Lookahead Optimizer: k steps forward, 1 step back

Rithwik KSV
IIT Gandhinagar
kukunuri.sai@iitgn.ac.in

Shivji Bhagat
IIT Gandhinagar
shivji.bhagat@iitgn.ac.in

Abstract

Neural networks have helped achieve state of the art results in various challenging tasks. However, training neural networks is a difficult affair and consumes a lot of time and memory. Optimizers constitute an important component of any neural network and an efficient optimization algorithm can significantly reduce the training time required to achieve sate of the art results. We replicate the novel lookahead optimizer as proposed by Zhang et al. and demonstrate its performance on two deep learning tasks, namely classification of Cifar10 and Cifar100.

1 Introduction

We present the replication of the novel lookahead optimizer as proposed by Zhang et al along with an the examination of the related baseline expriments.

1.1 Problem Statement

The problem statement for our reproducibility challenge as proposed by us is as follows:-
Reproducibility track - Replication of the work proposed by Zhang et al.
We aim to implement the novel optimizer as proposed in the paper and restrict to the following two experiments to evaluate the performance of the optimizer (considering the limited time and compute resources).

- Classification task for Cifar10 using ResNet18
- Classification task for Cifar100 using ResNet18

We implement all our works in keras framework as the authors have already provided codes in pytorch and tensorflow. We use SGD as the inner optimizer for Cifar 10 experiments and Adam for the Cifar 100. For the purpose of comparison, we also performed one of the baselines for each of the above two experiments i.e. same experiments using Adam optimizer.

NeurIPS 2019 Reproducibility Challenge

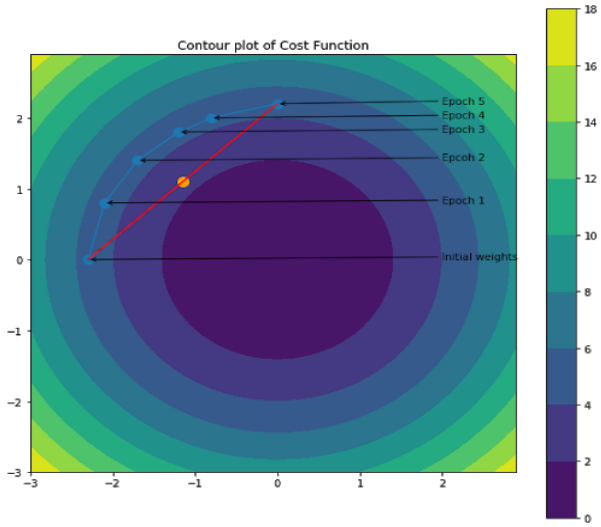

Figure 1: Visualisation of lookahead

1.2 Algorithm

The algorithm as provided in [1].

Algorithm 1: Lookahead Optimizer

L ← Loss function ;
A ← Inner Optimizer ;
Initialize weights ϕ_0, slow weights step size α ;
for i ←*1,2,3,...* **do**
  $\theta_{i,0}$=ϕ_{i-1};
  for j ← *1 to k* **do**
    Sample a mini-batch of data, d;
    $\theta_{i,j} \leftarrow \theta_{i,j-1}$ + A(L,$\theta_{i,j-1}, d$)
  end
  $\phi_i = \phi_{i-1} + \alpha(\theta_{i,k} - \phi_{i-1})$
end
return ϕ

The algorithm uses a standard optimizer such as Adam or SGD as its inner optimizer and improves upon it in terms convergence training time required. It maintains two sets of weights, the slow weights and the fast weights. For every k updates to the fast weights, the slow weights are updated once.

1.3 Intution

The neural network takes small steps in the direction of minimum cost. By taking many small such steps, the neural network converges to a minima. This can be thought of as a greedy algorithm, where we greedily go in the direction of minimum cost. The greedy approach might not be a good approach. The Lookahead optimizer is an extension over the greedy approach. Initially we start will a set of random weights. Call these weights slow weights. The neural network, takes 'k' greedy steps for minimizing the cost. The lookahead step, takes the slow weights and takes a small step in the direction of fast-weights (Figure 1). This can be thought as taking a decision(small step) after considering the result of 'k' epochs.

It starts with the point labelled "Initial weights". It then trains for five epochs and follows the path in blue. Generally, this is the situation that happens as the training progresses. The training loss

starts to converge, so the weights stay in a particular band of the contour plot. The slow-weights (initial weights) waits for "k" training epochs and then takes a small step in the final direction of fast-weights (epoch 5). Based on the step size then the initial weights change to some point on the red line. Hypothetically, say that the point end up being the orange point. The cost associated with this orange point is much lower than the cost during the 'k' epochs. Now, this orange point becomes the "slow-weights"(initial weights) and the process goes on.

2 Dataset

The Lookahead optimizer was evaluated on CIFAR10 and CIFAR 100
Cifar 10: This data set has 50,000 training images and 10,000 testing images. Each image is of shape (32,32,3). The data set consists of images of 10 different classes, where each class has 5000 training images and 1000 testing images
Cifar 100: This dataset has 50,000 training images and 10,000 testing images. Each image is of shape (32,32,3). The dataset consists of images of 100 different classes, where each class has 500 training images and 100 testing images

Preprocessing for Experiments:
The datasets are preprocessed such that mean is zero and the standard deviation is one. This preprocessing ensures better learning in neural networks.

3 Hyper Parameter search

For the hyper parameter search, we performed grid search over the parameters provided by the authors. Since we lacked more compute power, we made a simplifications on these experiments. These experiments were run for one seed, and each seed is trained for 40 epochs each. This method of hyper parameter search might be the reason for the differences in the results.

4 Experiments

The original paper, demonstrated the power of the optimizer on a variety of tasks that neural networks excel at such as language modelling and image classification. Due to lack of compute power and time, we restricted our experiments to dealing with image datasets Cifar 10 and Cifar 100. All the codes are available in the github repository at `https://github.com/shv07/lookahead-optimizer`.[1]

4.1 Experimental Setup and procedure

We implemented the lookahead optimizer in keras and performed all our experiments in keras framework in Google Cloud (8 CPUs and NVIDIA P4 GPU).

4.2 Cifar 10 Experiments

Experiments were conducted on the CIFAR-10 dataset with the experiment details provided in [1] and [2]. We use the official keras code of ResNet-18 [3] for this experiment. Considering the limited time and computation, we limit the evaluation to two random seeds. Since Adam is the current state of the art optimizer, we evaluated the performance of the new optimizer (with SGD as the inner optimizer) against Adam. Both the experiments are run for 200 epochs. The results are as shown in the Figure 2, 3 and Table 1.

4.3 Cifar 100 Experiments

Similarly, we run Cifar 100 on ResNet-18 architecture with the details provided in [1], [2]. In this experiment we implemented the Lookahead optimizer with Adam as internal optimizer. The model performance was really similar to the Adam model, since both of them had the same learning rates

[1] `The github repostory for the replication codes - https://github.com/shv07/lookahead-optimizer`

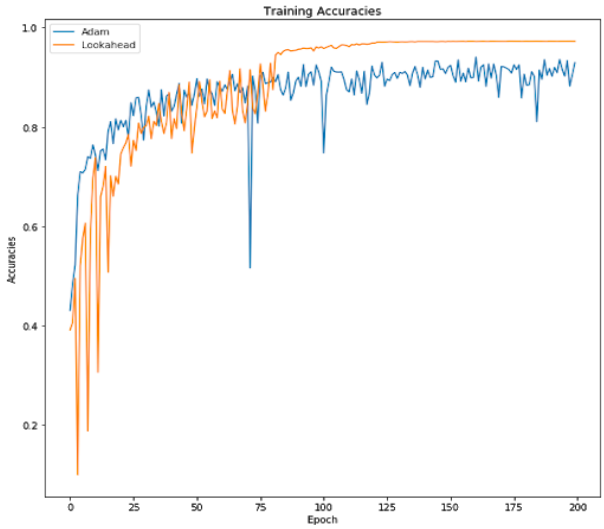

Figure 2: Comparing the performance of different optimizers on Cifar10 train set

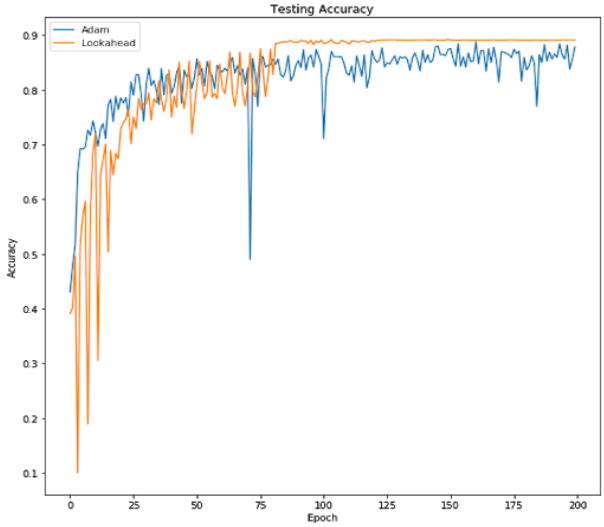

Figure 3: Comparing the performance of different optimizers on Cifar10 test set

Table 1: Performance of different optimizers on Cifar 10

Optimizer	Accuracy	No. of epochs
Adam	88.89 %	200
Lookahead with SGD	89.27 %	200

Table 2: Performance of different optimizers on Cifar 100

Optimizer	Accuracy	No. of epochs
Adam	67.32 %	200
Lookahead with Adam	67.98 %	200

and decays.

4.4 Discussion

In the plot, Fig. 3, where the mean testing accuracy over the two seeds plotted against the epoch number. In the initial phase, we can clearly see that the Adam model is doing a better job than the Lookahead model, this is trivial since during the initial phases the slow-weights update only once every k epochs. Slowly the lookahead model catches up and it surpasses the accuracy of the Adam model. Also, we can see the accuracy obtained by the Adam model is much more noisier, this is a property of the Adam optimizer. The learning rate depends on the gradient obtained. So, as time progresses the the gradients get smaller, and the optimizer starts to increase the learning rate to achieve significant learning. This change is learning rate is leading to overshooting and this is leading to the jumps in testing accuracies.
A similar plot can be observed with the training accuracy. The training accuracy of the lookahead model converge way before the adam model.

5 Conclusion

We can confirm that lookahead optimizer converges faster than a traditional optimizer and can reach a better optima. In order to increase the effectiveness of lookahead optimizer, one needs to carefully understand the effect of k and the step size between slow weights and fast weights. They inversely complement each other, there needs to be more studies to determine an optimal parameters. Also, for the experiments we concluded, lookahead optimizer can reach a better optima compared to the model which only uses the inner optimizer. This lookahead property can be extended to various other state-of-the-art optimizers and be used to further improve upon them. Building upon this idea, one can also experiment with recursive lookahead, where it is a series of lookaheads, to reach a better minima.
Some more recommendations for reproducibility:-

- Since the aim of the paper is to introduce a novel and efficient optimizer, for reproducibility purpose, a lighter architecture could be used to save time and resources.
- To verify the results more thoroughly, both the baseline experiments and the one with lookahead should be run for more different random seeds, so as to generalise better.

References

[1] Zhang, M. R., Lucas, J., Hinton, G., et al. 2019, arXiv e-prints, arXiv:1907.08610

[2] He, K., Zhang, X., Ren, S., et al. 2015, arXiv e-prints, arXiv:1512.03385

[3] `https://keras.io/examples/cifar10_resnet/`

[4] `https://codeocean.com/` - for compute resources

[5] `https://cloud.google.com/` - for compute resources