RESCIENCE C

ML Reproducibility Challenge 2022

Reproducing results for Adaptive Stochastic Variance Reduction for Non-convex Finite-Sum Minimization method

Isaac Battles[1], Patryk Będkowski[2], Wojciech Rokicki[3]
[1,2,3]Swiss Federal Institute of Technology Lausanne (Ecole Polytechnique Fédérale de Lausanne)

Published
22 December 2022

Reproducibility Summary

In the following paper we managed to sucessfully reproduce the results from the original paper [1]. We executed experiments comparing several methods of gradient optimization including variance-reduction algorithms. For non-convex setting it turned out, that variance reduction algorithms outperforms other non-VR algorithms in the sense of sample complexity. Moreover it was confirmed, that Spider is the only algorithm which outstands from other VR methods. We cannot tell the difference betweeen the rest of tested VR methods. It can prompt, that the further investigation and experiments are needed. AdaSpider could have a significant advantage in specific setting, but not proved here.

The obtained results vary from the original paper's outcomes. We suspect, that it is due to differece in algorithms initialization. The authors didn't specify the initialization of the models. Therefore the results are not exactly the same, but they preserve characteristics and dynamics of the algorithms.

Additionally we have tried to reproduce the second part of the paper considering the Neural Nets learning experiments. Unfortunately we didn't manage to find the proper aproach to implement the optimizers. We were able to obtain the code from the authors, so we started with the premise that we will implement it in TensorFlow to add a value. It turned out, that this framework was very hard to work with and we had problems with compatibility, as well as with modifying core optimizers classes. We found only deprecated implementations of VR optimizers (e.g. SVRG), which uses previous gradients.

The authors have declared that no competing interests exists.
Code is available at https://github.com/Wojtek-Rokicki/ML-Reproducibility-Challange.

1 Introduction

The aim of the work is to reproduce the results from the "Adaptive stochastic variance reduction for non-convex finite-sum minimization" paper [1].

The algorithm implemented by the team from Laboratory for Information and Inference Systems at the EPFL combines an adaptive step-size schedule with the recursive stochastic path integrated estimator proposed in Fang et al. [2]. It does not require any problem-dependent parameters such as smoothness constant L, target accuracy ϵ or any bound on gradient norms and its ϵ-stationary point is $\tilde{O}(n + \sqrt{n}/\epsilon^2)$.

First-order methods such as AdaSpider have the potential to provide low-accuracy solutions at low computational complexity which makes them an attractive set of tools in large-scale optimization problems. That's the reason for trying to reproduce and investigate further the properties of this algorithm.

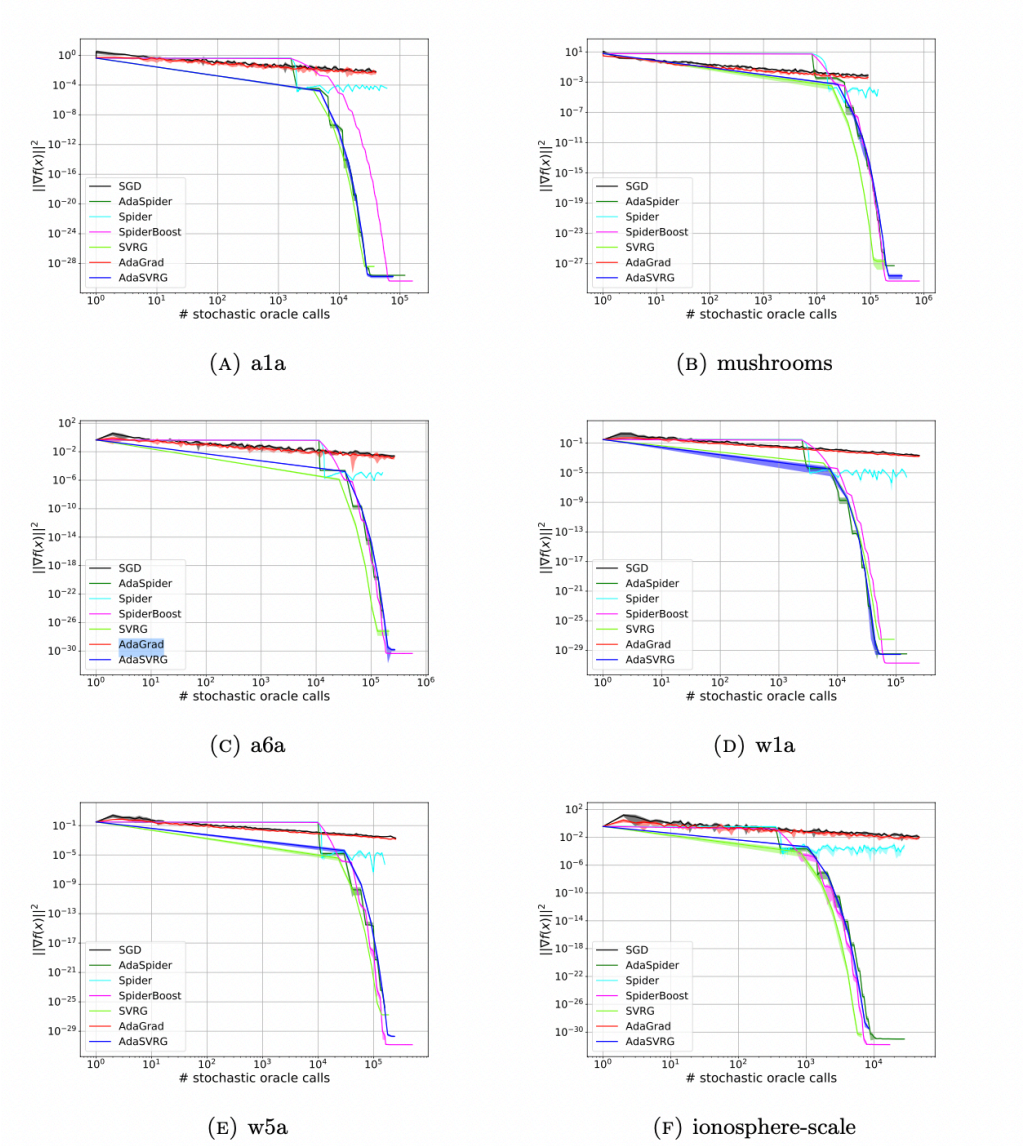
2 Scope of reproducibility

In the aforementioned paper, the authors carried out experiments comparing different first-order optimization methods. They compare AdaSpider against Spider, SpiderBoost, SVRG, AdaSVRG, and two non-VR methods SGD and AdaGrad. For that purpose they consider the minimization of a convex loss with a non-convex regularizer in the sense of Wang et al. [3].

They consider the following problem:

$$\min_{x \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} l(x, (a_i, b_i)) + \lambda g(x)$$

where $l(x, (a_i, b_i))$ is the loss with respect to the decision variable/weights x with a_i, b_i denoting the (feature vector, label) pair. We select $g = \sum_{i=1}^{d} \frac{x_i^2}{1+x_i^2}$, similar to Wang et al. [3], where the subscript denotes the corresponding dimension of x. They optimize the loss on 6 datasets from LibSVM, namely a1a, mushrooms, a6a, w1a, w5a, ionosphere-scale. Algorithms are tunned by the execution of parameter sweep for their initial step-size over an interval of values which are exponentially scaled as $\{10^{-3}, 10^{-2}, ..., 10^{2}, 10^{3}\}$. After tuning the algorithms on one dataset, they run them with the same parameters as the others. In the beginning, each algorithm is initialized at the same point and run 5 times on the dataset. Then they report the mean convergence with standard deviation as the shaded region around the mean curves.



(A) a1a (B) mushrooms

(C) a6a (D) w1a

(E) w5a (F) ionosphere-scale

Figure 1. Original paper results for logistic regression with non-convex regularizer on LibSVM datasets

Their claims are:

1. VR methods in general demonstrate superior sample complexity as compared to SGD and AdaGrad
2. Among VR algorithms, there does not seem to be any concrete differences with similar convergence, except for Spider
3. The performance of AdaSpider is on par with other VR methods, and superior to Spider

In our work, we try to reproduce the outcome of the results and try to accept or reject the original paper's claims.

The second part of the original paper's experiments is focusing on testing methods as neural net optimizers. The authors have chosen 3-layer fully connected network with

dimensions [28 * 28, 512, 512, 10] and ELU activation function. They have tested the model on two well-known benchmark datasets: MNIST and FashionMNIST. Due to the problems with custom optimizers implementation in TensorFlow, we won't focus on this part of the experiments. Later in the results section 4, we will include shortly our best try to implement SpiderBoost optimizer.

3 Methodology

The original paper does not provide any source code. Thus, for our experiment, we will implement all methods from scratch. As a base for AdaSpider, we will take the description in the original paper and for other methods' external references.

After implementing algorithms, we will test them for a small number of iterations and see if they are working - if the gradient and loss are decreasing.

The next step will be parameters sweep tuning, which will be carried out for the a1a dataset. Determined best hyperparameters will be used unchanged for all of the rest datasets.

In order to decrease the learning time for algorithms we use AWS machines.

3.1 Methods descriptions

In the following section, we will describe the methods, which were implemented and tested. We include the most important information such us the source of algorithms pseudocodes and the number of parameters they take. All the code is available at github repository.

- SGD
  - Source: Machine Learning CS-433 "Optimization" Lecture
  - Parameters:
    - stepsize λ
- AdaGrad
  - Source: Sebastian Ruder's blog post "An overview of gradient descent optimization algorithms"
  - Parameters:
    - stepsize λ
    - smoothing term ϵ (not tunned, usually $1e-8$)
- SVRG
  - Source: "A Variance Controlled Stochastic Method with Biased Estimation for Faster Non-convex Optimization"[4]
  - Parameters:
    - stepsize λ
    - full gradient computation interval q
    - Lipschitz gradient constant L

- AdaSVRG
  - Source: "SVRG Meets AdaGrad: Painless Variance Reduction"[5]
  - Parameters:
    * stepsize λ
    * full gradient computation interval q
    * Lipschitz gradient constant L
- Spider
  - Source: "Spider: Near-Optimal Non-Convex Optimization via Stochastic Path Integrated Differential Estimator"[2]
  - Parameters:
    * stepsize λ
    * full gradient computation interval q
    * approximate second-order stationary point ϵ
    * free parameter to choose n_0
    * Lipschitz gradient constant L
- SpiderBoost
  - Source: "SpiderBoost and Momentum: Faster Stochastic Variance Reduction Algorithms"[3]
  - Parameters:
    * stepsize λ
    * full gradient computation interval q
    * Lipschitz gradient constant L
- AdaSpider
  - Source: "Adaptive Stochastic Variance Reduction for Non-convex Finite-Sum Minimization"[1]
  - Parameters:
    * stepsize λ
    * full gradient computation interval q

3.2 Datasets

Datasets are coming from the well-known UCI's binary classification datasets collection. In papers we used the following ones:

a1a:

- Preprocessing: The original Adult data set has 14 features, among which six are continuous and eight are categorical. In this data set, continuous features are discretized into quantiles, and each quantile is represented by a binary feature. Also, a categorical feature with m categories is converted to m binary features
- Number of data: 1,605 / 30,956 (testing)
- Number of features: 123 / 123 (testing)

mushrooms:

- Preprocessing: Each nominal attribute is expanded into several binary attributes. The original attribute #12 has missing values and is not used.
- Number of data: 8124
- Number of features: 112

a6a:

- Preprocessing: The same as a1a
- Number of data: 11,220 / 21,341 (testing)
- Number of features: 123 / 123 (testing)

w1a:

- Number of data: 2,477 / 47,272 (testing)
- Number of features: 300 / 300 (testing)

w5a:

- Number of data: 9,888 / 39,861 (testing)
- Number of features: 300 / 300 (testing)

ionosphere-scale:

- Number of data: 351
- Number of features: 34

For convenience, we have used DSDL library. It is essentially UCI's LIBSVM datasets downloader written in python. It automatically divides the dataset in train and test data.

3.3 Hyperparameters

Algorithms are tunned by the execution of parameter sweep for their initial step-size over an interval of values which are exponentially scaled as $\{10^{-3}, 10^{-2}, ..., 10^{2}, 10^{3}\}$. The choice of the right parameter is based on the minimal loss from testing each method on the a1a dataset.

3.4 Experimental setup and code

For convenience, we have used Jupyter Notebook with Python language. It speeds up the process of testing small snippets of code. The core methods and functions were implemented in separate packages.

The main file was called `run_tests.py`. The file was divided into 4 main sections. The first consists of methods initialization and functions for loading datasets definitions. Then there is the main loop for testing the methods. Each algorithm returns the full gradient norm for each stochastic gradient oracle call. The third part is for plotting the results. Last but not least was used for hyperparameters sweep tunning.

In order to obtain the results you need to install all the python package's requirements and simply run the `run_test.py` notebook.

3.5 Computational requirements

We decided to use the AWS Sagemaker cloud machine-learning platform as our testing environment. It provides a variety of different computational instances which come in form of JupyterLab notebooks.

Our tests were run on ml.m5.4xlarge AWS Sagemaker instance which comes with 16 CPU cores optimized for python computations. It is recommended to run mentioned tests on a machine with multiple cores. Taking advantage of parallelism can also speed up computations. The below table presents the average time for each algorithm to perform 100'000 iteration

	Average computation time of 100'000 iterations
SGD	58.20s
AdaSpider	139.61s
Spider	75.43s
SpiderBoost	73.62s
AdaGrad	64.21s
SVRG	78.59s
AdaSVRG	93.84s

Table 1. Average time of computing 100'000 iterations with each method on ionosphere-scale dataset

4 Results

The results produced from experiment one include plots of each method mentioned in 3.1 for each dataset mentioned in section 3.2. Every method was then run for 10^5 iterations to achieve the same amount of stochastic oracles calls as in the paper and then was plotted against the norm of the full gradient squared at the corresponding iteration. The final result was then produced by repeating the following experiments 5 times, plotting the average convergence of each method and its standard deviation. The plots produced to show that the AdaSpider has a better average convergence when compared to other variance reduction methods.

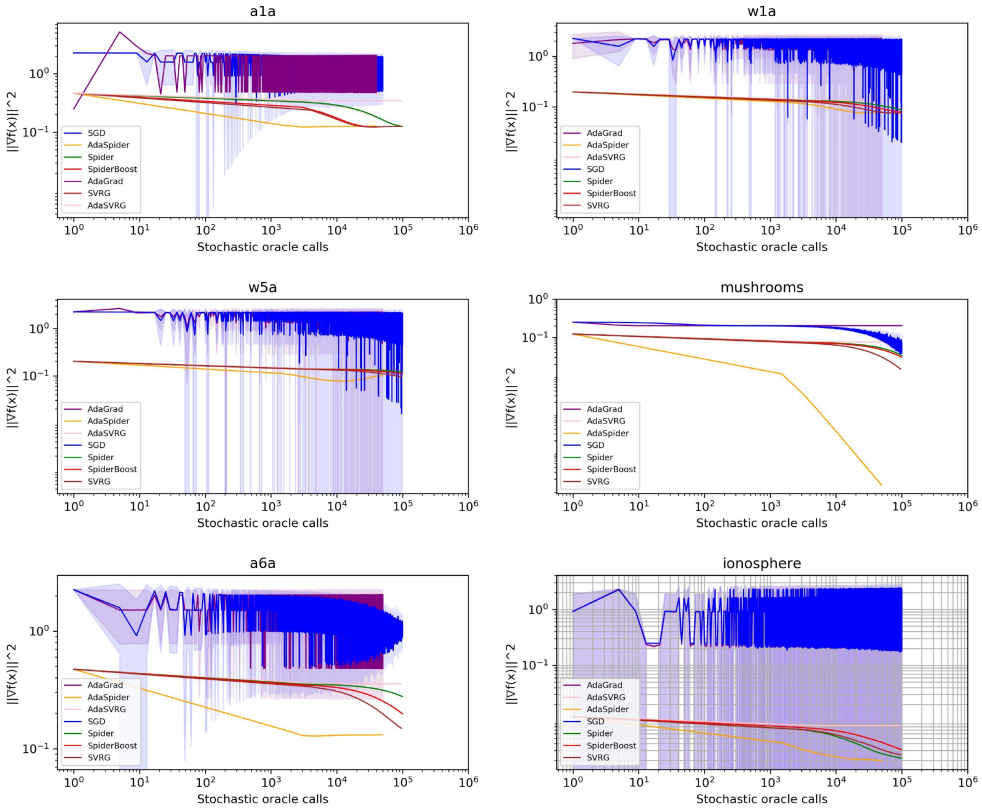


Figure 2. Methods results for UCI's datasets

4.1 Results reproducing original paper

a1a — (1) For the a1a dataset, claim 1 is supported by observing that SGD and AdaGrad behave, on average, worse than the variance reduction methods. Claim 2 can also be verified by observing that Spider converges, on average, worse than the other variance reduction methods, excluding SVRG. Claim 3 can also be verified by observing that AdaSpider converges, on average, faster than the other variance reduction methods.

(2) As for the success of reproducing the exact results from the original paper, it was not possible to achieve the same level of convergence for each method seen in the paper. This can be attributed to our lack of knowledge surrounding initialization and parameter tuning for the given dataset. It should also be mentioned that the standard deviation, the shading surrounding the methods, is much more extreme than the original paper. The potential explanation for this could be the same as the convergence. Of the methods, it can be observed that the SVRG method does not perform correctly. We are unsure why this method is behaving weirdly.

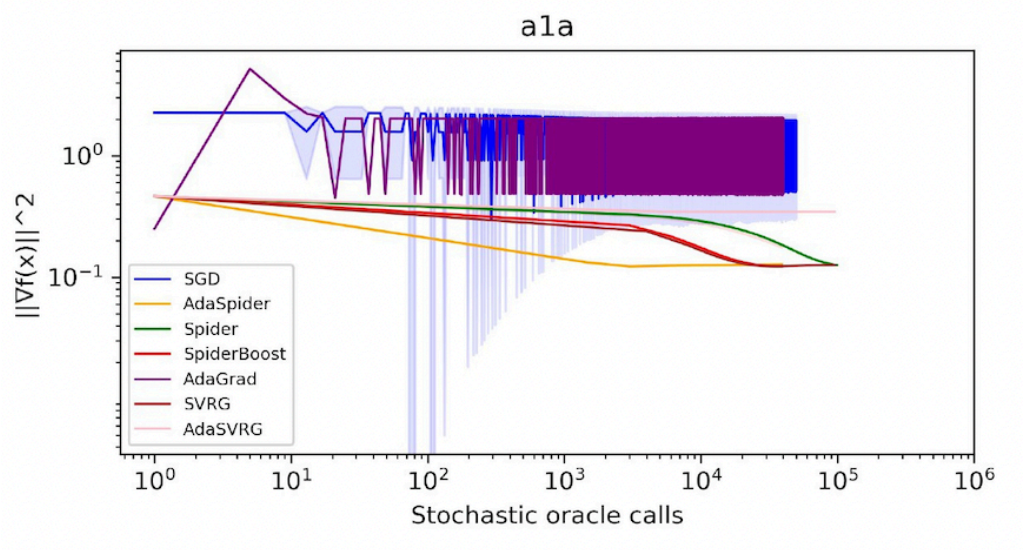


Figure 3. Methods results for the a1a dataset

mushrooms — (1) For the mushrooms dataset, claim 1 is supported by observing that SGD and AdaGrad behave, on average, worse than the variance reduction methods, though SGD performs significantly better than AdaGrad. Claim 2 can also be verified by observing that Spider converges, on average, worse than the other variance reduction methods, excluding SVRG. Claim 3 can also be verified by observing that AdaSpider converges, on average, significantly better than the other variance reduction methods.

(2) As for the success of reproducing the exact results from the original paper, it was not possible to achieve the same level of convergence for each method seen in the paper. This can be attributed to our lack of knowledge surrounding initialization and parameter tuning for the given dataset. Of the methods, it can be observed that the SVRG method does not perform correctly. We are unsure why this method is behaving weirdly.

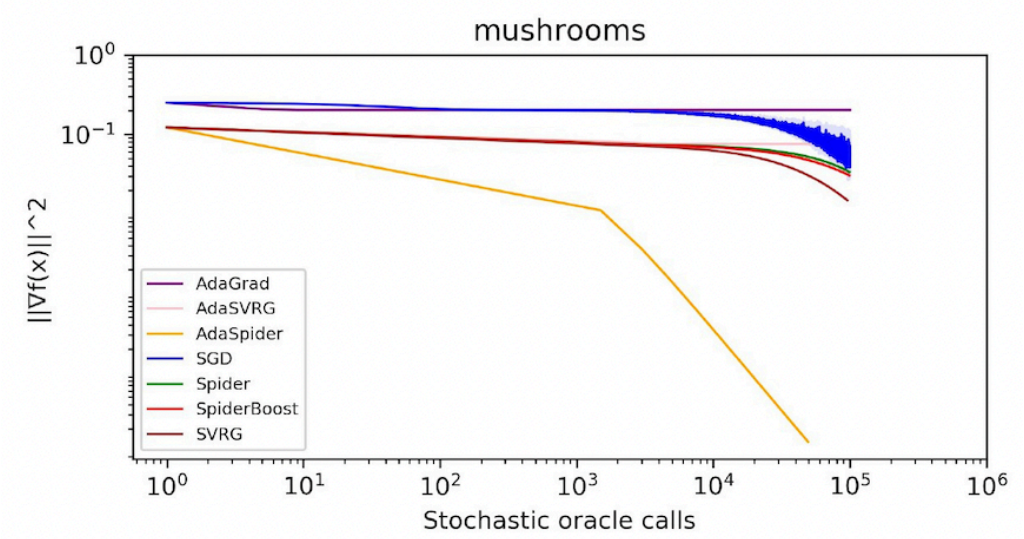


Figure 4. Methods results for the a1a dataset

a6a – (1) For the a6a dataset, claim 1 is supported by observing that SGD and AdaGrad behave, on average, worse than the variance reduction methods. Claim 2 can also be verified by observing that Spider converges, on average, worse than the other variance reduction methods, excluding SVRG. Claim 3 can also be verified by observing that AdaSpider converges, on average, faster than the other variance reduction methods.

(2) As for the success of reproducing the exact results from the original paper, it was not possible to achieve the same level of convergence for each method seen in the paper. This can be attributed to our lack of knowledge surrounding initialization and parameter tuning for the given dataset. It should also be mentioned that the standard deviation, the shading surrounding the methods, is much more extreme than the original paper. The potential explanation for this could be the same as the convergence. Of the methods, it can be observed that the SVRG method does not perform correctly. We are unsure why this method is behaving weirdly.

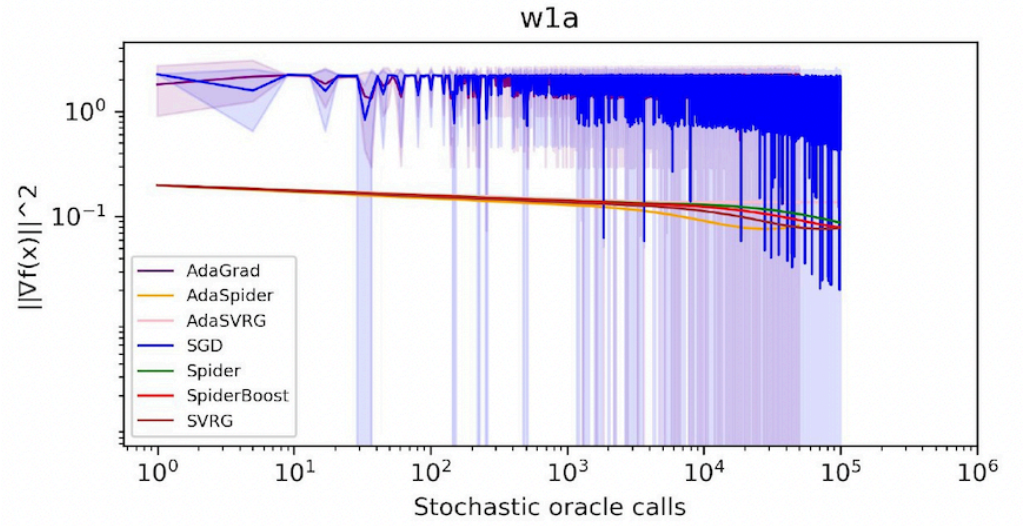


Figure 5. Methods results for the a1a dataset

w1a – (1) For the w1a dataset, claim 1 is supported by observing that SGD and AdaGrad behave, on average, worse than the variance reduction methods. Claim 2 can also be verified by observing that Spider converges, on average, worse than the other variance reduction methods, excluding SVRG. Claim 3 can also be verified by observing that AdaSpider converges, on average, faster than the other variance reduction methods.

(2) As for the success of reproducing the exact results from the original paper, it was not possible to achieve the same level of convergence for each method seen in the paper. This can be attributed to our lack of knowledge surrounding initialization and parameter tuning for the given dataset. It should also be mentioned that the standard deviation, the shading surrounding the methods, is much more extreme than the original paper. The potential explanation for this could be the same as the convergence. Of the methods, it can be observed that the SVRG method does not perform correctly. We are unsure why this method is behaving weirdly.

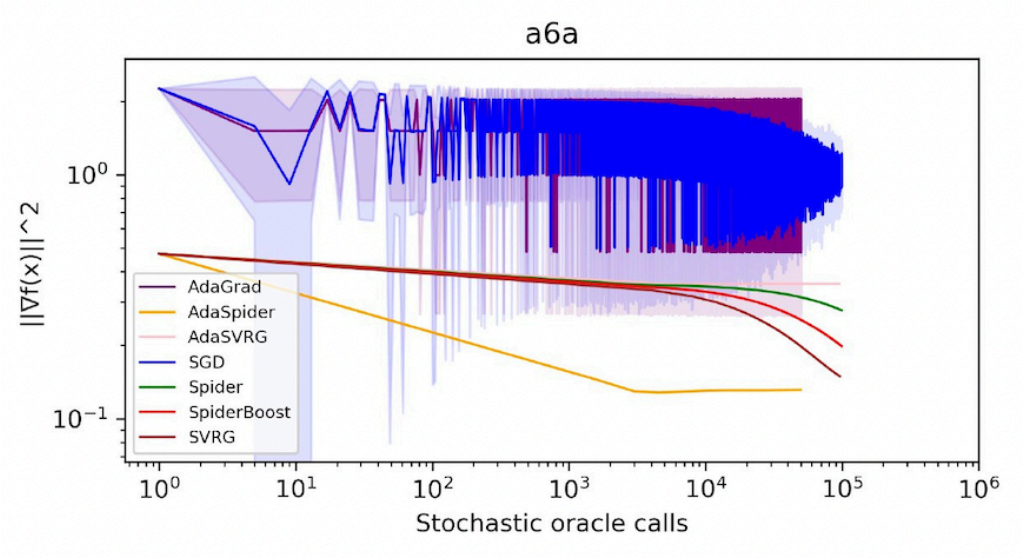


Figure 6. Methods results for the a1a dataset

w5a – (1) For the w5a dataset, claim 1 is supported by observing that SGD and AdaGrad behave, on average, worse than the variance reduction methods. Claim 2 can also be verified by observing that Spider converges, on average, worse than the other variance reduction methods, excluding SVRG. However, claim 3 cannot be accurately verified due to the odd behavior AdaSpider displays. It can be observed that after around 10^4 iterations, the full gradient norm increases. A potential explanation for this anomaly is the parameters were not tuned properly for this dataset.

(2) As for the success of reproducing the exact results from the original paper, it was not possible to achieve the same level of convergence for each method seen in the paper. This can be attributed to our lack of knowledge surrounding initialization and parameter tuning for the given dataset. It should also be mentioned that the standard deviation, the shading surrounding the methods, is much more extreme than the original paper. The potential explanation for this could be the same as the convergence. Of the methods, it can be observed that the SVRG method does not perform correctly. We are unsure why this method is behaving weirdly.

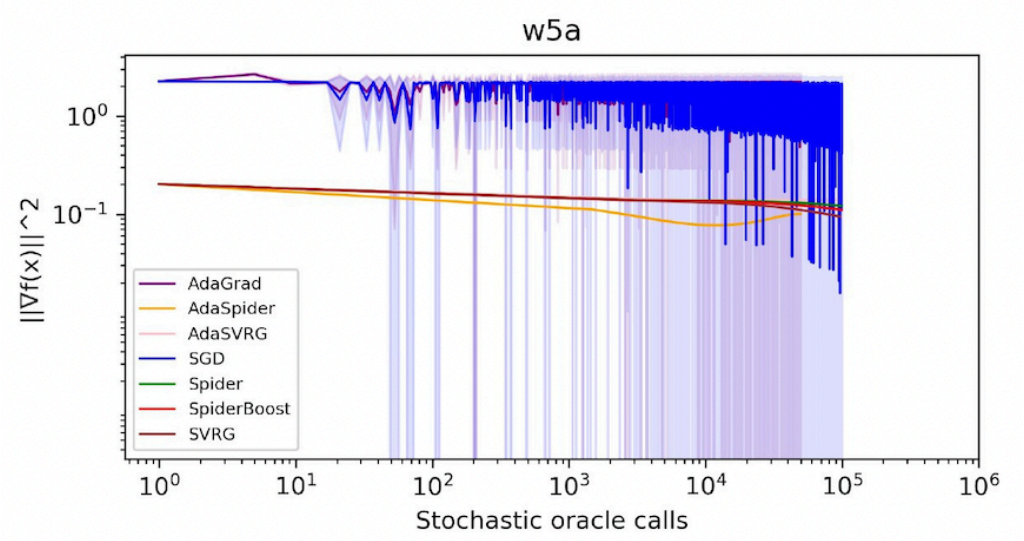


Figure 7. Methods results for the a1a dataset

ionosphere-scale — (1) For the ionosphere-scale dataset, claim 1 is supported by observing that SGD and AdaGrad behave, on average, worse than the variance reduction methods. Claim 2 can also be verified by observing that Spider converges, on average, worse than the other variance reduction methods, excluding SVRG. Claim 3 can also be verified by observing that AdaSpider converges, on average, faster than the other variance reduction methods.

(2) As for the success of reproducing the exact results from the original paper, it was not possible to achieve the same level of convergence for each method seen in the paper. This can be attributed to our lack of knowledge surrounding initialization and parameter tuning for the given dataset. It should also be mentioned that the standard deviation, the shading surrounding the methods, is much more extreme than the original paper. The potential explanation for this could be the same as the convergence. Of the methods, it can be observed that the SVRG method does not perform correctly. We are unsure why this method is behaving weirdly.

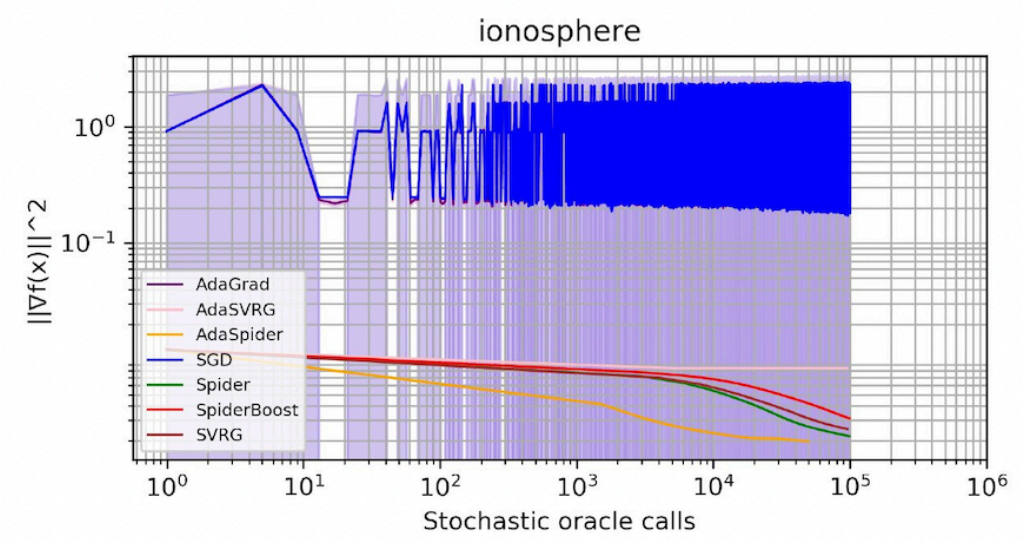


Figure 8. Methods results for the a1a dataset

Results for Neural Nets part — For the neural net part, we've managed to implement an optimizer that was able to learn but not efficiently. It was due to the problem with previous weight saving. In TensorFlow, the model's variables are complex objects combined by references with the model and learning process. When we were saving variables and using them as previous weights in the next optimizer iteration, either weights were the same (due to the same reference), or the model updating rules were collapsing (due to the change in references).

Implemented SpiderBoost optimizer was learning, but it was using the same weights. The results from learning are presented in figure 10. It was tested on the MNIST dataset for the neural net architecture shown in figure 9. The source code is available in the Colab notebook at this link.

```
# Build the model
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

Figure 9. Neural Network architecture for custom optimizer

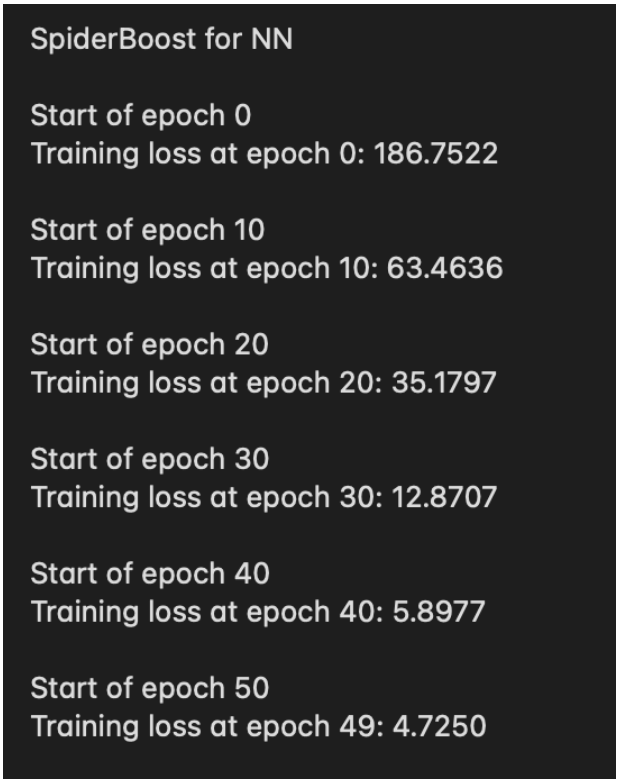


Figure 10. Neural Network SpiderBoost optimizer learning

5 Discussion

Overall, we believe our reproduction results reflect the claims that were presented in the original paper. In most cases, it was shown that AdaSpider performed on par, if not better, than the other variance reduction methods.

However, there are some apparent differences between the results that were reproduced and those of the original paper. For instance, the main difference was the convergences of the methods, and as explained above, this can be attributed to our lack of parameter and initialization knowledge. Another issue that played a large role was the time frame for reproducing the paper's results (i.e. neural network section and convex setting). All in all, even with the setbacks, we believe the results successfully show the claims presented in the original paper.

5.1 What was easy

The parts of the paper that were easy to reproduce were the logistic loss function with a non-convex regularization term, the optimization algorithms that are compared to the AdaSpider algorithm, and the plotting of the gradient norms.

The Logistic loss function is well-studied. As a result, we already had a working implementation of it at our disposal. As for the non-convex regularization, it was a relatively simple function to implement.

The optimization algorithms used in the paper were also easy to implement. Each algorithm had its own paper describing in detail what the algorithm did and the parameters

used. With this, it was rather simple to adjust the algorithms to fit the setting we were applying them in.

The plotting of the performance of the algorithms posed some issues, but overall it was easy to implement the framework. The norms of the full gradients squared were plotted on the y-axis using a log scale, and stochastic oracle calls were plotted on the x-axis. The issue that we faced when plotting initially was due to the fact that it was not clearly specified that only the full gradients were taken into account when computing the gradient norms. However, it was an easy mistake to fix.

5.2 What was difficult

We faced three glaring roadblocks that made reproducing the results of the paper challenging. These being the lack of knowledge of parameters and initialization needed to produce exact reproductions of the results, the computational power needed to produce the reproductions of the plots, and the inefficiency of Tensorflow for implementing novel optimizers.

First, not having access to the code for the first experiment made it challenging to define the parameters and initialization for the optimization methods, which all had their own set of unique parameters. For example, in the SVRG methods, we did not have the q parameter to determine when the full gradients are computed. This is the reason why our results do not entirely reflect those produced in the original paper. To alleviate this, we had to estimate the parameters such that they produced results as close to the original paper's results.

Second, we lacked the computational power to produce the plots quickly as a group. This is mainly due to the fact that we needed to train the logistic model on 10^5 iterations for each of the seven algorithms. Having no access to any computing clusters and being limited to only 16 CPU cores put us at a massive disadvantage. Making our testing phase and parameter tuning take an unreasonable amount of time to compute.

Lastly, our initial goal was to reproduce all of the original paper's results, including the neural network section. To do this, we used Tensorflow as our framework before gaining access to the code. Unbeknownst to us, Tensorflow was not an optimal framework for implementing novel optimizers. This is because it is hard for Tensorflow to record previous gradients at each iteration, which some of the algorithms require. It was then recommended that we try the Jax framework to implement the neural networks. However, due to time constraints, we decided to focus our efforts entirely on reproducing the results for experiment one.

5.3 Communication with original authors

We only had a meeting with the Laboratory for Information and Inference Systems members that authored the original paper. The goal of this meeting was to clear up some uncertainties surrounding the plots and for advice regarding the neural network portion of the paper. This is where we found that the authors only recorded the full gradients and that they suggest using the Jax framework, as discussed above. We also attempted to contact the authors again regarding the code used to create the results for experiment one; however, to no avail.

References

1. A. Kavis, E. P. SKOULAKIS, K. Antonakopoulos, L. T. Dadi, and V. Cevher. "Adaptive Stochastic Variance Reduction for Non-convex Finite-Sum Minimization." In: **Advances in Neural Information Processing Systems**. Ed. by A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho. 2022.
2. C. Fang, C. J. Li, Z. Lin, and T. Zhang. **SPIDER: Near-Optimal Non-Convex Optimization via Stochastic Path Integrated Differential Estimator**. 2018.
3. Z. Wang, K. Ji, Y. Zhou, Y. Liang, and V. Tarokh. **SpiderBoost and Momentum: Faster Stochastic Variance Reduction Algorithms**. 2018.
4. J. Bi and S. R. Gunn. **A Variance Controlled Stochastic Method with Biased Estimation for Faster Non-convex Optimization**. 2021.
5. B. Dubois-Taine, S. Vaswani, R. Babanezhad, M. Schmidt, and S. Lacoste-Julien. **SVRG Meets AdaGrad: Painless Variance Reduction**. 2021.