CS 169 Group Project

Lookahead Optimizer

Chenhao Li 46157018
Xu Chen 67237167
Yuexin Chen 88831640

Abstract

Micheal Zhang from University of Toronto discovered a new method of optimization, called Lookahead. This optimization method can utilize any optimization method as the backbone. In this project, we did a comparison test on this optimization method on the top layer, with other widely used optimization methods, such as Adam, SGD and Gradient Descent to verify whether the Lookahead optimization method can improve convergence as Micheal stated in his paper.

Introduction

This new method uses two sets of weights - a fast weight that is generated by the backbone optimization method, and a slow weight whose value depends on how the fast weight changes. For each step of the Lookahead optimizer, there will be k steps done by the backbone optimizer. The backbone optimizer can be any standard optimization method as stated in the paper. Then, the slow weight will change by a proportion of the change of fast weight before and after the k steps of the backbone optimizer. Figure 1 shows the pseudocode of the Lookahead optimizer. Micheal concluded in his paper that the Lookahead optimization method can improve learning stability, and lower the variance of its inner optimizer to enhance the convergence. Moreover, this new method can use any standard

Algorithm 1 Lookahead Optimizer:

Require: Initial parameters ϕ_0, objective function L
Require: Synchronization period k, slow weights step size α, optimizer A
for $t = 1, 2, \ldots$ **do**
  Synchronize parameters $\theta_{t,0} \leftarrow \phi_{t-1}$
  for $i = 1, 2, \ldots, k$ **do**
    sample minibatch of data $d \sim \mathcal{D}$
    $\theta_{t,i} \leftarrow \theta_{t,i-1} + A(L, \theta_{t,i-1}, d)$
  end for
  Perform outer update $\phi_t \leftarrow \phi_{t-1} + \alpha(\theta_{t,k} - \phi_{t-1})$
end for
return parameters ϕ

Figure 1. Lookahead Optimizer pseudocode

optimization methods as stated in the paper. We tested it by comparing Lookahead optimizer with some widely using optimizer, such as Adam, SGD, momentum. We tested its convergence by both finding the optimal point of functions and running it on neural networks. We did a comparison based on fixed steps, the total time to find the optimal point, and the number of function calls with the same level of accuracy. We also tested it on neural network by using CIFA-10 as our dataset and ResNet18 as base network, and compared the loss by using different optimization methods. From the result, we did not see any significant improvement by using the Lookahead optimization method in general.

State of Arts

The idea of the Lookahead optimization method is that the slow weight can reach to a point that is closer to the optimal point with the updates of fast weight as shown in Figure 2. The author Micheal compared his optimization method with SGD, Adam, and Polyak by using ResNet50 and ResNet152 based on CIFAR-10, CIFAR-100 and ImageNet dataset. One of the training loss results is shown as Figure 3. As we can see from the graph, the loss of Lookahead decreases much faster than any other method on the CIFAR-10 dataset. And its loss is much more stable than other optimization methods. In the end, Lookahead optimization method also yields the best

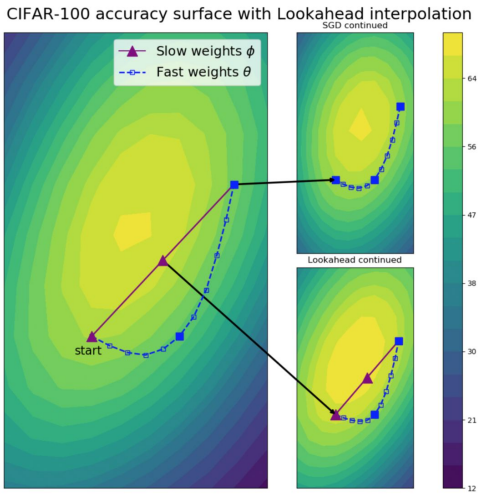


Figure 2.

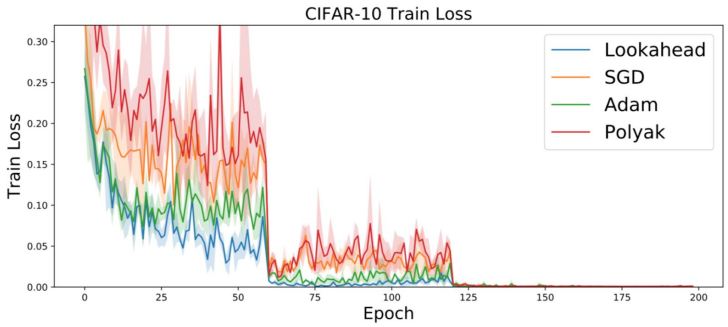


Figure 3.

validation accuracy among all the optimization methods that Micheal used. In the meantime, Micheal changed the parameters of optimizers, such as the learning rate of each optimization to test on a broader result of the comparison, and all of them gave positive outcomes.

Discussion/Method

We first tested the Lookahead method on Rosenbrock's banana function and flower function by comparing it with Momentum optimizer, Adam, gradient descent and conjugate gradient descent. We compared them by running fixed 2000 steps and see how the x changes. We tracked the convergent path to test whether Lookahead gives a better convergence. This time we only used two variables so that we could easily visualize the path for us to analyze. And we believed it is easier for us to test on simple function optimization first. The slow weight step size we used this time was 0.5 and the number of updates of fast weight was 5.

Then, we did an optimizer methods comparison among Momentum, Adam, Gradient Descent, Conjugate Gradient Descent and Augmented Lagrange by comparing absolute value of error (compared with the exact known x values where f(x) is the global optimum), time taken during computation and number of function calls when reaching the same level of accuracy. All of the comparisons are made after running the Lookahead optimizer on top of the testing optimizer method to ensure the standard baseline for comparison. The objective function we used here was Rosenbrock's banana function.

Finally, we thought it was also necessary for us to do some tests on neural networks, which were much more complex and it is what the original paper did. It was essential for us to compare the results computed by us using neural networks with that of the original paper. Since

we wanted to test on optimization but not neural network, and based on the conclusion of the original paper, it should be a generalized phenomenon that Lookahead has better convergence. Then, we chose ResNet18 as our base neural network model, and used CIFAR-10 dataset with normalization. The training data was loaded as batches that each batch contains 4 images and label information. The loss function we chose was the cross-entropy function. We compared the Lookahead with Adam and SGD of learning rate 0.001 and momentum 0.9 by using PyTorch. The source code of Lookahead implemented on PyTorch is from the author[1]. The slow weight step size we used here was 0.8, and the number of updates of fast weight was set as 5. We ran a total 30 epochs and recorded the mean loss of every 2000 batches of images.

Result

The result of path of x changes is obtained and recorded in Appendix A for Rosenbrock's function and flower function. For both Momentum and Adam, we can see that the Lookahead optimizer does not improve the convergence. The paths of x by applying the Lookahead optimization method highly overlap with the one without the Lookahead. However, for both the gradient descent method, the original optimization methods do not yield a smooth path of convergence while the one with the Lookahead optimization method provides a better convergence that is much more smooth and quick. We also find that Adam and momentum does not go very close to the optimal point in flower with fixed 2000 steps, but with the Lookahead optimization method, they can easily reach the optimal point with the same steps.

[1] @michaelrzhang https://github.com/michaelrzhang/lookahead

The result of our optimizer methods comparison is obtained and recorded in Appendix B for Rosenbrock's banana function. The first three plots (top left, top right and bottom left) indicate the five optimizer's performance with the Lookahead optimizer wrapping on top as the standard base line. We can see that even though Momentum has the smallest absolute value of error, the computation time and the function calls that this method has taken are not worth the trade. The computation time spent by Momentum is almost 10 times bigger than that spent by Gradient Descent and Conjugate Gradient. On the other hand, Adam seemed to be a mediocre method of choice as neither the convergence accuracy nor the convergence efficiency stands out, as it ran slower than both Gradient Descent and Conjugate Gradient did, while its absolute value of error is the largest among the five tested optimizers. Similar to Appendix A, the last plot (bottom right) shows a side-by-side relative comparison of Gradient Descent, Conjugate Gradient and Augmented Lagrange optimizers, both with and without wrapping of the Lookahead optimizer on top. The results here generally agree with what we've discovered in Appendix A - the Lookahead optimizer did not significantly improve the convergence performance of the three optimizers being tested; in fact, it slowed down the computation time by about 0.25 seconds for Gradient Descent, Conjugate Gradient and Augmented Lagrange optimizers. The computation time for Augmented Lagrange method with the Lookahead optimizer appeared to be an exception in this case, as the computation time increased by almost 900% compared to the base method without the Lookahead optimizer.

The result we get from running ResNet18 is shown in Appendix C. We can see from the graph that the loss of both Adam and Stochastic Gradient Descent methods does not vary significantly from one implemented with the Lookahead optimizer. Two loss lines are very close

to each other. We did not see the difference by using the Lookahead optimizer as shown in the original paper. We also did not find any improvements of using the Lookahead optimizer for a long training session.

Conclusion

Based on what we get from implementing the Lookahead optimization method on simple optimization problem and neural network, we find that we have different outcomes with the original paper. First of all, we do not see that the improvement of convergence is a general situation, that is it does not mean that it is necessary to apply the Lookahead optimization method over any standard optimization methods. From the plots of path we can see that only when the base optimizer is gradient descent method, Lookahead can make the convergence better that fits to the idea of the original paper. Lookahead applying on Adam and Momentum does not make the path go towards the optimal point in a better way. We guess it is because Adam and Momentum already yields a smooth path to the optimal point while gradient descent method does not have a smooth path but jagged. Therefore, Lookahead can improve the convergence of gradient descent method. While Lookahead may slightly increase the computation time for gradient descent and conjugate gradient descent method, in trade of smaller absolute value of error and fewer function calls. The loss graph is totally different from the original paper. This is not what we expected. We may guess that this is may be because some steps of we running the neural network is different from what the author did. And the parameters we use may not be the same as those used by the original paper. However, based on the purpose of Lookahead, these should not be the problems that Lookahead should improve the convergence in whatever nets

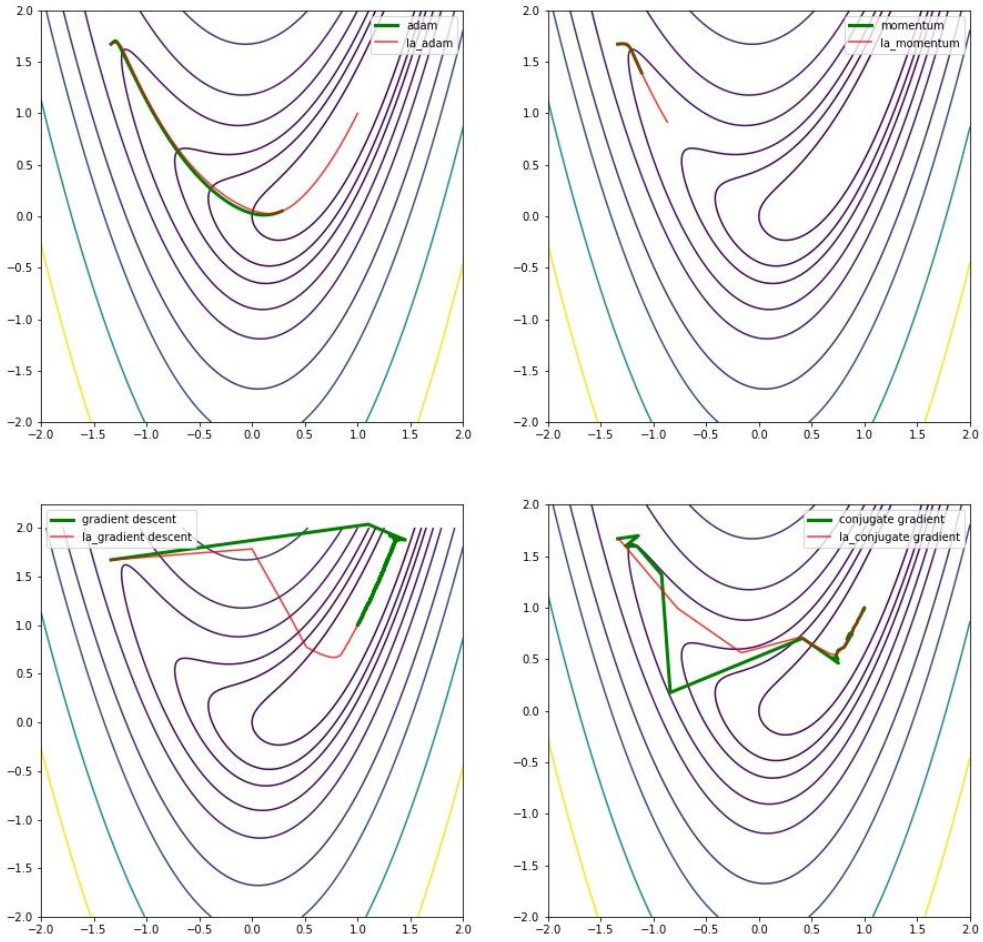
and using any parameters. Otherwise, we cannot say it can be used for any standard optimization method. And this is exactly what we get from our path tracking. Therefore, we guess that the Lookahead optimization method can somehow improve the convergence but for some specific backbone optimizers, parameters, nets if using neural networks and datasets. And there is one problem we find during the coding, the original paper does not states whether initialize the backbone optimizer for each step of the Lookahead or we continue using the states from the previous one. This can make a difference on running Lookahead, and it still needs to be tested out.

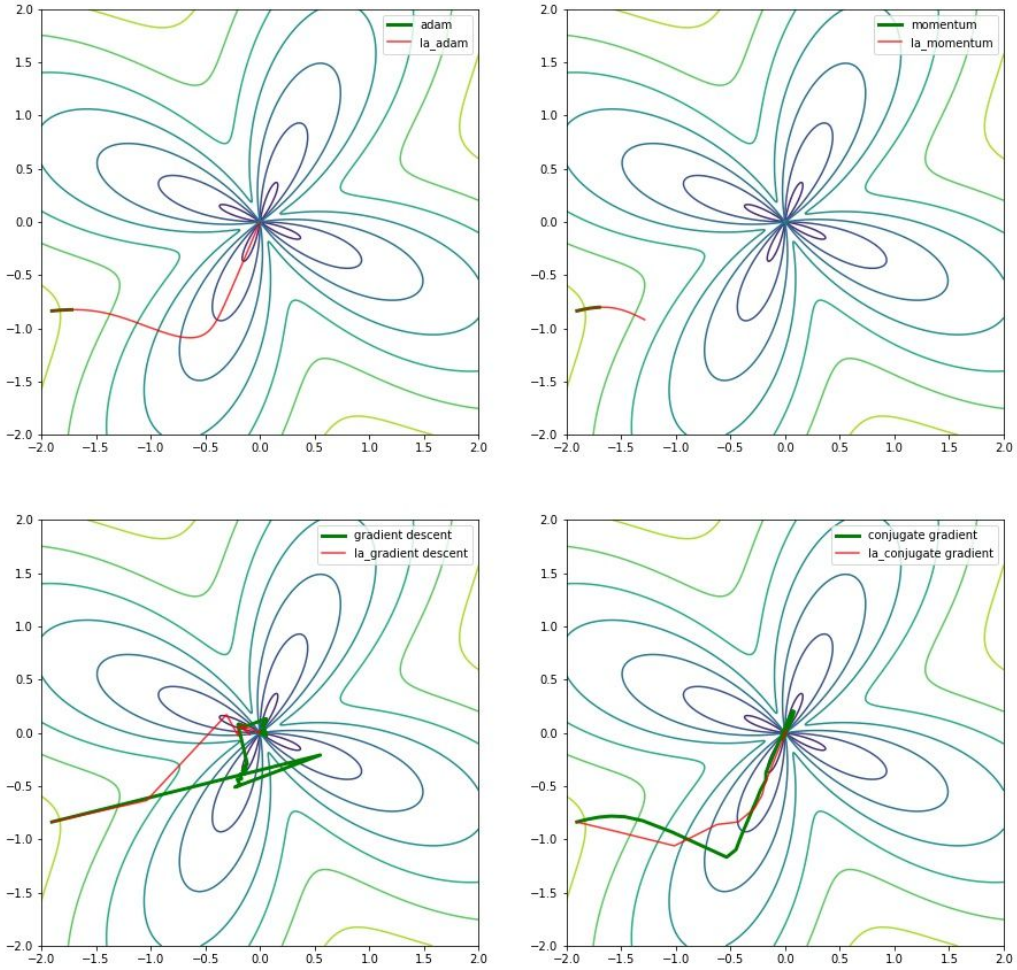
Bibliography

Zhang, Michael, et al. "Lookahead Optimizer: k steps forward, 1 step back." Advances in Neural Information Processing Systems. 2019.

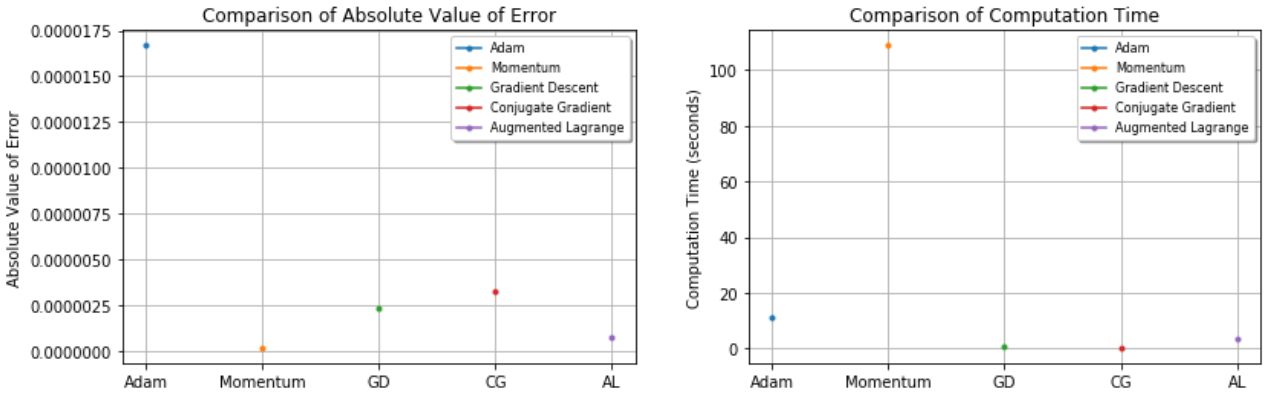
Appendix

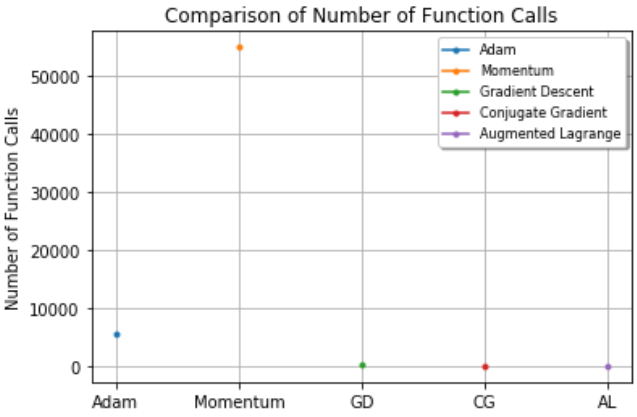
A. Path of convergence in Rosenbrock's function and flower function

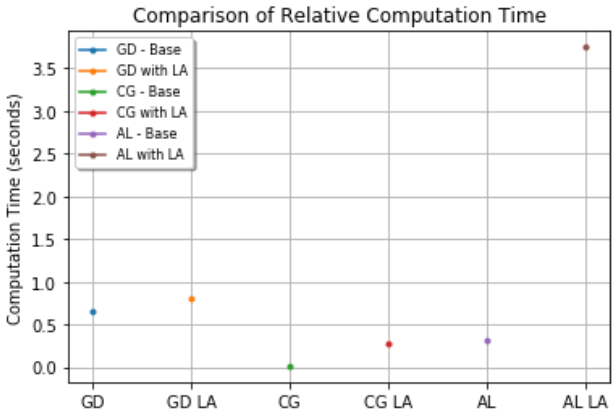




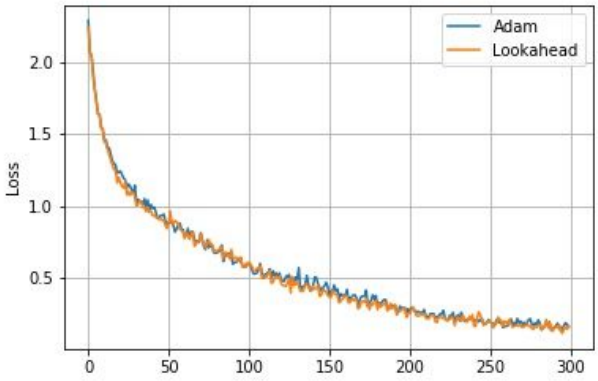
B. Optimizer Comparisons between Momentum, Adam, Gradient Descent, Conjugate Gradient Descent and Augmented Lagrange Methods

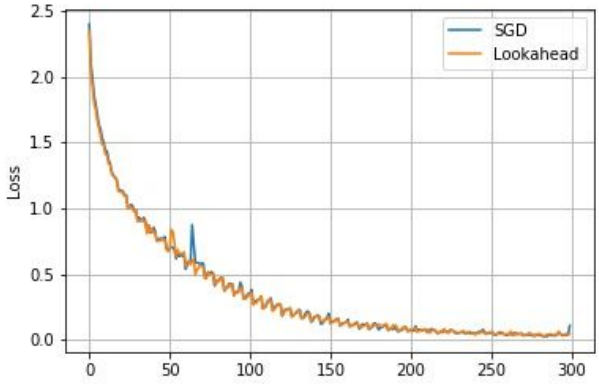






C. Loss of running on ResNet18 by using Adam and SGD, and Comparison with using Lookahead





D. Decomposition of the Project and Contribution of Group Member

a. Chenhao Li

Chenhao Li was responsible for writing the Lookahead optimization method on simple optimization problem, writing the Adam and Momentum, writing the neural networks scripts, running and recording neural network. Chenhao was also responsible for writing analytical experience of testing Lookahead optimizer on neural networks and generating “path of convergence” plots and “loss” plots.

b. Yuexin Chen

Yuexin was responsible for translating common optimizing methods (Golden Section Search, Gradient Descent, Conjugate Gradient Descent and Augmented Lagrange) written in Julia to Python and verifying code integrity with testing of objective functions. Yuexin was also responsible for writing data collection functions after evaluating the performance of class-based optimizers and plotting various data in Optimizer_Comparison.ipynb. Yuexin wrote the data analysis of comparing the common optimizers with and without the Lookahead optimizer.

c. Xu Chen

Xu Chen was responsible for writing out and debugging the optimization methods including gradient descent, conjugate descent, Augmented Lagrange and BFGS that is used for evaluation later on. He then refactor Yuexin's code from function based to class based which are included in the file optimizer.py. He standardized the code for each optimization method's class to suitable for the evaluate function used in Optimizer_Comparison.