EMPIRICAL STUDY OF VARIANCE-REDUCED METHODS FOR MACHINE LEARNING

Sathvik Joel K
Dept. of Computer Science and Engineering
IIT Madras
cs19b025@smail.iitm.ac.in

Vishnu Vinod
Dept. of Computer Science and Engineering
IIT Madras
cs19b048@smail.iitm.ac.in

***K*eywords** Gradient Descent · Variance Reduction · SVRG · SAG · SDCA · ADAM

1 Introduction

The literature on variance reduced (VR) stochastic gradient algorithms has extensively studied their iterative bounds[1]. However, there has been a noticeable lack of empirical studies evaluating and comparing the practical performance of these algorithms on both convex and non-convex problems, and across various datasets. Moreover, many of the proposed algorithms lack sufficient empirical evaluations in their original papers.

In this work, we aim to address this gap by conducting a thorough empirical analysis of several VR stochastic gradient algorithms. We explore their performance in diverse problem settings, including both convex and non-convex cases, and test them on multiple datasets. We hope that our study provides valuable insights into the practical effectiveness of these algorithms and helps to bridge the gap between theory and practice. Additionally, we aim to fill the gap in the literature by providing empirical evaluations for many algorithms that lack sufficient experimental results in their original papers.

2 Related Work

Stochastic gradient descent (SGD) is a widely used optimization algorithm for training deep neural networks, but its convergence can be slow due to high variance in stochastic gradients. To address this issue, several variance reduction methods have been proposed by various authors, in the past decade, including Stochastic Average Gradient (SAG)[2][3], Stochastic Variance Reduced Gradient (SVRG)[4], and Stochastic Dual Coordinate Ascent (SDCA)[5], among others.

Initial experimental results show that all these methods can achieve faster convergence and better generalization performance compared to traditional SGD. SAG is memory-based and can reduce variance in gradients by keeping a history of past gradients. SVRG reduces bias by periodically computing the full gradient. SDCA updates one dual variable at a time and uses a cyclic coordinate descent scheme to minimize the primal objective function.

Variance reduced methods suffer from significant limitations too. SVRG and related methods have shown promising results in various tasks, such as image classification, but require careful implementation and tuning of hyperparameters. It is more versatile than most other VR methods and a number of variations/improvements of SVRG have spawned over the past decade. In a recent study by Defazio et al.[6], it was shown that the Stochastic Variance Reduced Gradient (SVRG) algorithm exhibits an increase in variance for a majority of each epoch in deep neural networks such as ResNet and DesNet. This finding highlights a potential limitation of SVRG in the context of deep learning.

In recent years, momentum-based methods have dominated both research interest and effort in enhancing the convergence of Stochastic Gradient Descent (SGD). This acceleration in convergence has been achieved by introducing a momentum term that smooths out the gradient updates. Nesterov Accelerated Gradient (NAG)[7] is one the first such widely used momentum-based method that has shown faster convergence than traditional SGD and has been applied successfully in various deep learning tasks. Another popular momentum-based method is Adaptive Moment Estimation (Adam)[8], which can handle non-convex functions and is robust to noisy gradients.

Despite their success, momentum-based methods have limitations, such as slower convergence and poorer generalization performance in certain scenarios, including flat regions of the loss surface. Additionally, the selection of hyperparameters can significantly impact their performance, requiring careful tuning.

3 Problem Statement

This project paper aims to implement and extensively compare various variance reduced stochastic gradient algorithms, including Stochastic Average Gradient (SAG), Stochastic Average Gradient “Amélioré”(SAGA) [9], Stochastic Dual Coordinate Ascent (SDCA), and Stochastic Variance Reduced Gradient (SVRG). Additionally, we want to compare these algorithms with momentum-based approaches such as Adaptive Moment Estimation (Adam) and Adaptive Gradient Algorithm (Adagrad).

To evaluate the performance of these algorithms, we will conduct experiments in various problem settings, including both convex and non-convex cases. For the convex setting, we will use Logistic Regression and L2-regularized Logistic Regression.

We also aim to investigate the claim made in [6] by testing the performance of SVRG on shallow neural networks. We will use a range of shallow neural network architectures and evaluate the performance of SVRG in terms of variance and convergence rate. Our experiments will enable us to gain a better understanding of the performance characteristics of SVRG on both shallow networks. Moreover, our findings will provide insights into the suitability of SVRG for practical applications, particularly in the context of deep learning[10][11].

We will use standard datasets that are readily available on the LibSVM website [12], such as adult (a9a) , web (w8a), rcv1 (rcv1.binary), MNIST, and CIFAR-10. By evaluating the performance of these algorithms across a range of problem settings and datasets, we aim to provide a comprehensive analysis of their effectiveness in practice.

4 Experiments: Convex Paradigm

In the convex case, we conducted experiments using six datasets: adult (a9a), web (w8a), gisette, news20, diabetes, and australian from LibSVM. By comparing the performance of multiple algorithms on these datasets, we aimed to gain insight into their practical implications. The algorithms used in this comparison were Stochastic Variance Reduced Gradient (SVRG), Stochastic Dual Coordinate Ascent (SDCA), Coordinate Descent (CD), Adagrad, Stochastic Average Gradient "Amélioré" (SAGA), and Stochastic Average Gradient (SAG).

To minimize the function, we used the appropriate loss function, which in this case was the **log** loss. Specifically, we sought to minimize the following function:

$$\underset{\mathrm{w}}{\text{minimize}} \left(\frac{1}{n_{\text{samples}}} * \sum_i \text{loss}(\mathbf{w}^T \mathbf{x}_i, y_i) + \text{alpha} * 0.5 * ||\mathbf{w}||_2^2 \right) \tag{1}$$

Log loss is a loss function used in binary classification that measures the difference between predicted probabilities and true labels. It is defined as:

$$L_{log} = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log(\sigma(\mathbf{w}^T \mathbf{x}_i)) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}_i)) \right]$$

It is further important to note that log loss for logistic regression is a convex function and the overall function we optimize is also thus a convex function. The optimization problem thus lies entirely in the convex regime.

4.1 Finer Details

When employing algorithms that require a step size, such as SVRG, SAG, SAGA, and Adagrad, it is crucial to fine-tune the step size to achieve optimal performance for each dataset and algorithm. To achieve this, we perform hyperparameter optimization by sampling step sizes within the range of $(1, 1e-4)$ for each run. Specifically, we select a step size that best fits the training set and evaluate its accuracy on the test set. We repeat this process for multiple runs and select the one that yields the highest test accuracy. The selected step size is then used for the corresponding algorithm and dataset. The resulting step sizes for each algorithm and dataset are presented below 1

dataset /opt	a9a	w8a	gisette	news20	diabetes	australian
svrg	0.571	0.976	0.002	0.996	0.669	0.741
saga	0.433	0.793	0.025	0.098	0.989	0.702
sag	0.274	0.089	0.536	0.991	0.658	0.915
adagrad	0.168	0.007	0.958	0.889	0.015	0.166

Table 1: Optimal step sizes for each optimizer and dataset, obtained through a hyperparameter search of 100 runs.

To implement the algorithms, we used the Lightning-Sklearn-contrib library [13], which follows the Scikit-learn API conventions and supports both dense and sparse data representations. Additionally, computationally demanding parts of the library are implemented in Python, making the algorithms efficient and fast to run. This implementation is particularly suitable for practical applications in large-scale machine learning and our experiments

4.2 Results

In Figure 1, the training results are displayed, where the x-axis represents the objective value at a given time minus the optimal value, while the y-axis represents the CPU time. This plot gives a rough estimate of the gradient iterate since the library used does not allow for plotting the loss per iterate, but instead provides a callback functionality. Therefore, the CPU time (in seconds) has been used as a proxy measure for tracking the progress of the optimization algorithm.

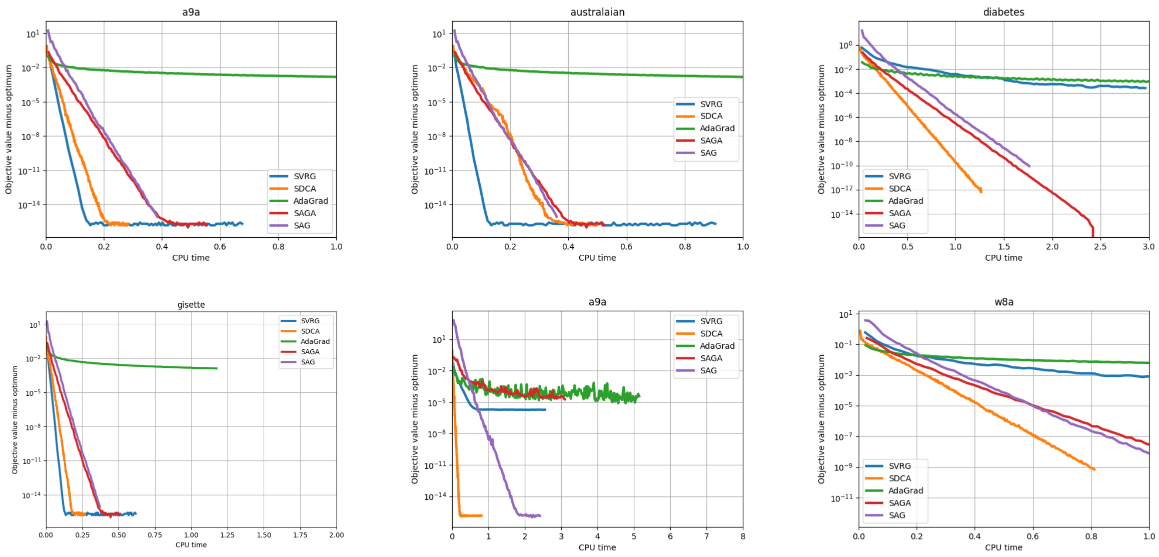


Figure 1: Comparison of training curves for different datasets with a convex objective function

4.3 A note on Error Bounds

We have repeated each experiment 20 times and plotted the average curves with the standard error bars, but given that the y scale is log, the error bars are not visible, but on zooming one can see thr error bars as shown in 2. To calculate the error bars we have used the formula

$$\text{error} = \frac{\text{data.std}}{\sqrt{n_{\text{experiments}}}}$$

where:

- error represents the standard error
- data.std represents the standard deviation of the data points from the repeated experiments.
- $n_{\text{experiments}}$ represents the number of experiments conducted.

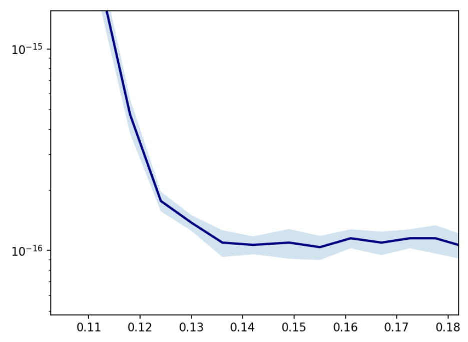


Figure 2: Example of error bounds

4.4 Observations and Inferences

From the graphs we can make the following observations

- Adagrad is known to take longer to converge compared to other optimization algorithms due to its significant computational overhead. This overhead is primarily due to the computation-intensive adagrad update step, which involves computing a sum of squares of gradients over all previous iterations. As a result, the algorithm requires significantly more computation per iteration than other methods, which can lead to slower convergence rates. Nonetheless, Adagrad remains a popular optimization algorithm due to its effectiveness in handling sparse data and noisy gradients. Practitioners should be aware of its computational demands when selecting an optimization algorithm for their specific problem.
- SDCA consistently performs better across datasets, but it is not used in practice because SDCA has high per-iteration computational cost and requires solving a large number of optimization problems in each iteration, which can make it impractical for large datasets or high-dimensional feature spaces. Its performance can be sensitive to the choice of regularization parameter, making it challenging to tune for different tasks. As a result, despite its good performance, SDCA is not widely used in practice.
- Performance of SAG and SAGA was similar on most of the datasets, this can be expected due to the similarity of both the gradient updates, Both methods maintain an estimate of the gradient for each sample and use this estimate to update the model parameters. The main difference between SAG and SAGA is in the way they update the gradient estimate.
- SVRG's performance varies significantly depending on the dataset used. It performs well on some datasets, such as a9a, australian, and gisette, but poorly on others, such as diabetes and w8a. The reasons for this variability are not fully understood but may be related to dataset characteristics or the need to tune hyperparameters. Nonetheless, SVRG is widely used for large-scale optimization problems due to its potential for high accuracy and fast convergence rates in certain cases. However, practitioners must evaluate its performance on their specific datasets to select the best algorithm.

5 Experiments: Neural Networks

Neural networks have emerged as a critical tool in the field of artificial intelligence, offering significant utility due to their capacity to learn and adapt from data. Therefore, it is crucial to conduct a comprehensive survey to investigate and analyze the performance and characteristics of these algorithms when applied in neural network training scenarios. In our experimentation, we shall focus on SVRG and try to provide a comprehensive comparison with SGD and other momentum-based methods.

5.1 Experimental Setup

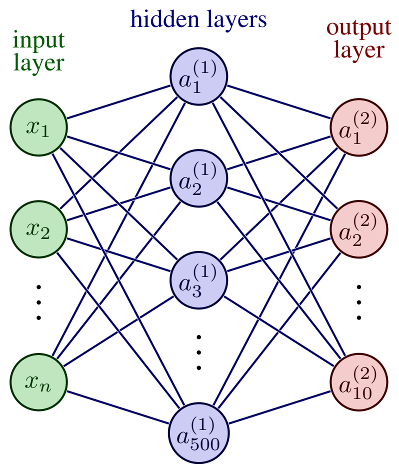
Dataset

In our experimental study, we have employed two widely recognized datasets, **MNIST** and **CIFAR-10**, which are commonly used for image classification tasks. Prior to training our models on them, we normalized the images from both datasets. The classification task for both datasets encompasses ten classes: the ten digits (0-9) for MNIST and ten distinct image classes for CIFAR-10. The selection of these datasets aimed at leveraging their established status as benchmark datasets for exploring advancements in machine learning techniques. Additionally, we chose these datasets due to their contrasting difficulty levels for an MLP-based dense model to effectively learn and classify the data. With the absence of convolutional layers and other modern machine learning innovations, MLP-based models while comparatively successful at classifying images from MNIST, tend to perform significantly poorly even when trained optimally on CIFAR-10.

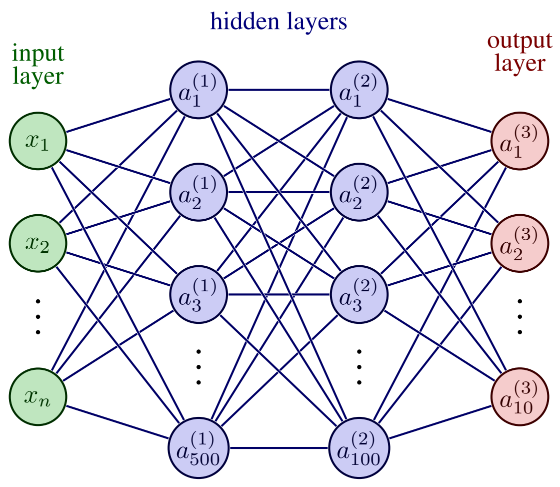
Models

The models that we have used for the classification task are given below in Fig.3. Each of these models has been trained from scratch (no pretraining) in order to avoid tarnishing the progress of each algorithm. Throughout the models, ReLU activation was applied at each layer, except for the final layer where softmax activation was used. It is worth noting that the use of softmax activation introduces non-convexity to both the final loss function and the entire problem. Using a softmax activation is crucial to framing the problem of training neural networks within the non-convex regime.

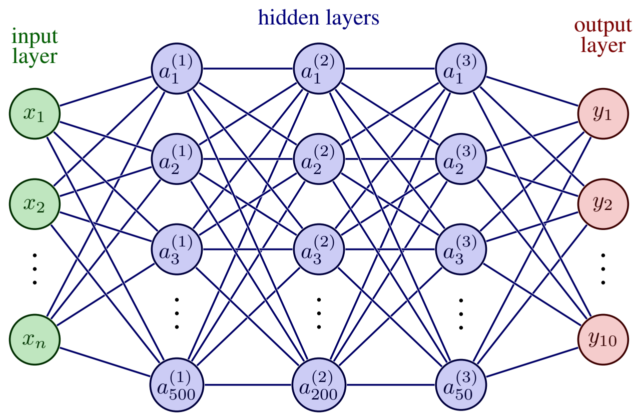
- **Model 1**: 1 hidden layer - 500 nodes
- **Model 2**: 2 hidden layers - 500 and 100 nodes respectively
- **Model 3**: 3 hidden layers - 500, 200 and 50 nodes respectively



(a) Model 1



(b) Model 2



(c) Model 3

Figure 3: Neural Networks used for training and reporting results

Cost Function

It is now important to consider the cost function. As we can clearly see the overall structure of the cost function to be optimized is nearly the same as in the convex case. However, with the introduction of regularization, we are no-longer dealing with simply a non-convex function. The overall cost function now has two parts: the non-convex loss function and the convex regularization term. This is shown below in Eqn. 2:

$$\text{Cost function} \quad = \quad \underset{\mathbf{w}}{\text{minimize}} \left(\overbrace{\frac{1}{n} \cdot \sum_i \text{loss}(\mathbf{w}^T \cdot \mathbf{x}_{\text{out},i}, y_i)}^{\text{non-convex}} \quad + \quad \underbrace{\frac{\lambda}{2}||\mathbf{w}||_2^2}_{\text{convex}} \right) \tag{2}$$

The loss function used here is the Cross-Entropy loss function which is commonly used for classification tasks. The inclusion of the second L_2-regularization term in the cost function introduces convexity to the cost landscape. Notably, as the value of the regularization parameter λ increases so does the convexity of the graph and the properties of the learning curves of the algorithms begin to closely mimic the convex case.

Algorithms

The selection of algorithms investigated in this study focuses on those suitable for neural networks - thus excluding SDCA and SAG due to their higher computational and memory requirements. The comparison was primarily conducted between SVRG and other algorithms, including SGD with constant step size (henceforth referred to as SGD1), SGD with decaying step size (henceforth referred to as SGD2), and ADAM. This allows us to carry out a comprehensive analysis of these algorithms across various datasets and different values of the regularization parameter λ.

It is worth mentioning that a batch-based implementation of SVRG was employed, where the gradient snapshot is computed once for each batch. Throughout experimentation, a consistent batch size of 50 was used for all algorithms.

Learning Rate

In order to obtain the best results, the learning rate was tuned by picking from the set $[0.1, 0.01, 0.001, 0.0001]$ for both datasets and comparing the validation accuracies achieved by each setting of learning rate when training using a constant step-size SGD optimizer, for SGD1, SGD2 and SVRG, and a constant step-size Adam optimizer for ADAM. This optimal setting of learning rate has then been used to compare the performance of all the other algorithms. Except in the case of SGD with decaying step size, the learning rate remains constant for all algorithms throughout the training of the model. In order to get a good idea of decaying learning rate, we use a decay rate of $\gamma = 0.955$. This value was chosen since a decay of γ after every step would cause the learning rate to decay by a factor of 100 after 100 steps (since $\gamma^{100} = 0.955^{100} \approx 0.01$).

The final chosen values for the learning rate for each combination of dataset and algorithm is given below:

Dataset \Optimizer	SGD1	SGD2	SVRG	ADAM
MNIST	0.001	0.001	0.001	0.001
CIFAR-10	0.01	0.01	0.01	0.001

Table 2: Tuned Step-Sizes for each optimizer and dataset

Metrics

The chosen metric for comparing algorithms in this study is the training loss, which provides valuable insights into algorithm performance. Both the magnitude of the training loss and its rate of descent are considered. While the validation set accuracy could have been an alternative metric for assessing algorithm quality and how well it learns, we opted for a more direct approach by using the training loss as a metric for evaluating the algorithm's convergence to the optimal value.

Training Settings

For each of the 4 algorithms chosen, we have carried out training for 100 epochs in 24 different settings.

- **Datasets**: MNIST and CIFAR-10
- **Models**: Model 1, Model 2 and Model 3 in Fig.3
- **Regularization**: $\lambda = 0$, $\lambda = 10^{-2}$, $\lambda = 10^{-4}$ and $\lambda = 10^{-6}$

The average training time for each of the four algorithms in case of both the datasets as well as the overall average training time has been reported in the table below. It is to be noted that all time is given in **mm : ss** format

Algorithm	Avg. Time (MNIST)	Avg. Time (CIFAR)	Total Average Time
SVRG	36:25.53	31:47.45	34:06.21
SGD1	18:11.82	16:25.68	17:18.75
SGD2	18:59.90	16:07.90	17:33.90
Adam	21:00.80	16:44.30	18:52.55

Table 3: Average training times for each algorithm (over 100 epochs)

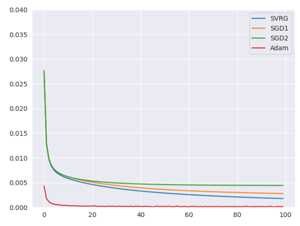
As we can see SVRG takes significantly more time to train than all the other methods (by a factor of 2). This is due to the calculation as well as backward propagation of gradient snapshots associated with the training process. All the other algorithms take roughly equal time to train the models for 100 epochs.

Device Specifications

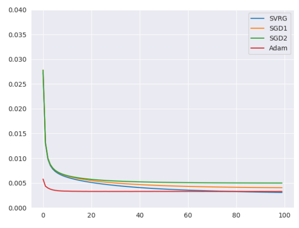
All training was carried out on a device with an AMD Ryzen 9 5900HS processor and NVIDIA RTX3050 GPU with 4 GB VRAM. The training times reported have minimal interference from background processes and the machine was kept idle throughout the training phase. It is also important to note that an implementation of SAG was attempted but aborted due to the process causing an overflow in memory.

5.2 Results: MNIST

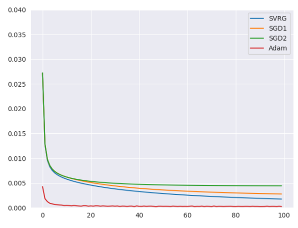
The training results are presented in the following plots, where the y-axis represents the training loss and the x-axis represents the epoch number. Unlike the convex scenario, all algorithms share the same range for both axes, enhancing the comparability across different configurations. The final training progress of each algorithm on the MNIST dataset, considering various combinations of regularization and models, is displayed below in Fig. 4:



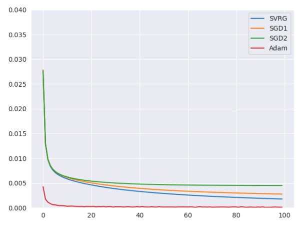
(a) Model 1: $\lambda = 0$



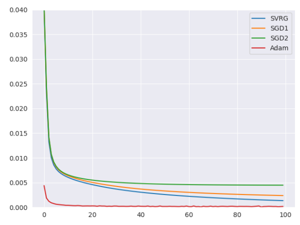
(b) Model 1: $\lambda = 10^{-2}$



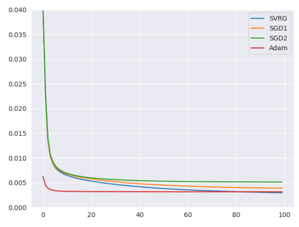
(c) Model 1: $\lambda = 10^{-4}$



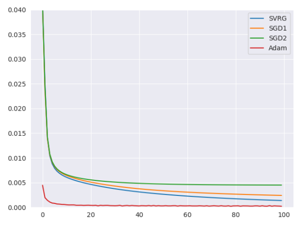
(d) Model 1: $\lambda = 10^{-6}$



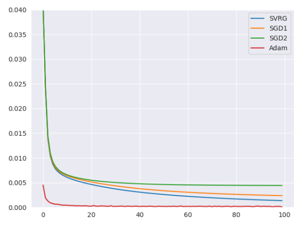
(e) Model 2: $\lambda = 0$



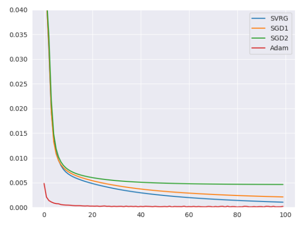
(f) Model 2: $\lambda = 10^{-2}$



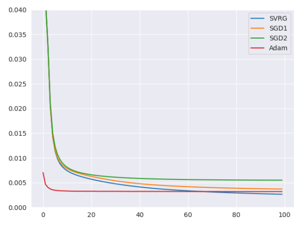
(g) Model 2: $\lambda = 10^{-4}$



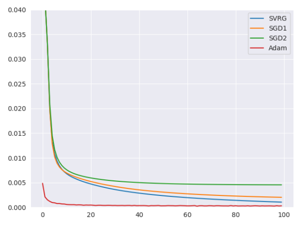
(h) Model 2: $\lambda = 10^{-6}$



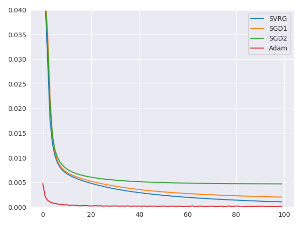
(i) Model 3: $\lambda = 0$



(j) Model 3: $\lambda = 10^{-2}$



(k) Model 3: $\lambda = 10^{-4}$

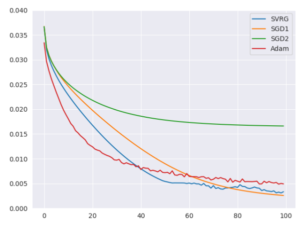


(l) Model 3: $\lambda = 10^{-6}$

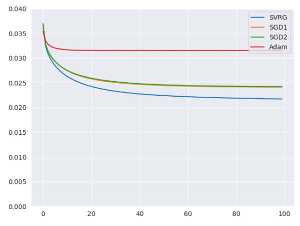
Figure 4: Training Loss v/s Epoch for MNIST dataset

5.3 Results: CIFAR-10

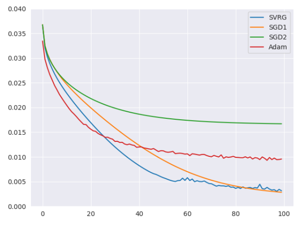
The training results are presented in the following plots, where the y-axis represents the training loss and the x-axis represents the epoch number. Unlike the convex scenario, all algorithms share the same range for both axes, enhancing the comparability across different configurations. The final training progress of each algorithm on the CIFAR-10 dataset, considering various combinations of regularization and models, is displayed below in Fig. 5



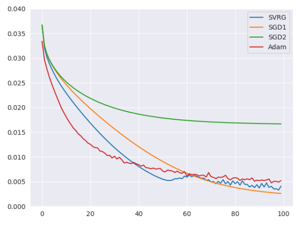
(a) Model 1: $\lambda = 0$



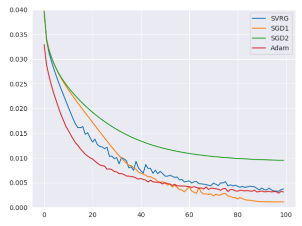
(b) Model 1: $\lambda = 10^{-2}$



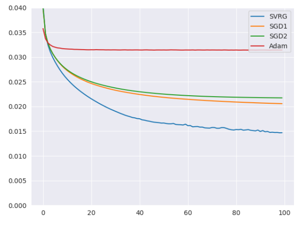
(c) Model 1: $\lambda = 10^{-4}$



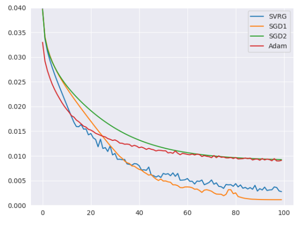
(d) Model 1: $\lambda = 10^{-6}$



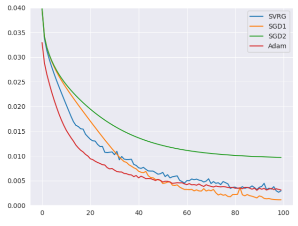
(e) Model 2: $\lambda = 0$



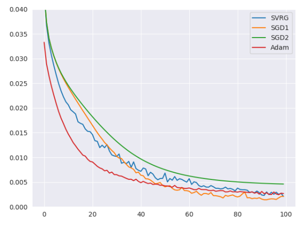
(f) Model 2: $\lambda = 10^{-2}$



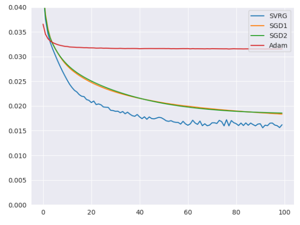
(g) Model 2: $\lambda = 10^{-4}$



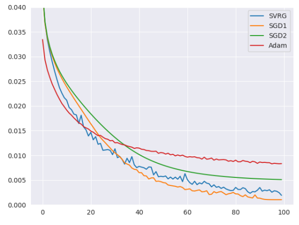
(h) Model 2: $\lambda = 10^{-6}$



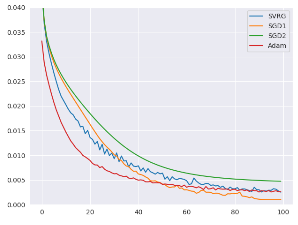
(i) Model 3: $\lambda = 0$



(j) Model 3: $\lambda = 10^{-2}$



(k) Model 3: $\lambda = 10^{-4}$



(l) Model 3: $\lambda = 10^{-6}$

Figure 5: Training Loss v/s Epoch for CIFAR-10 dataset

5.4 Observations and Inferences

Based on our rigorous experimentation, we have made several observations regarding the performance of different algorithms under various experimental conditions. These insights provide valuable guidance in selecting suitable algorithms for specific problem scenarios.

- In the case of MNIST, satisfactory convergence rates are observed for all optimizers. However, among the set of optimizers, SGD2 exhibits the poorest performance in terms of both convergence rate and achieved minimum training loss. A slight improvement is seen with SGD1, while SVRG surpasses both SGD optimizers despite its computational overhead. Notably, ADAM emerges as the best algorithm, demonstrating lightning-fast convergence to an exceptionally low training loss value (which appears to reach zero within the scale shown in the visualizations).
  It is important to mention that while SGD1, SGD2, and SVRG exhibit smooth convergence curves, noticeable undulations are observed in the training loss curves of ADAM in the unregularized case for all three models.
- We can further see that for MNIST, the three models give almost identical training loss curves. Despite Model 3 being marginally better with respect to testing accuracy achieved, the convergence of all the optimizers for the three models does not give us any useful information.
- The observed result of SGD1 outperforming SGD2 is unexpected and counterintuitive. In the case of a non-convex cost landscape, one would typically anticipate that the decaying step size of SGD2 allows it to descend deeper into a cost function minima compared to SGD1. It is further expected that for an undecayed step-size, the optimizer may shoot out of the minima leading to a sub-optimal outcome. However, an important

caveat we hypothesize is that a decaying learning rate can potentially cause SGD2 to become more susceptible to getting trapped in sub-optimal local minima of the cost landscape. The decaying step size prevents it from escaping these minima, leading to inferior performance compared to its constant step-size counterpart.

- When considering the results for CIFAR-10, it becomes apparent that all optimizers struggle to achieve rapid convergence compared to the results obtained with MNIST. This difficulty can be attributed to CIFAR-10 being a more challenging dataset for a simple MLP-based neural network without convolutional or residual blocks. Among the optimizers, there is no clear discernible trend regarding which optimizer converges to the most optimal training loss value. However, it is noteworthy that ADAM consistently demonstrates the fastest convergence with a highly convex training loss curve.
  SGD2 exhibits the poorest overall performance, particularly in settings involving Model 1. The reason for this observation will be discussed later. Additionally, it is interesting to observe that both SGD optimizers exhibit notably smoother convergence curves compared to SVRG and ADAM in the unregularized case.
- One of the intriguing findings pertains to the impact of the regularization parameter λ on the training curves. In a previous section discussing the cost section, it was noted that the overall cost does not belong strictly to the non-convex regime and can be considered as the sum of a convex and non-convex function. As the value of the regularization parameter λ increases from 0 to 10^{-6} to 10^{-4} and finally 10^{-2} we can note several interesting observations. It is worth mentioning that $\lambda = 10^{-2}$ represents an over-regularized value across all combinations of optimizers and datasets. This allows us to gain insights from an exaggerated scenario where the overall cost function becomes predominantly convex, thereby diminishing the effects of non-convexities.
- As the value of λ increases, the training loss curves for the various optimizers become much smoother, with the setting of $\lambda = 10^{-2}$ giving very smooth training curves across all settings. It is also interesting to note that with regularization, the optimal training loss achieved by each of the optimizers increases as expected. However, the effect of this increase is most pronounced for ADAM which goes from being the best overall performer across all unregularized settings to being the worst overall performer in all settings with $\lambda = 10^{-2}$. A similar observation is that the effect of regularization is least pronounced in SVRG. In fact for all settings with $\lambda = 10^{-2}$, SVRG emerges as the best performer with respect to optimal training loss achieved, significantly outperforming its competitors.
- It was observed that for CIFAR-10, SGD2 performs very poorly for Model 1 in all settings (except for the over-regularized setting of $\lambda = 10^{-2}$, while somehow regaining its performance for Model 2 and Model 3. We hypothesize the reason for this to be that the magnitude of depth of local minima in the cost landscape of Model 1 is significantly higher compared to the cost landscapes of Model 2 and Model 3.

Based on the comprehensive experimentation conducted and the aforementioned observations, we are now able to draw several significant inferences. The key inferences are as follows:

- ADAM and momentum-based methods, significantly outperform SVRG and similar methods in the case of neural networks, for both regularized and unregularized methods. We see this reflected in the significant differences in training loss convergence as well as training time.
- The computational overhead for SVRG leads to almost double the training time as that taken for SGD, Adam etc. This makes it unusable for real-world applications where fast convergence is measured with respect to time taken and not just number of training epochs completed.
- SVRG is quite resistant to regularization across datasets. On the other hand, ADAM is surprisingly pliant and is greatly affected by the value of the regularization parameter chosen. It is thus important to consider the type of optimizer being used before choosing a value for the regularization parameter λ.

6 Conclusion

In summary, it can be concluded that Stochastic Variance Reduced Gradient (SVRG) and other variance reduced methods exhibit notable advancements compared to Stochastic Gradient Descent (SGD) in the convex regime, despite their slightly increased computational complexity. However, in the non-convex regime, particularly when dealing with neural networks, SVRG, although worthy of investigation, does not provide substantial performance improvements or the relatively low computational burden offered by ADAM. For this reason, SVRG has not achieved the level of ubiquity across thousands of different applications of machine learning.

References

[1] Robert Mansel Gower, Mark W. Schmidt, Francis R. Bach, and Peter Richtárik. Variance-reduced methods for machine learning. *Proceedings of the IEEE*, 108:1968–1983, 2020.

[2] Nicolas Roux, Mark Schmidt, and Francis Bach. A stochastic gradient method with an exponential convergence _rate for finite training sets. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[3] Mark Schmidt, Nicolas Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162, 09 2013.

[4] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[5] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(16):567–599, 2013.

[6] Aaron Defazio and Leon Bottou. On the ineffectiveness of variance reduced optimization for deep learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[7] Yurii Nesterov. A method for solving the convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, 269:543–547, 1983.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[9] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives, 2014.

[10] Zeyuan Allen Zhu and Elad Hazan. Variance reduction for faster non-convex optimization. In *International Conference on Machine Learning*, 2016.

[11] Sashank J. Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alex Smola. Stochastic variance reduction for nonconvex optimization. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 314–323, New York, New York, USA, 20–22 Jun 2016. PMLR.

[12] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3), May 2011.

[13] Mathieu Blondel and Fabian Pedregosa. Lightning: large-scale linear classification, regression and ranking in Python, 2016.