YellowFin: Adaptive optimization for (A)synchronous systems

Extended Abstract*

Jian Zhang
Stanford University
zjian@cs.stanford.edu

Ioannis Mitliagkas
University of Montreal
ioannis@iro.umontreal.ca

ABSTRACT

Hyperparameter tuning is one of the most time-consuming steps in machine learning. Adaptive optimizers, like AdaGrad and Adam, reduce this labor by tuning an individual learning rate for each variable. Lately, researchers have shown interest in simpler methods like momentum SGD as they often yield better results. We ask: can simple adaptive methods based on SGD perform well? We show empirically that hand-tuning a single learning rate and momentum makes SGD competitive with Adam. We analyze momentum's robustness to learning rate misspecification and curvature variation. We use this robustness to design YellowFin, an automatic tuner for momentum and learning rate in SGD. YellowFin uses a negative-feedback loop to compensate for the added dynamics in asynchronous-parallel settings on the fly. We empirically show YellowFin can converge in fewer iterations than Adam on ResNet and LSTM models, with a speedup of up to 3.28x in synchronous and up to 2.69x in asynchronous settings.

ACM Reference Format:
Jian Zhang and Ioannis Mitliagkas. . YellowFin: Adaptive optimization for (A)synchronous systems: Extended Abstract. In *Proceedings of SysML (SysML'18).* ACM, New York, NY, USA, 3 pages.

1 INTRODUCTION

Accelerated forms of stochastic gradient descent (SGD) [10, 12], are the de-facto training algorithms for deep learning [13]. Their use requires a sane choice for their *hyperparameters*: typically a *learning rate* and *momentum parameter* [13]. Adaptive optimizers aim to eliminate hyperparameter search by tuning on the fly for a single training run: algorithms like AdaGrad [3], RMSProp [14] and Adam [6] use the magnitude of gradient elements to tune learning rates *individually for each variable* and have been helpful in relieving practitioners of tuning the learning rate.

Recently some researchers have started favoring simple momentum SGD over the previously mentioned adaptive methods [1, 4], often reporting better test scores [15]. Motivated by this trend, we ask the question: can simpler adaptive methods based on momentum SGD perform as well or better? We empirically show that, with hand-tuned learning rate, Polyak's momentum SGD achieves faster convergence than Adam for a large class of models. We then formulate the optimization update as a dynamical system and study certain robustness properties of the momentum operator. Guided by our analysis, we design YellowFin, an automatic hyperparameter tuner for momentum SGD (Section 2). It tunes both the learning rate and momentum on the fly and can be faster than Adam (Section 3). Finally, we discuss closed-loop YellowFin, designed for asynchronous training. It uses a novel component for measuring the total momentum in a running system, including any asynchrony-induced momentum, a phenomenon described in [9]. This measurement is used in a negative feedback loop to control the value of algorithmic momentum (Section 4). We release PyTorch and TensorFlow implementations[1][2] that can be used as drop-in replacements for any optimizer. YellowFin has also been implemented in various other packages and deployed in industry.

*This extended abstract summarizes the most important contributions in our full manuscript, with the same name, available on arXiv [17].

2 THE YELLOWFIN TUNER

In this section we summarize the main insight behind our tuner: *when the momentum value is high enough, gradient descent's convergence rate can be robust to learning rate misspecification and to curvature variation.* Robustness to learning rate misspecification means tolerance to a less-carefully-tuned learning rate. Robustness to curvature variation can imply a linear convergence on a class of non-convex objectives where curvature varies significantly. To demonstrate, we rewrite the momentum update

$$x_{t+1} = x_t - \alpha \nabla f(x_t) + \mu(x_t - x_{t-1}), \tag{1}$$

on a scalar quadratic, $f(x)$ with curvature h, with learning rate α and momentum μ in terms of the *momentum operator* $\boldsymbol{A}_t$ at time t

$$\begin{pmatrix} x_{t+1} - x^* \\ x_t - x^* \end{pmatrix} = \begin{bmatrix} 1 - \alpha h + \mu & -\mu \\ 1 & 0 \end{bmatrix} \begin{pmatrix} x_t - x^* \\ x_{t-1} - x^* \end{pmatrix} \triangleq \boldsymbol{A}_t \begin{pmatrix} x_t - x^* \\ x_{t-1} - x^* \end{pmatrix}.$$

Lemma 2.1 (Robustness of the momentum operator). *Our robustness property stems from an unexplored fact: the momentum operator's spectral radius is constant in a subset of the hyperparameter space, which we call* **the robust region**. *Specifically, if* $(1 - \sqrt{\mu})^2 \leq \alpha h \leq (1 + \sqrt{\mu})^2$ *then the spectral radius of the momentum operator at step* t *depends solely on the momentum parameter:* $\rho(\boldsymbol{A}_t) = \sqrt{\mu}$.

A homogeneous spectral radius for all operators $\boldsymbol{A}_t$ does not guarantee a constant rate of convergence, nonetheless, we observe this kind of behavior in practice. In our full manuscript we extend this result to non-quadratics and discuss extensions to multidimensional objectives [17]. We use these robustness insights and a standard quadratic model analysis to drive the design of YellowFin, an automatic tuner for momentum SGD. YellowFin uses on-the-fly measurements from the gradients to tune both a single learning rate and momentum. The measurement functions use a log-probability assumption to yield estimates of the curvatures encountered during training. Similarly, we get estimates of gradient variance and distance from a local minimum. These estimates are used to minimize

[1]PyTorch implementation: https://github.com/JianGoForIt/YellowFin_Pytorch
[2]TensorFlow implementation: https://github.com/JianGoForIt/YellowFin

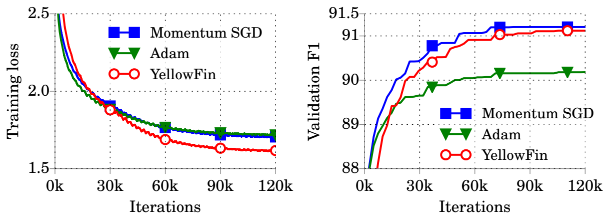

Figure 1: Training loss (left) and validation F1 (right) for the 3-layer constituency parsing LSTM on WSJ.

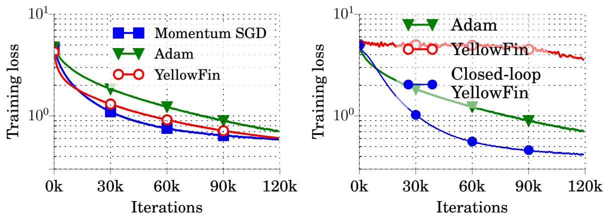

Figure 2: Synchronous (left) and asynchronous (right) training loss for ResNet164 on CIFAR100.

the expected squared error on a local quadratic approximation. The key point is that we force the tuner to work in the *robust region* by setting an adaptive lower bound on the momentum value [17].

3 SYNCHRONOUS EXPERIMENTS

In this section, we empirically validate the importance of momentum tuning and evaluate YellowFin, on both convolutional and recurrent neural networks. We tune baseline optimizers, Adam and momentum, SGD on learning rate grids with prescribed momentum 0.9 for SGD. We run YellowFin *without any hand tuning*. For visualization purposes, we smooth training losses with a uniform window of width 1000. For Adam and momentum SGD on each model, we pick the configuration achieving the lowest smoothed loss. To compare two algorithms, we record the lowest smoothed loss achieved by both. The speedup is reported as the ratio of iterations to achieve this loss. We use this setup to validate our claims.

Momentum SGD is competitive with adaptive methods. In Figure 1 (left) and Figure 2 (left), we compare the training loss from tuned momentum SGD and tuned Adam on ResNet164 [5] for CIFAR100 dataset [7] and a 3-layer LSTM [2] constituency parser for Penn TreeBank WSJ dataset [8]. We can observe that momentum SGD achieves 1.87x and 1.33x speedup to tuned Adam on ResNet and LSTM model respectively.

YellowFin can match hand-tuned momentum SGD and can outperform hand-tuned Adam. In our experiments, YellowFin, without any hand-tuning, yields training loss matching hand-tuned momentum SGD for the ResNet and LSTM in Figure 1 (left) and Figure 2 (left). When comparing to tuned Adam, YellowFin achieves 1.38x to 2.33x speedups in training losses on the ResNet and LSTM respectively. Noticeably, YellowFin also matches momentum SGD in validation F1, outperforming Adam by 1% on the constituency parsing LSTM. We refer to our full manuscript [17] for detailed experiment protocol and results on more models.

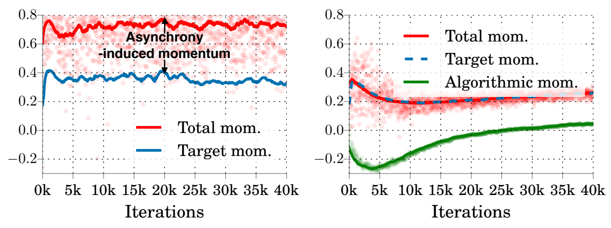

Figure 3: When running YellowFin, total momentum $\hat{\mu}_t$ is greater than algorithmic value on 16 asynchronous workers (left). Closed-loop YellowFin automatically lowers algorithmic momentum and matches total momentum to the target value (right). Red dots are measured $\hat{\mu}_t$ at every step with red line as its running average.

4 CLOSED-LOOP YELLOWFIN

Asynchrony is a parallelization technique that avoids synchronization barriers [11]. It yields better hardware efficiency, i.e. faster steps, but can increase the number of iterations to a given metric, i.e. statistical efficiency, as a tradeoff [16]. Mitliagkas et al. [9] interpret asynchrony as added momentum dynamics. We design closed-loop YellowFin, a variant of YellowFin to automatically control algorithmic momentum, compensate for asynchrony and accelerate convergence. We use the formula in (2) to model the dynamics in the system, where the total momentum, μ_T, includes both asynchrony-induced and algorithmic momentum μ,

$$\mathbb{E}[x_{t+1} - x_t] = \mu_T \mathbb{E}[x_t - x_{t-1}] - \alpha \mathbb{E}\nabla f(x_t) \tag{2}$$

We first use (2) to design a robust estimator $\hat{\mu}_T$ for the value of total momentum at every iteration. Then we use a simple negative feedback control loop to adjust the value of algorithmic momentum so that $\hat{\mu}_T$ matches the *target momentum* decided by the basic YellowFin algorithm. In Figure 3, we demonstrate momentum dynamics in an asynchronous training system. As it directly uses the target value as algorithmic momentum, the basic YellowFin (left) exhibits total momentum $\hat{\mu}_T$ strictly larger than the target momentum, due to asynchrony-induced momentum. Closed-loop YellowFin (right) automatically brings down the algorithmic momentum, match measured total momentum $\hat{\mu}_T$ to target value and, as we will see, significantly speeds up convergence compared to YellowFin.

We evaluate closed-loop YellowFin with focus on the number of iterations to reach a certain solution. To that end, we run 16 asynchronous workers on a single machine and force them to update the model in a round-robin fashion, i.e. the gradient is delayed for 15 iterations. Figure 2 (right) presents training losses on the CIFAR100 ResNet, using the basic YellowFin, closed-loop YellowFin and Adam with the learning rate achieving the best smoothed loss in Section 3. We can observe closed-loop YellowFin achieves 20.1x speedup to YellowFin, and consequently a 2.69x speedup to Adam. This demonstrates that 1) closed-loop YellowFin accelerates by reducing algorithmic momentum to compensate for asynchrony and 2) can converge in less iterations than Adam in asynchronous-parallel training.

REFERENCES

[1] Danqi Chen, Jason Bolton, and Christopher D Manning. 2016. A thorough examination of the cnn/daily mail reading comprehension task. *arXiv preprint arXiv:1606.02858* (2016).

[2] Do Kook Choe and Eugene Charniak. [n. d.]. Parsing as Language Modeling. ([n. d.]).

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

[4] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional Sequence to Sequence Learning. *arXiv preprint arXiv:1705.03122* (2017).

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[6] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[7] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2014. The CIFAR-10 dataset. (2014).

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

[9] Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Ré. 2016. Asynchrony begets momentum, with an application to deep learning. *arXiv preprint arXiv:1605.09774* (2016).

[10] Yurii Nesterov. 1983. A method of solving a convex programming problem with convergence rate O (1/k2). In *Soviet Mathematics Doklady*, Vol. 27. 372–376.

[11] Feng Niu, Benjamin Recht, Christopher Re, and Stephen Wright. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*. 693–701.

[12] Boris T Polyak. 1964. Some methods of speeding up the convergence of iteration methods. *U. S. S. R. Comput. Math. and Math. Phys.* 4, 5 (1964), 1–17.

[13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning (ICML-13)*. 1139–1147.

[14] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4, 2 (2012).

[15] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. 2017. The Marginal Value of Adaptive Gradient Methods in Machine Learning. *arXiv preprint arXiv:1705.08292* (2017).

[16] Ce Zhang and Christopher Ré. 2014. DimmWitted: A Study of Main-Memory Statistical Analytics. *PVLDB* 7, 12 (2014), 1283–1294. http://www.vldb.org/pvldb/vol7/p1283-zhang.pdf

[17] Jian Zhang, Ioannis Mitliagkas, and Christopher Ré. 2017. YellowFin and the Art of Momentum Tuning. *arXiv preprint arXiv:1706.03471* (2017).