Lookahead Optimizer: k steps forward, 1 step back

Michael R. Zhang **James Lucas** **Geoffrey Hinton**

Jimmy Ba
Department of Computer Science, University of Toronto, Vector Institute
{michael, jlucas, hinton,jba}@cs.toronto.edu

Abstract

The vast majority of successful deep neural networks are trained using variants of stochastic gradient descent (SGD) algorithms. Recent attempts to improve SGD can be broadly categorized into two approaches: (1) adaptive learning rate schemes, such as AdaGrad and Adam, and (2) accelerated schemes, such as heavy-ball and Nesterov momentum. In this paper, we propose a new optimization algorithm, Lookahead, that is orthogonal to these previous approaches and iteratively updates two sets of weights. Intuitively, the algorithm chooses a search direction by *looking ahead* at the sequence of "fast weights" generated by another optimizer. We show that Lookahead improves the learning stability and lowers the variance of its inner optimizer with negligible computation and memory cost. We empirically demonstrate Lookahead can significantly improve the performance of SGD and Adam, even with their default hyperparameter settings on ImageNet, CIFAR-10/100, neural machine translation, and Penn Treebank.

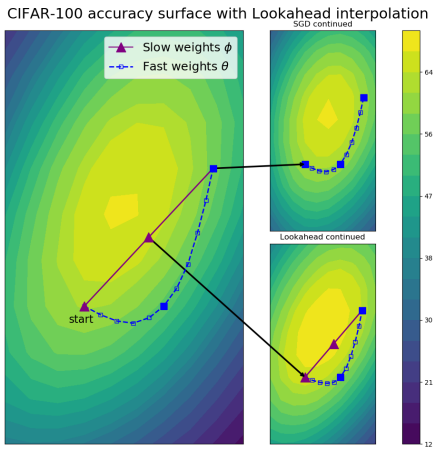
1 Introduction

Despite their simplicity, SGD-like algorithms remain competitive for neural network training against advanced second-order optimization methods. Large-scale distributed optimization algorithms [9, 44] have shown impressive performance in combination with improved learning rate scheduling schemes [41, 34], yet variants of SGD remain the core algorithm in the distributed systems. The recent improvements to SGD can be broadly categorized into two approaches: (1) adaptive learning rate schemes, such as AdaGrad [6] and Adam [17], and (2) accelerated schemes, such as Polyak heavy-ball [32] and Nesterov momentum [28]. Both approaches make use of the accumulated past gradient information to achieve faster convergence. However, to obtain their improved performance in neural networks often requires costly hyperparameter tuning [27].

In this work, we present Lookahead, a new optimization method, that is orthogonal to these previous approaches. Lookahead first updates the "fast weights" [11] k times using any standard optimizer in its inner loop before updating the "slow weights" once in the direction of the final fast weights. We show that this update reduces the variance. We find that Lookahead is less sensitive to suboptimal hyperparameters and therefore lessens the need for extensive hyperparameter tuning. By using Lookahead with inner optimizers such as SGD or Adam, we achieve faster convergence across different deep learning tasks with minimal computation overhead.

Empirically, we evaluate Lookahead by training classifiers on the CIFAR [18] and ImageNet datasets [5], observing faster convergence on the ResNet-50 and ResNet-152 architectures [10]. We also trained LSTM language models on the Penn Treebank dataset [23] and Transformer-based [41] neural machine translation models on the WMT 2014 English-to-German dataset. For all tasks, using Lookahead leads to improved convergence over the inner optimizer and often improved generalization performance while being robust to hyperparameter changes. Our experiments demonstrate that

Preprint. Under review.



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: (Left) Visualizing Lookahead through a ResNet-32 test accuracy surface at epoch 100 on CIFAR-100. We project the weights onto a plane defined by the first, middle, and last fast (inner-loop) weights. The fast weights are along the blue dashed path. All points that lie on the plane are represented as solid, including the entire Lookahead slow weights path (in purple). Lookahead (middle, bottom right) quickly progresses closer to the minima than SGD (middle, top right) is able to. (Right) Pseudocode for Lookahead.

Lookahead is robust to changes in the inner loop optimizer, the number of fast weight updates, and the slow weights learning rate.

2 Method

同步

In this section, we describe the Lookahead algorithm and discuss its properties. Lookahead maintains a set of slow weights ϕ and fast weights θ, which get synced with the fast weights every k updates. The fast weights are updated through applying A, any standard optimization algorithm, to batches of training examples sampled from the dataset $\mathcal{D}$. After k inner optimizer updates using A, the slow weights are updated towards the fast weights by linearly interpolating in weight space, $\theta - \phi$. We denote the slow weights learning rate as α. After each slow weights update, the fast weights are reset to the current slow weights value. Psuedocode is provided in Algorithm 1.

Standard optimization methods typically require carefully tuned learning rates to prevent oscillation and slow convergence. This is even more important in the stochastic setting [24, 42]. Lookahead, however, benefits from a larger learning rate in the inner loop. When oscillating in the high curvature direction, the fast weights updates make rapid progress along the low curvature direction. The slow weights help smooth out the oscillation through the parameter interpolation. The combination of fast weights and slow weights improves learning in high curvature directions, reduces variance, and enables Lookahead to converge rapidly in practice.

Figure 1 shows the trajectory of both the fast weights and slow weights during the optimization of a ResNet-32 model on CIFAR-100. While the fast weights explore around the minima, the slow weight update pushes Lookahead aggressively towards an area of improved test accuracy, a region which remains unexplored by SGD after 20 updates.

Slow weights trajectory We can characterize the trajectory of the slow weights as an exponential moving average (EMA) of the *final* fast weights within each inner-loop, regardless of the inner optimizer. After k inner-loop steps we have:

$$\phi_{t+1} = \phi_t + \alpha(\theta_{t,k} - \phi_t) \tag{1}$$

$$= \alpha[\theta_{t,k} + (1-\alpha)\theta_{t-1,k} + \ldots + (1-\alpha)^{t-1}\theta_{0,k}] + (1-\alpha)^t\phi_0 \tag{2}$$

Intuitively, the slow weights heavily utilize recent proposals from the fast weight optimization but maintain some influence from previous fast weights. We show that this has the effect of reducing variance in Section 3.1. While a Polyak-style average has further theoretical guarantees, our results match the claim that "an exponentially-decayed moving average typically works much better in practice" [24].

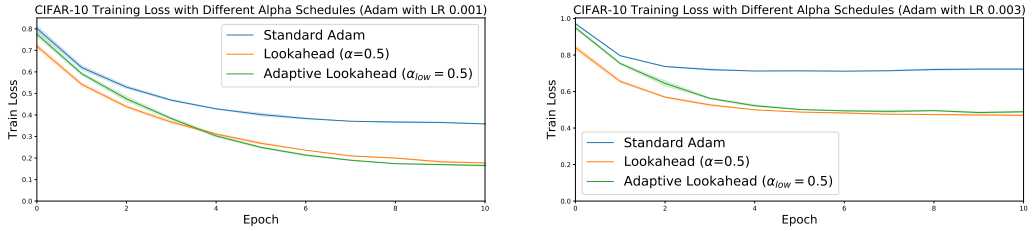


Figure 2: CIFAR-10 training loss with fixed and adaptive α. The adaptive α is clipped between $[\alpha_{low}, 1]$. (Left) Adam learning rate = 0.001. (Right) Adam learning rate = 0.003.

Fast weights trajectory Within each inner-loop, the trajectory of the fast weights depends on the choice of underlying optimizer. Given an optimization algorithm A that takes in an objective function L and the current mini-batch training examples d, we have the update rule for the fast weights:

$$\theta_{t,i+1} = \theta_{t,i} + A(L, \theta_{t,i-1}, d). \tag{3}$$

We have the choice of maintaining, interpolating, or resetting the internal state (e.g. momentum) of the inner optimizer. Every choice improves convergence of the inner optimizer. We describe this tradeoff on the CIFAR dataset in Appendix C.5 and maintain internal state for the other experiments.

Computational complexity Lookahead has a constant computational overhead due to parameter copying and basic arithmetic operations that is amortized across the k inner loop updates. The number of operations is $\mathcal{O}(\frac{k+1}{k})$ times that of the inner optimizer. Lookahead maintains a single additional copy of the number of learnable parameters in the model.

2.1 Selecting the Slow Weights Step Size

The step size in the direction $(\theta_{t,k} - \theta_{t,0})$ is controlled by α. By taking a quadratic approximation of the loss, we present a principled way of selecting α.

Proposition 1 (Optimal slow weights step size)**.** *For a quadratic loss function $L(x) = \frac{1}{2}x^T Ax - b^T x$, the step size α^* that minimizes the loss for two points $\theta_{t,0}$ and $\theta_{t,k}$ is given by:*

$$\alpha^* = \arg\min_{\alpha} L(\theta_{t,0} + \alpha(\theta_{t,k} - \theta_{t,0})) = \frac{(\theta_{t,0} - \theta^*)^T A(\theta_{t,0} - \theta_{t,k})}{(\theta_{t,0} - \theta_{t,k})^T A(\theta_{t,0} - \theta_{t,k})}$$

where $\theta^* = A^{-1}b$ minimizes the loss.

Proof is in the appendix. Using quadratic approximations for the curvature, which is typical in second order optimization [6, 17, 25], we can derive an estimate for the optimal α more generally. The full Hessian is typically intractable so we instead use aforementioned approximations, such as the diagonal approximate empirical Fisher used by the Adam optimizer [17]. This approximation works well in our numerical experiments if we clip the magnitude of the step size. At each slow weight update, we compute:

$$\hat{\alpha}^* = \text{clip}(\frac{(\theta_{t,0} - (\theta_{t,k} - \hat{A}^{-1}\nabla L(\theta_{t,k}))^T \hat{A}(\theta_{t,0} - \theta_{t,k})}{(\theta_{t,0} - \theta_{t,k})^T \hat{A}(\theta_{t,0} - \theta_{t,k})}, \alpha_{\text{low}}, 1)$$

Setting $\alpha_{\text{low}} > 0$ improves the stability of our algorithm. We evaluate the performance of this adaptive scheme versus a fixed scheme and standard Adam on a ResNet-18 trained on CIFAR-10 with two different learning rates and show the results in Figure 2. Additional hyperparameter details are given in appendix C. Both the fixed and adaptive Lookahead offer improved convergence.

In practice, a fixed choice of α offers similar convergence benefits and tends to generalize better. Fixing α avoids the need to maintain an estimate of the empirical Fisher, which incurs a memory and computational cost when the inner optimizer does not maintain such an estimate e.g. SGD. We thus use a fixed α for the rest of our deep learning experiments.

3 Convergence Analysis

3.1 Noisy quadratic analysis

We analyze Lookahead on the noisy quadratic model to shed light on its convergence guarantees. While simple, this model is a proxy for neural network optimization and effectively optimizing it remains a challenging open problem [36, 25, 42, 46].

Model definition We use the same model as in Schaul et al. [36] and Wu et al. [42].

$$\hat{\mathcal{L}}(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{c})^T \mathbf{A}(\mathbf{x} - \mathbf{c}), \tag{4}$$

with $\mathbf{c} \sim \mathcal{N}(\mathbf{x}^*, \Sigma)$. We assume that both $\mathbf{A}$ and Σ are diagonal and that $\mathbf{x}^* = 0$. [1] We use a_i and σ_i^2 to denote the diagonal elements of $\mathbf{A}$ and Σ respectively. Taking the expectation over $\mathbf{c}$, the expected loss of the iterates $\theta^{(t)}$ is,

$$\mathcal{L}(\theta^{(t)}) = \mathbb{E}[\hat{\mathcal{L}}(\theta^{(t)})] = \frac{1}{2}\mathbb{E}[\sum_i a_i {\theta_i^{(t)}}^2 + \sigma_i^2)] = \frac{1}{2}\sum_i a_i(\mathbb{E}[\theta_i^{(t)}]^2 + \mathbb{V}[\theta_i^{(t)}] + \sigma_i^2). \tag{5}$$

Analyzing the expected dynamics of the SGD iterates and the slow weights gives the following result.

Proposition 2 (Lookahead variance reduction)**.** *Let $0 < \gamma < 2/L$ be the learning rate of SGD and Lookahead where $L = \max_i a_i$. In the noisy quadratic model, the iterates of SGD and Lookahead with SGD as its inner optimizer converge to 0 in expectation and the variances converge to the following fixed points:*

$$V^*_{SGD} = \frac{\gamma^2 \mathbf{A}^2 \Sigma^2}{\mathbf{I} - (\mathbf{I} - \gamma\mathbf{A})^2} \tag{6}$$

$$V^*_{LA} = \frac{\alpha^2(\mathbf{I} - (\mathbf{I} - \gamma\mathbf{A})^{2k})}{\alpha^2(\mathbf{I} - (\mathbf{I} - \gamma\mathbf{A})^{2k}) + 2\alpha(1-\alpha)(\mathbf{I} - (\mathbf{I} - \gamma\mathbf{A})^k)} V^*_{SGD} \tag{7}$$

Remarks For the Lookahead variance fixed point, the first product term is always smaller than 1 for $\alpha \in (0, 1)$, and thus Lookahead has a variance fixed point that is strictly smaller than that of the SGD inner-loop optimizer. Evidence of this phenomenon is present in Figure 10.

In Proposition 2, we use the same learning rate. In order to compare the convergence of the two methods we should choose hyperparameters such that the variance fixed points are equal. In Figure 3 we show the expected loss after 1000 updates (computed analytically) for both Lookahead and SGD. At this stage in optimization (and onwards), Lookahead outperforms SGD across the broad spectrum of α values. Details and additional discussion are in Appendix B.

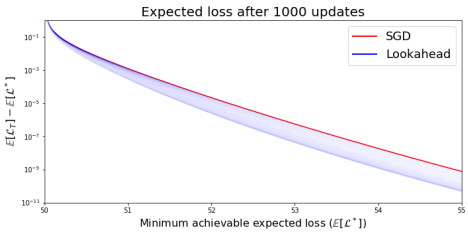


Figure 3: Comparing expected optimization progress between SGD and Lookahead($k = 5$) with a range of α values on the noisy quadratic model. Each vertical slice compares the convergence of optimizers with the same final loss values.

3.2 Deterministic quadratic convergence

In the previous section we showed that on the noisy quadratic model, Lookahead is able to reduce the variance of the SGD optimizer. Here we analyze the quadratic model without noise using gradient descent with momentum [32, 8] and show that when the system is under-damped, Lookahead is able to improve on the convergence rate.

As before, we restrict our attention to diagonal quadratic functions with minima at the origin. Given an initial point $\boldsymbol{\theta}_0$, we wish to measure the convergence rate of contraction $1.0 - ||\boldsymbol{\theta}_t||/||\boldsymbol{\theta}_{t-1}||$. We follow the approach of [30] and model the optimization of this function as a linear dynamical system. Details are in Appendix B.

[1] Classical momentum's iterates are invariant to translations and rotations (see e.g. Sutskever et al. [40]) and Lookahead's linear interpolation is also invariant to such changes. The assumption on noise is less trivial and we refer to Wu et al. [42] and Zhang et al. [46] for discussion.

As in Lucas et al. [22], to better understand the sensitivity of Lookahead to misspecified conditioning we fix the momentum coefficient of classical momentum and explore the convergence rate over varying condition number under the optimal learning rate. As expected, Lookahead has slightly worse convergence in the over-damped regime where momentum is set too low and CM is slowly, monotonically converging to the optimum. However, when the system is under-damped (and oscillations occur) Lookahead is able to significantly improve the convergence rate by skipping to a better parameter setting during oscillation.

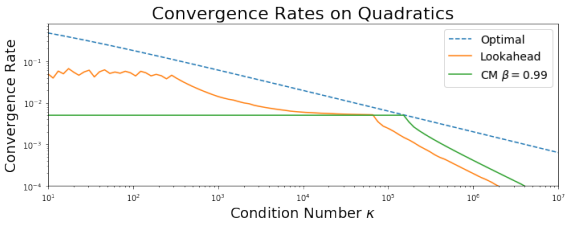


Figure 4: Quadratic convergence rates of classical momentum versus Lookahead wrapping classical momentum. For Lookahead, we fix $k = 20$ lookahead steps and $\alpha = 0.5$ for the slow weights step size. Lookahead is able to significantly improve on the convergence rate in the under-damped regime where oscillations are observed.

4 Related work

Our work is inspired by recent advances in understanding the loss surface of deep neural networks. While the idea of following the trajectory of weights dates back to Ruppert [35], Polyak and Juditsky [33], averaging weights in neural networks has not been carefully studied until more recently. Garipov et al. [7] observe that the final weights of two independently trained neural networks can be connected by a curve with low loss. Izmailov et al. [13] proposes Stochastic Weight Averaging (SWA), which averages the weights of different neural network obtained during training. Parameter averaging schemes are used to create ensembles in natural language processing tasks [14, 26] and in training Generative Adversarial Networks [43]. In contrast to previous approaches, which generally focus on generating a set of parameters at the *end* of training, Lookahead is an optimization algorithm which performs parameter averaging *during* the training procedure to achieve faster convergence.

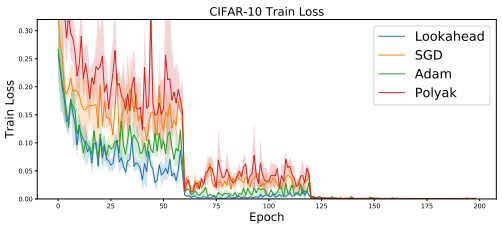
The Reptile algorithm, proposed by Nichol et al. [29], samples tasks in its outer loop and runs an optimization algorithm on each task within the inner loop. The initial weights are then updated in the direction of the new weights. While the functionality is similar, the application and setting are starkly different. Reptile samples different tasks and aims to find parameters which act as good initial values for new tasks sampled at test time. Lookahead does not sample new tasks for each outer loop and aims to take advantage of the geometry of loss surfaces to improve convergence.

Katyusha [1], an accelerated form of SVRG [16], also uses an outer and inner loop during optimization. Katyusha checkpoints parameters during optimization and within each inner loop step the parameters are pulled back towards the latest checkpoint. Lookahead computes the pullback only at the end of the inner loop and the gradient updates do not utilize the SVRG correction (though this would be possible). While Katyusha has theoretical guarantees in the convex optimization setting, the SVRG-based update does not work well for neural networks [4].

Anderson acceleration [2] and other related extrapolation techniques [3] have a similar flavor to Lookahead. These methods keep track of all iterates within an inner loop and then compute some linear combination which extrapolates the iterates towards their fixed point. This presents additional challenges first in the form of additional memory overhead as the number of inner-loop steps increases and also in finding the best linear combination. Scieur et al. [37, 38] propose a method by which to find a good linear combination and apply this approach to deep learning problems and report both improved convergence and generalization. However, their method requires on the order of k times more memory than Lookahead. Lookahead can be seen as a simple version of Anderson acceleration wherein only the first and last iterates are used.

5 Experiments

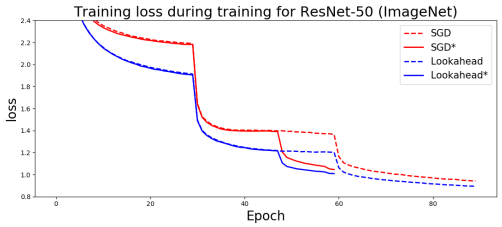
We completed a thorough evaluation of the Lookahead optimizer on a range of deep learning tasks against well-calibrated baselines. We explored image classification on CIFAR-10/CIFAR-100 [18] and ImageNet [5]. We also trained LSTM language models on the Penn Treebank dataset [23] and



OPTIMIZER	CIFAR-10	CIFAR-100
SGD	$95.23 \pm .19$	$78.24 \pm .18$
POLYAK	$95.26 \pm .04$	$77.99 \pm .42$
ADAM	$94.84 \pm .16$	$76.88 \pm .39$
LOOKAHEAD	$95.27 \pm .06$	$78.34 \pm .05$

Table 1: CIFAR Final Validation Accuracy.

Figure 5: Performance comparison of the different optimization algorithms. (**Left**) Train Loss on CIFAR-100. (**Right**) CIFAR ResNet-18 validation accuracies with various optimizers. We do a grid search over learning rate and weight decay on the other optimizers (details in appendix C). Lookahead and Polyak are wrapped around SGD.



OPTIMIZER	LA	SGD
EPOCH 50 - TOP 1	75.13	74.43
EPOCH 50 - TOP 5	92.22	92.15
EPOCH 60 - TOP 1	75.49	75.15
EPOCH 60 - TOP 5	92.53	92.56

Table 2: Top-1 and Top-5 single crop validation accuracies on ImageNet.

Figure 6: ImageNet training loss. The asterisk denotes the aggressive learning rate decay schedule, where LR is decayed at iteration 30, 48, and 58. We report validation accuracies for this schedule.

Transformer-based [41] neural machine translation models on the WMT 2014 English-to-German dataset. For all of our experiments, every algorithm consumed the same amount of training data.

5.1 CIFAR-10 and CIFAR-100

The CIFAR-10 and CIFAR-100 datasets for classification consist of 32×32 color images, with 10 and 100 different classes, split into a training set with 50,000 images and a test set with 10,000 images. We ran all our CIFAR experiments with 3 seeds and trained for 200 epochs on a ResNet-18 [10] with batches of 128 images and decay the learning rate by a factor of 5 at the 60th, 120th, and 160th epochs. Additional details are given in appendix C.

We summarize our results in Figure 5.[2] Note that Lookahead achieves significantly faster convergence throughout training even though the learning rate schedule is optimized for the inner optimizer—future work can involve building a learning rate schedule for Lookahead.

5.2 ImageNet

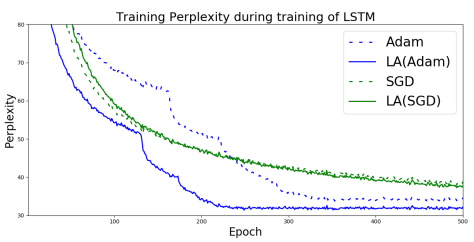
The 1000-way ImageNet task [5] is a classification task that contains roughly 1.28 million training images and 50,000 validation images. We use the official PyTorch implementation[3] and the ResNet-50 and ResNet-152 [10] architectures. Our baseline algorithm is SGD with an initial learning rate of 0.1 and momentum value of 0.9. We train for 90 epochs and decay our learning rate by a factor of 10 at the 30th and 60th epochs. For Lookahead, we set $k = 5$ and slow weights step size $\alpha = 0.5$.

Motivated by the improved convergence we observed in our initial experiment, we tried a more aggressive learning rate decay schedule where we decay the learning rate by a factor of 10 at the 30th, 48th, and 58th epochs. Using such a schedule, we reach 75% single crop top-1 accuracy on ImageNet in just 50 epochs and reach 75.5% top-1 accuracy in 60 epochs. The results are shown in Figure 6.

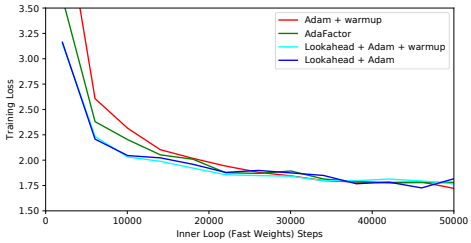
To test the scalability of our method, we ran Lookahead with the aggressive learning rate decay on ResNet-152. We reach 77% single crop top-1 accuracy in 49 epochs (matching what is reported in He et al. [10]) and 77.96% top-1 accuracy in 60 epochs. Other approaches for improving convergence on ImageNet can require hundreds of GPUs, or tricks such as ramping up the learning rate and adaptive

[2] We refer to SGD with heavy ball momentum [32] as SGD

[3] Implementation available at `https://github.com/pytorch/examples/tree/master/imagenet`.



(a) Training perplexity of LSTM models trained on the Penn Treebank dataset



(b) Training Loss on Transformer. Adam and AdaFactor both use a linear warmup scheme described in Vaswani et al. [41].

Figure 7: Optimization performance on Penn Treebank and WMT-14 machine translation task.

Table 3: LSTM training, validation, and test perplexity on the Penn Treebank dataset.

OPTIMIZER	TRAIN	VAL.	TEST
SGD	43.62	66.0	63.90
LA(SGD)	35.02	65.10	63.04
ADAM	33.54	61.64	59.33
LA(ADAM)	**31.92**	**60.28**	**57.72**
POLYAK	-	61.18	58.79

Table 4: Transformer Base Model trained for 50k steps on WMT English-to-German. "Adam-" denote Adam without learning rate warm-up.

OPTIMIZER	NEWSTEST13	NEWSTEST14
ADAM	24.6	24.6
LA(ADAM)	24.68	24.70
LA(ADAM-)	24.3	24.4
ADAFACTOR	24.17	24.51

batch-sizes [9, 15]. The fastest convergence we are aware of uses an approximate second-order method to train a ResNet-50 to 75% top-1 accuracy in 35 epochs with 1,024 GPUs [31]. In contrast, Lookahead requires changing one single line of code and can easily scale to ResNet-152.

5.3 Language modeling

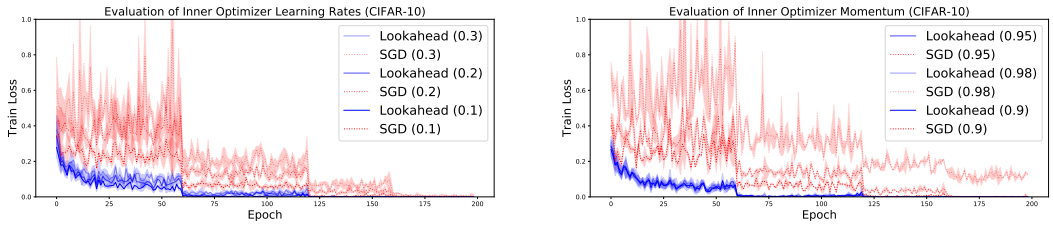
We trained LSTMs [12] for language modeling on the Penn Treebank dataset. We followed the model setup of Merity et al. [26] and made use of their publicly available code in our experiments. We did not include the fine-tuning stages. We searched over hyperparameters for both Adam and SGD (without momentum) to find the model which gave the best validation performance. We then performed an additional small grid search on each of these methods with Lookahead. Each model was trained for 750 epochs. We show training curves for each model in Figure 7a.

Using Lookahead with Adam we were able to achieve the fastest convergence and best training, validation, and test perplexity. The models trained with SGD took much longer to converge (around 700 epochs) and were unable to match the final performance of Adam. Using Polyak weight averaging [33] with SGD, as suggested by Merity et al. [26] and referred to as ASGD, we were able to improve on the performance of Adam but were unable to match the performance of Lookahead. Full results are given in Table 3 and additional details are in appendix C.

5.4 Neural machine translation

We trained Transformer based models [41] on the WMT2014 English-to-German translation task on a single Tensor Processing Unit (TPU) node. We took the base model from Vaswani et al. [41] and trained it using the proposed warmup-then-decay learning rate scheduling scheme and, additionally, the same scheme wrapped with Lookahead. We found Lookahead speedups the early stage of the training over Adam and the later proposed AdaFactor [39] optimizer. All the methods converge to similar training loss and BLEU score at the end, see Figure 7b and Table 4.

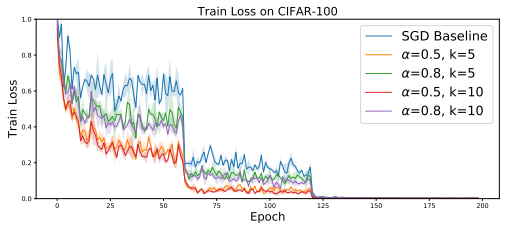
Our NMT experiments further confirms Lookahead improves the robustness of the inner loop optimizer. We found Lookahead enables a wider range of learning rate {0.02, 0.04, 0.06} choices for the Transformer model that all converge to similar final losses. Full details are given in Appendix C.4.



(a) CIFAR-10 Train Loss: Different LR

(b) CIFAR-10 Train Loss: Different momentum

Figure 8: We fix Lookahead parameters and evaluate on different inner optimizers.



K \ α	0.5	0.8
5	$78.24 \pm .02$	$78.27 \pm .04$
10	$78.19 \pm .22$	$77.94 \pm .22$

Table 5: All settings have higher validation accuracy than SGD (77.72%)

Figure 9: CIFAR-100 train loss and final test accuracy with various k and α.

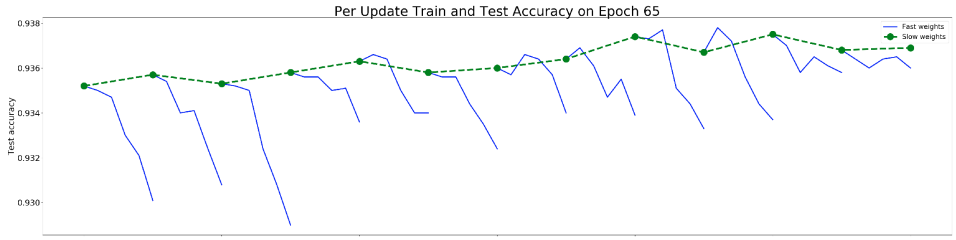


Figure 10: Visualizing Lookahead accuracy for 60 fast weight updates. We plot the test accuracy after every update (the training accuracy and loss behave similarly). The inner loop update tends to degrade both the training and test accuracy, while the interpolation recovers the original performance.

5.5 Empirical analysis

Robustness to inner optimization algorithm, k, and α We demonstrate empirically on the CIFAR dataset that Lookahead consistently delivers fast convergence across different hyperparameter settings. We fix slow weights step size $\alpha = 0.5$ and $k = 5$ and run Lookahead on inner SGD optimizers with different learning rates and momentum; results are shown in Figure 8. In general, we observe that Lookahead can train with higher learning rates on the base optimizer with little tuning on k and α. This agrees with our discussion of variance reduction in Section 3.1. We also evaluate robustness to the Lookahead hyperparameters by fixing the inner optimizer and evaluating runs with varying updates k and step size α; these results are shown in In Figure 9.

Inner loop and outer loop evaluation To get a better understanding of the Lookahead update, we also plotted the test accuracy for every update on epoch 65 in Figure 10. We found that within each inner loop the fast weights may lead to substantial degradation in task performance—this reflects our analysis of the higher variance of the inner loop update in section 3.1. The slow weights step recovers the outer loop variance and restores the test accuracy.

6 Conclusion

In this paper, we present Lookahead, an algorithm that can be combined with any standard optimization method. Our algorithm computes weight updates by *looking ahead* at the sequence of "fast weights" generated by another optimizer. We illustrate how Lookahead improves convergence by reducing variance and show strong empirical results on many deep learning benchmark datasets.

References

[1] Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1200–1205. ACM, 2017.

[2] Donald G Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4):547–560, 1965.

[3] Claude Brezinski and M Redivo Zaglia. *Extrapolation methods: theory and practice*, volume 2. Elsevier, 2013.

[4] Aaron Defazio and Léon Bottou. On the ineffectiveness of variance reduced optimization for deep learning, 2018.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[6] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[7] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. *arXiv preprint arXiv:1802.10026*, 2018.

[8] Gabriel Goh. Why momentum really works. *Distill*, 2017. doi: 10.23915/distill.00006. URL `http://distill.pub/2017/momentum`.

[9] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Geoffrey E Hinton and David C Plaut. Using fast weights to deblur old memories. 1987.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[13] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[14] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*, 2014.

[15] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

[16] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[19] Laurent Lessard, Benjamin Recht, and Andrew Packard. Analysis and design of optimization algorithms via integral quadratic constraints. *SIAM Journal on Optimization*, 26(1):57–95, 2016.

[20] Ren-cang Li. Sharpness in rates of convergence for cg and symmetric lanczos methods. Technical report, 2005.

[21] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 2017.

[22] James Lucas, Richard Zemel, and Roger Grosse. Aggregated momentum: Stability through passive damping. *arXiv preprint arXiv:1804.00325*, 2018.

[23] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[24] James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

[25] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[26] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

[27] Grégoire Montavon, Geneviève Orr, and Klaus-Robert Müller. *Neural networks: tricks of the trade*, volume 7700. springer, 2012.

[28] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

[29] Alex Nichol, Joshua Achiam, and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2018.

[30] Brendan O'Donoghue and Emmanuel Candes. Adaptive restart for accelerated gradient schemes. *Foundations of computational mathematics*, 15(3):715–732, 2015.

[31] Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. Second-order optimization method for large mini-batch: Training resnet-50 on imagenet in 35 epochs, 2018.

[32] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[33] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[34] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*, 2018.

[35] David Ruppert. Efficient estimations from a slowly convergent robbins-monro process. Technical report, Cornell University Operations Research and Industrial Engineering, 1988.

[36] Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *International Conference on Machine Learning*, pages 343–351, 2013.

[37] Damien Scieur, Edouard Oyallon, Alexandre d'Aspremont, and Francis Bach. Nonlinear acceleration of deep neural networks. *arXiv preprint arXiv:1805.09639*, 2018.

[38] Damien Scieur, Edouard Oyallon, Alexandre d'Aspremont, and Francis Bach. Nonlinear acceleration of cnns. *arXiv preprint arXiv:1806.00370*, 2018.

[39] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. *arXiv preprint arXiv:1804.04235*, 2018.

[40] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[42] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021*, 2018.

[43] Stefan Winkler Kim-Hui Yap Georgios Piliouras Vijay Chandrasekhar Yasin Yazıcı, Chuan-Sheng Foo. The unusual effectiveness of averaging in gan training. *arXiv preprint arXiv:1806.04498*, 2018.

[44] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*, page 1. ACM, 2018.

[45] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[46] Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George E Dahl, Christopher J Shallue, and Roger Grosse. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. *arXiv preprint arXiv:1907.04164*, 2019.

A Noisy quadratic analysis

Here we present the details of the noisy quadratic analysis, and the proof of Proposition 2.

Stochastic dynamics of SGD From Wu et al. [42], we can compute the dynamics of SGD with learning rate γ as follows:

$$\mathbb{E}[\mathbf{x}^{(t+1)}] = (\mathbf{I} - \gamma\mathbf{A})\,\mathbb{E}[\mathbf{x}^{(t)}] \tag{8}$$

$$\mathbb{V}[\mathbf{x}^{(t+1)}] = (\mathbf{I} - \gamma\mathbf{A})^2\,\mathbb{V}[\mathbf{x}^{(t)}] + \gamma^2\mathbf{A}^2\Sigma \tag{9}$$

Stochastic dynamics of Lookahead SGD We now compute the dynamics of the slow weights of Lookahead.

Lemma 1. *The Lookahead slow weights have the following trajectories:*

$$\mathbb{E}[\boldsymbol{\phi}_{t+1}] = [1 - \alpha + \alpha(\mathbf{I} - \gamma\mathbf{A})^k]\,\mathbb{E}[\boldsymbol{\phi}_t] \tag{10}$$

$$\mathbb{V}[\boldsymbol{\phi}_{t+1}] = [1 - \alpha + \alpha(\mathbf{I} - \gamma\mathbf{A})^k]^2\,\mathbb{V}[\boldsymbol{\phi}_t] + \alpha^2\sum_{i=0}^{k-1}(\mathbf{I} - \gamma\mathbf{A})^{2i}\gamma^2\mathbf{A}^2\Sigma \tag{11}$$

Proof. The expectation trajectory follows from SGD,

$$\begin{aligned}\mathbb{E}[\boldsymbol{\phi}_{t+1}] &= (1-\alpha)\,\mathbb{E}[\boldsymbol{\phi}_t] + \alpha\,\mathbb{E}[\boldsymbol{\theta}_{t,k}] \\ &= (1-\alpha)\,\mathbb{E}[\boldsymbol{\phi}_t] + \alpha(\mathbf{I} - \gamma\mathbf{A})^k\,\mathbb{E}[\boldsymbol{\phi}_t] \\ &= [1 - \alpha + \alpha(\mathbf{I} - \gamma\mathbf{A})^k]\,\mathbb{E}[\boldsymbol{\phi}_t]\end{aligned}$$

For the variance, we can write $\mathbb{V}[\boldsymbol{\phi}_{t+1}] = (1-\alpha)^2\,\mathbb{V}[\boldsymbol{\phi}_t] + \alpha^2\,\mathbb{V}[\boldsymbol{\theta}_{t,k}] + 2\alpha(1-\alpha)\text{cov}(\boldsymbol{\phi}_t, \boldsymbol{\theta}_{t,k})$. We proceed by computing the covariance term recursively. For simplicity, we work with a single element, θ, of the vector $\boldsymbol{\theta}$ (as $\mathbf{A}$ is diagonal, each element evolves independently).

$$\begin{aligned}\text{cov}(\theta_{t,k-1}, \theta_{t,k}) &= \mathbb{E}[(\theta_{t,k-1} - \mathbb{E}[\theta_{t,k-1}])(\theta_{t,k} - \mathbb{E}[\theta_{t,k}])] \\ &= \mathbb{E}[(\theta_{t,k-1} - \mathbb{E}[\theta_{t,k-1}])(\theta_{t,k} - (1-\gamma a)\,\mathbb{E}[\theta_{t,k-1}])] \\ &= \mathbb{E}[\theta_{t,k-1}\theta_{t,k}] - (1-\gamma a)\,\mathbb{E}[\theta_{t,k-1}]^2 \\ &= \mathbb{E}[(1-\gamma a)\theta_{t,k-1}^2] - (1-\gamma a)\,\mathbb{E}[\theta_{t,k-1}]^2 \\ &= (1-\gamma a)\,\mathbb{V}[\theta_{t,k-1}]\end{aligned}$$

A similar derivation yields $\text{cov}(\boldsymbol{\phi}_t, \boldsymbol{\theta}_{t,k}) = (\mathbf{I} - \gamma\mathbf{A})^k\,\mathbb{V}[\phi_t]$. After substituting the SGD variance formula and some rearranging we have,

$$\mathbb{V}[\boldsymbol{\phi}_{t+1}] = [1 - \alpha + \alpha(\mathbf{I} - \gamma\mathbf{A})^k]^2\,\mathbb{V}[\boldsymbol{\phi}_t] + \alpha^2\sum_{i=0}^{k-1}(\mathbf{I} - \gamma\mathbf{A})^{2i}\gamma^2\mathbf{A}^2\Sigma$$

□

We now proceed with the proof of Proposition 2.

Proof. First note that if the learning rate is chosen as specified, then each of the trajectories is a contraction map. By Banach's fixed point theorem, they each have a unique fixed point. Clearly the expectation trajectories contract to zero in each case.

For the variance we can solve for the fixed points directly. For SGD,

$$V^*_{SGD} = (1-\gamma\mathbf{A})^2 V^*_{SGD} + \gamma\mathbf{A}^2\Sigma,$$
$$\Rightarrow V^*_{SGD} = \frac{\gamma^2\mathbf{A}^2\Sigma}{\mathbf{I}-(\mathbf{I}-\gamma\mathbf{A})^2}.$$

For Lookahead, we have,

$$V^*_{LA} = [1-\alpha+\alpha(\mathbf{I}-\gamma\mathbf{A})^k]^2 V^*_{LA} + \alpha^2 \sum_{i=0}^{k-1} (\mathbf{I}-\gamma\mathbf{A})^{2i}\gamma^2\mathbf{A}^2\Sigma$$
$$\Rightarrow V^*_{LA} = \frac{\alpha^2 \sum_{i=0}^{k-1}(\mathbf{I}-\gamma\mathbf{A})^{2i}}{\mathbf{I}-[(1-\alpha)\mathbf{I}+\alpha(\mathbf{I}-\gamma\mathbf{A})^k]^2}\gamma^2\mathbf{A}^2\Sigma$$
$$V^*_{LA} = \frac{\alpha^2(\mathbf{I}-(\mathbf{I}-\gamma\mathbf{A})^{2k})}{\mathbf{I}-[(1-\alpha)\mathbf{I}+\alpha(\mathbf{I}-\gamma\mathbf{A})^k]^2}\frac{\gamma^2\mathbf{A}^2\Sigma}{\mathbf{I}-(\mathbf{I}-\gamma\mathbf{A})^2}$$

where for the final equality, we used the identity $\sum_0^k a^i = (1-a^k)/(1-a)$. Some standard manipulations of the denominator on the first term lead to the final solution,

$$V^*_{LA} = \frac{\alpha^2(\mathbf{I}-(\mathbf{I}-\gamma\mathbf{A})^{2k})}{\alpha^2(\mathbf{I}-(\mathbf{I}-\gamma\mathbf{A})^{2k}) + 2\alpha(1-\alpha)(\mathbf{I}-(\mathbf{I}-\gamma\mathbf{A})^k)}\frac{\gamma^2\mathbf{A}^2\Sigma^2}{\mathbf{I}-(\mathbf{I}-\gamma\mathbf{A})^2}$$

□

For the same learning rate, Lookahead will achieve a smaller loss as the variance is reduced more. However, the convergence speed of the expectation term will be slower as we must compare $1-\alpha+\alpha(\mathbf{I}-\gamma\mathbf{A})^k$ to $(\mathbf{I}-\gamma\mathbf{A})^k$ and the latter is always smaller for $\alpha < 1$. In our experiments, we observe that Lookahead typically converges much faster than its inner optimizer. We speculate that the learning rate for the inner optimizer is set sufficiently high such that the variance reduction term is more important–this is the more common regime for neural networks that attain high validation accuracy, as higher initial learning rates are used to overcome the short-horizon bias [42]

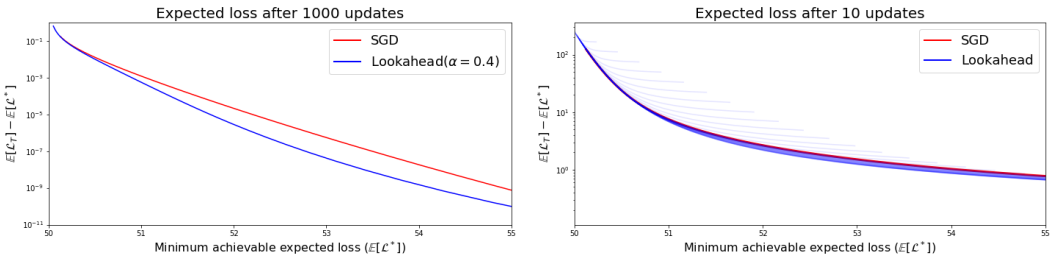
A.1 Comparing convergence rates

In Figure 3 we compared the convergence rates of SGD and Lookahead. We specified the eigenvalues of A according to the worst-case model from Li [20] (also used by Wu et al. [42] and set $\Sigma = A^{-1}$. We computed the expected loss (Equation 5) for learning rates in the range $(0, 1)$ for SGD and Lookahead with a range of α values, with $k = 5$, at time $T = 1000$ (by unrolling the above dynamics). We computed the variance fixed point for each learning rate under each optimizer and use this value to compute the optimal loss. Finally, we plot the difference between the expected loss at T and the final loss, as a function of the final loss. This allows us to compare the convergence performance between SGD and Lookahead optimization settings which converge to the same solution.

Further convergence plots In Figure 11 we present additional plots comparing the convergence performance between SGD and Lookahead. In (a) we show the convergence of Lookahead for a single choice of α, where our method is able to outperform SGD even for this fixed value. In (b) we show the convergence after only a few updates. Here SGD outperforms lookahead for some smaller choices of α, this is because SGD is able to make progress on the expectation more rapidly and reduces this part of the loss quickly — this is related to the short-horizon bias phenomenon [42]. However, even with only a few updates there are choices of α which are able to outperform SGD.

B Deterministic quadratic convergence analysis

Here we present additional details on the quadratic convergence analysis.



(a) (b)

Figure 11: Convergence of SGD and Lookahead on the noisy quadratic model. (a): We show the convergence of Lookahead with a single fixed choice of $\alpha = 0.4$. (b): We compare the early stage performance of Lookahead to SGD over a range of α values.

B.1 Lookahead as a dynamical system

As in the main text, we will assume that the optimum lies at $\boldsymbol{\theta}^* = \mathbf{0}$ for simplicity, but the argument easily generalizes. Here we consider the more general case of a quadratic function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x}$. We use η to denote the CM learning rate and β for it's momentum coefficient.

First we can stack together a full set of fast weights and write the following,

$$\begin{bmatrix} \boldsymbol{\theta}_{t,0} \\ \boldsymbol{\theta}_{t-1,k} \\ \vdots \\ \boldsymbol{\theta}_{t-1,1} \end{bmatrix} = AB^{(k-1)}T \begin{bmatrix} \boldsymbol{\theta}_{t-1,0} \\ \boldsymbol{\theta}_{t-2,k} \\ \vdots \\ \boldsymbol{\theta}_{t-2,1} \end{bmatrix}$$

Here, A represents the Lookahead interpolation, B represents the update corresponding to classical momentum in the inner-loop and T is a transition matrix which realigns the fast weight iterates.

Each of these matrices takes the following form,

$$A = \begin{bmatrix} \alpha I & 0 & \cdots & 0 & (1-\alpha)I \\ I & 0 & \cdots & \cdots & 0 \\ 0 & I & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & I & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} (1+\beta)I - \eta A & -\beta I & 0 & \cdots & 0 \\ I & 0 & \cdots & \cdots & 0 \\ 0 & I & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & I & 0 \end{bmatrix}$$

$$T = \begin{bmatrix} I - \eta A & \beta I & -\beta I & 0 & \cdots & 0 \\ I & 0 & \cdots & \cdots & 0 & \vdots \\ 0 & I & \ddots & \cdots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots & \vdots \\ \vdots & \cdots & 0 & I & 0 & 0 \\ 0 & \cdots & 0 & 0 & I & 0 \end{bmatrix}$$

Each matrix consists of four blocks. The bottom left block is always an identity matrix that shifts the iterates along one index. The bottom right column is all zeros with the top-right column being

non-zero only for A which applies the Lookahead interpolation. The top left row is used to apply the Lookahead/CM updates in each matrix.

After computing the appropriate product of these matrices, we can use standard solvers to compute the eigenvalues which bound the convergence of the linear dynamical system (see e.g. Lessard et al. [19] for a recent exposition). Finally, note that because this linear dynamical systems corresponds to k updates (or one slow-weight update) we must compute the k^{th} root of the eigenvalues to recover the correct convergence bound.

B.2 Optimal slow weight step size

We present the proof of Proposition 1 for the optimal slow weight step size α^*.

Proof. We compute the derivative with respect to α

$$\nabla_\alpha L(\theta_{t,0} + \alpha(\theta_{t,k} - \theta_{t,0})) = (\theta_{t,k} - \theta_{t,0})^T A(\theta_{t,0} + \alpha(\theta_{t,k} - \theta_{t,0})) - (\theta_{t,k} - \theta_{t,0})^T b$$

Setting the derivative to 0 and using $b = A\theta^*$:

$$\alpha[(\theta_{t,k} - \theta_{t,0})^T A(\theta_{t,k} - \theta_{t,0})] = (\theta_{t,k} - \theta_{t,0})^T A(\theta^* - \theta_{t,0}) \tag{12}$$

$$\implies \alpha^* = \arg\min_\alpha L(\theta_{t,0} + \alpha(\theta_{t,k} - \theta_{t,0})) = \frac{(\theta_{t,0} - \theta^*)^T A(\theta_{t,0} - \theta_{t,k})}{(\theta_{t,0} - \theta_{t,k})^T A(\theta_{t,0} - \theta_{t,k})} \tag{13}$$

□

C Experimental setup

Here we present additional details on the experiments appearing in the main paper.

C.1 CIFAR-classification

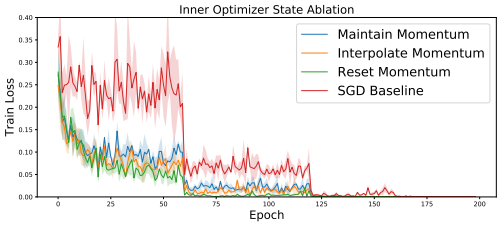
We run every experiment with three random seeds. Our plots show the mean value with error bars of one standard deviation. We use a standard training procedure that is the same as that of Zagoruyko and Komodakis [45]. That is, images are zero-padded with 4 pixels on each side and then a random 32×32 crop is extracted and mirrored horizontally 50% of the time. Inputs are normalized with per-channel means and standard deviations. For computing the loss curves, we iterate through the entire training dataset at the end of very epoch. Lookahead is evaluated on the slow weights of its inner optimizer. To make this evaluation consistent, we evaluate the training loss at the end of each epoch by iterating through the training set again, without performing any gradient updates.

For SGD, we set the momentum to 0.9 and sweep over the learning rates {0.03, 0.05, 0.1, 0.2, 0.3} and weight decay values of {0.0003, 0.001, 0.003}. We found AdamW [21] to perform better than Adam and refer it to as Adam throughout our CIFAR experiment section. For Adam, we sweep do a grid search on learning rate of {3e-4, 1e-3, 3e-3} and weight decay values of {0.1, 0.3, 1, 3}. For Polyak averaging, we compute the moving average of SGD use the best weight decay from SGD and sweep over the learning rates {0.05, 0.1, 0.2, 0.3, 0.5}.

For Lookahead, we set the inner optimizer SGD learning rate to $\{0.1, 0.2\}$ and do a grid search over $\alpha = \{0.2, 0.5, 0.8\}$ and $k = \{5, 10\}$. We report the verison of Lookahead that resets momentum in our CIFAR experiments.

C.2 ImageNet

We directly wrapped Lookahead around the settings provided in the official PyTorch repository repository with $k = 5$ and $\alpha = 0.5$. Observing the improved convergence of our algorithm, we tested Lookahead with the aggressive learning rate decay schedule (decaying at the 30th, 48th, and 58th epochs) and $\alpha = \{0.2, 0.5, 0.8\}$. We run our experiments on 4 Nvidia P100 GPUs with a batch size of 256 and weight decay of 1e-4.



OPTIMIZER	CIFAR-10
MAINTAIN	$95.15 \pm .08$
INTERPOLATE	$95.16 \pm .13$
RESET	$94.91 \pm .05$

Table 6: CIFAR Final Validation Accuracy.

Figure 12: Evaluation of maintaining, interpolating, and resetting momentum on CIFAR-10

C.3 Language modeling

For the language modeling task we used the model and code provided by Merity et al. [26]. We used the default settings suggested in this codebase at the time of usage which we report here. The LSTM we trained had 3 layers each containing 1150 hidden units. We used word embeddings of dimension 400. Within each hidden layer we apply dropout with probability 0.3 and the input embedding layers use dropout with probability 0.65. We applied dropout to the embedding layer itself with probability 0.1. We used the weight drop method proposed in Merity et al. [26] with probability 0.5. We adopt the regularization proposed in section 4.6 in Merity et al. [26]: RNN activations have L2 regularization applied to them with a scaling of 2.0, and temporal activation regularization is applied with scaling 1.0. Finally, all weights receive a weight decay of 1.2e-6.

We trained the model using variable sequence lengths and batch sizes of 80. We apply gradient clipping of 0.25 to all optimizers. During training, if validation loss has not decreased for 15 epochs then we reduce the learning rate by half. Before applying Lookahead, we completed a grid search over the Adam and SGD optimizers to find competitive baseline models. For SGD we did not apply momentum and searched learning rates in the range $\{50, 30, 10, 5, 2.5, 1, 0.1\}$. For Adam we kept the default momentum values of $(\beta_1, \beta_2) = (0.9, 0.999)$ and searched over learning rates in the range $\{0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001\}$. We chose the best model by picking the model which achieved the best validation performance at any point during training.

After picking the best SGD/Adam hyperparameters we trained the models again using Lookahead with the best baseline optimizers for the inner-loop. We tried using $k = \{5, 10, 20\}$ inner-loop updates and $\alpha = \{0.2, 0.5, 0.8\}$ interpolation coefficients. Once again, we reported Lookahead's final performance by choosing the parameters which gave the best validation performance during training.

For this task, $\alpha = 0.5$ or $\alpha = 0.8$ and $k = 5$ or $k = 10$ worked best. As in our other experiments, we found that Lookahead was largely robust to different choices of k and α. We expect that we could achieve even better results with Lookahead if we jointly optimized the hyperparameters of Lookahead and the underlying optimizer.

C.4 Neural machine translation

For this task, we trained on a single TPU core that has 8 workers each with a minibatch size of 2048. We use the default hyperparameters for Adam [41] and AdaFactor [39] in the experiments. For Lookahead, we did a minor grid search over the learning rate $\{0.02, 0.04, 0.06\}$ and $k = \{5, 10\}$ while setting $\alpha = 0.5$. We found learning rate 0.04 and $k = 10$ worked best. After we train those models for 250k steps, they can all reach around 27 BLEU on Newstest2014 respectively.

C.5 Inner Optimizer State

Throughout our paper, we maintain the state of our inner optimizer for simplicity. For SGD with heavy-ball momentum, this corresponds to preserving the momentum. Here, we present a sensitivity study by comparing the convergence of Lookahead when maintaining the momentum, interpolating the momentum, and resetting the momentum. All three improve convergence versus SGD.