j}^{(j)}) - \mathbb{E}_j f(x_{N_j+1}^{(j)}) = \frac{1}{m}\mathbb{E}_j(f(x_0^{(j)}) - f(x_{N_j}^{(j)})) = \frac{1}{m}\mathbb{E}_j(f(x_{j-1}) - f(x_j)).$$

The proof is then completed. □

Combining Lemma 6 and Lemma 8, we arrive at the following key result on one inner loop.

Theorem 3. *For any j,*

$$\begin{aligned} \mathbb{E}\|\nabla f(x_j)\|^2 \leq \frac{2}{\eta m}\mathbb{E}_j(f(x_{j-1}) - f(x_j)) + \frac{\sigma^2 I(B<n)}{B} + \frac{(d-k_1-k_2)m}{k_2}R_j \\ - \left(1 - \eta L - \frac{m\eta^2 L^2}{b}\right)\mathbb{E}_j\|\nu_{N_j}^{(j)}\|^2. \end{aligned}$$

A.3 Complexity analysis

Proof of Theorem 1. By definition equation 7 of R_j and the smoothness assumption **A**1,

$$\mathbb{E}R_j \leq \frac{b+1}{b}L^2\mathbb{E}\|x_{N_j+1}^{(j)} - x_{N_j}^{(j)}\|^2 \leq 2\eta^2 L^2 \mathbb{E}\|\nu_{N_j}^{(j)}\|^2.$$

By Theorem 3,

$$\mathbb{E}\|\nabla f(x_j)\|^2 \leq \frac{2}{\eta m}\mathbb{E}_j(f(x_{j-1}) - f(x_j)) + \frac{\sigma^2 I(B<n)}{B} - \left(1 - \eta L - \frac{m\eta^2 L^2}{b} - \frac{2(d-k_1-k_2)m\eta^2 L^2}{k_2}\right)\mathbb{E}_j\|\nu_{N_j}^{(j)}\|^2.$$

Since $\eta = \sqrt{k_2/6dm}$,

$$\eta L + \frac{m\eta^2 L^2}{b} + \frac{2(d-k_1-k_2)m\eta^2 L^2}{k_2} \leq \frac{1}{\sqrt{6}} + \frac{1}{6} + \frac{1}{3} \leq 1.$$

As a result,

$$\mathbb{E}\|\nabla f(x_j)\|^2 \leq \frac{2}{\eta m}\mathbb{E}_j(f(x_{j-1}) - f(x_j)) + \frac{\sigma^2 I(B<n)}{B}.$$

Since $x_{\text{out}} = x_{T'}$ where $T' \sim \text{Unif}([T])$, we have

$$\mathbb{E}\|x_{\text{out}}\|^2 \leq \frac{2}{\eta m T}\mathbb{E}(f(x_0) - f(x_{T+1})) + \frac{\sigma^2 I(B<n)}{B} \leq \frac{2\Delta_f}{\eta m T} + \frac{\sigma^2 I(B<n)}{B}.$$

The setting of T and B guarantees that

$$\frac{2\Delta_f}{\eta m T} \leq \frac{\epsilon^2}{2}, \quad \frac{\sigma^2 I(B<n)}{B} \leq \frac{\epsilon^2}{2}.$$

Therefore,

$$\mathbb{E}\|\nabla f(x_{\text{out}})\|^2 \leq \epsilon^2.$$

By Cauchy-Schwarz inequality,

$$\mathbb{E}\|\nabla f(x_{\text{out}})\| \leq \sqrt{\mathbb{E}\|\nabla f(x_{\text{out}})\|^2} \leq \epsilon.$$

In this case, the average computation cost is

$$\mathbb{E}C_{\text{comp}}(\epsilon) = T(\epsilon)\left(B + \frac{2(k_1+k_2)}{d}bm\right) = 3BT(\epsilon) = O\left(\frac{BL\Delta_f}{\eta m \epsilon^2}\right) = O\left(\frac{\sqrt{Bb}L\Delta_f}{\epsilon^2}\sqrt{\frac{k_1+k_2}{k_2}}\right).$$

The proof is then proved by the setting of B. □

Proof of Theorem 2. Under the setting of η,

$$\eta L + \frac{m\eta^2 L^2}{b} \leq \frac{1}{\sqrt{3}} + \frac{1}{3} \leq 1.$$

By Theorem 3,

$$\mathbb{E}\|\nabla f(x_j)\|^2 \leq \frac{2}{\eta m}\mathbb{E}_j(f(x_{j-1}) - f(x_j)) + \frac{\sigma^2 I(B<n)}{B} + \frac{d-k_1-k_2}{k_2}R_j.$$

By definition of x_{out},

$$\mathbb{E}\|\nabla f(x_{\text{out}})\|^2 \leq \frac{2\Delta_f}{\eta m T} + \frac{\sigma^2 I(B<n)}{B} + \frac{(d-k_1-k_2)m}{k_2}\mathbb{E}\bar{R}_T.$$

Under the settings of T and B,

$$\frac{2\Delta_f}{\eta m T} \leq \frac{\epsilon^2}{3}, \quad \frac{\sigma^2 I(B<n)}{B} \leq \frac{\epsilon^2}{3}.$$

This proves the first result. The second result follows directly. For the computation cost, similar to the proof of Theorem 1, we have

$$\mathbb{E}C_{\text{comp}}(\epsilon) = O(BT) = O\left(\frac{L\Delta_f}{\epsilon^2}\frac{B}{\sqrt{m(b\wedge m)}}\right).$$

The proof is then completed by trivial algebra. □

B EXPERIMENTS

B.1 DESCRIPTION OF SIMPLE CONVOLUTIONAL NEURAL NETWORK

The simple convolutional neural network used in the experiments consists of a convolutional layer with a kernel size of 5, followed by a max pool layer with kernel size 2, followed by another convolutional layer with kernel size 5, followed by a fully connected layer of input size $16 * side^2 \times 120$ (side is the size of the second dimension of the input), followed by a fully connected layer of size 120×84, followed by a final fully connected layer of size $84\times$ the output dimension.

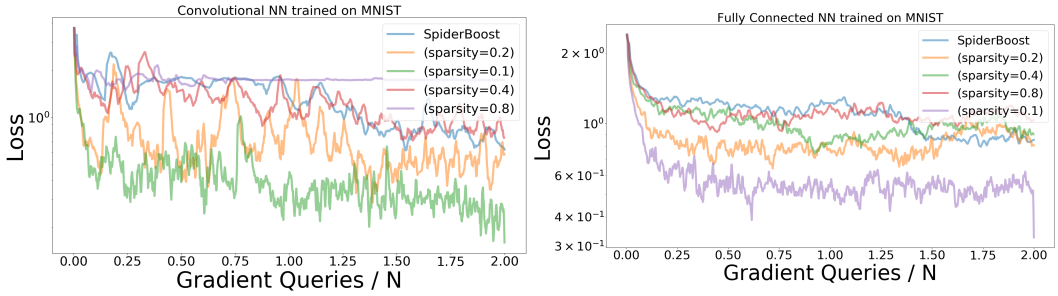


Figure 2: SpiderBoost with various values of sparsity, where (sparsity=k/d) corresponds to SpiderBoost with sparsity k/d. Both figures use MNIST. The x-axis measures gradient queries over N, where N is the size of the respective datasets. Plots are in log-scale.