OPT2020: 12th Annual Workshop on Optimization for Machine Learning

Adaptive Gradient Tracking In Stochastic Optimization

Zhanhong Jiang STARKJIANG@GMAIL.COM
Johnson Controls International

Xian Yeow Lee XYLEE@IASTATE.EDU
Iowa State University

Sin Yong Tan TSYONG98@IASTATE.EDU
Iowa State University

Aditya Balu BADITYA@IASTATE.EDU
Iowa State University

Young M Lee YOUNG.M.LEE@JCI.COM
Johnson Controls International

Chinmay Hegde CHINMAY.H@NYU.EDU
New York University

Soumik Sarkar SOUMIKS@IASTATE.EDU
Iowa State University

Abstract

Adaptive gradient descent algorithms such as Adam have played a critical role in driving deep learning success and emerged as the current state-of-the-art stochastic optimizer. However, adaptive learning rate usage causes such optimizers to underperform than SGD in terms of generalization, due to large variance, when the dataset is large and complex. In this work, we perceive the problem from another perspective of penalizing the gradient fluctuation using the *gradient tracking* technique that has been quite popular in decentralized optimization. We develop an algorithm, termed AdaTrack, using the *adaptive* gradient tracking to control the degree of penalty throughout the optimization process. We present the theoretical analysis to show the sublinear regret bound. Via empirical experiments, we have found that AdaTrack compares favorably with RAdam by reducing the variance in terms of gradient penalty with fewer intermediate parameters and outperforms AdaBound and Adam convincingly by improving the training performance significantly.

1. Introduction

Machine learning (in particular deep learning) success is primarily attributed to stochastic optimizers, such as stochastic gradient descent (SGD), which has been used tremendously in various science and engineering problems. To accelerate stochastic optimization, *adaptive learning rate* recently attracted considerable attention such that adaptive gradient descent algorithms, in particular, Adagrad [2], RMSProp [19], Adadelta [22], and Adam [7], stand out to be superior alternatives to SGD in numerous applications, due to their faster convergence. However, it was observed that the generalization of these adaptive gradient-descent approaches on unseen data could be poor when the model and data are both complex [5, 11, 13]. Recent work [10] has revealed that the poor generalization was due to the unbounded variance of adaptive learning rates in the early stages, causing convergence to poor local optima. As remedies, different empirical techniques have been applied, including warmup [16] and varying the stability parameter (ϵ) [9] in the update rule, which unfortunately lacks theoretical underpinnings.

In Liu et al. [10], the authors mathematically analyzed the origin of the variance and developed a rectification term to reduce the negative impact on the performance. However, based on Adam's algorithmic framework, two more intermediate parameters are required in the update law, which increases the memory storage during the optimization, particularly for high-dimension problems. Another recent work [12] proposed to employ dynamic bounds on learning rates such that a gradual and smooth transition from adaptive methods to SGD was achieved to improve the generalization performance. Intuitively, such bounds act like trust regions for the adaptive learning rate. Though theoretical proof of convergence was provided, studying the best lower and upper bounds of learning rates for different application scenarios was missing.

In this work, to effectively reduce the gradient's variance, we consider the problem from the perspective of *gradient tracking* [17, 20, 21], which keeps track of the difference between two consecutive steps of (stochastic) gradients. This technique has been quite popular in decentralized optimization, either deterministic or stochastic. We propose AdaTrack, an adaptive gradient descent algorithm that leverages the exponential moving average (EMA) of gradient tracking, which plays a role in penalizing the significant variations of gradients during the optimization process, enhancing the generalization capability. To facilitate the theoretical understanding of the adaptive gradient tracking in AdaTrack, we provide a detailed analysis of the proposed algorithm. Additionally, we note that gradient tracking is also close to recently developed algorithms, SARAH [14] and SPIDER [3], and provides a discussion on the difference among SARAH, SPIDER, and gradient tracking. We give another variant of RAdam incorporating adaptive gradient tracking (termed RAT) and empirically compare AdaTrack and RAT to RAdam, AdaBound, Adam, and SGD with benchmark four image classification datasets. The image classification tasks show that RAT performs the best in training and that both AdaTrack and RAT have better convergence speed than AdaBound. Regarding the generalization for the testing dataset, AdaTrack compares favorably with RAdam and RAT with fewer intermediate parameters in the update rule.

2. Preliminaries

In this section, we formulate the stochastic optimization problem and provide a brief recap of the adaptive gradient descent algorithm and gradient tracking technique. Consider an expected risk minimization problem for generic deep learning models as follows:

$$x^* = \text{argmin}_{x\in\mathbb{R}^d}\mathbb{E}[f(x)] \tag{1}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is the corresponding continuously differentiable loss. At each iteration $t \in [T] = \{0, 1, ..., T\}$, we denote by $f_0, f_1, ..., f_T$ each realization of the loss function. To solve Eq. 1, adaptive gradient descent algorithms have been adopted popularly in diverse applications. Their update rules can be summarized in a unified way at any time step t.

$$\begin{aligned} g_t = \nabla f_t(x_t, \zeta_t), \quad m_{t+1} = h_t(g_1, ..., g_t, t) \\ v_{t+1} = c_t(g_1, ..., g_t, t), \quad x_{t+1} = x_t - \alpha_t m_{t+1} v_{t+1}, \end{aligned} \tag{2}$$

where ζ_t signifies the random seed at time step t, $h_t(\cdot)$ is for calculating the first-order moment, $c_t(\cdot)$ is for attaining the second-order moment, and α_t represents the learning rate. For example, Adam enables the second and third equations to become

$$m_{t+1} = \frac{(1-\beta_1)\sum_{k=1}^{t}\beta_1^{t-k} g_k}{1-\beta_1^t}, \quad v_{t+1} = \frac{\sqrt{1-\beta_2^t}}{\epsilon + \sqrt{(1-\beta_2)\sum_{k=1}^{t}\beta_2^{t-k} g_k^2}}, \tag{3}$$

where β_1, β_2 are positive coefficients strictly less than 1, ϵ is a small constant for maintaining the numerical stability and typically set 10^{-8}. We next review the gradient tracking that is expressed as

$$y_{t+1} = y_t + \nabla f_t(x_t, \zeta_t) - \nabla f_{t-1}(x_{t-1}, \zeta_{t-1}), \tag{4}$$

where y in this context is an intermediate parameter to accumulate the difference between the current and last steps of stochastic gradients. In decentralized optimization [17], Eq. 4 has a slightly different variant where y_t is multiplied by a (doubly) stochastic matrix to form the average of local gradients. In a centralized setting, though without a matrix, Eq. 4 can still be interpreted by a simple moving average, which always updates the successive values using only the last two steps of information. Please refer to Supplementary for a brief preliminary overview of related works.

3. Proposed Algorithm

We present the proposed algorithm, AdaTrack, using the following relationship instead of the vanilla gradient tracking in Eq. 4:

$$y_{t+1} = \beta y_t + (1-\beta)(\nabla f_t(x_t, \zeta_t) - \nabla f_{t-1}(x_{t-1}, \zeta_{t-1})), \tag{5}$$

which can be interpreted as the EMA of gradient tracking, where $\beta \in [0, 1)$. Stochastic recursive gradient schemes such as gradient tracking and SARAH cannot induce an unbiased estimate of $\nabla f_t(x_t)$, but being able to reduce variance. In Nguyen et al. [14], the intermediate parameter y_{t+1} was periodically reset using the full gradient to reduce the bias. In our work, the reset step is dropped for the gradient tracking to avoid full gradient calculation. Instead, we apply the EMA to reduce the bias, with some sacrifice on variance reduction. This also makes a difference from the update law in Carnevale et al. [1] and detailed analysis is provided in Section 4. Algorithm 1 presents the algorithmic framework of AdaTrack, followed by an overview.

Algorithm 1 Adaptive Gradient Tracking Descent

```
1:  Input: $\alpha_t, \beta_1, \beta_2, \beta_3, \epsilon, x_0, \nabla f_{-1}(x_{-1}, \zeta_{-1})$        ▷ Input params
2:  $m_0, v_0, y_0, t = 0$        ▷ Initializations
    while $t \leq T$ do
3:      $m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla f_t(x_t, \zeta_t)$        ▷ Approximate first moment
4:      $v_{t+1} = \beta_2 v_t + (1-\beta_2)\nabla f_t^2(x_t, \zeta_t)$        ▷ Approximate second moment
5:      $y_{t+1} = \beta_3 y_t + (1-\beta_3)(\nabla f_t(x_t, \zeta_t) - \nabla f_{t-1}(x_{t-1}, \zeta_{t-1}))$        ▷ EMA of gradient tracking
6:      $\hat{m}_{t+1} = \frac{m_{t+1}}{1-\beta_1^t}$        ▷ Bias-correction for first moment
7:      $\hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^t}$        ▷ Bias-correction for second moment
8:      $\hat{y}_{t+1} = \frac{y_{t+1}}{1-\beta_3^t}$        ▷ Bias-correction for adaptive gradient tracking
9:      $x_{t+1} = x_t - \alpha_t \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}+\epsilon}} - \alpha_t \hat{y}_{t+1}$        ▷ Update using adaptive gradient tracking descent
10:     $t = t + 1$
11: return $x_T$
```

In Algorithm 1, Lines 3 to 5 approximate the first or second moment of gradient or gradient tracking by computing the associated EMAs. Similar to Adam, Lines 6 to 8 correct the bias. In Line

9, the adaptive learning rate is still applied to the gradient, but not the EMA of gradient tracking. As we have known from [12], the adaptive learning rate can cause large variance at the early stage. Hence, for the term $\hat{y}_{t+1}$, it is only multiplied by the negative learning rate instead of the adaptive learning rate, which may weaken the effort of variance reduction. A different perspective that can be imposed in this context for Line 9 is that the search for the optimal solution is via the negative direction of $\frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}+\epsilon}}$ and the variance reduction is conducted by penalizing the significant gradient variations along the gradient trajectory, signified by the intermediate parameter $\hat{y}_{t+1}$. Therefore, such an update explicitly shows the combined efforts between the optimality and generalization capability for a model f. A more formal analysis of the algorithm is presented in Section 4. It is noted that the EMA of gradient tracking can be incorporated into any adaptive gradient descent algorithm, such as RAdam or AdaBound. Additionally, compared to Adam, AdaTrack has one additional hyperparameter β_3 to be determined. For simplicity, we set $\beta_3 = \beta_1$, and empirically found that AdaTrack performs robustly with different values of β_3 as long as it is strictly less than 1. For the following analysis, we still keep β in Eq. 5 instead of β_3, which is for the algorithmic implementation. In this context, we also extend the gradient tracking technique to RAdam to develop a counterpart, termed RAT (included in Supplementary Materials). While RAT is more complex than both RAdam and AdaTrack due to the additional intermediate parameters, we empirically investigate if gradient tracking further improves RAdam on top of variance rectification.

4. Theoretical Analysis

4.1. Gradient Tracking

As shown in Eq. 4, the vanilla gradient tracking can be rewritten as

$$y_{t+1} = \sum_{k=0}^{t} \nabla f_k(x_k, \zeta_k) - \nabla f_{k-1}(x_{k-1}, \zeta_{k-1}), \tag{6}$$

which is the cumulative gradient variations of any two consecutive steps along with the gradient trajectory $\{\nabla f_{-1}(x_{-1}, \zeta_{-1}), \nabla f_0(x_0, \zeta_0), ..., \nabla f_t(x_t, \zeta_t)\}$. By recalling the core update law in Algorithm 1, the following relationship can be obtained by directly substituting y_{t+1}

$$x_{t+1} = x_t - \alpha_t \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \epsilon} - \alpha_t y_{t+1}. \tag{7}$$

Suppose that after a sufficiently large number of iterations, $T_0 \in \mathbb{N}$, x_t converges to an ϵ_1-ball centered at a local minimum x^*, where $\epsilon_1 > 0$ can be an arbitrary constant, such that

$$y_{t+1} \approx \sum_{k=0}^{T_0} \nabla f_k(x_k, \zeta_k) - \nabla f_{k-1}(x_{k-1}, \zeta_{k-1}), \tag{8}$$

$\forall t \geq T_0$, which implies that $\nabla f_t(x_t, \zeta_t) \approx \nabla f_{t-1}(x_{t-1}, \zeta_{t-1})$. Hence substituting Eq. 8 into Eq. 7 suggests that the gradient tracking term in the update law becomes a constant that is independent of $x_t, \forall t > T_0$. Thus, instead of being the penalty on the significant gradient variations to reduce variance, such a term may have a negative impact on updating x_t afterwards. Since for any iteration ahead of T_0, y_t acts like an constraint to prevent x_t from deviating significantly from the trajectory of $\{x_{-1}, x_0, ..., x_{T_0}\}$, which may not necessarily be approaching the minimum x^*. Hence, after T_0, the impact of the historical gradients prior to $\nabla f_{T_0}(x_{T_0}, \zeta_{T_0})$ on x_t should be alleviated accordingly.

We instead use Eq. 5 in the update law such that after T_0, the following approximate equality can be obtained

$$y_{t+1} \approx (1-\beta)\sum_{k=0}^{T_0} \beta^{T_0-k}(\nabla f_k(x_k,\zeta_k) - \nabla f_{k-1}(x_{k-1},\zeta_{k-1})), \tag{9}$$

$\forall t \geq T_0$, such that,

$$x_{t+1} \approx x_t - \alpha_t \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}}+\epsilon} - \alpha\beta^{t-T_0}\hat{y}_{t+1}, \tag{10}$$

where it can be observed that the third term on the right-hand side decays exponentially; hence, the EMA of gradient tracking is more resilient than the vanilla gradient tracking in terms of controlling the variance reduction during the optimization. Intuitively, either SARAH or SPIDER may have the same underlying problem the vanilla gradient tracking has, though we are unaware of any algorithms that combine either with adaptive gradient descent. Existing results on SARAH or SPIDER are alternatives to achieve variance reduction in stochastic optimization and focus mostly on simple datasets and models. While SARAH or SPIDER can be extended with the exponential moving average, the following Table 1 motivates us to select the gradient tracking instead.

Table 1: *Comparisons between different stochastic recursive gradient schemes based on usage of different variables* $\Big(\nabla f_t(x_t,\zeta_t)$ *&* $\nabla f_{t-1}(x_{t-1},\zeta_{t-1})$ *&* $\nabla f_t(x_{t-1},\zeta_t)$ *&* $x_{t-1}\Big)$ *used in the scheme.*

Method	$\nabla f_t(x_t,\zeta_t)$	$\nabla f_{t-1}(x_{t-1},\zeta_{t-1})$	$\nabla f_t(x_{t-1},\zeta_t)$	x_{t-1}	Computation
SARAH	✓	✗	✓	✓	$\mathcal{O}(2dT)$
SPIDER	✓	✗	✓	✓	$\mathcal{O}(2dT)$
ROOT-SGD [8]	✓	✗	✓	✓	$\mathcal{O}(2dT)$
Gradient Tracking	✓	✓	✗	✗	$\mathcal{O}(dT)$

4.2. Sublinear Regret

To investigate the convergence of AdaTrack, we present the regret bound instead of the static error bound. The regret analysis is based on the online learning framework given an arbitrary unknown sequence of convex loss functions, $\{f_0(x), f_1(x), ..., f_T(x)\}$. The regret analysis aims at predicting the parameter x_t at each iteration t and evaluate it on the previously unknown cost function f_t. Hence, the static regret is defined as the sum of all the previous difference between the online prediction $f_t(x_t)$ and the best-fixed point parameter $f_t(x^*)$ from a feasible set $\mathcal{X} \subset \mathbb{R}^d$. Specifically, the regret is expressed as:

$$\mathcal{R}_T^S := \sum_{t=0}^{T}[f_t(x_t) - f_t(x^*)], \tag{11}$$

where $x^* = \text{argmin}_{x\in\mathcal{X}} \sum_{t=0}^{T} f_t(x)$. Thus, we have the following informal result for AdaTrack.

Proposition 1 *Assume that f_t is Lipschitz continuous and that $\mathcal{X}$ is compact. Let $\beta_1, \beta_2, \beta_3 \in [0,1)$ satisfy $\frac{\beta_1^2}{\sqrt{\beta_2}} < 1$ and $\beta_{1,t} = \beta_1\lambda^{t-1}, \lambda \in (0,1)$. Thus, for all $T \geq 0$, when the learning rate $\alpha_t = \mathcal{O}(\frac{1}{\sqrt{t+1}})$, AdaTrack has the sublinear regret, i.e., $\mathcal{R}_T^S = \mathcal{O}(\sqrt{T})$.*

5. Numerical Experiments

We evaluate AdaTrack on several benchmarks: Fashion-MNIST, SVHN, CIFAR 10 and CIFAR 100. Please also refer to Supplementary for details on the model training details.

Comparing to Adam & SGD. The performance on image classification is presented in Table 2 and Fig. 1. The results show that AdaTrack outperforms SGD and Adam in all four datasets, though for the Fashion MNIST dataset, AdaTrack's performance is close to Adam and SGD. However, Fig. 1 demonstrates that in training performance, the convergence speed for AdaTrack is significantly faster than either SGD or Adam. In other words, by reducing the stochastic gradient variance, AdaTrack obtains both faster convergence and better performance. It also empirically shows from Table 2 that for different model architectures (ranging from simple to complex), AdaTrack keeps consistent outperforming capabilities over the two most popular baseline methods, SGD and Adam. Additional results are shown in the supplementary materials to show the training accuracies as well.

Comparing to RAdam & AdaBound. Two other recently developed advanced optimizers, AdaBound and RAdam, are also used to compare with AdaTrack and one extensive variant equipped with the gradient tracking, RAT, on top of RAdam. Based on Fig. 1, the results suggest that in terms of convergence speed, the gradient tracking enables AdaTrack to perform significantly better than AdaBound and to be favorably compared to or slightly better than RAdam. Although in Fashion MNIST, AdaTrack performs slightly slower than RAdam. For RAT, it has almost the same performance as RAdam. In Table 2, when datasets are simpler, such as Fashion MNIST and SVHN, RAdam, AdaTrack, and RAT stand to be the best alternatives in the generalization performance. However, for CIFAR datasets, AdaBound becomes the best optimizer, mainly attributed to the use of a dynamic learning rate with the lower and upper bounds. However, these bounds can be difficult to determine for different tasks. When the training weights approach the optimal solution, reducing the learning rate can stabilize the convergence. Averagely speaking, RAdam has slightly better generalization capability than AdaTrack, but the former has a more complex algorithmic framework involving one more intermediate parameter in the update law than the latter. Overall, AdaTrack effectively reduces the variance to achieve faster convergence and to match the best performance of the state-of-the-art. Hence, this can provide helpful insights to the community that stochastic recursive gradients are another way to reduce variance in adaptive gradient descent algorithms.

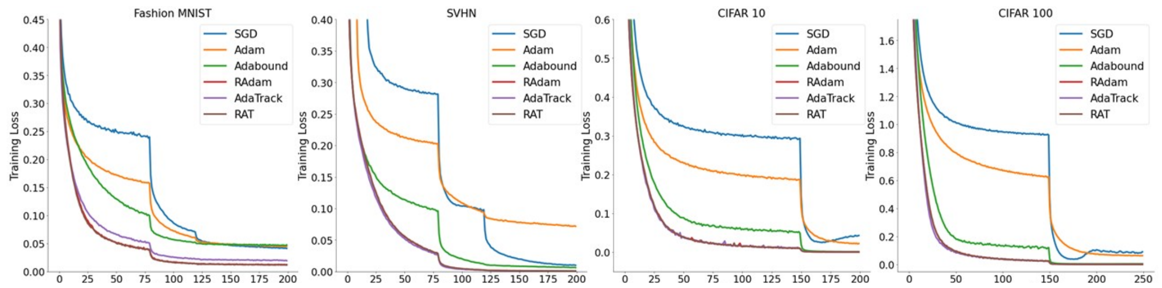

Figure 1: Comparison of training loss trends for six optimizers across four image datasets.

6. Conclusions

This work presented a new stochastic optimizer by leveraging the adaptive gradient tracking developed to reduce the gradient variance. We showed the difference among different stochastic recursive gradient schemes and presented the analytical results for the proposed algorithm, AdaTrack, which enabled a decent sublinear regret bound for convex loss functions. Empirical results demon-

Table 2: *Testing accuracies of different optimizers for image datasets. A multi-layer CNN network architecture was used for Fashion MNIST, VGG19 was used for SVHN and ResNet34 was used for both CIFAR datasets.*

	Fashion MNIST	SVHN	CIFAR 10	CIFAR 100
	Test Acc.(%)	Test Acc.(%)	Test Acc.(%)	Test Acc.(%)
SGD	93.4 ± 0.17	95.8 ± 0.12	92.9 ± 0.30	71.9 ± 0.63
Adam	93.5 ± 0.09	95.6 ± 0.13	92.9 ± 0.17	71.5 ± 0.22
Adabound	93.2 ± 0.10	95.8 ± 0.10	94.9 ± 0.17	76.6 ± 0.23
RAdam	93.6 ± 0.15	96.0 ± 0.09	94.6 ± 0.24	74.4 ± 0.13
AdaTrack	93.5 ± 0.12	96.0 ± 0.11	94.3 ± 0.12	72.5 ± 0.84
RAT	93.6 ± 0.21	96.0 ± 0.04	94.4 ± 0.10	74.1 ± 0.30

strated that AdaTrack outperformed SGD and Adam and that it is competitive compared with the state-of-the-art optimizers by introducing a different way to reduce variance.

7. Acknowledgements

This paper is based upon research partially supported by the National Science Foundation under grant No. CAREER-1845969. Any claims, methodology and conclusions discussed in this research work correspond to those of the authors and do not necessarily reflect the opinions of the funding agency.

References

[1] Guido Carnevale, Francesco Farina, Ivano Notarnicola, and Giuseppe Notarstefano. Distributed online optimization via gradient tracking with adaptive momentum. *arXiv preprint arXiv:2009.01745*, 2020.

[2] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[3] Cong Fang, Chris Junchi Li, Zhouchen Lin, and Tong Zhang. Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator. In *Advances in Neural Information Processing Systems*, pages 689–699, 2018.

[4] Elad Hazan, Karan Singh, and Cyril Zhang. Efficient regret minimization in non-convex games. *arXiv preprint arXiv:1708.00075*, 2017.

[5] 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.

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

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

[8] Chris Junchi Li, Wenlong Mou, Martin J Wainwright, and Michael I Jordan. Root-sgd: Sharp nonasymptotics and asymptotic efficiency in a single algorithm. *arXiv preprint arXiv:2008.12690*, 2020.

[9] Liyuan Liu, Xiang Ren, Jingbo Shang, Jian Peng, and Jiawei Han. Efficient contextualized representation: Language model pruning for sequence labeling. *arXiv preprint arXiv:1804.07827*, 2018.

[10] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[11] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[12] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*, 2019.

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

[14] Lam M Nguyen, Jie Liu, Katya Scheinberg, and Martin Takáč. Sarah: A novel method for machine learning problems using stochastic recursive gradient. *arXiv preprint arXiv:1703.00102*, 2017.

[15] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch, 2017.

[16] Martin Popel and Ondřej Bojar. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43–70, 2018.

[17] Shi Pu and Angelia Nedić. Distributed stochastic gradient tracking methods. *Mathematical Programming*, pages 1–49, 2020.

[18] Arun Sai Suggala and Praneeth Netrapalli. Online non-convex learning: Following the perturbed leader is optimal. In *Algorithmic Learning Theory*, pages 845–861, 2020.

[19] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2): 26–31, 2012.

[20] Ran Xin and Usman A Khan. Distributed heavy-ball: A generalization and acceleration of first-order methods with gradient tracking. *IEEE Transactions on Automatic Control*, 2019.

[21] Ran Xin, Anit Kumar Sahu, Usman A Khan, and Soummya Kar. Distributed stochastic optimization with gradient tracking over strongly-connected networks. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 8353–8358. IEEE, 2019.

[22] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Supplementary Materials

Related works

SARAH & SPIDER. To reduce the gradient variance, the authors in [14] and [3] respectively developed the recursive stochastic gradient update that has slight difference compared with gradient tracking:

$$y_{t+1} = y_t + \nabla f_t(x_t, \zeta_t) - \nabla f_t(x_{t-1}, \zeta_t), \tag{12}$$

where at the time step t, the difference is obtained by calculating the gradients of f at x_t and x_{t-1} using the same sampled data. They also periodically reset y_{t+1} using the full gradient $\nabla f_t(x_t)$ to reduce the bias as y_{t+1} is not the unbiased estimate of $\nabla f_t(x_t)$, different from another approach, SVRG [6]. Either Eq. 4 or Eq. 12 is interpreted as the stochastic recursive gradient scheme to estimate the full gradient by using the stochastic gradient $\nabla f_t(x_t, \zeta_t)$ and the gradient correction term $y_t - \nabla f_t(x_{t-1}, \zeta_t)$ (or $y_t - \nabla f_{t-1}(x_{t-1}, \zeta_{t-1})$), which has been theoretically shown to improve the convergence rate compared to vanilla SGD form [14]. In this work we perceive the problem in a different perspective, namely, the penalty on gradient variations based on Eq. 4, which motivated us to present a novel algorithm in Section 3, followed by the detailed analysis of why gradient tracking is selected for reducing variance instead of SARAH or SPIDER.

ROOT-SGD. Li et al. [8] recently proposed a novel algorithm that is still based on stochastic recursive gradients, termed ROOT-SGD (Recursive One-Over-T SGD) for reducing the variance. They considered the first-order stochastic optimization problem in a general statistical point of view using the recursive averaging of past stochastic gradients. The core update of ROOT-SGD is expressed as

$$y_{t+1} = \nabla f_t(x_t, \zeta_t) + \frac{t}{t+1}(y_t - \nabla f_t(x_{t-1}, \zeta_t)), \tag{13}$$

which can be connected with SARAH with the following rewritten formula:

$$y_{t+1} = \frac{1}{t+1}\nabla f_t(x_t, \zeta_t) + \frac{t}{t+1}(y_t + \nabla f_t(x_t, \zeta_t) - \nabla f_t(x_{t-1}, \zeta_t)). \tag{14}$$

We can clearly observe that the above equation can be viewed as the convex combination between the current stochastic gradient and the SARAH, with a time-varying ratio. When t is sufficiently large, the SARAH part becomes dominating. ROOT-SGD is a generalized SARAH and we mainly discuss the property of SARAH in this paper, but the conclusions can also be adaptive to ROOT-SGD.

Concurrent Work. One concurrent work [1] leverages the gradient tracking and adaptive momentum, resulting in a new approach called GTAdam. While the idea sounds similar to ours, the specific implementation is quite different. The authors approximated the first and second moment using directly the gradient tracking term instead of the stochastic gradient and defined a saturation for the second moment. However, determining the saturation constant can be challenging, depending on different problems. Additionally, GTAdam was developed for the distributed online optimization with strongly convex losses, which is quite different from our problem formulation. Though we did not theoretically investigate the difference between the GTAdam and AdaTrack, in our empirical work we tested the performance of GTAdam-like approaches for the stochastic optimization, without a saturation constant. The results showed us that GTAdam-like approaches performed poorly with complex datasets and models (e.g., CIFAR 100 and ResNet), which haven't been adopted in [1]. Thus, for a fair comparison, GTAdam will not be included in the final baseline methods.

A Generalization Gap. A generalization gap in this context is referred to as the difference between training and testing accuracies. Such a gap can be underlyingly attributed to the tradeoff between model bias and variance. Due to the usage of deep models that typically have millions or even billions of parameters, high variance in the training is one of most challenging topics in modern machine learning, which yields the overfitting that weakens the model generalization to unseen data. Instead of using the bias-variance tradeoff, we leverage the tradeoff between convergence speed and generalization capability in terms of optimizers. It has been well acknowledged that SGD has the averagely best generalization capability, while under some constraint of computational budget such as time, it may not be the best optimizer. Alternatively, most adaptive gradient descent algorithms using adaptive learning rates can enjoy the faster convergence rate, but unfortunately the ultimate generalization error is larger than that of SGD. The proposed AdaTrack may not thoroughly solve this problem, but leading us to explore another way for reducing variance, while maintaining the fast training convergence speed in adaptive gradient descent type of algorithms.

Comparison among different optimizers

A detailed comparison between different optimization algorithms is shown in Table 3.

Table 3: *Comparisons between different optimization algorithms*

Method	V.R.	I.P.	Regret	Regret Bound
SGD	–	–	Static	$\mathcal{O}(\sqrt{T})$
Adam	–	2	Static	$\mathcal{O}(\sqrt{T})$
AdaBound	Bounded L.R.	2	Static	$\mathcal{O}(\sqrt{T})$
RAdam	Rectification	4	–	–
AdaTrack	Gradient Tracking	3	Static	$\mathcal{O}(\sqrt{T})$
			Dynamic	$\mathcal{O}(\sqrt{T+P_T T})$

V.R.: variance reduction
I.P: intermediate parameters
P_T is defined below
L.R.: learning rate

Iterative Performance

We have observed that for the recursive stochastic gradient update (i.e., gradient tracking, SARAH, or SPIDER), an intermediate parameter is defined as the summation between the stochastic gradient and a correction term. The update for SARAH or SPIDER is expressed as

$$y_{t+1} = y_t + \nabla f_t(x_t, \zeta_t) - \nabla f_t(x_{t-1}, \zeta_t).$$

We can observe that in an iteration, the correction term necessitates the computation of the stochastic gradient of f at the point x_{t-1} using a sampled data at the current iteration, represented by ζ_t and the memory storage of x_{t-1}. However, based on Eq. 4,

$$y_{t+1} = y_t + \nabla f_t(x_t, \zeta_t) - \nabla f_{t-1}(x_{t-1}, \zeta_{t-1}).$$

the correction term only requires the memory storage of $\nabla f_{t-1}(x_{t-1}, \zeta_{t-1})$ from the last iteration. The memory storage for either x_{t-1} or $\nabla f_{t-1}(x_{t-1}, \zeta_{t-1})$ is the same in this context as the dimension for x is assumed to be $\mathbb{R}^d$ and f is a nonlinear mapping from $\mathbb{R}^d$ to $\mathbb{R}$. For an iteration of computation, SARAH or SPIDER requires $\mathcal{O}(d)$ more computation than the gradient tracking. So in T iterations, the extra computation is $\mathcal{O}(dT)$ for SARAH or SPIDER.

Dynamic regret

Proposition 1 reveals that using a static regret in Eq. 11, AdaTrack achieves the sublinear regret that matches the best result in online learning when the losses are convex. We also observe that the learning rate in Proposition 1 is diminishing, which is consistent with the setup in [7] for Adam. However, the static regret is not always applicable to be an effective metric due to the fixed x^*. We then naturally extend the regret to a dynamic scenario where it is expressed by the following equation

$$\mathcal{R}_T^D := \sum_{t=0}^{T}[f_t(x_t) - f_t(x_t^*)], \tag{15}$$

where $x_t^* = \text{argmin}_{x\in\mathcal{X}} f_t(x)$. Before characterizing the regret bound, we introduce another important concept to set a constraint on the unknown losses, which is a regularity for the path variation in terms of the optimizer.

Assumption 1 *An arbitrary unknown sequence of convex loss function, $\{f_0(x), f_1(x), ..., f_T(x)\}$ is assumed to be selected from the following set,*

$$\mathcal{P} := \left\{\{f_0, f_1, ..., f_T\} : \max_{x_t^*\in\text{argmin}_{x\in\mathcal{X}} f_t(x)} \sum_{t=0}^{T-1} \|x_t^* - x_{t+1}^*\| \leq P_T, P_T > 0\right\}, \tag{16}$$

where $\|\cdot\|$ is the l_2 norm.

This assumption signifies the bounded worst-case variation of the optimal solution x_t^* of $f_t(\cdot)$ along the trajectories $\{x_0^*, x_1^*, ...x_T^*\}$. Intuitively, such an assumption restricts that each optimal solution cannot deviate from the trajectory significantly. If for any $t \in [T], x_{t+1}^* = x_t^*$, $P_T = 0$ such that $\mathcal{R}_T^D$ degenerates to $\mathcal{R}_T^S$. With this assumption in hand, we have the following claim for AdaTrack.

Proposition 2 *Let Assumption 1 hold. Suppose that f_t is Lipschitz continuous and that $\mathcal{X}$ is compact. Let $\beta_1, \beta_2, \beta_3 \in [0, 1)$ satisfy $\frac{\beta_1^2}{\sqrt{\beta_2}} < 1$ and $\beta_{1,t} = \beta_1\lambda^{t-1}, \lambda \in (0, 1)$. Thus, for all $T \geq 0$, when the learning rate $\alpha_t = \mathcal{O}(\frac{1}{\sqrt{T}})$, AdaTrack has the sublinear regret, i.e.,*

$$\mathcal{R}_T^D = \mathcal{O}(\sqrt{T + P_T T}).$$

Proposition 2 shows clearly that due to the adoption of dynamic regret, the regret bound is more complex than that of Proposition 1. Roughly speaking, $\mathcal{R}_T^D$ still remains sublinear, but its order depends on the order of P_T. If $P_T = \mathcal{O}(T^{\frac{1}{3}})$, then $\mathcal{R}_T^D = \mathcal{O}(T^{\frac{2}{3}})$, which is obviously worse than that of $\mathcal{R}_T^S$. However, this intuitively makes sense as the dynamic regret also takes into account the variations of optimizers. For most deep learning models, they are highly non-convex and in this paper, we still assume that the loss functions are convex. This is because the regret instead of static error has been applied in this context. Different from offline non-convex learning, online non-convex learning is still an active research area where there is no a well-known regret bound, though the authors [18] showed a sublinear regret $\mathcal{O}(\sqrt{T})$, which required an approximate offline optimization oracle. This is unfortunately out of the scope of our work and will be left as a future direction. Optionally, one can leverage the computationally tractable notion of local regret proposed in [4], but this cannot guarantee vanishing regret for general non-convex losses.

Training details

Results shown in Table 2 were trained using a Nvidia Titan RTX GPU with codes implemented using PyTorch [15] framework. Specific training details and procedures for each dataset are listed below.

- **Fashion-MNIST**: We trained a convolutional neural network with three convolutional layers with 32, 64 and 64 filters respectively, followed by one max-pooling later and two fully-connected layers. A 3 x 3 filter size and stride values of 1 was used for all convolutional layers. ReLU activation functions were also used in between all layers. The model was trained for 200 epochs for this dataset. All optimizers were initialized with a initial learning rate of 1E-3, except SGD, which was initialized with a learning rate of 1E-1. For this dataset, we implemented a learning rate schedule which scales the initial learning rate by a factor of 0.1 at the 80th and 120th epoch. A constant weight decay of 5E-4 was also used for all optimizers.
- **SVHN**: A standard VGG-19 model architecture was used for this dataset. The model was trained for 200 epochs for this dataset and all optimizers were initialized with a initial learning rate of 1E-3, except SGD, which was initialized with a learning rate of 1E-1. For this dataset, we implemented a learning rate schedule which scales the initial learning rate by a factor of 0.1 at the 80th and 120th epoch. A constant weight decay of 5E-4 was also used for all optimizers.
- **CIFAR 10 and CIFAR 100**: For both CIFAR 10 and CIFAR 100 dataset, we train the model using a standard ResNet34 architecture. For CIFAR 10, the model was trained for 200 epochs and for CIFAR 100, the model was trained for 250 epochs. All optimizers were initialized with a initial learning rate of 1E-3, except SGD, which was initialized with a learning rate of 1E-1. For this dataset, we implemented a learning rate schedule which scales the initial learning rate by a factor of 0.1 at the 150th epoch. A constant weight decay of 5E-4 was also used for all optimizers.

Simple nonconvex functions

We leverage two nonconvex functions, i.e., Rastrigin and Rosenbrock, in this context to test the performance of different optimizers. Though Rastrigin and Rosenbrock functions are simple non-convex problems, they have been used widely to test the performance for many numerical optimizers. Also, they both possess only one global optimal solution such that it is easy to evaluate the specific performance of any optimizer. In this context, we test four different baseline methods, SGD, Adam, AdaBound and RAdam with the proposed AdaTrack. According to Fig. 2, it can be visualized that SGD has the worse performance as it converges directly to a local optimum for either of them. Both Adam and AdaBound perform well on Rosenbrock, but fail to converge to the global optimum of Rastrigin function. RAdam and AdaTrack perform similarly as they all converge to the uniquely global optimum for either Rastrigin or Rosenbrock function. As analyzed before, AdaTrack utilizes a novel way that is different from what has been adopted in AdaBound as well as RAdam to reduce the variance. Hence, we use four benchmark image datasets to show the comparison among SGD, Adam and AdaTrack in terms of convergence speed (training performance) and generalization capability (testing accuracy). Additionally, we also compare our proposed schemes to the recently proposed AdaBound and RAdam to investigate empirically how the gradient tracking reduces variance.

Training scores

Table 4 details the training accuracies of each optimizer for the four image dataset shown in the main manuscript.

Additional Results

We provide additional experimental results for CIFAR 10 and CIFAR 100 datasets here. A ResNet20 architecture was used to train the model for the results shown in Fig. 3 and Table 5 while

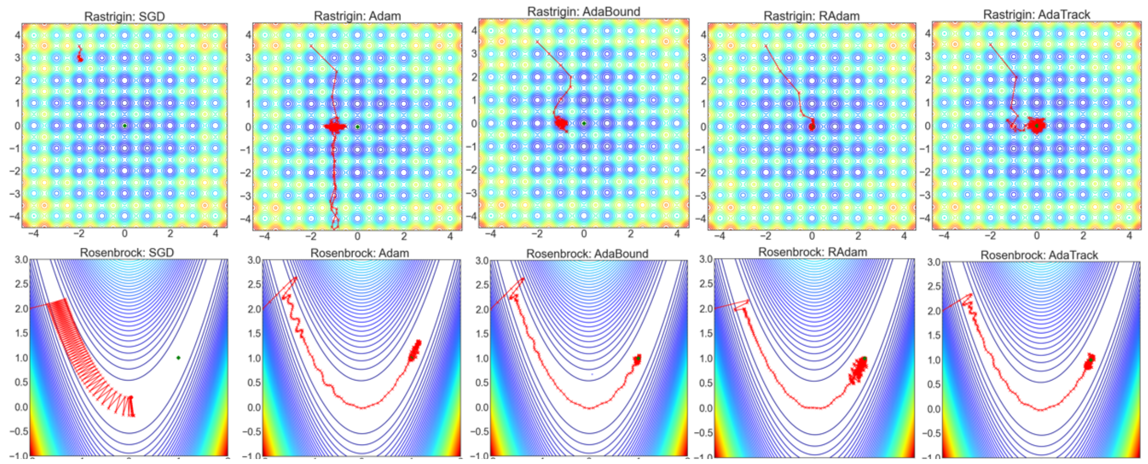

Figure 2: Convergence trajectories of different optimizers for Rastrigin and Rosenbrock functions. Green dots signify the global optima.

Table 4: Training accuracies of different optimizers corresponding to results in main manuscript.

	Fashion MNIST	SVHN	CIFAR 10	CIFAR 100
	Train Acc.(%)	Train Acc.(%)	Train Acc.(%)	Train Acc.(%)
SGD	98.7 ± 0.13	99.8 ± 0.03	98.6 ± 0.10	97.6 ± 0.12
Adam	98.6 ± 0.09	98.3 ± 0.41	99.3 ± 0.08	98.4 ± 0.15
Adabound	98.3 ± 0.12	99.9 ± 0.01	99.9 ± 0.01	99.9 ± 0.01
RAdam	99.5 ± 0.03	99.9 ± 0.01	99.9 ± 0.01	99.9 ± 0.01
AdaTrack	99.3 ± 0.06	99.9 ± 0.01	99.9 ± 0.01	99.9 ± 0.01
RAT	99.6 ± 0.08	99.9 ± 0.01	99.9 ± 0.01	99.9 ± 0.01

a VGG-19 architecture is used to train the model for the results shown in Fig. 4 and Table 6. The models were trained for 200 and 250 epochs for CIFAR 10 and CIFAR 100 respectively. An initial learning rate of 1E-3 was used for all optimizers except SGD, which had an initial learning rate of 1E-1. A learning rate schedule which scales the initial learning rate by a factor of 0.1 at the 80th and 120th epoch was used to train the ResNet20 models and a learning rate schedule which scales the initial learning rate by a factor of 0.1 at the 150th epoch was used to train the VGG-19 models . All optimizers had a weight-decay factor of 5E-4.

Table 5: Comparison of different optimizers for Cifar10 and Cifar100 datasets using ResNet20 architectures with two steps in the learning rate schedule.

	Cifar10		Cifar100	
	Train Acc.(%)	Test Acc.(%)	Train Acc.(%)	Test Acc.(%)
SGD	99.4 ± 0.06	91.9 ± 0.23	87.6 ± 0.17	67.2 ± 0.21
Adam	99.0 ± 0.13	91.0 ± 0.18	84.4 ± 0.34	66.6 ± 0.14
Adabound	98.9 ± 0.05	90.7 ± 0.19	83.3 ± 0.14	65.6 ± 0.18
RAdam	99.2 ± 0.05	89.7 ± 0.20	83.5 ± 0.35	63.0 ± 0.24
AdaTrack	99.2 ± 0.05	89.8 ± 0.38	83.6 ± 0.20	62.6 ± 0.45
RAT	99.2 ± 0.10	89.7 ± 0.29	83.5 ± 0.19	62.6 ± 0.58

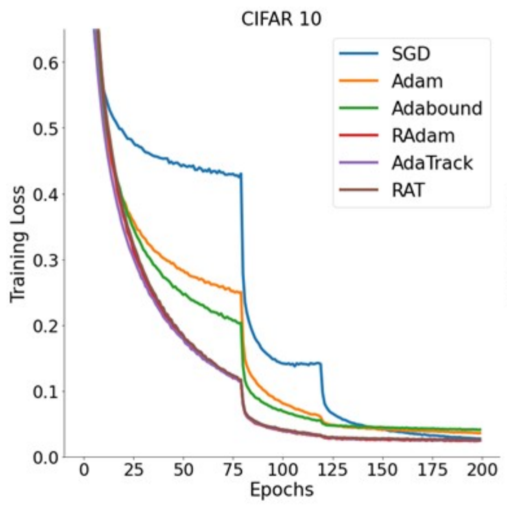

Figure 3: Additional results for CIFAR datasets using a ResNet20 architecture with two steps in the learning rate schedule.

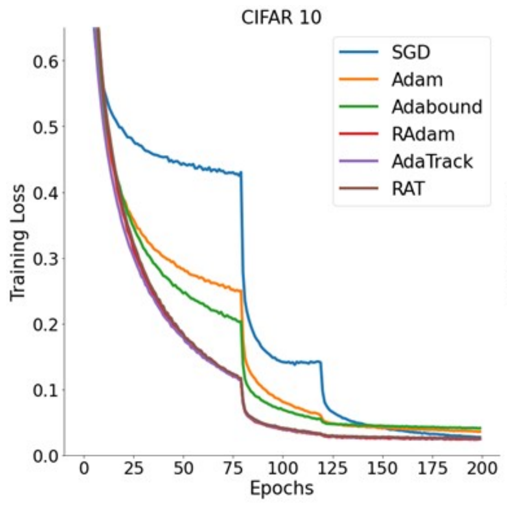

Figure 4: Additional results for CIFAR datasets using a VGG-19 architecture with one step in the learning rate schedule.

Table 6: Comparison of different optimizers for Cifar10 and Cifar100 datasets using VGG19 architectures with one step in the learning rate schedule.

	Cifar10		Cifar100	
	Train Acc.(%)	Test Acc.(%)	Train Acc.(%)	Test Acc.(%)
SGD	97.8 ± 0.08	90.7 ± 0.40	92.1 ± 0.46	65.9 ± 0.45
Adam	99.2 ± 0.08	91.8 ± 0.04	92.5 ± 2.11	65.0 ± 0.93
Adabound	99.9 ± 0.01	93.1 ± 0.11	99.9 ± 0.02	71.5 ± 0.22
RAdam	99.9 ± 0.01	93.2 ± 0.13	99.9 ± 0.01	70.7 ± 0.21
AdaTrack	99.9 ± 0.01	92.4 ± 0.16	99.8 ± 0.01	67.0 ± 0.25
RAT	99.9 ± 0.01	93.2 ± 0.21	99.9 ± 0.02	70.7 ± 0.28

Algorithm 2 Rectified AdaTrack

```
**Input**: $\alpha_t, \beta_1, \beta_2, \beta_3, \epsilon, x_0, \nabla f_{-1}(x_{-1}, \zeta_{-1}), m_0, v_0, y_0, t = 0$    ▷ Input params & initialization
$\rho_\infty = 2/(1-\beta_2) - 1$    ▷ Compute the maximum length of the approximated SMA
while $t \leq T$ do
    $m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla f_t(x_t, \zeta_t)$    ▷ Approximate first moment
    $v_{t+1} = \beta_2 v_t + (1-\beta_2)\nabla f_t^2(x_t, \zeta_t)$    ▷ Approximate second moment
    $y_{t+1} = \beta_3 y_t + (1-\beta_3)(\nabla f_t(x_t, \zeta_t) - \nabla f_{t-1}(x_{t-1}, \zeta_{t-1}))$    ▷ EMA of gradient tracking
    $\hat{m}_{t+1} = \frac{m_{t+1}}{1-\beta_1^t}$    ▷ Bias-correction for first moment
    $\hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^t}$    ▷ Bias-correction for second moment
    $\rho_t = \rho_\infty - 2t\beta_2^t/(1-\beta_2^t)$    ▷ Compute the length of the approximated SMA
    $\hat{y}_{t+1} = \frac{y_{t+1}}{1-\beta_3^t}$    ▷ Bias-correction for adaptive gradient tracking
        if the variance is tractable, i.e., $\rho_t > 4$ then
            $r_t = \sqrt{\frac{(\rho_t-4)(\rho_t-2)\rho_\infty}{(\rho_\infty-4)(\rho_\infty-2)\rho_t}}$    ▷ Compute the rectification term
            $x_{t+1} = x_t - \alpha_t r_t \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}+\epsilon}} - \alpha_t \hat{y}_{t+1}$    ▷ Update using adaptive gradient tracking descent
        else
            $x_{t+1} = x_t - \alpha_t \hat{m}_{t+1} - \alpha_t \hat{y}_{t+1}$    ▷ Update parameters with un-adapted momentum
    $t = t + 1$
return $x_T$
```