An investigation on Gradient Descent, Stochastic Gradient Descent and Stochastic Average Gradient Accelerated Method

Nguyen Hoang Nhat Tan[1*], Truong Tran Nhat Huy[1*]

[1]Fulbright University Vietnam
{tan.nguyen.210129, huy.truong.210287}@student.fulbright.edu.vn

Abstract

In this report, we evaluate the performance of different optimization algorithms including gradient descent (GD), stochastic gradient descent (SGD) and stochastic average gradient accelerated method (SAGA) on a simple toy dataset (Tiep, 2017) and Arabic Handwritten Digits Dataset (Loey et al., 2017). We compare those algorithms on different aspects including: variance of gradient, convergence and loss graphs. We also have a part on the effects of different batch size on the variance of gradient for SGD. The results came out that for the toy dataset, SGD converge the fastest then SAGA then SGD. The lowest loss values do not differ too much for all 3 algorithms. There is a quite significant reduction in variance of gradient when using SAGA comparing to SGD, however, the reduction becomes less significant as we proceed with the training iterations. For the Handwritten Dataset, we observe that SAGA has more stable training process, in terms of gradient variance and smaller gap between training accuracy and testing accuracy. However, in our specific settings, due to the simplicity of problem formulations and datasets, we observe that SAGA does not have significant advantages in term of convergence rate and running time. We suggest a direction for investigation that for different problem settings with more complicated datasets, SAGA may align with other theoretical results. The code for this paper can be find in this link: GitHub

1 Introduction

In the quest for efficient optimization method in machine learning, various algorithms have been developed to navigate the complex landscape of loss functions. Among these, Gradient Descent (GD) stands as a foundational method, guiding the iterative process towards the minima with steepest descent direction at each step. However, the advent of large-scale data processing necessitated more sophisticated approaches, as the cost of computation for gradients in GD is too large, leading to the emergence of Stochastic Gradient Descent (SGD) and its enhanced variant, the Stochastic Average Gradient Accelerated (SAGA) method.

1.1 Gradient Descent (GD)

Gradient Descent is an iterative optimization algorithm for finding the minimum of a function f. The update rule for GD is given by:

$$\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$$

where θ_t is the parameter vector at iteration t, η is the learning rate, and $\nabla f(\theta_t)$ is the gradient of the function f at θ_t.

1.2 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is a variant of GD that performs a parameter update for each training example. The update rule for SGD is given by:

$$\theta_{t+1} = \theta_t - \eta \nabla f_i(\theta_t)$$

where f_i is the function corresponding to the i-th observation.

1.3 Stochastic Average Gradient Accelerated (SAGA)

SAGA (Defazio et al., 2014) is a variant of SGD that includes a term for the average of the gradients of the loss function. The update rule for SAGA is given by:

$$\theta_{t+1} = \theta_t - \eta \left(\nabla f_i(\theta_t) - g_i + \frac{1}{n} \sum_{j=1}^{n} g_j \right)$$

where $\nabla f_i(\theta_t)$ is the gradient at the i-th data sample at iteration t, g_k is the last observed gradient of the k-th data sample, and n is the total number of observations.

[*] Equal contribution, random order.

These algorithms are fundamental to machine learning and have different trade-offs in terms of speed, accuracy, and suitability for different problem sizes. Notably, SAGA requires $O(NP)$ memory complexity where N is the number of data sample and P is the number of parameters. Further investigation into their behavior under various conditions can lead to more efficient and effective models.

Please note that these are simplified explanations and the actual algorithms may involve more parameters and considerations. Moreover, this is a simplified version of SAGA that is widely used and implemented. The original version should also contain a 'proximal' function in the update process, but for the scope of this project, we will just use the simplified version.

1.4 Variance reduction

The so called method SAGA is widely known as a variance reduction method. Recall SGD, when we use only 1 observation to compute the gradient, this introduce randomness in the process. If we repeat the process, we might select another observation that is different. People usually measure the difference of the observations when we repeat the process using variance, more specifically, in our case, it is the variance of the gradient vector.

Here come the question: How do we actually compute the variance of a vector-valued random variable? A standard way is the covariance matrix, however, matrices do not have specific ordering. So how do we know that the variance are being 'reduced'? This is a tricky questions and there are different sources that have different answers for that question. Nevertheless, for simplicity, in our experiments, we calculate the variance elementwise and take the mean (average).

2 Mathematical formulation

In order to perform classification task on any specific dataset, a common approach is to use a neural network and an optimization algorithm to find an optimal set of weights (parameters) of the neural network that fit the dataset well. We will use cross entropy as our loss objective function for classification task and try to minimize the loss on the dataset by optimizing the parameters.

Let call the parameters of the network θ, we have to find a θ^* by solving the optimization problem as follows:

$$\theta^* = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, f_\theta\left(\mathbf{x}_i\right)\right)$$

where:

- n is the total number of datapoint in the training dataset (the dataset that we use to optimize the parameters of the network)
- y_i is the label of the i^{th} data point
- $\mathbf{x}_i$ is the feature of the i^{th} data point.

Furthermore, L is the cross entropy loss function and f is the neural network with the parameters θ which receive features of inputs and produce outputs. In the setting of classification task, the outputs usually can be interpreted as the probability of samples belong to different classes.

3 Datasets

We use 2 differents datasets in our experiments:

1. A simple toy dataset
2. Arabic Handwritten Digits Dataset

3.1 Toy dataset

We took a simple toy dataset that is not linearly separable with 3 classes.

The datapoint is in 2-d, each datapoint has 2 features (features are in the range of $(-1, 1)$). Each class has 200 datapoints results in a total of 600 samples.

It is generated based on the properties of cosine and sine function with some noises added in. The more specific function used to generate the data can be found in the notebook file.

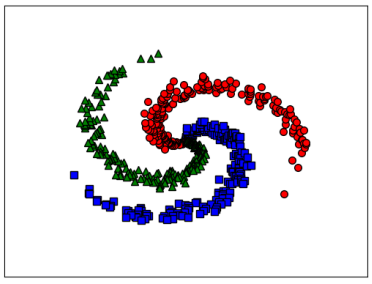

Figure 1: Toy dataset visualization (Tiep, 2017)

3.2 Arabic Handwritten Digits Dataset

Although, the full dataset is 60k gray images, due to limited resource, we only randomly sample 5k. Each gray image has 1 channel, which is the gray scale intensity of image pixels, the size of the image is 28x28.

There are total 10 classes which are Arabic digits. Below is a sample from the dataset.

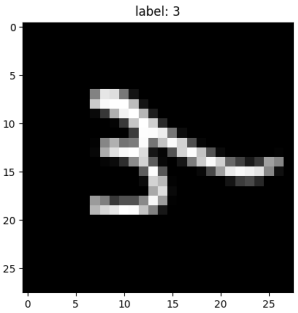

Figure 2: Arabic Handwritten Digits Dataset (El-Sawy et al., 2017)

4 Methodology

4.1 Neural network setup

When experimenting with different datasets, we use different neural network architectures:

1. For toy dataset: a small and simple fully connected neural network is used

2. For Arabic Handwritten Digits Dataset: a small convolutional neural network is used

The three algorithms will be used to optimize the parameters of the network will be GD, SGD and SAGA.

4.1.1 Fully connected (FC) neural network

The FC architecture is as follow:

1. An input layer of size 2 (dimension of feature)

2. A hidden layer with 100 hidden units using ReLU activation

3. An output layer of size 3 (number of classes) using softmax classification function.

$\rightarrow$ Total of $(2 \times 100) + (100 \times 3) = 500$ parameters

4.1.2 Convolutional neural network (CNN)

The CNN architecture is as follow:

- The model is a Convolutional Neural Network (CNN), which is effective for image recognition and classification tasks.

- It consists of two main parts: feature extraction (front-end) and classification (back-end).

- The feature extraction part includes two sequences of layers: convolutional layer, batch normalization layer, ReLU activation function, and max pooling layer.
  - The first sequence starts with a convolutional layer with 16 filters, followed by batch normalization, ReLU activation, and max pooling.
  - The second sequence is similar but has 32 filters in the convolutional layer.

- The classification part is a fully connected layer that outputs a 1x10 vector, each element representing the probability of the corresponding class of the input image.

- The weights of the convolutional and fully connected layers are initialized using the Kaiming normal initialization method, and the biases are initialized to zero.

- The model has a total of 29,034 parameters, making it a lightweight model that can be trained efficiently.

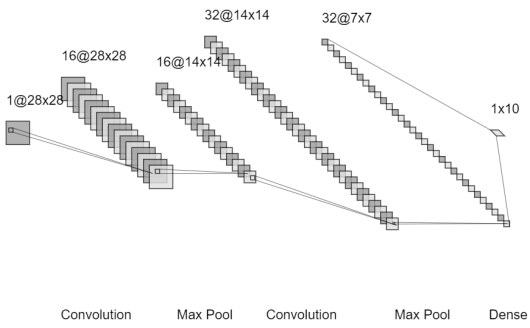

Figure 3: Visualization of CNN architecture

4.2 Type of experiments

Due to time and resource limitation, we conducted different experiments on different datasets:

1. Toy dataset: Mainly on SGD variance - SAGA variance, SGD batch size - variance.

2. Handwritten Digits: GD - SGD - SAGA: loss/time, accuracy/time; SGD variance - SAGA variance.

5 Results and discussion

5.1 Toy dataset

5.1.1 Convergence and loss graphs

We trained the FC network for 100k update iterations:

Optimizer	Lowest loss value	Runtime (s)
GD	0.02969	145
SGD	0.054315	68
SAGA	0.04067	109

Table 1: Lowest loss value and total runtime (in second) of GD, SGD and SAGA when training FC on toy dataset

.

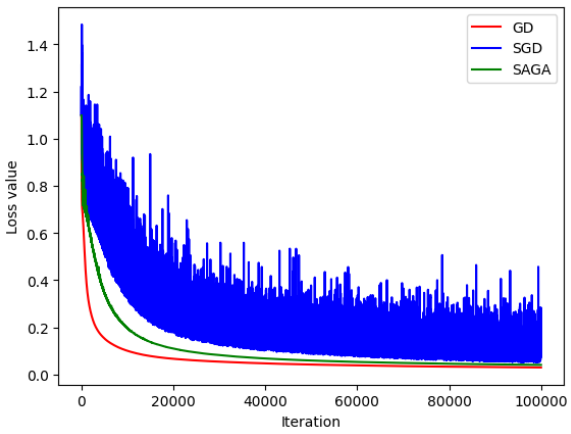

Figure 4: Loss graphs of GD, SGD and SAGA when training FC on toy dataset

We can see that

5.1.2 SGD variance - SAGA variance

We have to be really careful with this experiment as if we evaluate variance right at the initial point of the network (0 iteration), then it is almost zero due to initialization scheme of SAGA algorithm. So it is reasonable to evaluate from iteration 1 on.

The results can be seen in figure 7, for more specific numbers, please see table 2.

The first plot is not very good for visualization as the variance seems to get bigger as we keep on training which cause the two lines to be very close together at beginning iterations.

So we created the second plot in which we take the ratio of SGD variance over SAGA variance and plot it with respect to iteration. If the line is always above the line $y = 1$ then it is implied that in our setting, the variance of SGD is larger than of SAGA consistently. However, this time, we have another problem, the ratio seems to be quite good at the beginning but decreases as we train the network. And the scale differs a lot and change so quickly which creates difficulties in visualization.

Finally, we decided to take the natural logarithm of the ratio and it comes out to be quite nice. We can see that the line is always above the line $y = 0$ which implies that the variance of SAGA is smaller than the variance of SGD throughout iterations. On the other hand, we can also observe that the gap decreases meaning that the variance of SAGA and SGD get closer together as we keep on training. Our hypothesis that this behaviour is caused by some numerical instability which will be discussed in the next part (5.1.3).

By looking at the table 2, we can perform a quick double check and also confirm our the implication made above by using the plots.

	SGD	SAGA
1 iter	3.11E-06	1.67E-09
5 iter	3.20E-06	2.76E-08
20 iter	3.92E-06	4.21E-07
50 iter	1.04E-05	2.36E-06
100 iter	3.07E-05	5.96E-06
1000 iter	5.09E-03	1.98E-03

Table 2: Variance of SGD and SAGA with respect to training iterations

5.1.3 SGD batch size - variance

We have also made an extra experiment on the effect of batch size on the variance of SGD. The motivation behind this actually come from the slides of a lecture at University of British Columbia (Schmidt, 2019) when I was finding papers on variance reduction. The lecture mentioned that we can see mini-batch gradient descent as a method of variance reduction.

To me, an easy way to think about it is just similar to thinking about the random sampling process and the central limit theorem. When we take more sample, the sampling distribution has lower variance comparing to when we only take very few samples. And we think it is a good idea to also include this part in our experiment as it is also related

to variance reduction. The result of this part is in figure 6 and table 3.

As we can see from the graph, as we increase batch size from 1 upto the size of whole dataset, the variance of mini-batch gradient descent decreases. The shape of the decrease is similar to the graph $y = 1/x$ (see figure 5).

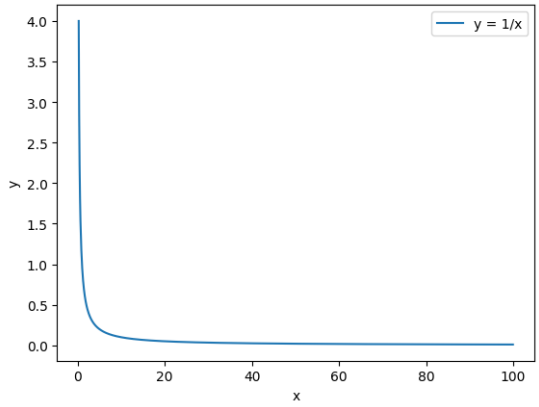

Figure 5: The graph of function $y = 1/x$

This implies that in our setting and problem, the batch size is directly inversely proportional to the variance. We can see this in another angle by using table 6. At any iteration, the variance decreases as we increase the batch size. Furthermore, the pattern can be easily spotted if we look at the batch size of 1, 10, 100 (highlighted in red). The variance decrease approximately at the same rate and very much close to the function $y = 1/x$.

Moreover, if we follow the pattern horizontally (as we keep on training), the variance seems to get bigger. However, there is a very strange thing happened when we take the whole dataset (if use sampling method here to take the batch then remember to set sample without replacement) and evaluate the variance, then the pattern we have mention earlier right above also occurs and the rate of (variance) increasing is nearly the same.

In fact, we expect the variance should be very small and consistent when we take the whole dataset for evaluation as we consider it is the true gradient (direction) which makes the variance to be 0.

We currently have no idea what cause this strange situation, but we have a hypothesis that may explain the case: It is due to the numerical error (floating point precision) in the calculation process. We come to this hypothesis after observing that due to the network parameters initialization scheme and the scale of the feature of datapoint in toy dataset is quite small which leads to the gradient for parameters is at very small scale (most of the time 1E-3, 1E-4). This can lead to small numerical error if we perform many algebra operators addition, multiplication,... many times. And the error will stack up as we keep on training the network which leads to the strange situation above. However, this is just a vague guessing, hypothesis and the question still there for us to have more exploration and readings.

5.2 Arabic Handwritten Digits Dataset

5.2.1 Training Results

The training results for our CNN model on Arabic Handwritten Digits dataset with GD, SGD, and SAGA can be found in table 4. We observe high accuracies for GD, SGD, SAGA with testing accuracies are 0.9492, 0.9568, 0.9541 respectively. Moreover, during the experiment we consistently observed that SAGA has smaller gap between train accuracy and test accuracy, comparing to GD and SGD.

5.2.2 Loss and training accuracy over time

The loss graph over time for GD, SGD, and SAGA is demonstrated in figure 8. We observe that GD has very stable convergence, while SGD and SAGA has lots of fluctuations due to their stochastic nature. We also observe that GD is the slowest to reduce loss values. This is understandable because GD needs to iterate over the entire dataset to perform one update. SAGA is slightly slower than SGD due to more computations.

We also analyzed the accuracy improvement trend for GD, SGD, and SAGA by looking at the time needed for each optimizer to reach milestones of accuracies. The graph is demonstrated in figure 9. We observed that GD is approximately 5 times slower to achieve 0.9 accuracy, in comparison to SGD and SAGA. There is no significant difference in the improvement trend between SGD and SAGA.

5.2.3 Variance Reduction Effect

By plotting the graph of loss values over iterations between SGD and SAGA (Figure 10), we can see that SAGA offers a more stable training process with smaller level of fluctuations. To verify our observation, we compute the change in variance between SGD and SAGA (Figure 11). We observe that for both SGD and SAGA, the gradient's variance decreases as the training process goes on.

Moreover, by using SAGA, we receive a consistently lower value of gradient's variance.

6 Conclusion and future work

To sum up, for our setting and scope of problem, for toy dataset, we do not see any significant difference between SGD and SAGA, except for the slower running time of SAGA. We also verified that there is variance reduction but there are still strange behaviours we still can not be able to explain clearly. However, due to the small set of toy data and simple pattern, we may not see the advantages of SAGA comparing to other two methods; moreover, the implementation of SAGA used for toy dataset can be further optimized to run faster. In more difficult problem and more complicated settings with satisfied assumptions of SAGA (as in the original paper analysis (Defazio et al., 2014)), we might see how SAGA can be more useful.

In practice, SAGA is less commonly used due to inefficient memory usage and the assumptions are hard to be satisfied. Furthermore, when working with AI in general, numeric stability and floating point precision is very important in order to guarantee that the process will run smoothly.

References

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

Ahmed El-Sawy, Hazem EL-Bakry, and Mohamed Loey. 2017. Cnn for handwritten arabic digits recognition based on lenet-5. In *Proceedings of the International Conference on Advanced Intelligent Systems and Informatics 2016*, pages 566–575, Cham. Springer International Publishing.

Loey, Mohamed, Ahmed El-Sawy, and Hazem El-Bakry. 2017. Deep learning autoencoder approach for handwritten arabic digits recognition.

Mark Schmidt. 2019. Sgd convergence rate.

Vu Huu Tiep. 2017. Bài 14: Multi-layer perceptron và backpropagation.

Appendices

A Tables and plots appendix

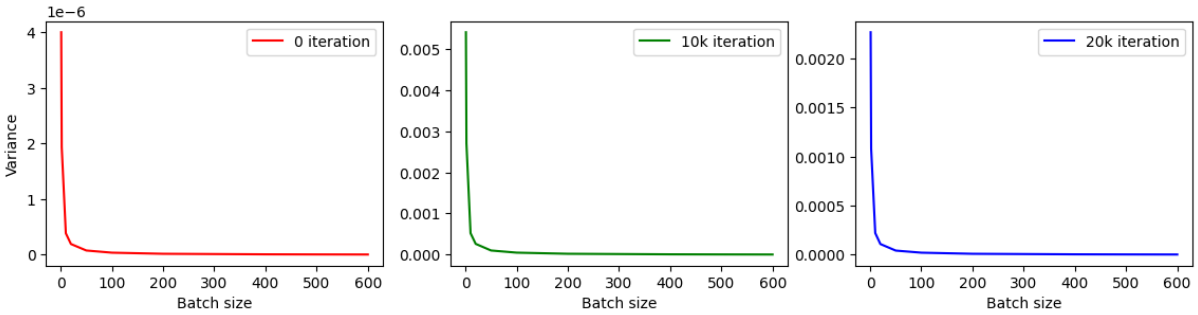

Figure 6: Variance with respect to batch size at different training iteration

Batch Size	Variance0	Variance10k	Variance20k
1	4.00E-06	5.42E-03	2.27E-03
2	1.93E-06	2.71E-03	1.08E-03
10	3.82E-07	5.20E-04	2.19E-04
50	7.22E-08	9.82E-05	4.01E-05
100	3.29E-08	4.54E-05	1.87E-05
600 (All)	2.97E-38	4.14E-36	1.69E-36

Table 3: Variance at different iterations with respect to different batch size

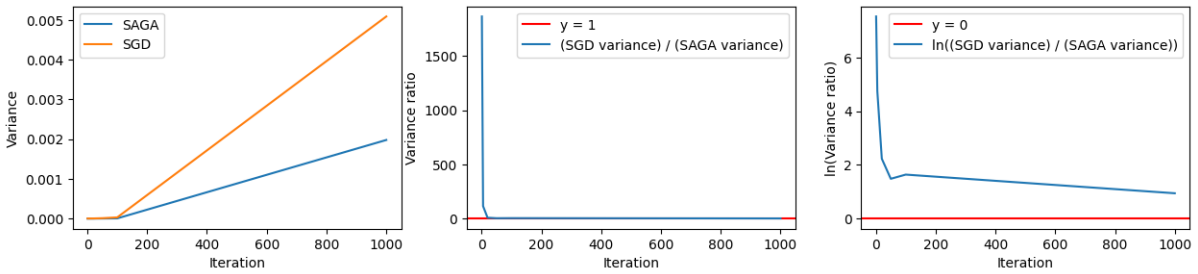

Figure 7: Results of SGD - SAGA variance comparison. From left to right: The first is the variance of two optimizers as with respect to training iteration, the second is the ratio of SGD variance over SAGA variance through training iterations and the last one is similar to the second one but we take natural logarithm of the ratio

Name	learning rate	epochs	train accuracy	test accuracy
GD	1e-2	100	0.962	0.9492
SGD	1e-3	2	0.9716	0.9568
SAGA	1e-3	2	0.9676	0.9541

Table 4: Training results of our CNN model on Arabic Handwritten Digits dataset

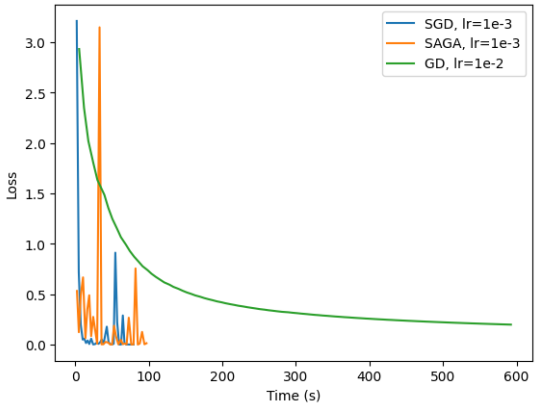

Figure 8: The loss graph over time of GD, SGD and SAGA

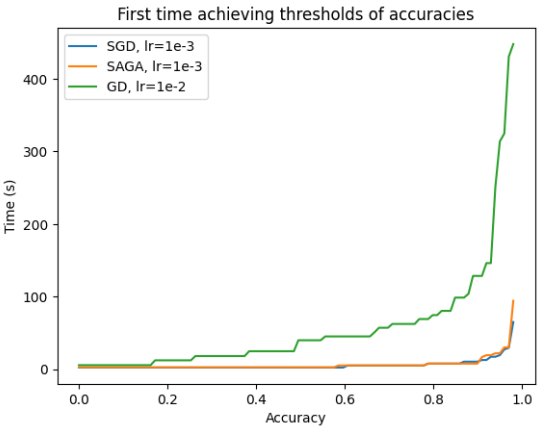

Figure 9: The accuracy improvement trend for GD, SGD, SAGA

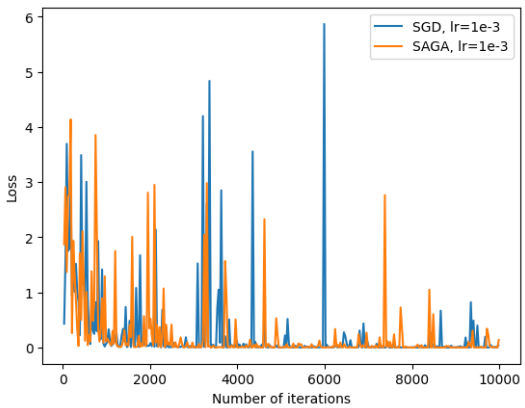

Figure 10: Loss graph between SGD and SAGA

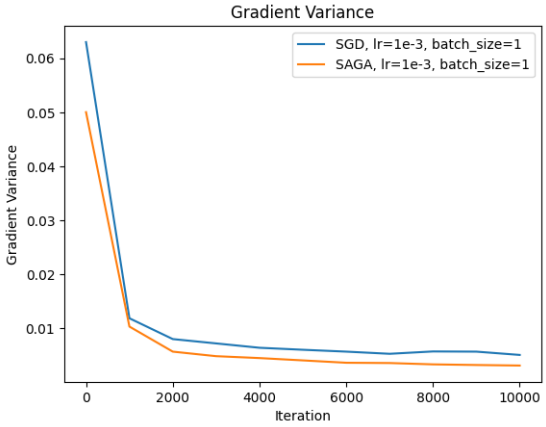

Figure 11: The change in gradient variance between SGD and SAGA