Adan: Adaptive Nesterov Momentum Algorithm for Faster Optimizing Deep Models

Xingyu Xie, Pan Zhou, Huan Li, Zhouchen Lin, *Fellow, IEEE*, and Shuicheng Yan, *Fellow, IEEE*

***Abstract*—In deep learning, different kinds of deep networks typically need different optimizers, which have to be chosen after multiple trials, making the training process inefficient. To relieve this issue and consistently improve the model training speed across deep networks, we propose the ADAptive Nesterov momentum algorithm, Adan for short. Adan first reformulates the vanilla Nesterov acceleration to develop a new Nesterov momentum estimation (NME) method, which avoids the extra overhead of computing gradient at the extrapolation point. Then Adan adopts NME to estimate the gradient's first- and second-order moments in adaptive gradient algorithms for convergence acceleration. Besides, we prove that Adan finds an ϵ-approximate first-order stationary point within $\mathcal{O}(\epsilon^{-3.5})$ stochastic gradient complexity on the non-convex stochastic problems (e.g., deep learning problems), matching the best-known lower bound. Extensive experimental results show that Adan consistently surpasses the corresponding SoTA optimizers on vision, language, and RL tasks and sets new SoTAs for many popular networks and frameworks, e.g., ResNet, ConvNext, ViT, Swin, MAE, DETR, GPT-2, Transformer-XL, and BERT. More surprisingly, Adan can use half of the training cost (epochs) of SoTA optimizers to achieve higher or comparable performance on ViT, GPT-2, MAE, *etc*, and also shows great tolerance to a large range of minibatch size, e.g., from 1 k to 32 k.**

***Index Terms*—Adaptive optimizer, fast DNN training, DNN optimizer.**

Manuscript received 14 February 2023; revised 18 April 2024; accepted 23 June 2024. Date of publication 4 July 2024; date of current version 5 November 2024. The work of Zhouchen Lin was supported in part by the National Key R&D Program of China under Grant 2022ZD0160300, in part by NSF China under Grant 62276004, and in part by Qualcomm. The work of Pan Zhou was supported in part by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant. Recommended for acceptance by S. J. Pan. *(Xingyu Xie and Pan Zhou equally contributed to this work.) (Corresponding authors: Zhouchen Lin; Shuicheng Yan.)*

Xingyu Xie is with the State Key Lab of General AI, School of Intelligence Science and Technology, Peking University, Beijing 100871, China (e-mail: xyxie@pku.edu.cn).

Pan Zhou is with the School of Computing and Information Systems, Singapore Management University, Singapore 188065.

Huan Li is with the Institute of Robotics and Automatic Information Systems, College of Artificial Intelligence, Nankai University, Tianjin 300192, China.

Zhouchen Lin is with the State Key Lab of General AI, School of Intelligence Science and Technology, Institute for Artificial Intelligence, Peking University, Beijing 100871, China, and also with Pazhou Laboratory (Huangpu), Guangzhou 510335, China (e-mail: zlin@pku.edu.cn).

Shuicheng Yan is with Sea AI Lab, Singapore 138680, and also with Skywork AI, Singapore 018936 (e-mail: shuicheng.yan@gmail.com).

Code is released at https://github.com/sail-sg/Adan, and has been used in multiple popular deep learning frameworks or projects.

This article has supplementary downloadable material available at https://doi.org/10.1109/TPAMI.2024.3423382, provided by the authors.

This article has supplementary downloadable material available at https://doi.org/10.1109/TPAMI.2024.3423382, provided by the authors.

Digital Object Identifier 10.1109/TPAMI.2024.3423382

I. Introduction

DEEP neural networks (DNNs) have made remarkable success in many fields, e.g., computer vision [1], [2], [3], [4] and natural language processing [5], [6]. A noticeable part of such success is contributed by the stochastic gradient-based optimizers, which find satisfactory solutions with high efficiency. Among current deep optimizers, SGD [7], [8] is the earliest and also the most representative stochastic optimizer, with dominant popularity for its simplicity and effectiveness. It adopts a single common learning rate for all gradient coordinates but often suffers unsatisfactory convergence speed on sparse data or ill-conditioned problems. In recent years, adaptive gradient algorithms [9], [10], [11], [12], [13], [14], [15], [16] have been proposed, which adjusts the learning rate for each gradient coordinate according to the current geometry curvature of the loss objective. These adaptive algorithms often offer a faster convergence speed than SGD in practice across many DNN frameworks.

However, none of the above optimizers can always stay undefeated among all its competitors across different DNN architectures and applications. For instance, for vanilla ResNets [2], SGD often achieves better generalization performance than adaptive gradient algorithms such as Adam [17], whereas on vision transformers (ViTs) [18], [19], [20], SGD often fails, and AdamW [21] is the dominant optimizer with higher and more stable performance. Moreover, these commonly used optimizers usually fail for large-batch training, which is a default setting of the prevalent distributed training. Although there is some performance degradation, we still tend to choose the large-batch setting for large-scale deep learning training tasks due to the unaffordable training time. For example, training the ViT-B with the batch size of 512 usually takes several days, but when the batch size comes to 32 K, we may finish the training within three hours [22]. Although some methods, e.g., LARS [23] and LAMB [24], have been proposed to handle large batch sizes, their performance may varies significantly across DNN architectures. This performance inconsistency increases the training cost and engineering burden since one has to try various optimizers for different architectures or training settings. This paper aims at relieving this issue.

When we rethink the current adaptive gradient algorithms, we find that they mainly combine the moving average idea with the heavy ball acceleration technique to estimate the first- and second-order moments of the gradient [15], [17], [21], [24], [25]. However, previous studies [26], [27], [28] have revealed that Nesterov acceleration can theoretically achieve a faster

convergence speed than heavy ball acceleration, as it uses gradient at an extrapolation point of the current solution and sees a slight "future". The ability to see the "future" may help optimizers better utilize the curve information of the dynamic training trajectory and show better robustness to DNN architectures. Moreover, recent works [29], [30] have shown the potential of Nesterov acceleration for large-batch training. Thus we are inspired to consider efficiently integrating Nesterov acceleration with adaptive algorithms.

The contributions of our work include: 1) We propose an efficient DNN optimizer, named Adan. Adan develops a Nesterov momentum estimation method to estimate stable and accurate first- and second-order moments of the gradient in adaptive gradient algorithms for acceleration. 2) Moreover, Adan enjoys a provably faster convergence speed than previous adaptive gradient algorithms such as Adam. 3) Empirically, Adan shows superior performance over the SoTA deep optimizers across vision, language, and reinforcement learning (RL) tasks. So it is possible that the effort on trying different optimizers for different deep network architectures can be greatly reduced. Our *detailed* contributions are highlighted below.

First, we propose an efficient Nesterov-acceleration-induced deep learning optimizer termed Adan. Given a function f and the current solution $\boldsymbol{\theta}_k$, Nesterov acceleration [26], [27], [28] estimates the gradient $\mathbf{g}_k = \nabla f(\boldsymbol{\theta}'_k)$ at the extrapolation point $\boldsymbol{\theta}'_k = \boldsymbol{\theta}_k - \eta(1-\beta_1)\mathbf{m}_{k-1}$ with the learning rate η and momentum coefficient $\beta_1 \in (0,1)$, and updates the moving gradient average as $\mathbf{m}_k = (1-\beta_1)\mathbf{m}_{k-1} + \mathbf{g}_k$. Then it runs a step by $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\mathbf{m}_k$. Despite its theoretical advantages, the implementation of Nesterov acceleration in practice reveals several significant challenges: 1) The process requires estimating the gradient at a point, $\boldsymbol{\theta}'_k$, which is not the current parameter set but an extrapolation. This two-step operation involves storing the original parameters $\boldsymbol{\theta}_k$ and updating them to $\boldsymbol{\theta}'_k$ solely for the purpose of gradient computation. Such a mechanism increases complexity in implementation, adds computational overhead, and escalates memory demands within deep learning frameworks. This additional step interrupts the workflow, as the optimizer cannot proceed directly to forward propagation without first completing this parameter extrapolation, thereby complicating the training process; 2) Distributed training, essential for handling large models, splits optimizer states and model weights across multiple GPUs. The requirement to manually update model weights to reflect the extrapolated position introduces significant communication burdens. Effective synchronization of these updates and their corresponding gradients across various nodes is crucial but challenging, often leading to inefficiencies and potential delays. To resolve the above incompatibility issues, we propose an alternative Nesterov momentum estimation (NME). We compute the gradient $\mathbf{g}_k = \nabla f(\boldsymbol{\theta}_k)$ at the current solution $\boldsymbol{\theta}_k$, and estimate the moving gradient average as $\mathbf{m}_k = (1-\beta_1)\mathbf{m}_{k-1} + \mathbf{g}'_k$, where $\mathbf{g}'_k = \mathbf{g}_k + (1-\beta_1)(\mathbf{g}_k - \mathbf{g}_{k-1})$. Our NME is provably equivalent to the vanilla one yet can avoid the extra cost. Then by regarding $\mathbf{g}'_k$ as the current stochastic gradient in adaptive gradient algorithms, e.g., Adam, we accordingly estimate the first- and second-moments as $\mathbf{m}_k = (1-\beta_1)\mathbf{m}_{k-1} + \beta_1\mathbf{g}'_k$ and $\mathbf{n}_k = (1-\beta_2)\mathbf{n}_{k-1} + \beta_2(\mathbf{g}'_k)^2$, respectively. Finally, we update $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\mathbf{m}_k/(\sqrt{\mathbf{n}_k} + \varepsilon)$. In this way, Adan enjoys the merit of Nesterov acceleration, namely faster convergence speed and tolerance to large minibatch size [39], which is verified in our experiments in Section V.

Second, as shown in Table I, we theoretically justify the advantages of Adan over previous SoTA adaptive gradient algorithms on nonconvex stochastic problems.

- Given the Lipschitz gradient condition, to find an ϵ-approximate first-order stationary point, Adan has the stochastic gradient complexity $\mathcal{O}(c_\infty^{2.5}\epsilon^{-4})$ which accords with the lower bound $\Omega(\epsilon^{-4})$ (up to a constant factor) [40]. This complexity is lower than $\mathcal{O}(c_2^6\epsilon^{-4})$ of Adabelief [15] and $\mathcal{O}(c_2^2 d\epsilon^{-4})$ of LAMB, especially on over-parameterized networks. Specifically, for the d-dimensional gradient, compared with its ℓ_2 norm c_2, its ℓ_∞ norm c_∞ is usually much smaller, and can be $\sqrt{d}\times$ smaller for the best case. Moreover, Adan's results still hold when the loss and the ℓ_2 regularizer are separated, which could significantly benefit the generalization [18] but the convergence analysis for Adam-type optimizers remains open.
- Given the Lipschitz Hessian condition, Adan has a complexity $\mathcal{O}(c_\infty^{1.25}\epsilon^{-3.5})$ which also matches the lower bound $\Omega(\epsilon^{-3.5})$ in [36]. This complexity is superior to $\mathcal{O}(\epsilon^{-3.5}\log\frac{c_2}{\epsilon})$ of A-NIGT [34] and also $\mathcal{O}(\epsilon^{-3.625})$ of Adam$^+$ [35]. Indeed, Adam$^+$ needs the minibatch size of order $\mathcal{O}(\epsilon^{-1.625})$ which is prohibitive in practice. For other optimizers, e.g., Adam, their convergence has not been provided yet under the Lipschitz Hessian condition.

Finally, Adan simultaneously surpasses the corresponding SoTA optimizers across vision, language, and RL tasks, and establishes new SoTAs for many networks and settings, e.g., ResNet, ConvNext [4], ViT [18], Swin [19], MAE [41], LSTM [42], Transformer-XL [43] and BERT [44]. More importantly, with half of the training cost (epochs) of SoTA optimizers, Adan can achieve higher or comparable performance on ViT, Swin, ResNet, *etc*. Besides, Adan works well in a large range of minibatch sizes, e.g., from 1 k to 32 k on ViTs. Due to the consistent improvement for various architectures and settings, Adan is supported in several popular deep learning frameworks or projects including Timm [45] (a library containing SoTA CV models), Optax from DeepMind [46] and MMClassification [47] from OpenMMLab, see our Github repo for more projects.

II. Related Work

Current DNN optimizers can be grouped into two families: SGD and its accelerated variants, and adaptive gradient algorithms. SGD computes stochastic gradient and updates the variable along the gradient direction. Later, heavy-ball acceleration [48] movingly averages stochastic gradient in SGD for faster convergence. Nesterov acceleration [28] runs a step along the moving gradient average and then computes the gradient at the new point to look ahead for correction. Typically, Nesterov

TABLE I
COMPARISON OF DIFFERENT ADAPTIVE GRADIENT ALGORITHMS ON NONCONVEX STOCHASTIC PROBLEMS

Smoothness Condition	Optimizer	Separated Reg.	Batch Size Condition.	Grad. Bound	Complexity	Lower Bound
Lipschitz Gradient	Adam-type [31]	✗	✗	$\ell_\infty \le c_\infty$	$\mathcal{O}(c_\infty^2 d\epsilon^{-4})$	$\Omega(\epsilon^{-4})$
	RMSProp [10], [37]	✗	✗	$\ell_\infty \le c_\infty$	$\mathcal{O}(\sqrt{c_\infty} d\epsilon^{-4})$	$\Omega(\epsilon^{-4})$
	AdamW [21]	✓	—	—	—	—
	Adabelief [15]	✗	✗	$\ell_2 \le c_2$	$\mathcal{O}(c_2^6 \epsilon^{-4})$	$\Omega(\epsilon^{-4})$
	Padam [38]	✗	✗	$\ell_\infty \le c_\infty$	$\mathcal{O}(\sqrt{c_\infty} d\epsilon^{-4})$	$\Omega(\epsilon^{-4})$
	LAMB [24]	✗	$\mathcal{O}(\epsilon^{-4})$	$\ell_2 \le c_2$	$\mathcal{O}(c_2^2 d\epsilon^{-4})$	$\Omega(\epsilon^{-4})$
	Adan (ours)	✓	✗	$\ell_\infty \le c_\infty$	$\mathcal{O}(c_\infty^{2.5} \epsilon^{-4})$	$\Omega(\epsilon^{-4})$
Lipschitz Hessian	A-NIGT [34]	✗	✗	$\ell_2 \le c_2$	$\mathcal{O}(\epsilon^{-3.5} \log \frac{c_2}{\epsilon})$	$\Omega(\epsilon^{-3.5})$
	Adam$^+$ [35]	✗	$\mathcal{O}(\epsilon^{-1.625})$	$\ell_2 \le c_2$	$\mathcal{O}(\epsilon^{-3.625})$	$\Omega(\epsilon^{-3.5})$
	Adan (ours)	✓	✗	$\ell_\infty \le c_\infty$	$\mathcal{O}(c_\infty^{1.25} \epsilon^{-3.5})$	$\Omega(\epsilon^{-3.5})$

"Separated Reg." refers to whether the ℓ_2 regularizer (weight decay) can be separated from the loss objective like AdamW. "Complexity" denotes stochastic gradient complexity to find an ϵ-approximate first-order stationary point. Adam-type methods [31] includes Adam, AdaMomentum [32], and AdaGrad [9], AdaBound [13] and AMSGrad [11], *etc*. AdamW has no available convergence result. For SAM [33], A-NIGT [34] and Adam$^+$ [35], we compare their adaptive versions. d is the variable dimension. The lower bound is proven in [36] and please see Section A in the supplementary for the discussion on why the lower bound is $\Omega(\epsilon^{-3.5})$.

acceleration converges faster both empirically and theoretically at least on convex problems, and also has superior generalization results on DNNs [33], [49].

Unlike SGD, adaptive gradient algorithms, e.g., AdaGrad [9], RMSProp [10] and Adam, view the second momentum of gradient as a precontioner and also use moving gradient average to update the variable. Later, many variants have been proposed to estimate more accurate and stable first momentum of gradient or its second momentum, e.g., AMSGrad [11], Adabound [13], and Adabelief [15]. To avoid gradient collapse, AdamP [16] proposes to clip gradient adaptively. Radam [14] reduces gradient variance to stabilize training. To improve generalization, AdamW [21] splits the objective and trivial regularization, and its effectiveness is validated across many applications; SAM [33] and its variants [22], [49], [50] aim to find flat minima but need forward and backward twice per iteration. LARS [23] and LAMB [24] train DNNs with a large batch but suffer unsatisfactory performance on small batch. [51] reveal the generalization and convergence gap between Adam and SGD from the perspective of diffusion theory and propose the optimizers, Adai, which accelerates the training and provably favors flat minima. Padam [38] provides a simple but effective way to improve the generalization performance of Adam by adjusting the second-order moment in Adam. The most related work to ours is NAdam [52]. It simplifies Nesterov acceleration to estimate the first moment of the gradient in Adam. But its acceleration does not use any gradient from the extrapolation points and thus does not look ahead for correction. Moreover, there is no theoretical result to ensure its convergence. See more difference discussion in Section III-B, especially for (3).

In addition to the optimization techniques that form the core focus of our work, it is pertinent to acknowledge the breadth of research dedicated to enhancing training efficiency across various domains. Notable among these is the domain of data augmentation, where techniques such as mixup have been proposed, which performs the training on convex combinations of pairs of examples and their labels [53], [54]. This approach significantly enriches the training data without the need for additional data collection. Furthermore, innovative training strategies play a crucial role in the efficient training of compact deep neural networks. For instance, the concept of multi-way BP offers a more efficient gradient calculation mechanism [55]. Additionally, the design of loss functions, as explored in works like Sphere Loss [56], introduces novel approaches to learning discriminative features. Each of these areas contributes to the overarching goal of training efficiency, offering complementary avenues to optimization techniques.

III. METHODOLOGY

In this work, we study the following regularized nonconvex optimization problem:

$$\min_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) := \mathop{\mathbb{E}}_{\boldsymbol{\zeta}\sim\mathcal{D}}[f(\boldsymbol{\theta},\boldsymbol{\zeta})] + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2, \tag{1}$$

where loss $f(\cdot,\cdot)$ is differentiable and possibly nonconvex, data $\boldsymbol{\zeta}$ is drawn from an unknown distribution $\mathcal{D}$, $\boldsymbol{\theta}$ is learnable parameters, and $\|\cdot\|$ is the classical ℓ_2 norm. Here we consider the ℓ_2 regularizer as it can improve generalization performance and is widely used in practice [21]. The formulation (1) encapsulates a large body of machine learning problems, e.g., network training problems, and least square regression. Below, we first introduce the key motivation of Adan in Section III-A, and then give detailed algorithmic steps in Section III-B.

A. Preliminaries

Adaptive gradient algorithms, Adam [17] and AdamW [21], have become the default choice to train CNNs and ViTs. Unlike SGD which uses one learning rate for all gradient coordinates, adaptive algorithms adjust the learning rate for each gradient coordinate according to the current geometry curvature of the objective function, and thus converge faster. Take RMSProp [10] and Adam [17] as examples. Given stochastic gradient estimator $\mathbf{g}_k := \mathbb{E}_{\boldsymbol{\zeta}\sim\mathcal{D}}[\nabla f(\boldsymbol{\theta}_k,\boldsymbol{\zeta})] + \boldsymbol{\xi}_k$, e.g., minibatch gradient, where

$\boldsymbol{\xi}_k$ is the gradient noise, RMSProp updates the variable $\boldsymbol{\theta}$ as follows:

$$\text{RMSProp:} \begin{cases} \mathbf{n}_k = (1-\beta)\mathbf{n}_{k-1} + \beta \mathbf{g}_k^2 \\ \boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \boldsymbol{\eta}_k \circ \mathbf{g}_k, \end{cases}$$

where $\boldsymbol{\eta}_k := \eta/(\sqrt{\mathbf{n}_k} + \varepsilon)$, $\mathbf{m}_0 = \mathbf{g}_0$, $\mathbf{n}_0 = \mathbf{g}_0^2$, the scalar η is the base learning rate, $\circ$ denotes the element-wise product, and the vector square and the vector-to-vector or scalar-to-vector root in this paper are both element-wise.

Based on RMSProp, Adam (for presentation convenience, we omit the de-bias term in adaptive gradient methods), as follows, replaces the estimated gradient $\mathbf{g}_k$ with a moving average $\mathbf{m}_k$ of all previous gradient $\mathbf{g}_k$.

$$\text{Adam:} \begin{cases} \mathbf{m}_k = (1-\beta_1)\mathbf{m}_{k-1} + \beta_1 \mathbf{g}_k \\ \mathbf{n}_k = (1-\beta_2)\mathbf{n}_{k-1} + \beta_2 \mathbf{g}_k^2 \\ \boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \boldsymbol{\eta}_k \circ \mathbf{m}_k, \end{cases}$$

By inspection, one can easily observe that the moving average idea in Adam is similar to the classical (stochastic) heavy-ball acceleration (HBA) technique [48]

$$\text{HBA:} \begin{cases} \mathbf{g}_k = \nabla f(\boldsymbol{\theta}_k) + \boldsymbol{\xi}_k \\ \mathbf{m}_k = (1-\beta_1)\mathbf{m}_{k-1} + \mathbf{g}_k \\ \boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \mathbf{m}_k, \end{cases}$$

Both Adam and HBA share the spirit of moving gradient average, though HBA does not have the factor β_1 on the gradient $\mathbf{g}_k$. That is, given one gradient coordinate, if its gradient directions are more consistent along the optimization trajectory, Adam/HBA accumulates a larger gradient value in this direction and thus goes ahead for a bigger gradient step, which accelerates convergence.

In addition to HBA, Nesterov's accelerated (stochastic) gradient descent (AGD) [26], [27], [28] is another popular acceleration technique in the optimization community

$$\text{AGD:} \begin{cases} \mathbf{g}_k = \nabla f(\boldsymbol{\theta}_k - \eta(1-\beta_1)\mathbf{m}_{k-1}) + \boldsymbol{\xi}_k \\ \mathbf{m}_k = (1-\beta_1)\mathbf{m}_{k-1} + \mathbf{g}_k \\ \boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \mathbf{m}_k \end{cases}. \quad (2)$$

Unlike HBA, AGD uses the gradient at the extrapolation point $\boldsymbol{\theta}'_k = \boldsymbol{\theta}_k - \eta(1-\beta_1)\mathbf{m}_{k-1}$. Hence when the adjacent iterates share consistent gradient directions, AGD sees a slight future to converge faster. Indeed, AGD theoretically converges faster than HBA and achieves optimal convergence rate among first-order optimization methods on the general smooth convex problems [28]. It also relaxes the convergence conditions of HBA on the strongly convex problems [27]. Meanwhile, since the over-parameterized DNNs have been observed/proved to have many convex-alike local basins [57], [58], [59], [60], [61], [62], [63], [64], AGD seems to be more suitable than HBA for DNNs. For large-batch training, [29] showed that AGD has the potential to achieve comparable performance to some specifically designed optimizers, e.g., LARS and LAMB. With its advantage and potential in convergence and large-batch training, we consider applying AGD to improve adaptive algorithms.

B. Adaptive Nesterov Momentum Algorithm

Main Iteration: We temporarily set $\lambda = 0$ in (1). As aforementioned, AGD computes gradient at an extrapolation point $\boldsymbol{\theta}'_k$ instead of the current iterate $\boldsymbol{\theta}_k$, which however brings extra computation and memory overhead for computing $\boldsymbol{\theta}'_k$ and preserving both $\boldsymbol{\theta}_k$ and $\boldsymbol{\theta}'_k$. To solve the issue, Lemma 1 with proof in supplementary Section C.1 reformulates AGD (2) into its equivalent but more DNN-efficient version.

Lemma 1: Assume $\mathbb{E}(\boldsymbol{\xi}_k) = \mathbf{0}$, $\text{Cov}(\boldsymbol{\xi}_i, \boldsymbol{\xi}_j) = 0$ for any $k, i, j > 0$, $\bar{\boldsymbol{\theta}}_k$ and $\bar{\mathbf{m}}_k$ be the iterate and momentum of the vanilla AGD in (2), respectively. Let $\boldsymbol{\theta}_{k+1} := \bar{\boldsymbol{\theta}}_{k+1} - \eta(1-\beta_1)\bar{\mathbf{m}}_k$ and $\mathbf{m}_k := (1-\beta_1)^2\bar{\mathbf{m}}_{k-1} + (2-\beta_1)(\nabla f(\boldsymbol{\theta}_k) + \boldsymbol{\xi}_k)$. The vanilla AGD in (2) becomes AGD-II

$$\text{AGD II:} \begin{cases} \mathbf{g}_k = \mathbb{E}_{\boldsymbol{\zeta}\sim\mathcal{D}}[\nabla f(\boldsymbol{\theta}_k, \boldsymbol{\zeta})] + \boldsymbol{\xi}_k \\ \mathbf{m}_k = (1-\beta_1)\mathbf{m}_{k-1} + \mathbf{g}'_k \\ \boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \mathbf{m}_k \end{cases},$$

where $\mathbf{g}'_k := \mathbf{g}_k + (1-\beta_1)(\mathbf{g}_k - \mathbf{g}_{k-1})$. Moreover, if vanilla AGD in (2) converges, so does AGD-II: $\mathbb{E}(\boldsymbol{\theta}_\infty) = \mathbb{E}(\bar{\boldsymbol{\theta}}_\infty)$.

The main idea in Lemma 1 is that we maintain $(\boldsymbol{\theta}_k - \eta(1-\beta_1)\mathbf{m}_{k-1})$ rather than $\boldsymbol{\theta}_k$ in vanilla AGD at each iteration since there is no difference between them when the algorithm converges. Like other adaptive optimizers, by regarding $\mathbf{g}'_k$ as the current stochastic gradient and movingly averaging $\mathbf{g}'_k$ to estimate the first- and second-moments of gradient, we obtain

$$\text{Vanilla Adan:} \begin{cases} \mathbf{m}_k = (1-\beta_1)\mathbf{m}_{k-1} + \beta_1 \mathbf{g}'_k \\ \mathbf{n}_k = (1-\beta_3)\mathbf{n}_{k-1} + \beta_3 (\mathbf{g}'_k)^2 \\ \boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \boldsymbol{\eta}_k \circ \mathbf{m}_k \end{cases},$$

where $\mathbf{g}'_k := \mathbf{g}_k + (1-\beta_1)(\mathbf{g}_k - \mathbf{g}_{k-1})$ and the vector square in the second line is element-wisely. The main difference of Adan with Adam-type methods and Nadam [52] is that, as compared in (3), the first-order moment $\mathbf{m}_k$ of Adan is the average of $\{\mathbf{g}_t + (1-\beta_1)(\mathbf{g}_t - \mathbf{g}_{t-1})\}_{t=1}^k$ while those of Adam-type and Nadam are the average of $\{\mathbf{g}_t\}_{t=1}^k$. So is their second-order term $\mathbf{n}_k$,

$$\mathbf{m}_k = \begin{cases} \sum_{t=0}^k c_{k,t}[\mathbf{g}_t + (1-\beta_1)(\mathbf{g}_t - \mathbf{g}_{t-1})], & \text{Adan}, \\ \sum_{t=0}^k c_{k,t}\mathbf{g}_t, & \text{Adam}, \\ \frac{\mu_{k+1}}{\mu'_{k+1}}\left(\sum_{t=0}^k c_{k,t}\mathbf{g}_t\right) + \frac{1-\mu_k}{\mu'_k}\mathbf{g}_k, & \text{Nadam}, \end{cases} \quad (3)$$

where $c_{k,t} = \beta_1(1-\beta_1)^{k-t}$ for $t > 0$ and $c_{k,t} = (1-\beta_1)^k$ for $t = 0$. $\mu_{t=1}^\infty$ is a predefined exponentially decaying sequence, $\mu'_k = 1 - \prod_{t=1}^k \mu_t$. Nadam is more like Adam than Adan, as their $\mathbf{m}_k$ averages the historical gradients instead of gradient differences in Adan. For the large k (i.e., small μ_k), $\mathbf{m}_k$ in Nadam and Adam are almost the same.

As shown in (3), the moment $\mathbf{m}_k$ in Adan consists of two terms, i.e., gradient term $\mathbf{g}_t$ and gradient difference term $(\mathbf{g}_t - \mathbf{g}_{t-1})$, which actually have different physic meanings. So here we decouple them for greater flexibility and also better trade-off between them. Specifically, we estimate

$$(\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k)/\boldsymbol{\eta}_k = \sum_{t=0}^k [c_{k,t}\mathbf{g}_t + (1-\beta_2)c'_{k,t}(\mathbf{g}_t - \mathbf{g}_{t-1})]$$

$$= \mathbf{m}_k + (1-\beta_2)\mathbf{v}_k, \tag{4}$$

where $c'_{k,t} = \beta_2(1-\beta_2)^{k-t}$ for $t>0$, $c'_{k,t} = (1-\beta_2)^k$ for $t=0$, and, with a little abuse of notation on $\mathbf{m}_k$, we let $\mathbf{m}_k$ and $\mathbf{v}_k$ be

$$\begin{cases} \mathbf{m}_k = (1-\beta_1)\mathbf{m}_{k-1} + \beta_1 \mathbf{g}_k \\ \mathbf{v}_k = (1-\beta_2)\mathbf{v}_{k-1} + \beta_2(\mathbf{g}_k - \mathbf{g}_{k-1}) \end{cases}.$$

This change for a flexible estimation does not impair convergence speed. As shown in Theorem 1, Adan's convergence complexity still matches the best-known lower bound. We do not separate the gradients and their difference in the second-order moment $\mathbf{n}_k$, since $\mathbb{E}(\mathbf{n}_k)$ contains the correlation term $\mathrm{Cov}(\mathbf{g}_k, \mathbf{g}_{k-1}) \neq 0$ which may have statistical significance.

Decay Weight by Proximation: As observed in AdamW, decoupling the optimization objective and simple-type regularization (e.g., ℓ_2 regularizer) can largely improve the generalization performance. Here we follow this idea but from a rigorous optimization perspective. Intuitively, at each iteration $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \boldsymbol{\eta}_k \circ \bar{\mathbf{m}}_k$, we minimize the first-order approximation of $F(\cdot)$ at the point $\boldsymbol{\theta}_k$

$$\boldsymbol{\theta}_{k+1} = \operatorname*{argmin}_{\boldsymbol{\theta}} \left(F(\boldsymbol{\theta}_k) + \langle \bar{\mathbf{m}}_k, \boldsymbol{\theta} - \boldsymbol{\theta}_k \rangle + \frac{1}{2\eta} \|\boldsymbol{\theta} - \boldsymbol{\theta}_k\|^2_{\sqrt{\mathbf{n}_k}} \right),$$

where $\|\mathbf{x}\|^2_{\sqrt{\mathbf{n}_k}} := \langle \mathbf{x}, (\sqrt{\mathbf{n}_k} + \varepsilon) \circ \mathbf{x} \rangle$ and $\bar{\mathbf{m}}_k := \mathbf{m}_k + (1-\beta_2)\mathbf{v}_k$ is the first-order derivative of $F(\cdot)$ in some sense. Follow the idea of proximal gradient descent [65], [66], we decouple the ℓ_2 regularizer from $F(\cdot)$ and only linearize the loss function $f(\cdot)$

$$\begin{aligned} \boldsymbol{\theta}_{k+1} &= \operatorname*{argmin}_{\boldsymbol{\theta}} \left(F'_k(\boldsymbol{\theta}) + \langle \bar{\mathbf{m}}_k, \boldsymbol{\theta} - \boldsymbol{\theta}_k \rangle + \frac{1}{2\eta} \|\boldsymbol{\theta} - \boldsymbol{\theta}_k\|^2_{\sqrt{\mathbf{n}_k}} \right) \\ &= \frac{\boldsymbol{\theta}_k - \boldsymbol{\eta}_k \circ \bar{\mathbf{m}}_k}{1 + \lambda_k \eta}, \end{aligned} \tag{5}$$

where $F'_k(\boldsymbol{\theta}) := \mathbb{E}_{\boldsymbol{\zeta}\sim\mathcal{D}}[f(\boldsymbol{\theta}_k, \boldsymbol{\zeta})] + \frac{\lambda_k}{2}\|\boldsymbol{\theta}\|^2_{\sqrt{\mathbf{n}_k}}$, and $\lambda_k > 0$ is the weight decay at the kth iteration. Interestingly, we can easily reveal the updating rule $\boldsymbol{\theta}_{k+1} = (1-\lambda\eta)\boldsymbol{\theta}_k - \boldsymbol{\eta}_k \circ \bar{\mathbf{m}}_k$ of AdamW by using the first-order approximation of (5) around $\eta = 0$: 1) $(1+\lambda\eta)^{-1} = (1-\lambda\eta) + \mathcal{O}(\eta^2)$; 2) $\lambda\eta\boldsymbol{\eta}_k = \mathcal{O}(\eta^2)/(\sqrt{\mathbf{n}_k} + \varepsilon)$. One can find that the optimization objective of Separated Regularization at the kth iteration is changed from the vanilla "static" function $F(\cdot)$ in (1) to a "dynamic" function $F_k(\cdot)$ in (6), which adaptively regularizes the coordinates with larger gradient more

$$F_k(\boldsymbol{\theta}) := \mathop{\mathbb{E}}_{\boldsymbol{\zeta}\sim\mathcal{D}}[f(\boldsymbol{\theta}, \boldsymbol{\zeta})] + \frac{\lambda_k}{2}\|\boldsymbol{\theta}\|^2_{\sqrt{\mathbf{n}_k}}. \tag{6}$$

We summarize our Adan in Algorithm 1. We reset the momentum term properly by the restart condition, a common trick to stabilize optimization and benefit convergence [67], [68]. But to make Adan simple, in all experiments except Table 22, we do not use this restart strategy although it can improve performance as shown in Table 22.

Algorithm 1: Adan (Adaptive Nesterov Momentum Algorithm).

Input: initialization $\boldsymbol{\theta}_0$, step size η, momentum $(\beta_1, \beta_2, \beta_3) \in [0,1]^3$, stable parameter $\varepsilon > 0$, weight decay $\lambda_k > 0$, restart condition.
Output: some average of $\{\boldsymbol{\theta}_k\}_{k=1}^K$.

1 **while** $k < K$ **do**
2 estimate the stochastic gradient $\mathbf{g}_k$ at $\boldsymbol{\theta}_k$;
3 $\mathbf{m}_k = (1-\beta_1)\mathbf{m}_{k-1} + \beta_1 \mathbf{g}_k$;
4 $\mathbf{v}_k = (1-\beta_2)\mathbf{v}_{k-1} + \beta_2(\mathbf{g}_k - \mathbf{g}_{k-1})$;
5 $\mathbf{n}_k = (1-\beta_3)\mathbf{n}_{k-1} + \beta_3[\mathbf{g}_k + (1-\beta_2)(\mathbf{g}_k - \mathbf{g}_{k-1})]^2$;
6 $\boldsymbol{\eta}_k = \eta / \big(\sqrt{\mathbf{n}_k} + \varepsilon\big)$;
7 $\boldsymbol{\theta}_{k+1} = (1+\lambda_k\eta)^{-1}[\boldsymbol{\theta}_k - \boldsymbol{\eta}_k \circ (\mathbf{m}_k + (1-\beta_2)\mathbf{v}_k)]$;
8 **if** *restart condition holds* **then**
9 estimate stochastic gradient $\mathbf{g}_0$ at $\boldsymbol{\theta}_{k+1}$;
10 set $k = 1$ and update $\boldsymbol{\theta}_1$ by Line 7;
11 **end if**
12 **end while**

IV. Convergence Analysis

For analysis, we make several mild assumptions used in many works, e.g., [31], [32], [33], [34], [35], [37], [38], [49], [71]

Assumption 1 (L-smoothness): The function $f(\cdot,\cdot)$ is L-smooth w.r.t. the parameters. Denote $F(\mathbf{x}) := \mathbb{E}_{\boldsymbol{\zeta}}[f(\mathbf{x}, \boldsymbol{\zeta})]$. We have:

$$\|\nabla F(\mathbf{x}) - \nabla F(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|, \qquad \forall \mathbf{x}, \mathbf{y}.$$

Assumption 2 (Unbiased and bounded gradient oracle): The stochastic gradient oracle $\mathbf{g}_k = \mathbb{E}_{\boldsymbol{\zeta}}[\nabla f(\boldsymbol{\theta}_k, \boldsymbol{\zeta})] + \boldsymbol{\xi}_k$ is unbiased, i.e., $\mathbb{E}(\boldsymbol{\xi}_k) = \mathbf{0}$, and its magnitude and variance are bounded with probability 1

$$\|\mathbf{g}_k\|_\infty \leq c_\infty/3, \quad \mathbb{E}(\|\boldsymbol{\xi}_k\|^2) \leq \sigma^2, \quad \forall k \in [T].$$

Assumption 3 (ρ-Lipschitz continuous Hessian): The function $f(\cdot,\cdot)$ has ρ-Lipschitz Hessian w.r.t. the parameters.

$$\|\nabla^2 F(\mathbf{x}) - \nabla^2 F(\mathbf{y})\| \leq \rho\|\mathbf{x} - \mathbf{y}\|, \qquad \forall \mathbf{x}, \mathbf{y},$$

where $F(\mathbf{x}) := \mathbb{E}_{\boldsymbol{\zeta}}[f(\mathbf{x}, \boldsymbol{\zeta})]$, $\|\cdot\|$ is matrix spectral norm for matrix and ℓ_2 norm for vector.

For a general nonconvex problem, if Assumptions 1 and 2 hold, the lower bound of the stochastic gradient complexity (a.k.a. IFO complexity) to find an ϵ-approximate first-order stationary point (ϵ-ASP) is $\Omega(\epsilon^{-4})$ [40]. Moreover, if Assumption 3 further holds, the lower complexity bound becomes $\Omega(\epsilon^{-3.5})$ for a non-variance-reduction algorithm [36].

Lipschitz Gradient: Theorem 1 with proof in Supplementary Section C.2 proves the convergence of Adan on problem (6) with Lipschitz gradient condition.

Theorem 1: Suppose that Assumptions 1 and 2 hold. Let $\max \beta_1, \beta_2 = \mathcal{O}(\epsilon^2)$, $\mu := \sqrt{2\beta_3}c_\infty/\varepsilon \ll 1$, $\eta = \mathcal{O}(\epsilon^2)$, and $\lambda_k = \lambda(1-\mu)^k$. Algorithm 1 runs at most $K = \Omega(c_\infty^{2.5}\epsilon^{-4})$ iterations to achieve

$$\frac{1}{K+1}\sum_{k=0}^{K} \mathbb{E}(\|\nabla F_k(\boldsymbol{\theta}_k)\|^2) \leq 4\epsilon^2.$$

That is, to find an ϵ-ASP, the stochastic gradient complexity of Adan on problem (6) is $\mathcal{O}(c_\infty^{2.5}\epsilon^{-4})$.

Theorem 1 shows that under Assumptions 1 and 2, Adan can converge to an ϵ-ASP of a nonconvex stochastic problem with stochastic gradient complexity $\mathcal{O}(c_\infty^{2.5}\epsilon^{-4})$ which accords with the lower bound $\Omega(\epsilon^{-4})$ in [40]. For this convergence, Adan has no requirement on minibatch size and only assumes gradient estimation to be unbiased and bounded. Moreover, as shown in Table I in Section I, the complexity of Adan is superior to those of previous adaptive gradient algorithms. For Adabelief and LAMB, Adan always has lower complexity and respectively enjoys $d^3\times$ and $d^2\times$ lower complexity for the worst case. Adam-type optimizers (e.g., Adam and AMSGrad) enjoy the same complexity as Adan. But they cannot separate the ℓ_2 regularizer with the objective like AdamW and Adan. Namely, they always solve a static loss $F(\cdot)$ rather than a dynamic loss $F_k(\cdot)$. The regularizer separation can boost generalization performance [18], [19] and already helps AdamW dominate training of ViT-alike architectures. Besides, some previous analyses [13], [14], [70], [72] need the momentum coefficient (i.e., βs) to be close or increased to one, which contradicts with the practice that βs are close to zero. In contrast, Theorem 1 assumes that all βs are very small, which is more consistent with the practice. Note that when $\mu = c/T$, we have $\lambda_k/\lambda \in [(1-c), 1]$ during training. Hence we could choose the λ_k as a fixed constant in the experiment for convenience.

Lipschitz Hessian: To further improve the theoretical convergence speed, we introduce Assumption 3, and set a proper restart condition to reset the momentum during training. Consider an extension point $\mathbf{y}_{k+1} := \boldsymbol{\theta}_{k+1} + \boldsymbol{\eta}_k \circ [\mathbf{m}_k + (1-\beta_2)\mathbf{v}_k - \beta_1 \mathbf{g}_k]$, and the restart condition is

$$(k+1)\sum_{t=0}^{k} \|\mathbf{y}_{t+1} - \mathbf{y}_t\|^2_{\sqrt{\mathbf{n}_t}} > R^2, \tag{7}$$

where the constant R controls the restart frequency. Intuitively, when the parameters have accumulated enough updates, the iterate may reach a new local basin. Resetting the momentum at this moment helps Adan to better use the local geometric information. Besides, we change $\boldsymbol{\eta}_k$ from $\eta/(\sqrt{\mathbf{n}_k}+\varepsilon)$ to $\eta/(\sqrt{\mathbf{n}_{k-1}}+\varepsilon)$ to ensure $\boldsymbol{\eta}_k$ to be independent of noise $\boldsymbol{\zeta}_k$. See its proof in Supplementary C.3.

Theorem 2: Suppose that Assumptions 1–3 hold. Let $R = \mathcal{O}(\epsilon^{0.5})$, $\max\beta_1, \beta_2 = \mathcal{O}(\epsilon^2)$, $\beta_3 = \mathcal{O}(\epsilon^4)$, $\eta = \mathcal{O}(\epsilon^{1.5})$, $K = \mathcal{O}(\epsilon^{-2})$, $\lambda = 0$. Then Algorithm 1 with restart condition (7) satisfies

$$\mathbb{E}(\|\nabla F_k(\bar{\boldsymbol{\theta}})\|) = \mathcal{O}(c_\infty^{0.5}\epsilon), \quad \text{where } \bar{\boldsymbol{\theta}} := \frac{1}{K_0}\sum_{k=1}^{K_0}\boldsymbol{\theta}_k,$$

and $K_0 = \operatorname{argmin}_{\lfloor \frac{K}{2}\rfloor \le k \le K-1} \|\mathbf{y}_{t+1} - \mathbf{y}_t\|^2_{\sqrt{\mathbf{n}_t}}$. Moreover, to find an ϵ-ASP, Algorithm 1 restarts at most $\mathcal{O}(c_\infty^{0.5}\epsilon^{-1.5})$ times in which each restarting cycle has at most $K = \mathcal{O}(\epsilon^{-2})$ iterations, and hence needs at most $\mathcal{O}(c_\infty^{1.25}\epsilon^{-3.5})$ stochastic gradient complexity.

From Theorem 2, one can observe that with an extra Hessian condition, Assumption 3, Adan improves its stochastic gradient complexity from $\mathcal{O}(c_\infty^{2.5}\epsilon^{-4})$ to $\mathcal{O}(c_\infty^{1.25}\epsilon^{-3.5})$, which also matches the corresponding lower bound $\Omega(\epsilon^{-3.5})$ [36]. This complexity is lower than $\mathcal{O}(\epsilon^{-3.5}\log\frac{c_2}{\epsilon})$ of A-NIGT [34] and $\mathcal{O}(\epsilon^{-3.625})$ of Adam$^+$ [35]. For other DNN optimizers, e.g., Adam, their convergence under Lipschitz Hessian condition has not been proved yet.

Moreover, Theorem 2 still holds for the large batch size. For example, by using minibatch size $b = \mathcal{O}(\epsilon^{-1.5})$, our results still hold when $R = \mathcal{O}(\epsilon^{0.5})$, $\max\beta_1, \beta_2 = \mathcal{O}(\epsilon^{0.5})$, $\beta_3 = \mathcal{O}(\epsilon)$, $\eta = \mathcal{O}(1)$, $K = \mathcal{O}(\epsilon^{-0.5})$, and $\lambda = 0$. In this case, our η is of the order $\mathcal{O}(1)$, and is much larger than $\mathcal{O}(\text{ploy}(\epsilon))$ of other optimizers (e.g., LAMB [24] and Adam$^+$) for handling large minibatch. This large step size often boosts convergence speed in practice, which is actually desired.

V. Experimental Results

We evaluate Adan on vision tasks, natural language processing (NLP) tasks and reinforcement learning (RL) tasks. For vision classification tasks, we test Adan on several representative SoTA backbones under the conventional supervised settings, including 1) CNN-type architectures (ResNets [2] and ConvNexts [4]) and 2) ViTs (ViTs [3], [18] and Swins [19]). We also investigate Adan via the self-supervised pretraining by using it to train MAE-ViT [41]. Moreover, we test Adan on the vision object detection and instance segmentation tasks with two frameworks Deformable-DETR [73] and Mask-RCNN [74] (choosing ConvNext [4] as the backbone). For NLP tasks, we train LSTM [42], Transformer-XL [43], and BERT [44] for sequence modeling. We also provide the Adan's results on large language models, like GPT-2 [75], on the code generation tasks. For more results on general large-language model can be found supplementary material, available online. On RL tasks, we evaluate Adan on four games in MuJoCo [76]. We also conduct the experiments on GNNs.

Due to space limitation, we defer the ablation study, additional experimental results, and implementation details into supplementary materials: We compare Adan with the model's default/SoTA optimizer in all the experiments but may miss some representative optimizers, e.g., Adai, Padam, and AdaBlief in some cases. This is because they report few results for larger-scale experiments. For instance, Adablief only tests ResNet-18 performance on ImageNet and actually does not test any other networks. So it is really hard for us to compare them on ViTs, Swins, ConvNext, MAEs, etc, due to the challenges for hyper-parameter tuning and limited GPU resources. The other reason is that some optimizers may fail or achieve poor performance on transformers. For example, SGD and Adam achieve much lower accuracy than AdamW. See Table 16 in supplementary materials.

A. Experiments for Vision Classification Tasks

1) Training Setting: Besides the vanilla supervised training setting used in ResNets [2], we further consider the following two prevalent training settings on ImageNet [77].

TABLE II
TOP-1 ACC. (%) OF RESNET AND CONVNEXT ON IMAGENET UNDER THE OFFICIAL SETTINGS

Epoch	ResNet-50 100	ResNet-50 200	ResNet-50 300	ResNet-101 100	ResNet-101 200	ResNet-101 300
SAM [33]	77.3	78.7	79.4	79.5	81.1	81.6
SGD-M [26], [27], [28]	77.0	78.6	79.3	79.3	81.0	81.4
Adam [17]	76.9	78.4	78.8	78.4	80.2	80.6
AdamW [21]	77.0	78.9	79.3	78.9	79.9	80.4
LAMB [24], [70]	77.0	79.2	79.8*	79.4	81.1	81.3*
Adan (ours)	**78.1**	**79.7**	**80.2**	**79.9**	**81.6**	**81.8**

Epoch	ConvNext Tiny 150	ConvNext Tiny 300
AdamW [21], [4]	81.2	82.1◇
Adan (ours)	**81.7**	**82.4**
Epoch	**ConvNext Small 150**	**ConvNext Small 300**
AdamW [21], [4]	82.2	83.1◇
Adan (ours)	**82.5**	**83.3**

* and ◇ are from [42], [69].
The bold values indicate the optimal performance metrics under the given settings, also highlighting the methods that achieved the results.

Training Setting I: The recently proposed "A2 training recipe" in [69] has pushed the performance limits of many SoTA CNN-type architectures by using stronger data augmentation and more training iterations. For example, on ResNet50, it sets new SoTA 80.4%, and improves the accuracy 76.1% under vanilla setting in [2]. Specifically, for data augmentation, this setting uses random crop, horizontal flipping, Mixup (0.1) [53]/CutMix (1.0) [54] with probability 0.5, and RandAugment [78] with $M = 7, N = 2$ and MSTD $= 0.5$. It sets stochastic depth (0.05) [79], and adopts cosine learning rate decay and binary cross-entropy (BCE) loss. For Adan, we use batch size 2048 for ResNet and ViT.

Training Setting II: We follow the same official training procedure of ViT/Swin/ConvNext. For this setting, data augmentation includes random crop, horizontal flipping, Mixup (0.8), CutMix (1.0), RandAugment ($M = 9$, MSTD $= 0.5$) and Random Erasing ($p = 0.25$). We use CE loss, the cosine decay for base learning rate, the stochastic depth (with official parameters), and weight decay. For Adan, we set batch size 2048 for Swin/ViT/ConvNext and 4096 for MAE. We follow MAE and tune β_3 as 0.1.

2) Results on CNN-Type Architectures: To train ResNet and ConvNext, we respectively use their official Training Setting I and II. For ResNet/ConvNext, its default official optimizer is LAMB/AdamW. From Table II, one can observe that on ResNet, 1) in most cases, Adan only running 200 epochs can achieve higher or comparable top-1 accuracy on ImageNet [77] compared with the official SoTA result trained by LAMB with 300 epochs; 2) Adan gets more improvements over other optimizers, when training is insufficient, e.g., 100 epochs. The possible reason for observation 1) is the regularizer separation, which can dynamically adjust the weight decay for each coordinate instead of sharing a common one like LAMB. For observation 2), this can be explained by the faster convergence speed of Adan than other optimizers. As shown in Table I, Adan converges faster than many adaptive gradient optimizers. This faster speed partially comes from its large learning rate guaranteed by Theorem 2, almost $3\times$ larger than that of LAMB, since the same as Nestrov acceleration, Adan also looks ahead for possible correction. Note, we have tried to adjust learning rate and warmup-epoch for Adam and LAMB, but observed unstable training behaviors. On ConvNext (tiny and small), one can observe similar comparison results on ResNet.

Since some well-known deep optimizers test ResNet-18 for 90 epochs under the official vanilla training setting [2], we also run Adan 90 epochs under this setting for more comparison. Table III shows that Adan consistently outperforms SGD and all compared adaptive optimizers. Note for this setting, it is not easy for adaptive optimizers to surpass SGD due to the absence of heavy-tailed noise, which is the crucial factor helping adaptive optimizers beat AGD [81].

Additionally, we have extended our experiments to include smaller datasets, specifically running tests on the CIFAR-10 [82] dataset using a ResNet-34 model to evaluate Adan against nine other optimizers. These experiments were conducted using AdaBelief's codebase [15] as a benchmark for settings and hyperparameters, ensuring consistency and comparability. The results, now included in Table IV, reveal that Adan not only maintains its superior performance in comparison with other optimizers but also confirms its efficacy on smaller datasets. This evidence underlines Adan's robust performance across various dataset sizes and its capability to adapt to diverse training conditions.

3) Results on ViTs. Supervised Training: We train ViT and Swin under their official training setting, i.e., Training Setting II. Table V shows that across different model sizes of ViT and Swin, Adan outperforms the official AdamW optimizer by a large margin. For ViTs, their gradient per iteration differs much from the previous one due to the much sharper loss landscape than CNNs [83] and the strong random augmentations for training. So it is hard to train ViTs to converge within a few epochs. Thanks to its faster convergence, as shown in Fig. 1, Adan is very suitable for this situation. Moreover, the direction correction term from the gradient difference $\mathbf{v}_k$ of Adan can also better correct the first- and second-order moments. One piece of evidence is that the first-order moment decay coefficient $\beta_1 = 0.02$ of Adan is much smaller than 0.1 used in other deep optimizers. Besides AdamW, we also compare Adan with several other popular optimizers, including Adam, SGD-M, and LAMB, on ViT-S, please see Table 16 in supplementary materials.

Self-supervised MAE Training (pre-train + finetune): We follow the MAE training framework to pre-train and finetune ViT-B on ImageNet, i.e., 300/800 pretraining epochs and 100 fine-tuning epochs. Table VI shows that 1) with 300 pre-training epochs, Adan makes 0.5% improvement over AdamW; 2) Adan pre-trained 800 epochs surpasses AdamW pre-trained 1,600

TABLE III
TOP-1 ACC. (%) OF RESNET-18 UNDER THE OFFICIAL SETTING IN [2]

Adan	SGD [7]	Nadam [52]	AdaBound [13]	Adam [17]	Radam [14]	Padam [38]	LAMB [24]	AdamW [21]	AdaBlief [15]
70.90	70.23*	68.82	68.13*	63.79*	67.62*	70.07	68.46	67.93*	70.08*

* are reported in [15].
The bold values indicate the optimal performance metrics under the given settings, also highlighting the methods that achieved the results.

TABLE IV
TOP-1 ACC. (%) OF RESNET-34 UNDER THE SETTING FROM ADABLIEF [15] ON CIFAR-10 DATASET

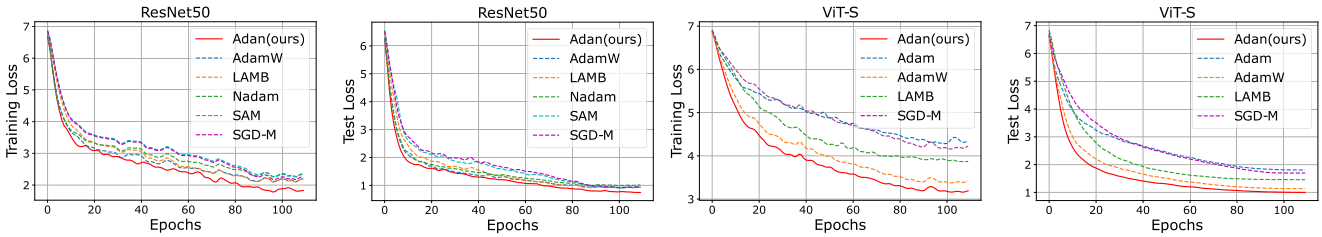
Adan	SGD [7]	Nadam [52]	AdaBound [13]	Adam [17]	Radam [14]	LAMB [24]	AdamW [21]	AdaBlief [15]	Yogi [71]
95.07	94.65	92.98	94.69	93.17	94.39	94.01	94.28	94.11	94.52

The bold values indicate the optimal performance metrics under the given settings, also highlighting the methods that achieved the results.

TABLE V
TOP-1 ACC. (%) OF VIT AND SWIN ON IMAGENET

	ViT Small		ViT Base		Swin Tiny		Swin small		Swin Base	
Epoch	150	300	150	300	150	300	150	300	150	300
AdamW [18], [19], [21]	78.3	79.9*	79.5	81.8*	79.9	81.2◇	82.1	83.2◇	82.6	83.5◇
Adan (ours)	**79.6**	**80.9**	**81.7**	**82.6**	**81.3**	**81.6**	**82.9**	**83.7**	**83.3**	**83.8**

We use their official training setting ii to train them. ∗ and ◇ are respectively reported in [18], [19].
The bold values indicate the optimal performance metrics under the given settings, also highlighting the methods that achieved the results.



(a) Training and test curves on ResNet-50. (b) Training and test curves on ViT-S.

Fig. 1. Training and test curves of various optimizers on ImageNet. The different magnitude of training and test loss is due to data argumentation. **Best viewed in 2×-sized color pdf file.**

TABLE VI
TOP-1 ACC. (%) OF VIT-B AND VIT-L TRAINED BY MAE UNDER THE OFFICIAL TRAINING SETTING II

	MAE-ViT-B			MAE-ViT-L	
Epoch	300	800	1600	800	1600
AdamW [21], [41]	82.9*	—	83.6◇	85.4◇	85.9◇
Adan (ours)	**83.4**	**83.8**	—	**85.9**	—

* and ◇ are respectively reported in [41], [80].
The bold values indicate the optimal performance metrics under the given settings, also highlighting the methods that achieved the results.

TABLE VII
TOP-1 ACC. (%) OF VIT-S ON IMAGENET TRAININED BY ADAM AND LAMB UNDER THE TRAINING SETTING I WITH DIFFERENT BATCH SIZES

Batch Size	1k	2k	4k	8k	16k	32k
LAMB [24], [30]	78.9	79.2	79.8	79.7	79.5	78.4
Adan (ours)	**80.9**	**81.1**	**81.1**	**80.8**	**80.5**	**80.2**

The bold values indicate the optimal performance metrics under the given settings, also highlighting the methods that achieved the results.

epochs by non-trial 0.2%. All these results show the superior convergence and generalization performance of Adan.

Large-Batch Training: Although large batch size can increase computation parallelism to reduce training time and is heavily desired, optimizers often suffer performance degradation, or even fail. For instance, AdamW fails to train ViTs when batch size is beyond 4,096. How to solve the problem remains open [30]. At present, LAMB is the most effective optimizer for large batch size. Table VII reveals that Adan is robust to batch sizes from 2 k to 32 k, and shows higher performance and robustness than LAMB.

4) Comparison of Convergence Speed: In Fig. 1(a), we plot the curve of training and test loss along with the training epochs on ResNet50. One can observe that Adan converges faster than the compared baselines and enjoys the smallest training and test losses. This demonstrates its fast convergence property and good generalization ability. To sufficiently investigate the fast convergence of Adan, we further plot the curve of training and test loss on the ViT-Small in Fig. 1(b). From the results, we can

TABLE VIII
DETECTION BOX-AP OF DEFORMABLE-DETR [73] ON COCO

Method (Backbone)	Optimizer	AP^b	AP^b_{50}	AP^b_{75}
Deformable-DETR (ResNet-50)	AdamW	43.8*	62.6*	47.7*
	AdamW	44.5◇	63.2◇	48.9◇
	Adan	**45.3**	**64.4**	**49.3**

* and ◇ are respectively reported in official setting [73] and mmdection's improved settings [84]. The official optimizer is adamw and the training epoch is 50.
The bold values indicate the optimal performance metrics under the given settings, also highlighting the methods that achieved the results.

TABLE IX
INSTANCE SEGMENTATION BOX/MASK-AP OF MASK-RCNN [74], CHOOSING CONVNEXT-T AS THE BACKBONE, ON COCO

Method (Backbone)	Optimizer	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}
Mask R-CNN (ConvNeXt-T)	AdamW	46.2◇	68.1◇	50.8◇	41.7◇	65.0◇	44.9◇
	Adan	**46.7**	**68.5**	**51.0**	**42.2**	**65.5**	**45.3**

◇ is from [84]. The official optimizer of these settings is adamw and the training epoch is 36.
The bold values indicate the optimal performance metrics under the given settings, also highlighting the methods that achieved the results.

see that Adan consistently shows faster convergence behaviors than other baselines in terms of both training loss and test loss. This also partly explains the good performance of Adan.

5) Experiments for Detection and Segmentation Tasks: In this experiment, we test Adan on the detection and segmentation tasks via the COCO dataset [85] which is a large-scale dataset for detection, segmentation and captioning tasks. We accomplish the experiments with Deformable-DETR [73] and Mask R-CNN [74] (with ConvNext [4] as the backbone) to compare Adan and their official optimizer AdamW.

Table VIII reports the box Average Precision (AP) of objection detection by Deformable-DETR. For AdamW, its results on Deformable-DETR are quoted from the reported results under the official setting [73] and improved setting from MMdection [84]. For fairness, we also follow the setting in MMdection to test Adan. The results in Table VIII show that Adan improves the box AP by $1.6\% \sim 1.8\%$ compared to the official optimizer AdamW. Meanwhile, Table IX reports both the box AP and mask AP of instance segmentation by Mask R-CNN with ConvNext backbone. Adan achieves $0.5\% \sim 1.2\%$ mask/box AP improvement over the official optimizer AdamW. All These results show the effectiveness of the proposed Adan.

B. Experiments for Language Processing Tasks

1) Results on LSTM: To begin with, we test our Adan on LSTM [42] by using the Penn TreeBank dataset [86], and report the perplexity (the lower, the better) on the test set in Table X. We follow the exact experimental setting in Adablief [15]. Indeed, all our implementations are also based on the code provided by Adablief[15].[1] We use the default setting for all the hyper-parameters provide by Adablief, since it provides more baselines for fair comparison. For Adan, we utilize its default weight decay (0.02) and βs ($\beta_1 = 0.02, \beta_2 = 0.08$, and $\beta_3 = 0.01$). We choose learning rate as 0.01 for Adan.

Table X shows that on the three LSTM models, Adan always achieves the lowest perplexity, making about 1.0 overall average perplexity improvement over the runner-up. Moreover, when the LSTM depth increases, the advantage of Adan becomes more remarkable.

2) Results on BERT: Similar to the pretraining experiments of MAE which is also a self-supervised learning framework on vision tasks, we utilize Adan to train BERT [44] from scratch, which is one of the most widely used pretraining models/frameworks for NLP tasks. We employ the exact BERT training setting in the widely used codebase—Fairseq [88]. We replace the default Adam optimizer in BERT with our Adan for both pretraining and fune-tuning. Specifically, we first pretrain BERT-base on the Bookcorpus and Wikipedia datasets, and then finetune BERT-base separately for each GLUE task on the corresponding training data. Note, GLUE is a collection of 9 tasks/datasets to evaluate natural language understanding systems, in which the tasks are organized as either single-sentence classification or sentence-pair classification.

Here we simply replace the Adam optimizer in BERT with our Adan and do not make other changes, e.g., random seed, warmup steps and learning rate decay strategy, dropout probability, *etc*. For pretraining, we use Adan with its default weight decay (0.02) and βs ($\beta_1 = 0.02, \beta_2 = 0.08$, and $\beta_3 = 0.01$), and choose learning rate as 0.001. For fine-tuning, we consider a limited hyper-parameter sweep for each task, with a batch size of 16, and learning rates $\in \{2e-5, 4e-5\}$ and use Adan with $\beta_1 = 0.02, \beta_2 = 0.01$, and $\beta_3 = 0.01$ and weight decay 0.01.

Following the conventional setting, we run each fine-tuning experiment three times and report the median performance in Table XI. On MNLI, we report the mismatched and matched accuracy. And we report Matthew's Correlation and Person Correlation on the task of CoLA and STS-B, respectively. The performance on the other tasks is measured by classification accuracy. The performance of our reproduced one (second row) is slightly better than the vanilla results of BERT reported in Huggingface-transformer [87] (widely used codebase for transformers in NLP), since the vanilla Bookcorpus data in [87] is not available and thus we train on the latest Bookcorpus data version.

From Table XI, one can see that in the most commonly used BERT training experiment, Adan reveals a much better advantage over Adam. Specifically, in all GLUE tasks, on the BERT-base model, Adan achieves higher performance than Adam and makes 1.8 average improvements on all tasks. In addition, on some tasks of Adan, the BERT-base trained by Adan can outperform some large models. e.g., BERT-large which achieves 70.4% on RTE, 93.2% on SST-2, and 60.6 correlation on CoLA, and XLNet-large which has 63.6 correlation on CoLA. See [89] for more results.

3) Results on GPT-2: We evaluate Adan on the large language models (LLMs), GPT-2 [75], for code generalization tasks, which enables the completion and synthesis of code, both from other code snippets and natural language descriptions. LLMs work across a wide range of domains, tasks, and programming languages, and can, for example, assist professional and citizen developers with building new applications. We pre-train

[1] The reported results in [15] slightly differ from the those in [38] because of their different settings for LSTM and training hyper-parameters.

TABLE X
TEST PERPLEXITY (THE LOWER, THE BETTER) ON PENN TREEBANK FOR ONE-, TWO- AND THREE-LAYERED LSTMS

LSTM	**Adan**	AdaBelief [15]	SGD [7]	AdaBound [13]	Adam [17]	AdamW [21]	Padam [38]	RAdam [14]	Yogi [71]
1 layer	**83.6**	84.2	85.0	84.3	85.9	84.7	84.2	86.5	86.5
2 layers	**65.2**	66.3	67.4	67.5	67.3	72.8	67.2	72.3	71.3
3 layers	**59.8**	61.2	63.7	63.6	64.3	69.9	63.2	70.0	67.5

All results except adan and padam in the table are reported by adabelief [15].
The bold values indicate the optimal performance metrics under the given settings, also highlighting the methods that achieved the results.

TABLE XI
CORRELATION OR ACC. (%) (THE HIGHER, THE BETTER) OF BERT-BASE MODEL ON THE DEVELOPMENT SET OF GLUE

BERT-base	MNLI	QNLI	QQP	RTE	SST-2	CoLA	STS-B	**Average**
Adam [17] (from [88])	83.7/84.8	89.3	90.8	71.4	91.7	48.9	91.3	81.5
Adam [17] (reproduced)	84.9/84.9	90.8	90.9	69.3	92.6	58.5	88.7	82.5
Adan (ours)	**85.7/85.6**	**91.3**	**91.2**	**73.3**	**93.2**	**64.6**	**89.3**	**84.3 (+1.8)**

The bold values indicate the optimal performance metrics under the given settings, also highlighting the methods that achieved the results.

TABLE XII
PASS@K METRIC (THE HIGHER, THE BETTER), EVALUATING FUNCTIONAL CORRECTNESS, FOR GPT-2 (345 M) MODEL ON THE HUMANEVAL DATASET PRE-TRAINED WITH DIFFERENT STEPS

GPT-2 (345m)	Steps	pass@1	pass@10	pass@100
Adam	300k	0.0840	0.209	0.360
Adan	150k	**0.0843**	**0.221**	**0.377**

The bold values indicate the optimal performance metrics under the given settings, also highlighting the methods that achieved the results.

TABLE XIII
TEST PPL (THE LOWER, THE BETTER) FOR TRANSFORMER-XL-BASE MODEL ON THE WIKITEXT-103 DATASET WITH DIFFERENT TRAINING STEPS

Transformer-XL-base	Training Steps 50k	100k	200k
Adam [17]	28.5	25.5	24.2*
Adan (ours)	**26.2**	**24.2**	**23.5**

* is reported in the official implementation.
The bold values indicate the optimal performance metrics under the given settings, also highlighting the methods that achieved the results.

GPT-2 on The-Stack dataset (Python only) [90] from BigCode[2] and evaluated on the HumanEval dataset [91] by zero-shot learning. HumanEval is used to measure functional correctness for synthesizing programs from docstrings. It consists of 164 original programming problems, assessing language comprehension, algorithms, and simple mathematics, with some comparable to simple software interview questions. We set the temperature to 0.8 during the evaluation.

We report pass@k [92] in Table XII to evaluate the functional correctness, where k code samples are generated per problem, a problem is considered solved if any sample passes the unit tests and the total fraction of problems solved is reported. We can observe that on GPT-2, Adan surpasses its default Adam optimizer in terms of pass@k within only half of the pre-training steps, which implies that Adan has a much larger potential in training LLMs with fewer computational costs. *For more comprehensive results on LLMs, please refer to supplementary material Section B.1*, available online.

4) Results on Transformer-XL: Here we investigate the performance of Adan on Transformer-XL [43] which is often used to model long sequences. We follow the exact official setting to train Transformer-XL-base on the WikiText-103 dataset that is the largest available word-level language modeling benchmark with long-term dependency. We only replace the default Adam optimizer of Transformer-XL-base by our Adan, and do not make other changes for the hyper-parameter. For Adan, we set $\beta_1 = 0.1, \beta_2 = 0.1$, and $\beta_3 = 0.001$, and choose learning rate as 0.001. We test Adan and Adam with several training steps, including 50 k, 100 k, and 200 k (official), and report the results in Table XIII.

From Table XIII, one can observe that on Transformer-XL-base, Adan surpasses its default Adam optimizer in terms of test PPL (the lower, the better) under all training steps. Surprisingly, Adan using 100 k training steps can even achieve comparable results to Adam with 200 k training steps. All these results demonstrate the superiority of Adan over the default SoTA Adam optimizer in Transformer-XL.

C. Results on Reinforcement Learning Tasks

Here we evaluate Adan on reinforcement learning tasks. Specifically, we replace the default Adam optimizer in PPO [93], which is one of the most popular policy gradient methods, without making any other changes to PPO. For brevity, we call this new PPO version "PPO-Adan". Then we test PPO and PPO-Adan on several games which are actually continuous control environments simulated by the standard and widely-used engine, MuJoCo [76]. For these test games, their agents receive a reward at each step. Following standard evaluation, we run each game under 10 different and independent random seeds (i.e., $1 \sim 10$), and test the performance for 10 episodes every 30,000 steps. All these experiments are based on the widely used codebase Tianshou [94]. For fairness, we use the default hyper-parameters in Tianshou, e.g., batch size, discount, and GAE parameter. We use Adan with its default β's ($\beta_1 = 0.02, \beta_2 = 0.08$, and

[2]https://www.bigcode-project.org

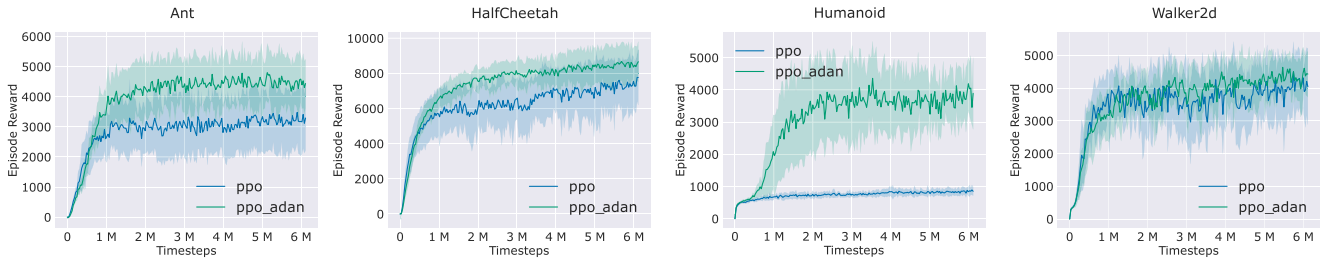


Fig. 2. Comparison of PPO and our PPO-Adan on several RL games simulated by MuJoCo. Here PPO-Adan simply replaces the Adam optimizer in PPO with our Adan and does not change others. **Best viewed in 2×-sized color pdf file.**

TABLE XIV
COMPARISON OF ROC-AUC METRICS FOR THE DEEPGCN GRAPH NEURAL NETWORK ON THE OGBN-PROTEINS DATASET

DeepGCN [98] layer=24, channel=64	Epochs 500	Epochs 1,000
Adam (official)	0.812	0.826
Adan	**0.828**	**0.831**

The bold values indicate the optimal performance metrics under the given settings, also highlighting the methods that achieved the results.

$\beta_3 = 0.01$). Following the default setting, we do not adopt the weight decay and choose the learning rate as 3e-4.

We report the results on four test games in Fig. 2, in which the solid line denotes the averaged episodes rewards in evaluation and the shaded region is its 75% confidence intervals. From Fig. 2, one can observe that on the four test games, PPO-Adan achieves much higher rewards than vanilla PPO which uses Adam as its optimizer. These results demonstrate the advantages of Adan over Adam since PPO-Adan simply replaces the Adam optimizer in PPO with our Adan and does not make other changes.

D. Results on Graph Neural Networks

To further assess the effectiveness of the Adan optimizer across different network architectures, this section focuses on graph neural networks using the Open Graph Benchmark (OGB) [95]. OGB encompasses several challenging large-scale datasets. Consistent with the settings used in DeepGCN [96], [97], our experiments were conducted on the ogbn-proteins dataset, optimizing the node feature prediction task at the level of the optimizer. The ogbn-proteins dataset is an undirected, weighted graph, classified by species type, comprising 132,534 nodes and 39,561,252 edges. Each edge is associated with an 8-dimensional feature, and every node features an 8-dimensional binary vector representing the species of the corresponding protein. Given that the prediction task for ogbn-proteins in DeepGCN is multi-label, ROC-AUC was chosen as the evaluation metric. As demonstrated in Table XIV, the Adan optimizer exhibits unique advantages in addressing the complex optimization challenges of graph convolutional networks. Particularly in the context of *deep* graph neural networks, Adan efficiently and effectively manages learning rate adjustments and model parameter update directions, enabling the DeepGCN model to achieve superior performance on the test dataset.

VI. CONCLUSION

In this paper, to relieve the plague of trying different optimizers for different deep network architectures, we propose a new deep optimizer, Adan. We reformulate the vanilla AGD to a more efficient version and use it to estimate the first- and second-order moments in adaptive optimization algorithms. We prove that the complexity of Adan matches the lower bounds and is superior to those of other adaptive optimizers. Finally, extensive experimental results demonstrate that Adan consistently surpasses other optimizers on many popular backbones and frameworks, including ResNet, ConvNext, ViT, Swin, MAE-ViT, LSTM, Transformer-XL, BERT, and GPT-2.

REFERENCES

[1] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[3] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Representations*, 2020.
[4] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 11976–11986.
[5] T. N. Sainath et al., "Improvements to deep convolutional neural networks for LVCSR," in *Proc. IEEE Workshop Autom. Speech Recognit. Understanding*, 2013, pp. 315–320.
[6] O. Abdel-Hamid, A. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE Trans. Audio Speech Lang. Process.*, vol. 22, no. 10, pp. 1533–1545, Oct. 2014.
[7] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, pp. 400–407, 1951.
[8] L. eon Bottou, "On-line Learning and Stochastic Approximations," *On-Line Learn. Neural Netw.*, D. DavidEditor, Ed., Cambridge, MA, USA: MIT Press, pp. 9–42, 1999.
[9] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. 7, pp. 2121–2159, 2011.
[10] T. Tijmen and H. Geoffrey, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Netw. Mach. Learn.*, vol. 4, pp. 26–31, 2012.
[11] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *Proc. Int. Conf. Learn. Representations*, 2018.
[12] X. Chen, S. Liu, R. Sun, and M. Hong, "On the convergence of a class of adam-type algorithms for non-convex optimization," in *Proc. Int. Conf. Learn. Representations*, 2018.

[13] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," in *Proc. Int. Conf. Learn. Representations*, 2018.

[14] L. Liu et al., "On the variance of the adaptive learning rate and beyond," in *Proc. Int. Conf. Learn. Representations*, 2019.

[15] J. Zhuang et al., "AdaBelief optimizer: Adapting stepsizes by the belief in observed gradients," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 18795–18806.

[16] B. Heo et al., "AdamP: Slowing down the slowdown for momentum optimizers on scale-invariant weights," in *Proc. Int. Conf. Learn. Representations*, 2020.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[18] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2021, pp. 10347–10357.

[19] Z. Liu et al., "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2021, pp. 10012–10022.

[20] W. Yu et al., "Metaformer is actually what you need for vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 10819–10829.

[21] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *Proc. Int. Conf. Learn. Representations*, 2018.

[22] Y. Liu, S. Mai, X. Chen, C.-J. Hsieh, and Y. You, "Towards efficient and scalable sharpness-aware minimization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 12360–12370.

[23] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," 2017, *arXiv: 1708.03888*.

[24] Y. You et al., "Large batch optimization for deep learning: Training bert in 76 minutes," in *Proc. Int. Conf. Learn. Representations*, 2019.

[25] P. Zhou, X. Xie, Z. Lin, K.-C. Toh, and S. Yan, "Win: Weight-decay-integrated nesterov acceleration for faster network training," *J. Mach. Learn. Res.*, vol. 25, no. 83, pp. 1–74, 2024.

[26] Y. Nesterov, "A method for solving the convex programming problem with convergence rate $\mathcal{O}(1/k^2)$," *Doklady Akademii Nauk*, vol. 269, no. 3, pp. 543–547, 1983.

[27] Y. Nesterov, "On an approach to the construction of optimal methods of minimization of smooth convex functions," *Ekonomika i Mateaticheskie Metody*, vol. 24, no. 3, pp. 509–517, 1988.

[28] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, vol. 87, Berlin, Germany: Springer Science & Business Media, 2003.

[29] Z. Nado, J. M. Gilmer, C. J. Shallue, R. Anil, and G. E. Dahl, "A large batch optimizer reality check: Traditional, generic optimizers suffice across batch sizes," 2021, *arXiv:2102.06356*.

[30] X. He, F. Xue, X. Ren, and Y. You, "Large-scale deep learning optimizations: A comprehensive survey," 2021, *arXiv:2111.00856*.

[31] Z. Guo, Y. Xu, W. Yin, R. Jin, and T. Yang, "A novel convergence analysis for algorithms of the adam family," 2021, *arXiv:2112.03459*.

[32] Y. Wang et al., "Adapting stepsizes by momentumized gradients improves optimization and generalization," 2021, *arXiv:2106.11514*.

[33] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," in *Proc. Int. Conf. Learn. Representations*, 2021.

[34] A. Cutkosky and H. Mehta, "Momentum improves normalized SGD," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2020, pp. 2260–2268.

[35] M. Liu, W. Zhang, F. Orabona, and T. Yang, "Adam$^+$: A stochastic method with adaptive variance reduction," 2020, *arXiv: 2011.11985*.

[36] Y. Arjevani, Y. Carmon, J. C. Duchi, D. J. Foster, A. Sekhari, and K. Sridharan, "Second-order information in non-convex stochastic optimization: Power and limitations," in *Proc. Conf. Learn. Theory*, PMLR, 2020, pp. 242–299.

[37] D. Zhou, J. Chen, Y. Cao, Y. Tang, Z. Yang, and Q. Gu, "On the convergence of adaptive gradient methods for nonconvex optimization," 2018, *arXiv: 1808.05671*.

[38] J. Chen, D. Zhou, Y. Tang, Z. Yang, Y. Cao, and Q. Gu, "Closing the generalization gap of adaptive gradient methods in training deep neural networks," in *Proc. 29th Int. Conf. Int. Joint Conf. Artif. Intell.*, 2021, pp. 3267–3275.

[39] Z. Lin, H. Li, and C. Fang, *Accelerated Optimization for Machine Learning*. Berlin, Germany: Springer, 2020.

[40] Y. Arjevani, Y. Carmon, J. C. Duchi, D. J. Foster, N. Srebro, and B. Woodworth, "Lower bounds for non-convex stochastic optimization," *Math. Program.*, vol. 199, pp. 165–214, 2023.

[41] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 15979–15988.

[42] J. Schmidhuber et al., "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[43] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 2978–2988.

[44] J. D. M.-W. C. Kenton and L. K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Annu. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Hum. Lang. Technol.*, 2019, pp. 4171–4186.

[45] R. Wightman, "PyTorch image models," 2019. [Online]. Available: https://github.com/rwightman/pytorch-image-models

[46] I. Babuschkin et al., "The DeepMind JAX ecosystem," 2020. [Online]. Available: http://github.com/deepmind

[47] M. Contributors, "OpenMMLab's image classification toolbox and benchmark," 2020. [Online]. Available: https://github.com/open-mmlab/mmclassification

[48] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *Ussr Comput. Math. Math. Phys.*, vol. 4, no. 5, pp. 1–17, 1964.

[49] J. Kwon, J. Kim, H. Park, and I. K. Choi, "ASAM: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2021, pp. 5905–5914.

[50] J. Du et al., "Efficient sharpness-aware minimization for improved training of neural networks," in *Proc. Int. Conf. Learn. Representations*, 2022.

[51] Z. Xie, X. Wang, H. Zhang, I. Sato, and M. Sugiyama, "Adaptive inertia: Disentangling the effects of adaptive learning rate and momentum," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2022, pp. 24430–24459.

[52] T. Dozat, "Incorporating nesterov momentum into adam," in *Proc. Int. Conf. Learn. Representations Workshops*, 2016.

[53] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," in *Proc. Int. Conf. Learn. Representations*, 2018.

[54] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "CutMix: Regularization strategy to train strong classifiers with localizable features," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 6023–6032.

[55] Y. Guo et al., "Multi-way backpropagation for training compact deep neural networks," *Neural Netw.*, vol. 126, pp. 250–261, 2020.

[56] J. Wang, H. Chen, L. Ma, L. Chen, X. Gong, and W. Liu, "Sphere loss: Learning discriminative features for scene classification in a hyperspherical feature space," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2021, Art. no. 5601819.

[57] B. Xie, Y. Liang, and L. Song, "Diverse neural network learns true target functions," in *Proc. Int. Conf. Artif. Intell. Statist.*, PMLR, 2017, pp. 1216–1224.

[58] Y. Li and Y. Yuan, "Convergence analysis of two-layer neural networks with ReLU activation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 597–607.

[59] Z. Charles and D. Papailiopoulos, "Stability and generalization of learning algorithms that converge to global optima," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2018, pp. 745–754.

[60] P. Zhou, H. Yan, X. Yuan, J. Feng, and S. Yan, "Towards understanding why lookahead generalizes better than SGD and beyond," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, Art. no. 2090.

[61] Q. Nguyen, M. Mondelli, and G. F. Montufar, "Tight bounds on the smallest eigenvalue of the neural tangent kernel for deep ReLU networks," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 8119–8129.

[62] Q. N. Nguyen and M. Mondelli, "Global convergence of deep networks with one wide layer followed by pyramidal topology," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 11961–11972.

[63] X. Xie, Q. Wang, Z. Ling, X. Li, G. Liu, and Z. Lin, "Optimization induced equilibrium networks: An explicit optimization perspective for understanding equilibrium models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 3, pp. 3604–3616, Mar. 2023.

[64] X. Xie, J. Wu, G. Liu, and Z. Lin, "SSCNet: Learning-based subspace clustering," *Vis. Intell.*, vol. 2, no. 1, 2024, Art. no. 11.

[65] N. Parikh and S. Boyd, "Proximal algorithms," *Found. Trends Optim.*, vol. 1, no. 3, pp. 127–239, 2014.

[66] Z. Zhuang, M. Liu, A. Cutkosky, and F. Orabona, "Understanding AdamW through proximal methods and scale-freeness," *Trans. Mach. Learn. Res.*, 2022. [Online]. Available: https://openreview.net/forum?id=IKhEPWGdwK

[67] H. Li and Z. Lin, "Restarted nonconvex accelerated gradient descent: No more polylogarithmic factor in the $\mathcal{O}(\epsilon^{-7/4})$ complexity," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2022, pp. 12901–12916.

[68] C. Jin, P. Netrapalli, and M. I. Jordan, "Accelerated gradient descent escapes saddle points faster than gradient descent," in *Proc. Conf. Learn. Theory*, PMLR, 2018, pp. 1042–1085.

[69] R. Wightman, H. Touvron, and H. Jégou, "ResNet strikes back: An improved training procedure in timm," 2021, *arXiv:2110.00476*.

[70] M. Zaheer, S. Reddi, D. Sachan, S. Kale, and S. Kumar, "Adaptive methods for nonconvex optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 9815–9825.

[71] P. Zhou, X. Xie, Z. Lin, and S. Yan, "Towards understanding convergence and generalization of AdamW," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Mar. 27, 2024, doi: 10.1109/TPAMI.2024.3382294.

[72] N. Shi, D. Li, M. Hong, and R. Sun, "RMSProp converges with proper hyper-parameter," in *Proc. Int. Conf. Learn. Representations*, 2020.

[73] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable transformers for end-to-end object detection," in *Proc. Int. Conf. Learn. Representations*, 2021.

[74] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2961–2969.

[75] A. Radford et al., "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, pp. 9, 2019.

[76] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 5026–5033.

[77] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[78] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2020, pp. 702–703.

[79] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 646–661.

[80] X. Chen et al., "Context autoencoder for self-supervised representation learning," 2022, *arXiv:2202.03026*.

[81] J. Zhang et al., "Why are adaptive methods good for attention models?," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 15383–15393.

[82] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Tech. Rep., 2009.

[83] X. Chen, C.-J. Hsieh, and B. Gong, "When vision transformers outperform resnets without pre-training or strong data augmentations," in *Proc. Int. Conf. Learn. Representation*, 2022.

[84] K. Chen et al., "MMDetection: Open MMLab detection toolbox and benchmark," 2019, *arXiv: 1906.07155*.

[85] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 740–755.

[86] M. A. Marcinkiewicz, "Building a large annotated corpus of English: The penn treebank," *Using Large Corpora*, vol. 273, 1994.

[87] T. Wolf et al., "Transformers: State-of-the-art natural language processing," in *Proc. Conf. Empir. Methods Natural Lang. Process.: Syst. Demonstrations*, 2020, pp. 38–45.

[88] M. Ott et al., "fairseq: A fast, extensible toolkit for sequence modeling," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics (Demonstrations)*, 2019, pp. 48–53.

[89] Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv: 1907.11692*.

[90] D. Kocetkov et al., "The stack: 3 tb of permissively licensed source code," *Trans. Mach. Learn. Res.*, 2023. [Online]. Available: https://openreview.net/forum?id=pxpbTdUEpD

[91] M. Chen et al., "Evaluating large language models trained on code," 2021, *arXiv:2107.03374*.

[92] S. Kulal et al., "SPoC: Search-based pseudocode to code," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, Art. no. 1066.

[93] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2016, pp. 1329–1338.

[94] J. Weng et al., "Tianshou: A highly modularized deep reinforcement learning library," *J. Mach. Learn. Res.*, vol. 23, 2022, Art. no. 267.

[95] W. Hu et al., "Open graph benchmark: Datasets for machine learning on graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 22118–22133.

[96] G. Li, M. Muller, A. Thabet, and B. Ghanem, "DeepGCNs: Can GCNs go as deep as CNNs?," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 9267–9276.

[97] G. Li, C. Xiong, A. Thabet, and B. Ghanem, "DeeperGCN: All you need to train deeper GCNs," 2020, *arXiv: 2006.07739*.

[98] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 315–323.

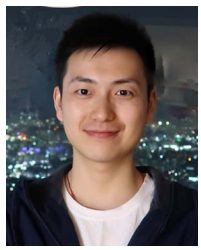
[99] S. S. Du, W. Hu, and J. D. Lee, "Algorithmic regularization in learning deep homogeneous models: Layers are automatically balanced," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 382–393.

[100] A. Q. Jiang et al., "Mixtral of experts," 2024, *arXiv:2401.04088*.

[101] T. Computer, "RedPajama: An open dataset for training large language models," 2023. [Online]. Available: https://github.com/togethercomputer/RedPajama-Data

[102] T. Xiao, M. Singh, E. Mintun, T. Darrell, P. Dollár, and R. Girshick, "Early convolutions help transformers see better," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 30392–30400.

[103] H. Touvron, M. Cord, and H. Jégou, "Deit III: Revenge of the ViT," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2022, pp. 516–533.

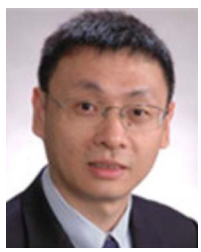


Xingyu Xie received the PhD degree from Peking University, in 2023. He is currently a research fellow with the Department of Mathematics, National University of Singapore. His current research interests include large-scale optimization and deep learning.

Pan Zhou received the master's degree from Peking University, in 2016, and the PhD degree from the National University of Singapore, in 2019. Now he is an assistant professor with Singapore Management University, Singapore. Before he also worked as a research scientist with Salesforce and Sea AI Lab, Singapore. His research interests include computer vision, machine learning, and optimization. He was the winner of the Microsoft Research Asia Fellowship 2018.

Huan Li received the PhD degree from Peking University, in 2019. He is currently an assistant researcher with the Institute of Robotics and Automatic Information Systems, Nankai University. His current research interests include optimization and machine learning.

Zhouchen Lin (Fellow, IEEE) received the PhD degree in applied mathematics from Peking University, in 2000. He is currently a Boya Special professor with the State Key Laboratory of General Artificial Intelligence, School of Intelligence Science and Technology, Peking University. His research interests include machine learning and numerical optimization. He has published more than 310 papers, collecting more than 35,000 Google Scholar citations. He is a fellow of the IEEE, the IAPR, the AAIA and the CSIG.



Shuicheng Yan (Fellow, IEEE) is currently the managing director of Kunlun 2050 Research and chief scientist of Kunlun Tech & Skywork AI, and the former Group chief scientist of Sea Group. He is a fellow of Singapore's Academy of Engineering, AAAI, ACM, and IAPR. His research areas include computer vision, machine learning, and multimedia analysis. Till now, he has published more than 800 papers with top international journals and conferences, with an H-index of more than 140. He has also been named among the annual World's Highly Cited Researchers nine times.