

训练ViT和MAE减少一半计算量！Sea和北大提出新优化器Adan：深度模型都能用！

计算机视觉工坊 2022-10-31 07:00 发表于江苏

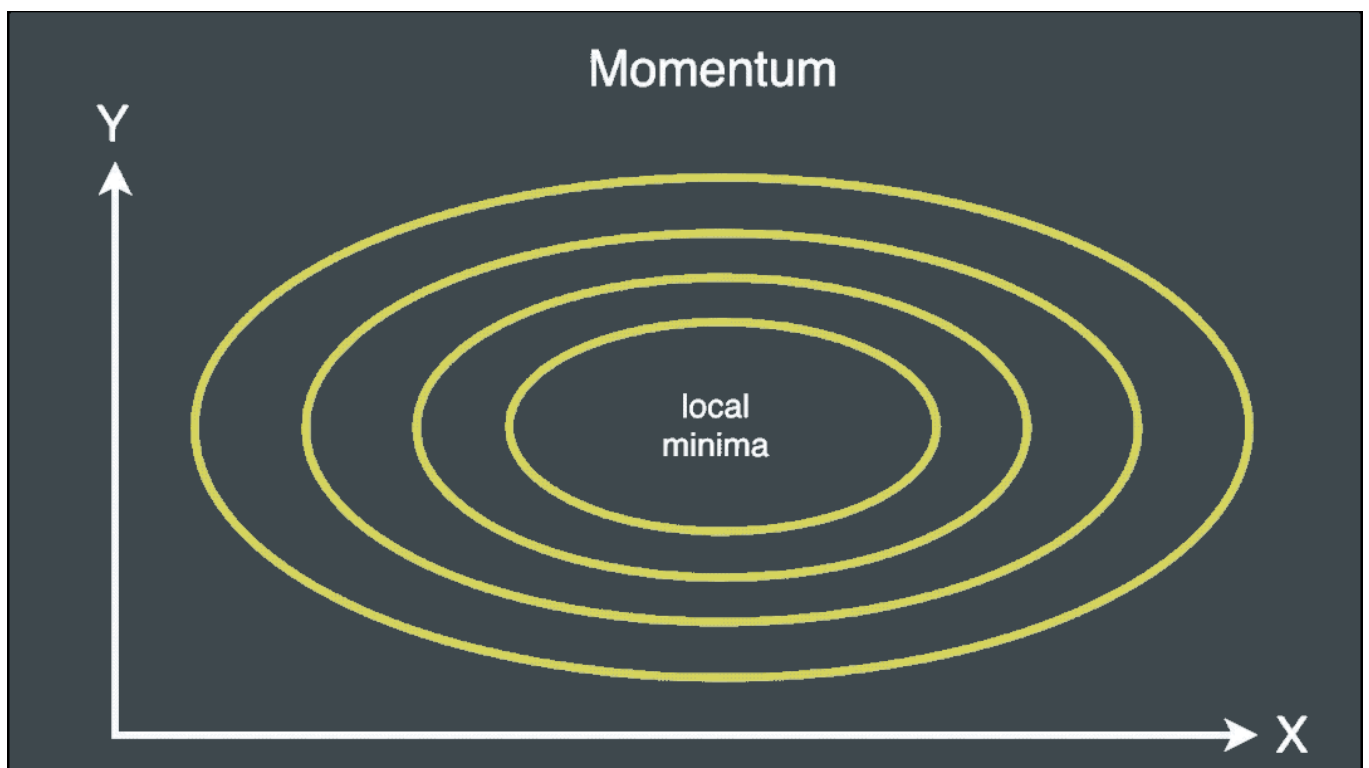


计算机视觉工坊

专注于计算机视觉、VSLAM、目标检测、语义分割、自动驾驶、深度学习、AI芯片、...

180篇原创内容

公众号




转载自：新智元 | 编辑：LRS 好困

【导读】 换个优化器，计算量少一半。

自Google提出**Vision Transformer(ViT)**以来，ViT渐渐成为许多视觉任务的默认backbone。凭借着ViT结构，许多视觉任务的SoTA都得到了进一步提升，包括图像分类、分割、检测、识别等。

然而，训练ViT并非易事。除了需要较复杂的训练技巧，模型训练的计算量往往也较之前的CNN大很多。近日，新加坡Sea AI LAB (SAIL) 和北大ZERO Lab的研究团队共同提出新的**深度模型优化器Adan**，该优化器可以仅用**一半的计算量就能完成ViT的训练**。

Adan: Adaptive Nesterov Momentum Algorithm for Faster Optimizing Deep Models

Xingyu Xie^{1,2*} Pan Zhou^{1*} Huan Li³ Zhouchen Lin² Shuicheng Yan¹
¹Sea AI Lab ²Peking University ³Nankai University  新智元
{xyxie,zhoupan,yansc}@sea.com {xyxie,zlin}@pku.cn lihuanss@nankai.edu.cn

Adan: Adaptive Nesterov Momentum Algorithm for Faster Optimizing Deep Models

论文链接：<https://arxiv.org/abs/2208.06677>

代码链接：<https://github.com/sail-sg/Adan>

此外，在计算量一样的情况下，Adan在多个场景（涉及CV、NLP、RL）、多种训练方式（有监督与自监督）和多种网络结构/算法（Swin、ViT、ResNet、ConvNext、MAE、LSTM、BERT、Transformer-XL、PPO算法）上，均获得了性能提升。

代码、配置文件、训练log均已开源。

深度模型的训练范式与优化器

随着ViT的提出，深度模型的训练方式变得越来越复杂。常见的训练技巧包括复杂的数据增强（如MixUp、CutMix、AutoRand）、标签的处理（如label smoothing和noise label）、模型参数的移动平均、随机网络深度、dropout等。伴随着这些技巧的混合运用，模型的泛化性与鲁棒性均得到了提升，但是随之而来的便是模型训练的计算量变得越来越大。

在ImageNet 1k上，训练epoch数从ResNet刚提出的90已经增长到了训练ViT常用的300。甚至针对一些自监督学习的模型，例如MAE、ViT，预训练的epoch数已经达到了1.6k。训练epoch增加意味着训练时间极大的延长，急剧增加了学术研究或工业落地的成本。目前一个普遍的解决方案是增大训练的batch size并辅助并行训练以减少训练时间，但是伴随的问题便是，大的batch size往往意味着performance的下降，并且batch size越大，情况越明显。

这主要是因为模型参数的更新次数随着batch size的增加在急剧减少。当前的优化器并不能在复杂的训练范式下以较少的更新次数实现对模型的快速训练，这进一步加剧了模型训练epoch数的增长。

因此，是否存在一种新的优化器能在较少的参数更新次数情况下更快更好地训练深度模型？在减少训练epoch数的同时，也能缓解batch size增加带来的负面影响？

被忽略的冲量

要想加速优化器的收敛速度，最直接的方法便是引入冲量。近年提出的深度模型优化器均沿用着Adam中使用的冲量范式——**重球法**：

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

其中 \mathbf{g}_k 是随机噪声， \mathbf{m}_k 是moment， η 是学习率。Adam将 \mathbf{m}_k 的更新由累积形式换成了移动平均的形式，并引入二阶moment (\mathbf{n}_k) 对学习率进行放缩，即：

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

然而随着Adam训练原始ViT失败，它的改进版本AdamW渐渐地变成了训练ViT甚至ConvNext的首选。但是AdamW并没有改变Adam中的冲量范式，因此在当batch size超过4,096的时候，AdamW训练出的ViT的性能会急剧下降。

在**传统凸优化领域**，有一个与重球法齐名的冲量技巧——**Nesterov冲量算法**：

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

Nesterov冲量算法在光滑且一般凸的问题上，拥有比重球法更快的理论收敛速度，并且理论上也能承受更大的batch size。同重球法不同的是，Nesterov算法不在当前点计算梯度，而是利用冲量找到一个外推点，在该点算完梯度以后再进行冲量累积。

外推点能帮助Nesterov算法提前感知当前点周围的几何信息。这种特性使得Nesterov冲量更加适合复杂的训练范式和模型结构（如ViT），因为它并不是单纯地依靠过去的冲量去绕开尖锐的局部极小点，而是通过提前观察周围的梯度，调整更新的方向。

尽管Nesterov冲量算法拥有一定的优势，但是在深度优化器中，却鲜有被应用与探索。其中一个主要的原因就是Nesterov算法需要在外推点计算梯度，在当前点更新，期间需要多次模型参数重载以及需要人为地在外推点进行back-propagation (BP)。这些不便利性极大地限制了Nesterov冲量算法在深度模型优化器中的应用。

Adan优化器

通过结合改写的Nesterov冲量与自适应优化算法，并引入解耦的权重衰减，可以得到最终的Adan优化器。利用外推点，Adan可以提前感知周围的梯度信息，从而高效地逃离尖锐的局部极小区域，以增加模型的泛化性。

1) 自适应的Nesterov冲量

为了解决Nesterov冲量算法中多次模型参数重载的问题，研究人员首先对Nesterov进行改写：

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

可以证明，改写的Nesterov冲量算法与原算法等价，两者的迭代点可以相互转化，且最终的收敛点相同。可以看到，通过引入梯度的差分项，已经可以避免手动的参数重载和人为地在外推点进行BP。

将改写的Nesterov冲量算法同自适应类优化器相结合——将 \mathbf{m}_k 的更新由累积形式替换为移动平均形式，并使用二阶moment对学习率进行放缩：

$$\text{Vanilla Adan: } \begin{cases} \mathbf{m}_k = (1 - \beta_1) \mathbf{m}_{k-1} + \beta_1 [\mathbf{g}_k + (1 - \beta_1)(\mathbf{g}_k - \mathbf{g}_{k-1})] \\ \mathbf{n}_k = (1 - \beta_3) \mathbf{n}_{k-1} + \beta_3 (\mathbf{g}_k + (1 - \beta_1)(\mathbf{g}_k - \mathbf{g}_{k-1}))^2 \\ \eta_k = \eta / (\sqrt{\mathbf{n}_k} + \varepsilon) \\ \boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \circ \mathbf{m}_k. \end{cases}$$

至此已经得到了Adan的算法的基础版本。

2) 梯度差分的冲量

可以发现， \mathbf{m}_k 的更新将梯度与梯度的差分耦合在一起，但是在实际场景中，往往需要对物理意义不同的两项进行单独处理，因此研究人员引入梯度差分的冲量 \mathbf{v}_k ：

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

这里对梯度的冲量和其差分的冲量设置不同的冲量/平均系数。梯度差分项可以在相邻梯度不一致的时候减缓优化器的更新，反之，在梯度方向一致时，加速更新。

3) 解耦的权重衰减

对于带L2权重正则的目标函数，目前较流行的AdamW优化器通过对L2正则与训练loss解耦，在ViT和ConvNext上获得了较好的性能。但是AdamW所用的解耦方法偏向于启发式，目前并不能得到其收敛的理论保证。

基于对L2正则解耦的思想，也给Adan引入解耦的权重衰减策略。目前Adan的每次迭代可以看成是在最小化优化目标F的某种一阶近似：

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

where

$$\|\mathbf{x}\|_{\sqrt{\mathbf{n}_k}}^2 := \langle \mathbf{x}, \sqrt{\mathbf{n}_k + \varepsilon} \circ \mathbf{x} \rangle, \quad \bar{\mathbf{m}}_k := \mathbf{m}_k + (1 - \beta_2) \mathbf{v}_k$$

由于F中的L2权重正则过于简单且光滑性很好，以至于不需要对其进行一阶近似。因此，可以只对训练loss进行一阶近似而忽略L2权重正则，那么Adan的最后一步迭代将会变成：

$$\theta_{k+1} = \theta_k - \eta_k \circ \bar{\mathbf{m}}_k = \operatorname{argmin}_{\theta} F(\theta_k) + \langle \bar{\mathbf{m}}_k, \theta - \theta_k \rangle + \frac{1}{2\eta} \|\theta - \theta_k\|_{\sqrt{\mathbf{n}_k}}^2,$$

有趣的是，可以发现AdamW的更新准则是Adan更新准则在学习率eta接近0时的一阶近似。因此，可从proximal 算子的角度给Adan甚至AdamW给出合理的解释而不是原来的启发式改进。

4) Adan优化器

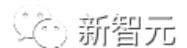
将2) 和3) 两个改进结合进Adan的基础版本，可以得到如下的Adan优化器。

Algorithm 1: Adan (Adaptive Nesterov Momentum Algorithm)

Input: initialization θ_0 , step size η , average parameter $(\beta_1, \beta_2, \beta_3) \in [0, 1]^3$, stable parameter $\varepsilon > 0$, weight decays $\lambda_k > 0$, restart condition.

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

```
1 while  $k < K$  do
2   compute the stochastic gradient estimator  $\mathbf{g}_k$  at  $\theta_k$ ;
3    $\mathbf{m}_k = (1 - \beta_1)\mathbf{m}_{k-1} + \beta_1\mathbf{g}_k$  /* set  $\mathbf{m}_0 = \mathbf{g}_0$  */;
4    $\mathbf{v}_k = (1 - \beta_2)\mathbf{v}_{k-1} + \beta_2(\mathbf{g}_k - \mathbf{g}_{k-1})$  /* set  $\mathbf{v}_1 = \mathbf{g}_1 - \mathbf{g}_0$  */;
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    $\eta_k = \eta / (\sqrt{\mathbf{n}_k} + \varepsilon)$ ;
7    $\theta_{k+1} = (1 + \lambda_k\eta)^{-1}[\theta_k - \eta_k \circ (\mathbf{m}_k + (1 - \beta_2)\mathbf{v}_k)]$ ;
8   if restart condition holds then
9     get stochastic gradient estimator  $\mathbf{g}_0$  at  $\theta_{k+1}$ ;
10     $\mathbf{m}_0 = \mathbf{g}_0$ ,  $\mathbf{v}_0 = \mathbf{0}$ ,  $\mathbf{n}_0 = \mathbf{g}_0^2$ , update  $\theta_1$  by Line 7,  $k = 1$ ;
11  end if
12 end while
```



Adan结合了自适应优化器、Nesterov冲量以及解耦的权重衰减策略的优点，能承受更大的学习率和batch size，以及可以实现对模型参数的动态L2正则。

5) 收敛性分析

这里跳过繁复的数学分析过程，只给出结论：

定理：在给定或未给定Hessian-smooth条件的两种情况下，Adan优化器的收敛速度在非凸随机优化问题上均能达到已知的理论下界，并且该结论在带有解耦的权重衰减策略时仍然成立。

实验结果

一、CV场景

1) 有监督学习——ViT模型

针对ViT模型，研究人员分别在ViT和Swin结构上，测试了Adan的性能。

Table 4: Top-1 accuracy (%) of ViT and Swin on ImageNet. We use their official Training Setting II to train them. * and \diamond are respectively reported in [9], [10].

Epoch	ViT Small		ViT Base		Swin Tiny		Swin small		Swin Base	
	150	300	150	300	150	300	150	300	150	300
AdamW [3, 9, 10]	78.3	79.9*	79.5	81.8*	79.9	81.2 \diamond	82.1	83.2 \diamond	83.5	83.5 \diamond
Adan (ours)	79.6	80.7	81.7	82.5	81.3	81.6	82.9	83.7	83.5	83.8



可以看到，例如在ViT-small、ViT-base、Swin-tiny以及Swin-base上，Adan仅仅消耗了一半的计算资源就获得了同SoTA优化器接近的结果，并且在同样的计算量下，Adan在两种ViT模型上均展现出较大的优势。

此外，也在大batch size下测试了Adan的性能：

Table 7: Top-1 accuracy (%) of ViT-S on ImageNet under the Training Setting I.

Batch Size	1k	2k	4k	8k	16k	32k
LAMB [4, 34]	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

可以看到，Adan在各种batch size下都表现得不错，且相对于专为大batch size设计的优化器（LAMB）也具有一定的优势。

2) 有监督学习——CNN模型

除了较难训练的ViT模型，研究人员也在尖锐局部极小点相对较少的CNN模型上也测试了Adan的性能——包括经典的ResNet与较先进的ConvNext。结果如下：

Table 2: Top-1 accuracy (%) of ResNet and ConvNext on ImageNet. We respectively use the official Training Setting I and II to train ResNet and ConvNext. * and ◊ are respectively reported in [64], [8].

Epoch	ResNet-50			ResNet-101		
	100	200	300	100	200	300
SAM [37]	77.3	78.7	79.4	79.5	81.1	81.6
SGD-M [5, 32, 33]	77.0	78.6	79.3	79.3	81.0	81.4
Adam [1]	76.9	78.4	78.8	78.4	80.2	80.6
AdamW [3]	77.0	78.9	79.3	78.9	79.9	80.4
LAMB [4, 64]	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	300
AdamW [3, 8]	81.2	82.1 [◊]
Adan (ours)	81.7	82.4

Epoch	ConvNext Small	
	150	300
AdamW [3, 8]	82.2	83.1 [◊]
Adan (ours)	82.5	83.3

Table 3: Top-1 accuracy (%) of ResNet18 under the official setting in [7]. * are reported in [2].

Adan	SGD [20]	Nadam [44]	AdaBound [26]	Adam [1]	Radam [27]	Padam [41]	LAMB [4]	AdamW [3]	AdaBlief [2]
70.60	70.23*	68.82	68.13*	63.79*	67.62*	70.07	68.46	67.93*	70.08*

可以观察到，不管是ResNet还是ConvNext，Adan均能在大约2/3训练epoch以内获得超越SoTA的性能。

3) 无监督学习

在无监督训练框架下，研究人员在最新提出的MAE上测试了Adan的表现。其结果如下：

Table 6: Top-1 accuracy (%) of ViT-B and ViT-L trained by self-supervised MAE on ImageNet. We use the official Training Setting II of MAE to train ViT-B and ViT-L. * and \diamond are respectively reported in [74], [11].

Epoch	MAE-ViT-B			MAE-ViT-L	
	300	800	1600	800	1600
AdamW [3, 11]	82.9*	—	83.6 \diamond	85.4 \diamond	85.9 \diamond
Adan	83.4	83.8	—	85.9	

新智元

同有监督学习的结论一致，Adan仅消耗了一半的计算量就追平甚至超过了原来的SoTA优化器，并且当训练epoch越小，Adan的优势就越明显。

二、NLP场景

1) 有监督学习

在NLP的有监督学习任务上，分别在经典的LSTM以及先进的Transformer-XL上观察Adan的表现。

Table 8: Test perplexity (the lower, the better) on Penn Treebank for one-, two- and three-layered LSTMs. All results except Adan and Padam in the table are reported by AdaBelief [2].

LSTM	Adan	AdaBelief [2]	SGD [20]	AdaBound [26]	Adam [1]	AdamW [3]	Padam [41]	RAdam [27]	Yogi [62]
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

Table 9: Test PPL (the lower, the better) for Transformer-XL-base model on the WikiText-103 dataset with different training steps. * is reported in the official implementation.

Transformer-XL-base	Training Steps		
	50k	100k	200k
Adam [1]	28.5	25.5	24.2*
Adan (ours)	26.2	24.2	23.5

新智元

Adan在上述两种网络上，均表现出一致的优越性。并且对于Transformer-XL，Adan在一半的训练步数内就追平了默认的Adam优化器。

2) 无监督学习

为了测试Adan在NLP场景下无监督任务上的模型训练情况。研究人员从头开始训练BERT：在经过1000k的预训练迭代后，在GLUE数据集的7个子任务上测试经过Adan训练的模型性能，结果如下：

Table 10: Results (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 [1] (from [76])	83.7/84.8	89.3	90.8	71.4	91.7	48.9	91.3	81.5
Adam [1] (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 (81.8)

Adan在所测试的7个词句分类任务上均展现出较大的优势。值得一提的是，经过Adan训练的BERT-base模型，在一些子任务上（例如RTE、CoLA以及SST-2）的结果甚至超过了Adam训练的BERT-large.

三、RL场景

研究人员将RL常用的PPO算法里的优化器替换为了Adan，并在MuJoCo引擎中的4个游戏上测试了Adan的性能。在4个游戏中，用Adan作为网络优化器的PPO算法，总能获得较高的reward。

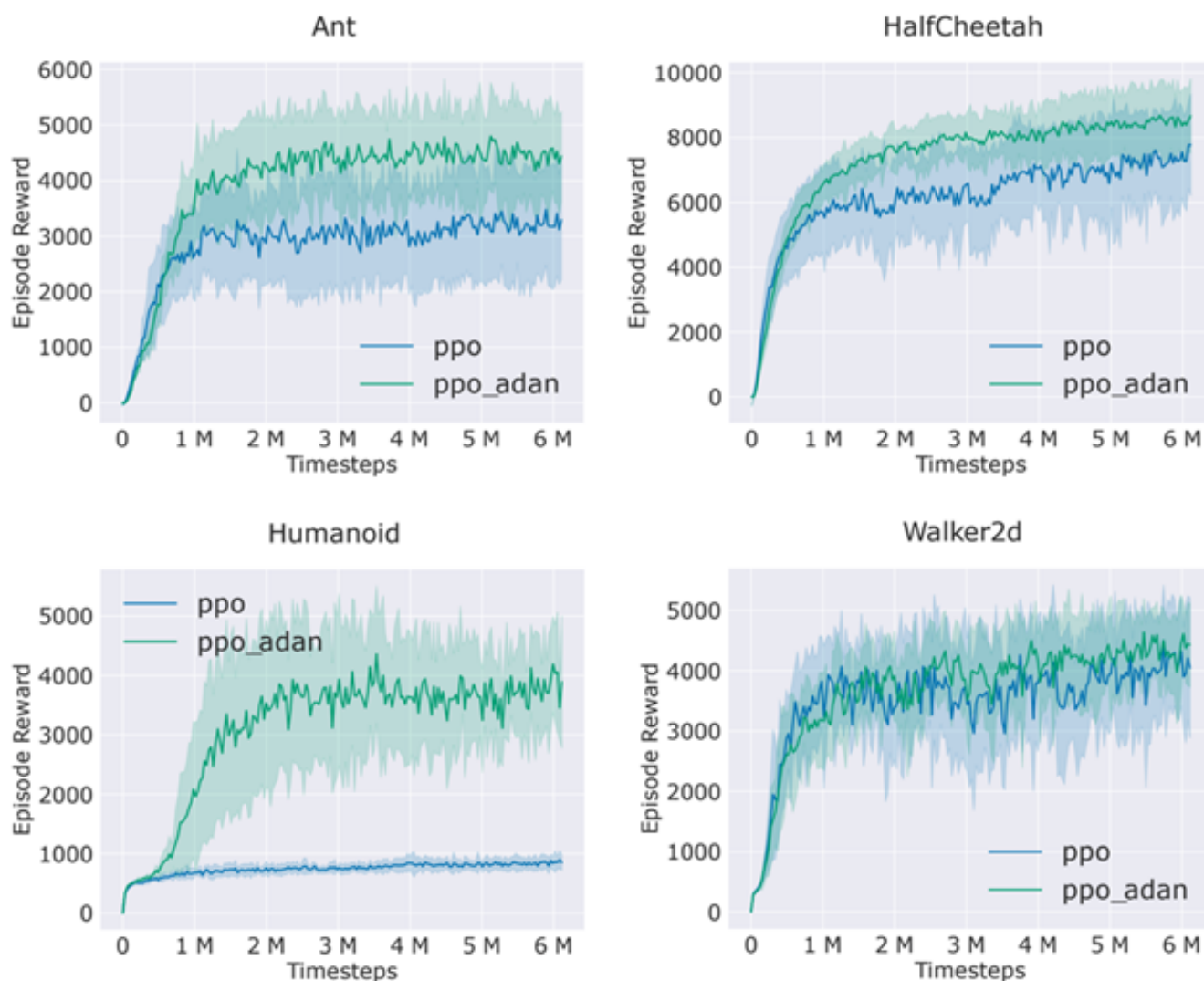



Figure 2: Comparison of PPO and our PPO-Adan on several RL games simulated by  [OpenAI Gym](#). PPO-Adan simply replaces the Adam optimizer in PPO with our Adan and does not change others.

Adan在RL的网络训练中，也表现出较大的潜力。

结论与展望

Adan优化器为目前的深度模型优化器引入了新的冲量范式。在复杂的训练范式下以较少的更新次数实现对模型的快速训练。

实验显示，Adan仅需1/2-2/3的计算量就能追平现有的SoTA优化器。

Adan在多个场景（涉及CV、NLP、RL）、多个训练方式（有监督与自监督）和多种网络结构（ViT、CNN、LSTM、Transformer等）上，均展现出较大的性能优势。此外，Adan优化器的收敛速度在非凸随机优化上也已经达到了理论下界。

本文仅做学术分享，如有侵权，请联系删除。