Variance Reduction in Deep Learning with Batches and Fine-Grained Snapshots

Mohamad Fakhouri, Davide Mazzali, Antonino Emanuele Scurria
CS-439 Optimization for Machine Learning, EPFL, Switzerland

Abstract—**The idea of variance-reduced SGD methods is to make stochastic steps concentrate more around their expectation, i.e. reducing their variance, so as to be better proxies to the full gradient. This can be achieved by adding a (zero-mean) variance-reducing term to the classical SGD step. Many such methods have been studied both theoretically and empirically, especially for convex objectives. Examples include SVRG and SAGA, which both rely on the idea of taking snapshots of some iterates and then using these for computing the variance-reducing term in later steps. In the context of deep learning, there is empirical evidence that SVRG-like approaches do not yield better performances. In this report, we further investigate the behaviour of SVRG on deep learning tasks and initiate the study of SAGA-like methods on non-convex objectives. In particular, we assess the influence of the batch size on the performance of SVRG, and propose a variant of SAGA that allows to take more fine-grained snapshots than SVRG without suffering from infeasible memory requirements like original SAGA. The motivating idea behind these experiments is that both the size of the batches and the relevance and correlation of snapshots intuitively could affect the reduction in variance and hence the overall performance of these methods. We put this intuition to the test.**

I. Introduction

The core idea of SGD is to take steps in the direction of an estimate of the gradient, where this estimate should be efficiently computable. However, high variance in the estimation of the total gradient can hinder the overall performance. Variance reduction techniques address the issue of high variance by reducing the noise in gradient estimates.

In this project we study the effect of variance reduction methods in a deep learning setting. We focus on optimization problems of the form $\min_{w\in\mathbb{R}^d} f(w)$ with the objective being $f(w) = \frac{1}{n}\sum_{i=1}^n f_i(w)$, where each $f_i : \mathbb{R}^d \to \mathbb{R}$ is a (not necessarily convex) loss function associated to the i-th datapoint-label pair in the training dataset.

II. Algorithms

We consider two algorithms for variance reduction, SVRG and SAGA. Both can be seen as generalizations of SGD. We recall that SGD is parameterized by a learning rate $\gamma \in [0,1]$ and a batch size $b \geq 1$, and it samples $B \sim \text{unif}(\binom{[n]}{b})$ at every iteration and follows the update rule

$$w^{(t)} \leftarrow w^{(t-1)} - \gamma \cdot u^{(t)} \quad \text{with} \quad u^{(t)} = s^{(t)} - v^{(t)}\,, \tag{1}$$

where $s^{(t)} = \nabla f_B(w^{(t)})$ and $v^{(t)} = 0$ (it will be clear later why we use this notation). Throughout, we use the notation $\nabla f_B(w) = \frac{1}{|B|}\sum_{i\in B}\nabla f_i(w)$ for all $w \in \mathbb{R}^d$ and $B \subseteq [n]$. We recall that the SGD step is an unbiased estimator of the true gradient, i.e. $\mathbb{E}_B[u^{(t)}] = \nabla f(w^{(t-1)})$. Variance reduced methods are variants of SGD that correct the classical SGD step with a zero-mean term that is meant to reduce the variance with respect to the true gradient.

A. SVRG

Stochastic Variance Reduced Gradient (SVRG) [1] is an optimization algorithm that corrects stochastic gradients with periodically updated full gradients. At every iteration we take a snapshot $\tilde{w}$ of the current iterate $w^{(t)}$ with some probability p. Then, the line between the stochastic gradient at the snapshot point and the full gradient at the snapshot point is used as the variance reduction term. In symbols, this method is parameterized by a learning rate $\gamma \in [0,1]$, a batch size $b \geq 1$, and a snapshot probability $p \in [0,1]$, and it follows the update rule (1) with

$$s^{(t)} = \nabla f_B(w^{(t-1)}),\ v^{(t)} = \nabla f_B(\tilde{w}) - \nabla f(\tilde{w})\,.$$

A formal description is deferred to Appendix A. We remark that in expectation the step taken by SVRG is in the direction of the full gradient, as for SGD, since $\mathbb{E}_B[v^{(t)}] = 0$.

B. SAGA

The Stochastic Average Gradient (SAGA) [2], [3] algorithm is similar in spirit to SVRG, but it takes more fine-grained snapshots: for each $i = 1, \ldots, n$, SAGA keeps track of the last iterate $\tilde{w}^{(i)}$ when i was sampled, and uses these gradients to reduce variance. We defer the pseudocode to Appendix B.

One prominent issue with this method is that it requires to explicitly store n d-dimensional points. This is especially problematic in deep learning, where the number of parameters d can be much larger than the number of features of the training dataset. In fact, there is a lack of studies of SAGA in deep learning.

We then propose a variant of SAGA that substantially reduces the memory usage and hence allows us to test a SAGA-like method in the non-convex setting. We call this variant Clustered SAGA (C-SAGA): the idea is to partition the dataset indices $[n]$ into k groups $C_1, \ldots, C_k$, and each group will have its own snapshot $\tilde{w}^{(\ell)}$ that will be used whenever we sample i that is in the ℓ-th cluster C_ℓ. We also allow for a snapshot probability $p \in (0,1)$, as in SVRG. More precisely, C-SAGA samples $i \sim \text{unif}([n])$ at every step, and follows the update rule (1) with

$$s^{(t)} = \nabla f_i(w^{(t-1)}),\ v^{(t)} = \nabla f_i(\tilde{w}^{(\ell)}) - \frac{1}{k}\sum_{h=1}^{k}\nabla f_{C_h}(\tilde{w}^{(h)})\,,$$

where ℓ is the index such that $i \in C_\ell$. A formal description is deferred to Appendix B.

One can see that setting $k = n$ and $p = 1$ recovers SAGA, while setting $k = 1$ recovers SVRG. Hence, C-SAGA can be seen as an attempt to interpolate between these two extremes, while allowing the user to tune the memory requirement, which is now $O(kd)$ as opposed to $O(nd)$ for classical SAGA.

We also remark that our variant still uses a variance reduction term that has mean zero, i.e. $\mathbb{E}_i[\nabla f_i(\tilde{w}^{(\ell)})] = \frac{1}{k}\sum_{h=1}^{k} \nabla f_{C_h}(\tilde{w}^{(h)})$, so $\mathbb{E}_i[v^{(t)}] = 0$.

Another property we want to highlight is that each datapoint i sees its snapshot $\tilde{w}^{(\ell)}$ updated once in k/p steps on average, where ℓ is such that $i \in C_\ell$. More precisely, if we denote by τ_i the number of steps between updates of i's cluster snapshot $\tilde{w}^{(\ell)}$, one has $\mathbb{E}[\tau_i] = \sum_{t\geq 1} t \cdot (1 - p|C_\ell|/n)^{t-1} p|C_\ell|/n = \frac{k}{p}$, where the expectation is taken over the randomness in the algorithm.

III. Experiments

All our experiments were run on the MNIST dataset [4], which consists of $n = 50'000$ training datapoints[1], with a LeNet5 architecture [5] and cross entropy loss. These three components define the objective function f that we want to minimize. We set the learning rate to $\gamma = 0.01$ for all algorithms[2], and do $T = 3'000$ iterations. We repeat every experiment five times, and average the results obtained.

We now describe that quantities that we track and analyse for our experiments.

- Training losses are computed once in 20 optimization steps.
- Whenever we refer to the variance of an algorithm, we mean the variance of the step that it takes, i.e. the variance of $u^{(t)}$. We estimate this quantity as the average distance squared of the step $u^{(t)}$ from the (estimated) average gradient, inspired by the approach of [6][3]. Even though this is just an estimate, it does reflect the expected behavior of the true variance (we briefly discuss why this is the case in Appendix D, Figure 6).

A. *Effect of batch size*

In this section we address the question of how the effect of the variance reduction term of SVRG is influenced by the batch size. We used batch sizes $b = 1, 16, 64, 128$, and used a fixed snapshot probability $p = 1/10$.

Looking at Figure 1 , we see that SGD and SVRG behave[4] similarly across batches. SGD and SVRG with $b = 1$ converge much slower than using larger batches for both algorithms,

[1]Each datapoint is a 28×28 black and white picture of digits from 0 to 9. We reshape the inputs from 28×28 pixels to 32×32 and shift and rescale the entries with their mean and standard deviation.

[2]Since the objective function is the same across all experiments, we used the same learning rate for all algorithms. We then searched for the step size by trying different values at a logarithmic scale.

[3]See https://github.com/facebookresearch/deep-variance-reduction/blob/main/torch_svrg.py

[4]We plot the training loss only, and not the test loss, as we focus on optimizing the function and not on the generalization error

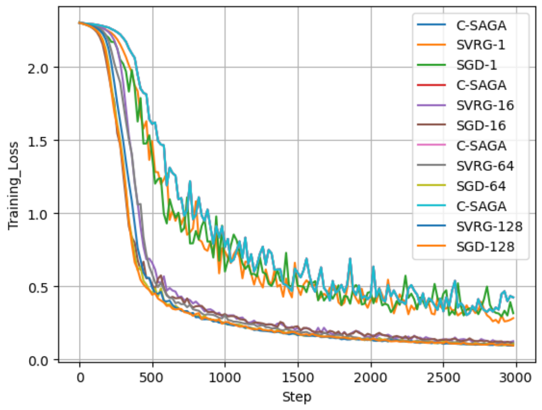

Fig. 1. Losses of SAGA, SGD, SVRG for batch sizes $b = 1, 16, 64, 128$. The top three lines correspond to SAGA, SGD ($b = 1$) and SVRG ($b = 1$).

and the noise of the losses for the $b = 1$ runs is quite similar. We see that using smaller batches leads to noisier steps (ie, steps with more variance), which is shown in in the Appendix (Figure 6 where we plot the variance of the SGD runs per step). We now proceed to investigate how SGD's variance compares to the other algorithms.

In Figure 3, we plot the ratio of the variance of SVRG to the variance of SGD, compared per batch size. At the start (for the first 500 steps), SVRG has much less variance than SGD, since the ratio is largely below 1. An interesting finding is that the average variance ratio for batch size $b = 1, 16, 64, 128$ is respectively $0.934, 0.936, 0.951, 0.956$. Hence, the larger the batch size, the less effective variance reduction is.

As shown in Figure 1, the variance reduction does not have strong impact on the convergence rate, as we see the train losses of SVRG follow a similar trend of SGD per same batch size. On the other hand in Figure 2, which plots the ratio of SVRG's and SGD's training loss at every time step, shows that in the early phase of training SVRG is outperformed by SGD for all batch sizes. With batch size 1, we do see some improvements in the later steps but the behaviour is very oscillating. Batch sizes $16, 64$ are instead consistently worse for SVRG than SGD.

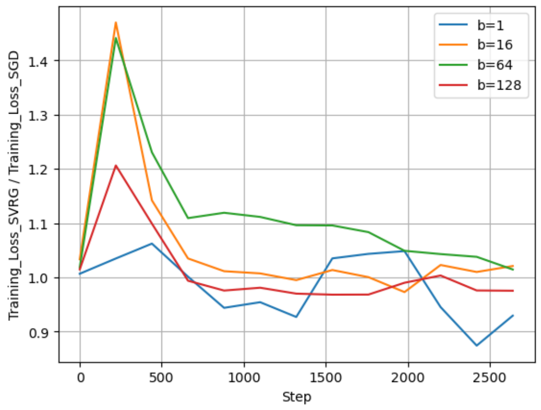

Fig. 2. Ratio of the training loss of SVRG to the training loss of SGD with varying batch size.

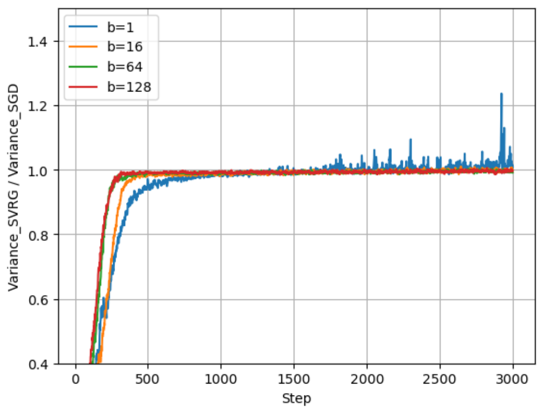

Fig. 3. Ratio of the Variance of SVRG to the Variance of SGD with batch size $b = 1, 16, 64, 128$. Values smaller than 1 indicate that SVRG had less variance at that step than SGD.

B. *Effect of more fine-grained snapshots*

The idea of the SAGA algorithm is to have "personalized" memory for each training datapoint, so as to remember the position in parameters' space of the last time the algorithm picked the i-th datapoint for the stochastic gradient. Instead, SVRG uses the same snapshot for every point. In this section we investigate whether more fine-grained snapshots, in a SAGA-like fashion, benefit the optimization of our non-convex objective. We use C-SAGA with $k = 10$ and $p = 1$, and we use SVRG with $p = 1/10$ and $b = 1$. Hence, as highlighted in Section II-B, each datapoint expects to see its snapshot updated once in 10 steps for both algorithms, hence making the comparison meaningful. We use SGD with $b = 1$ as the baseline benchmark.

In terms of training loss, Figure 1 shows that C-SAGA performs similarly to other algorithms with batch size $b = 1$ overall. However, a more refined comparison in Figure 4 shows that C-SAGA training loss is consistently worse than SGD, while SVRG is slightly better. In fact, the average ratio of C-SAGA's training loss over SGD's is 1.110, while the average ratio of SVRG's over SGD's is 0.978.

In terms of variance, we see from Figure 5 that the variance of both SVRG and C-SAGA is not higher than that of SGD, except for some fluctuations (as plotted lines are almost always below 1). The average ratio of C-SAGA's variance over SGD's is 0.978 while that of SVRG is 0.968. Hence, SVRG seems more effective than C-SAGA in reducing the variance of SGD overall. However, we see from the plot that, in the early phase of training, C-SAGA reduces the variance more than SVRG.

Overall, we conclude that having finer-grained snapshots does not improve the amount of variance reduction nor the training loss as compared to SVRG, that are in fact slightly worse on average.

IV. Conclusion

In [6], it was showed that the variance reduction of SVRG in deep learning is marginal (if not absent at all), and it was conjectured the same for SAGA. In this project, we noticed that the effect of the variance reduction on our non-convex

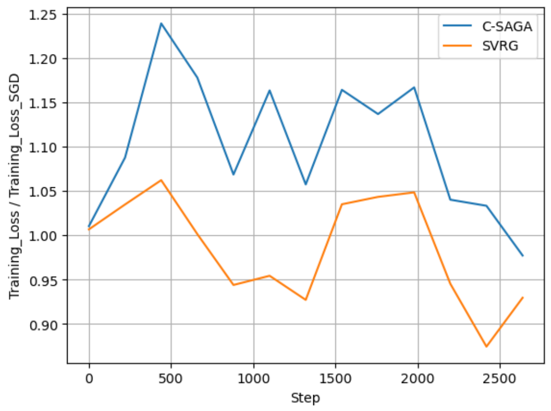

Fig. 4. Ratio of the training loss of SAGA and SVRG ($b = 1$) to the training loss of SGD ($b = 1$).

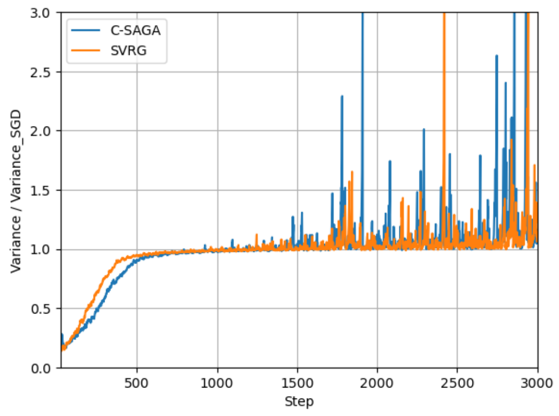

Fig. 5. Ratio of the Variance of C-SAGA and SVRG ($b = 1$) to the Variance of SGD ($b = 1$). Values smaller than 1 indicate that C-SAGA/SVRG had less variance at that step than SGD.

objective is minimal, hence confirming their finding about SVRG and their conjecture about SAGA. Therefore, these methods do not pay off, as they are more expensive than SGD, both FLOPs- and memory-wise.

Interestingly, Figures 3 and 5 show that the variance reduction of both SVRG (with all batch sizes) and C-SAGA is more prominent in the first ≈ 500 iterations, thus matching the finding of [6]. We observe that this roughly corresponds to the point where the training loss starts to plateau (see Figure 1). This may be an indication of the fact that the variance reduction is more effective on steeper regions of the optimization landscape. Further investigation of this aspect may shed more light on the effect of variance reduction methods.

Our findings suggest that running a variance reduction algorithm in the initial learning phases, and then switching to SGD, may give the best of both worlds. Moreover, one could potentially get better performances by not updating the weights using a GD-approach, but rather using the computed gradient vectors $s^{(t)} - v^{(t)}$ with other gradient-based optimizers such as Adam (and AdamW variants) [7], Adagrad [8] or other optimization algorithms such as SAM [9].

REFERENCES

[1] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'13. Red Hook, NY, USA: Curran Associates Inc., 2013, p. 315–323.

[2] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/ede7e2b6d13a41ddf9f4bdef84fdc737-Paper.pdf

[3] T. Hofmann, A. Lucchi, S. Lacoste-Julien, and B. McWilliams, "Variance reduced stochastic gradient descent with neighbors," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/effc299a1addb07e7089f9b269c31f2f-Paper.pdf

[4] Y. LeCun and C. Cortes, "MNIST handwritten digit database," http://yann.lecun.com/exdb/mnist/, 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[5] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[6] A. Defazio and L. Bottou, "On the ineffectiveness of variance reduced optimization for deep learning," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/84d2004bf28a2095230e8e14993d398d-Paper.pdf

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[8] J. Duchi and E. Hazan, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 07 2011.

[9] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," 2021.

APPENDIX

A. SVRG algorithm

We outline the SVRG algorithm [1].

Algorithm 1 SVRG

```
1: Hyper-params: learning rate $\gamma \in [0,1]$, batch size $b \geq 1$, snapshot probability $p \in [0,1]$
2: Take $w^{(0)} \in \mathbb{R}^d$
3: $\tilde{w} \leftarrow w_0$
4: for $t = 1, \ldots, T$ do
5:     $B \sim \text{unif}(\binom{[n]}{b})$
6:     $s^{(t)} \leftarrow \nabla f_B(w^{(t-1)})$
7:     $v^{(t)} \leftarrow \nabla f_B(\tilde{w}) - \nabla f(\tilde{w})$
8:     $w^{(t)} \leftarrow w^{(t-1)} - \gamma \cdot (s^{(t)} - v^{(t)})$
9:     $\tilde{w} \leftarrow \begin{cases} w^{(t)}, & \text{with probability } p \\ \tilde{w}, & \text{otherwise} \end{cases}$
```

B. Original SAGA algorithm

We outline the original SAGA algorithm [2], [3].

Algorithm 2 SAGA

```
1: Hyper-params: learning rate $\gamma \in [0,1]$
2: Take $w^{(0)} \in \mathbb{R}^d$
3: $\{\tilde{w}^{(i)}\}_{i=1}^n \leftarrow w^{(0)}$
4: for $t = 1, \ldots, T$ do
5:     $i \sim \text{unif}(n)$
6:     $s^{(t)} \leftarrow \nabla f_i(w^{(t-1)})$
7:     $v^{(t)} \leftarrow \nabla f_i(\tilde{w}^{(i)}) - \frac{1}{n}\sum_{j=1}^n \nabla f_j(\tilde{w}^{(j)})$
8:     $w^{(t)} \leftarrow w^{(t-1)} - \gamma \cdot (s^{(t)} - v^{(t)})$
9:     $\tilde{w}^{(i)} \leftarrow w^{(t)}$
```

C. C-SAGA

We outline the C-SAGA algorithm.

Algorithm 3 C-SAGA algorithm

```
1: Hyper-params: learning rate $\gamma \in [0,1]$, snapshot probability $p \in [0,1]$, number of groups $k \geq 1$
2: Randomly partition $[n]$ into equally sized clusters $\{C_\ell\}_{\ell=1}^k$
3: Take $w^{(0)} \in \mathbb{R}^d$
4: $\{\tilde{w}^{(\ell)}\}_{\ell=1}^k \leftarrow w^{(0)}$
5: for $t = 1, \ldots, T$ do
6:     $i \sim \text{unif}(n)$
7:     $s^{(t)} \leftarrow \nabla f_i(w^{(t-1)})$
8:     $\ell \leftarrow$ index $h$ such that $i \in C_h$
9:     $v^{(t)} \leftarrow \nabla f_i(\tilde{w}^{(\ell)}) - \frac{1}{k}\sum_{h=1}^k \nabla f_{C_h}(\tilde{w}^{(h)})$
10:    $w^{(t)} \leftarrow w^{(t-1)} - \gamma \cdot (s^{(t)} - v^{(t)})$
11:    $\tilde{w}^{(\ell)} \leftarrow \begin{cases} w^{(t)}, & \text{with probability } p \\ \tilde{w}^{(\ell)}, & \text{otherwise} \end{cases}$
```

D. Extra Plots

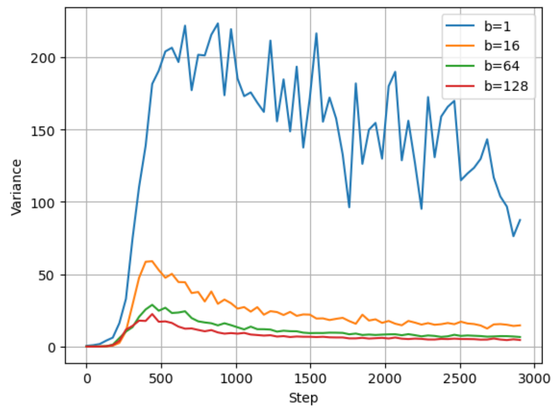

Fig. 6. Estimated variance of SGD with batch size $b = 1, 16, 64, 128$. This plot also serves as a validation for the way we estimate the variances. In fact, we see that the plotted estimated variance reflects the expected behaviour of the true variance of SGD: one can see that the (estimated) variance of SGD with larger batch sizes is lower at almost every time step. Moreover, the biggest difference is seen between $b = 1$ and $b = 16, 64, 128$, which reflects the fact that the largest multiplicative gap is the one between $b = 1$ and $b = 16$.

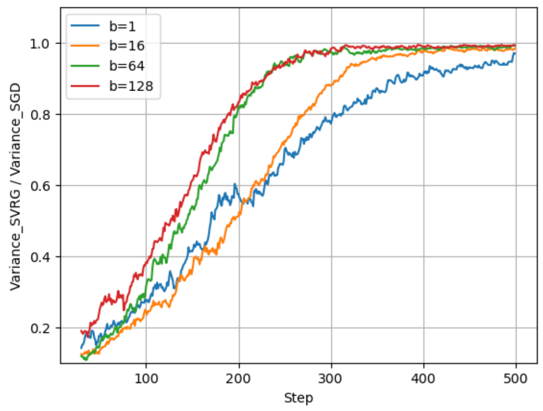

Fig. 7. Ratio of the Variance of SVRG to the Variance of SGD with batch size $b = 1, 16, 64, 128$ for the first 500 steps. Values smaller than 1 indicate that SVRG had less variance at that step than SGD.

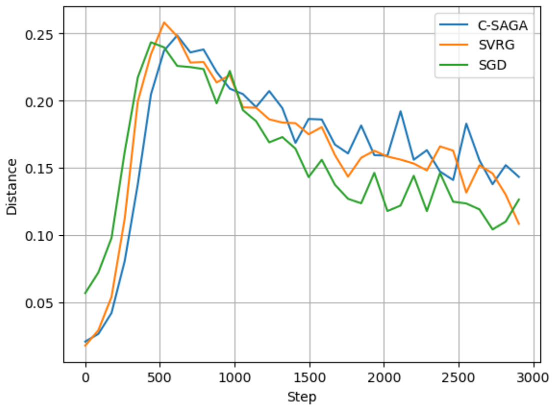

Fig. 9. The distance between consecutive iterates ($\|w^{(t)} - w^{(t-1)}\|$) of C-SAGA, SVRG and SGD.

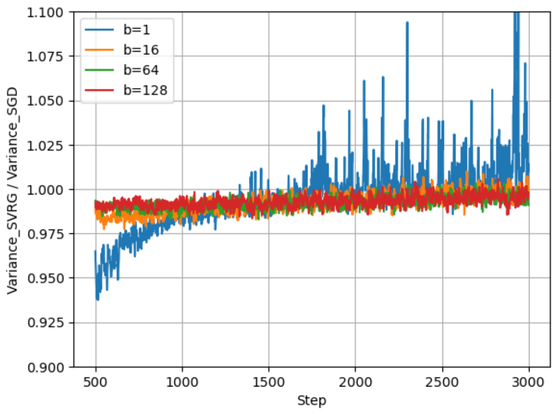

Fig. 8. Ratio of the Variance of SVRG to the Variance of SGD with batch size $b = 1, 16, 64, 128$ for the last 2500 steps. Values smaller than 1 indicate that SVRG had less variance at that step than SGD.

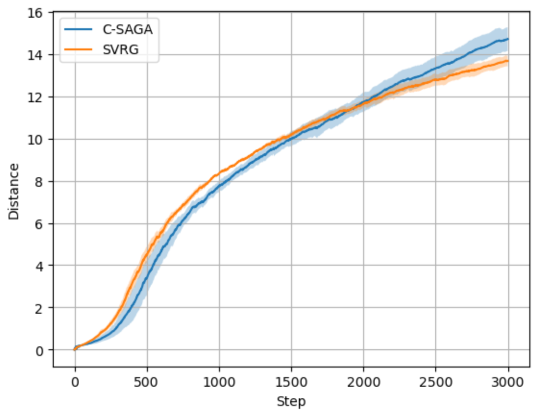

Fig. 10. The distance of an iterate $w^{(t)}$ from its snapshot for C-SAGA and SVRG. For SVRG, it is defined as $\|w^{(t)} - \tilde{w}\|$. For C-SAGA, it is defined as $\|w^{(t)} - \tilde{w}^{(\ell)}\|$ where ℓ is such that the index i sampled at the step t is $i \in C_\ell$. Shaded areas represent the 95% confidence interval over the five runs (i.e. the slab of $\pm 1.96 \cdot \sigma$ around the mean, where σ is the standard deviation over the five runs).