Adan: Adaptive Nesterov Momentum Algorithm for Faster Optimizing Deep Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

Adaptive gradient algorithms [1–4] combine the moving average idea with heavy ball acceleration to estimate accurate first- and second-order moments of gradient for accelerating convergence. But Nesterov acceleration which converges faster than heavy ball acceleration in theory [5] and also in many empirical cases [6] is much less investigated under the adaptive gradient setting. In this work, we propose the ADaptive Nesterov momentum algorithm (Adan) to speed up the training of deep neural networks. Adan first reformulates the vanilla Nesterov acceleration to develop a new Nesterov momentum estimation (NME) method that avoids the extra computation and memory overhead of computing gradient at the extrapolation point. Then Adan adopts NME to estimate the first- and second-order gradient moments in adaptive gradient algorithms for convergence acceleration. Besides, we prove that Adan finds an $\epsilon$-approximate stationary point within $O(\epsilon^{-4})$ stochastic gradient complexity on the non-convex stochastic problems, matching the best-known lower bound. Extensive experimental results show that Adan surpasses the corresponding SoTA optimizers for vision, language, and RL tasks and sets new SoTAs for many popular networks and frameworks, e.g. ResNet [7], ConvNext [8], ViT [9], Swin [10], MAE [11], Transformer-XL [12] and BERT [13]. More surprisingly, Adan can use half of the training cost (epochs) of SoTA optimizers to achieve higher or comparable performance on ViT, ResNet, MAE, etc, and also shows great tolerance to a large range of minibatch size, e.g. from 1k to 32k.

1 Introduction

Deep neural networks (DNNs) have made remarkable success in many fields, e.g. computer vision [7, 8, 14–16] and natural language processing [17, 18]. A noticeable part of such success is contributed by the stochastic gradient based optimizers which find satisfactory solutions with high efficiency. Starting from AdaGrad [19] and RMSProp [20], adaptive gradient algorithms [1–3, 19–25] have gained wide attention with faster convergence speed. They adjust the learning rate for each gradient coordinate according to the current geometry curvature of the loss objective. Indeed, Adam [1] and AdamW [3], as two representatives which often offer fast convergence speed across many DNN frameworks, have become the default choice to train CNNs and ViTs [9], respectively.

However, none of the above optimizers can always stay undefeated among all its competitors across different network architectures and application settings. For instance, for vanilla ResNet, SGD often achieves better generalization performance than adaptive gradient algorithms such as Adam, whereas on vision transformers (ViTs) [9, 10, 26], SGD often fails and AdamW is the dominant optimizer with higher and more stable performance. Moreover, these commonly used optimizers usually fail for

large-batch training, but 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. Though some methods, e.g. LARS [27]
and LAMB [4], have been proposed to handle large batch sizes, their performance often varies
significantly across batch sizes. This performance inconsistency increases the training cost and
engineering burden, since one usually has to try various optimizers for different architectures.

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 [1–4]. However, previous studies [5, 28, 29] 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”. Moreover, a
recent work [6] has shown the potential of Nesterov acceleration for large-batch training [30]. Thus
we are inspired to consider efficiently integrating Nesterov acceleration with adaptive algorithms.

Contributions: 1) We propose an efficient dnn optimizer, named Adan, to train DNNs. Adan
develops a Nesterov momentum estimation method to estimate stable and accurate first- and second-
order gradient moments in adaptive algorithms for acceleration. 2) Adan enjoys provably faster
convergence speed than previous adaptive algorithms, e.g. Adam. 3) Empirically, Adan shows
superior performance over the SoTA deep optimizers across vision, language, and RL tasks.

2 Methodology

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

\[
\min_{\theta} F(\theta) := E_{\xi \sim D} [f(\theta, \xi)] + \frac{\lambda}{2} \| \theta \|^2, \tag{1}
\]

where loss \( f(\cdot, \cdot) \) is differentiable and possibly nonconvex, data \( \xi \) is drawn from an unknown
distribution \( D \), \( \theta \) is learnable parameters, and \( \| \cdot \| \) is the classical \( \ell_2 \) norm. Here we consider the
square \( \ell_2 \) regularizer as it can improve generalization performance and is widely used in practice [3].

2.1 Preliminaries

Taking a deeper step into Adam, one can easily observe that the key moving average idea in Adam is
similar to the classical (stochastic) heavy-ball acceleration (HBA) technique [31]:

HBA:

\[
g_k = \nabla f(\theta_k) + \xi_k, \quad m_k = (1 - \beta_1)m_{k-1} + g_k, \quad \theta_{k+1} = \theta_k - \eta m_k,
\]

where \( g_k \) is the minibatch gradient \( g_k := E_{\xi \sim D}[\nabla f(\theta_k, \xi)] + \xi_k \), \( \xi_k \) is the gradient noise, and the
scalar constant \( \eta \) is the base learning rate.

In addition to HBA, Nesterov’s accelerated (stochastic) gradient descent (AGD) [5, 28, 29] is another
popular acceleration technique in the optimization community:

AGD:

\[
g_k = \nabla f(\theta_k - \eta(1 - \beta_1)m_{k-1}) + \xi_k, \quad m_k = (1 - \beta_1)m_{k-1} + g_k, \quad \theta_{k+1} = \theta_k - \eta m_k. \tag{2}
\]

Unlike HBA, AGD uses the gradient at the extrapolation point \( \theta'_k = \theta_k - (1 - \beta_1)(\theta_k - \theta_{k-1}) \).
Hence AGD sees a slight “future” to converge faster. Indeed, AGD theoretically converges faster than
HBA and achieves optimal convergence rate on the general smooth convex problems [3]. Meanwhile,
since the over-parameterized DNNs have been observed/proved to have many convex-alike local
basins [32–39], AGD seems more suitable than HBA for DNNs.

For large-batch training, the recent work [6] shows that AGD has the potential to achieve comparable
performance to some specifically designed optimizers, e.g. LARS and LAMB. With its advantage in
convergence and large-batch training, we consider applying AGD to improve adaptive algorithms.

2.2 Adaptive Nesterov Momentum Algorithm

Main iteration. We temporarily set \( \lambda = 0 \) in Eqn. (1). As aforementioned, AGD computes gradient
at an extrapolation point \( \theta'_k \) instead of the current iterate \( \theta_k \), which however brings extra computation
and memory overhead for computing \( \theta'_k \) and preserving both \( \theta_k \) and \( \theta'_k \). To solve the issue, we
reformulate AGD (2) into its equivalent (See Lemma 1 in Appendix) but more DNN-efficient version:

Reformulated AGD:

\[
m_k = (1 - \beta_1)m_{k-1} + [g_k + (1 - \beta_1)(g_k - g_{k-1})], \quad \theta_{k+1} = \theta_k - \eta m_k.
\]
Algorithm 1: Adan (Adaptive Nesterov Momentum Algorithm)

\textbf{Input:} initialization $\theta_0$, step size $\eta$, momentum $(\beta_1, \beta_2, \beta_3) \in [0, 1]^3$, weight decay $\lambda > 0$.

\textbf{Output:} some average of $\{\theta_k\}_{k=1}^K$.

\begin{algorithmic}
  \State \While{$k < K$} \Do
    \State compute the stochastic gradient estimator $g_k$ at $\theta_k$;
    \State $m_k = (1 - \beta_1)m_{k-1} + \beta_1 g_k$ \Comment{set $m_0 = g_0$};
    \State $v_k = (1 - \beta_2)v_{k-1} + \beta_2 (g_k - g_{k-1})$ \Comment{set $v_1 = g_1 - g_0$};
    \State $\eta_k = \eta/(\sqrt{m_k + \varepsilon})$, $\theta_{k+1} = \theta_k - \eta_k \cdot (m_k + (1 - \beta_2)v_k)$; 
  \EndWhile
\end{algorithmic}

where $g_k = \mathbb{E}_{\xi \sim D}[\nabla f(\theta_k, \xi)] + \xi_k$. The main idea here is that we maintain $(\theta_k - \eta(1 - \beta_1)m_{k-1})$ rather than $\theta_k$ in vanilla AGD per iteration, as there is no difference between them when the algorithm converges. Like Adam, by regarding $g_k' = g_k + (1 - \beta_1)(g_k - g_{k-1})$ as the current stochastic gradient and movingly averaging $g_k'$ to estimate the first- and second-moments of gradient, we obtain

Vanilla Adan:

\begin{align*}
   m_k &= (1 - \beta_1)m_{k-1} + \beta_1 [g_k + (1 - \beta_1)(g_k - g_{k-1})] \\
   n_k &= (1 - \beta_3)n_{k-1} + \beta_3 [g_k + (1 - \beta_1)(g_k - g_{k-1})]^2 \\
   \eta_k &= \eta/(\sqrt{m_k + \varepsilon}), \quad \theta_{k+1} = \theta_k - \eta_k \cdot (m_k + (1 - \beta_2)v_k).
\end{align*}

The main difference of Adan with Adam-type methods is that as compared in Eqn. (3), the moment $m_k$ of Adan is the average of $\{g_{t} + (1 - \beta_1)(g_t - g_{t-1})\}_{t=1}^k$ while those of Adam-type are the average of $\{g_{t}\}_{t=1}^k$. So is their second-order term $n_k$.

The first-order moment $m_k = \sum_{t=0}^{k} c_{k,t}g_t + (1 - \beta_1)(g_t - g_{t-1})$ consists of two terms, i.e., gradient term $g_t$ and gradient difference term $(g_t - g_{t-1})$, which actually have different physical meanings. So we further decouple them for greater flexibility and also better trade-off between them:

$$
(\theta_{k+1} - \theta_k)/\eta_k = \sum_{t=0}^{k} [c_{k,t}g_t + (1 - \beta_1)c_{k,t}(g_t - g_{t-1})] = m_k + (1 - \beta_2)v_k,
$$

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 $m_k$ and $v_k$ are defined as

$$
m_k = (1 - \beta_1)m_{k-1} + \beta_1 g_k, \quad v_k = (1 - \beta_2)v_{k-1} + \beta_2 (g_k - g_{k-1}).
$$

This change for a flexible estimation does not impair convergence speed. As we show in Theorem 1 (see Sec. C in Appendix), the complexity of Adan under this change matches the lower complexity bound. We do not separate the gradients and their difference in the second-order moment $n_k$, since $\mathbb{E}(n_k)$ contains the correlation term $\text{Cov}(g_k, g_{k-1}) \neq 0$ which may have statistical significance.

Decay Weight by Proximation. As observed in AdamW, decoupling the optimization objective and $\ell_2$ regularizer can largely improve the generalization performance. Intuitively, at each iteration, we minimize a first-order approximation of $F(\cdot)$ at the point $\theta_k$:

$$
\theta_{k+1} = \theta_k - \eta_k \cdot m_k = \arg\min_{\theta_k} F(\theta_k) + \langle \bar{m}_k, \theta - \theta_k \rangle + \frac{1}{2\eta} \|\theta - \theta_k\|^2_{\sqrt{m_k}}
$$

where $\|x\|^2_{\sqrt{m_k}} := \langle x, \sqrt{m_k + \varepsilon} \cdot x \rangle$ and $\bar{m}_k := m_k + (1 - \beta_2)v_k$ is the first-order derivative of $F(\cdot)$ in some sense. Interestingly, follow the idea of proximal gradient descent [40, 41], we decouple the $\ell_2$ regularizer from $F(\cdot)$ and only linearize the loss function $f(\cdot)$:

$$
\theta_{k+1} = \arg\min_{\theta} \left( \frac{\lambda}{2} \|\theta\|^2_{\sqrt{m_k}} + \langle \bar{m}_k, \theta - \theta_k \rangle + \frac{1}{2\eta} \|\theta - \theta_k\|^2_{\sqrt{m_k}} \right) = \theta_k - \eta_k \cdot \bar{m}_k.
$$

From the perspective of the proximal operator, we find that the optimization objective of Decoupled Regularization methods (e.g. Adan and AdamW) at the $k$-th iteration is changed from the
vanilla “static” function $F(\cdot)$ in (1) to a “dynamic” function $F_k(\cdot)$, which adaptively regularizes the coordinates with larger gradient square terms more. We summarize our Adan in Algorithm 1. One may replace the default optimizer with our Adan, and do not make other changes, e.g. network architectures and data augmentation.

**Convergence Analysis:** As shown in Theorems 1 in Appendix C, the convergence speed of Adan matches the best-known theoretical lower bound for non-convex stochastic optimization problems. This conclusion is still valid when it also uses the decoupled weight decay.

### 3 Experimental Results

For all the tested vision tasks, NLP, and RL tasks, we only replace the default optimizer with our Adan, and do not make other changes, e.g. network architectures and data augmentation.

#### Vision Results

1) supervised settings: we report the results on CNN-type architectures and ViTs in Tables 1 and 2, respectively.
2) self-supervised settings: we follow the MAE training framework to pretrain and fine-tune ViT-B and ViT-L, and report results in Table 3. All these results show that in most cases, Adan can use half of the training cost (epochs) of SoTA optimizers to achieve higher or comparable performance on ViT, ResNet, MAE, etc.

#### NLP Results

1) supervised settings: we investigate the performance of Adan on Transformer-XL, and report the results in Table 4. 2) self-supervised settings: we use Adan to train BERT from scratch, and report the results in Table 5. For all NLP tasks, Adan achieves higher performance than the default SoTAs, and suppress Adam within half training steps on Transformer-XL.

#### RL Results

We replace the default Adam optimizer in PPO [44] (one of the most popular policy gradient method), and do not make other change in PPO. Fig. 1 shows that on representative MuJoCo games, PPO-Adan achieves much higher rewards than PPO with Adam as its optimizer.

**More Extra Results.** Due to space limitation, we defer more extra results on vision, NLP and RL tasks (e.g. results with large batch size, loss curve, ablation study, etc.) to Appendix.
References


