HVAdam: A Full-Dimension Adaptive Optimizer

Yiheng Zhang[1*], Shaowu Wu[1*], Yuanzhuo Xu[1], Jiajun Wu[2], Shang Xu[3], Steve Drew[2], Xiaoguang Niu[1 †]

[1] School of Computer Science, Wuhan University
[2] Department of Electrical and Software Engineering, University of Calgary
[3] Department of Computer Science, University College London
{gjryhxt,wshaow,xyzxyz,xgniu}@whu.edu.cn, {jiajun.wu1, steve.drew}@ucalgary.ca, shang.xu.24@ucl.ac.uk

Abstract

Adaptive optimizers such as Adam and RMSProp have gained attraction in complex neural networks, including generative adversarial networks (GANs) and Transformers, thanks to their stable performance and fast convergence compared to non-adaptive optimizers. A frequently overlooked limitation of adaptive optimizers is that adjusting the learning rate of each dimension individually would ignore the knowledge of the whole loss landscape, resulting in slow updates of parameters, invalidating the learning rate adjustment strategy and eventually leading to widespread insufficient convergence of parameters. In this paper, we propose HVAdam, a novel optimizer that associates all dimensions of the parameters to find a new parameter update direction, leading to a refined parameter update strategy for an increased convergence rate. We validated HVAdam in extensive experiments, showing its faster convergence, higher accuracy, and more stable performance on image classification, image generation, and natural language processing tasks. Particularly, HVAdam achieves a significant improvement on GANs compared with other state-of-the-art methods, especially in Wasserstein-GAN (WGAN) and its improved version with gradient penalty (WGAN-GP).

Introduction

Optimizers play a crucial role in machine learning, efficiently minimizing loss and achieving generalization. There are two main types of state-of-the-art optimizers: Nonadaptive optimizers, for instance, stochastic gradient descent (SGD) (Robbins and Monro 1951), use a global learning rate for all parameters. In contrast, adaptive optimizers, such as RMSProp (Graves 2013) and Adam (Kingma and Ba 2014), use the partial derivative of each parameter to adjust their learning rates separately. When applied correctly, adaptive optimizers can significantly accelerate the convergence of parameters, even in cases where partial derivatives are minimal. These adaptive optimizers are crucial for overcoming saddle points or navigating regions characterized by small partial derivatives, which can otherwise hinder the convergence process (Xie et al. 2022).

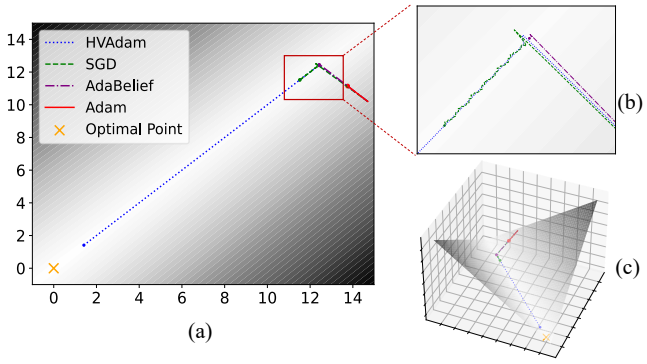


Figure 1: A typical example of the valley dilemma. (a) (c) depict the trajectories of SGD, Adam, AdaBelief and HVAdam in both 2D and 3D plots. (b) is a close-up of the red box area in (a), showing the slow convergence and zigzagging behavior of Adam, AdaBelief, and SGD. However, HVAdam demonstrates rapid convergence along the hidden vector direction.

In deep learning, model parameters often change significantly during training, and these changes frequently reverse direction, especially when the model is getting close to its optimal performance (near the minimum of the loss function). Adam dynamically adjusts the learning rate according to the magnitude of the gradients, while SGD incorporates momentum to smooth the gradient and does not directly modify the learning rate. AdaBelief (Zhuang et al. 2020), a variant of Adam, adjusts the learning rate based on the deviation of the gradient from the moving average. However, these approaches may be problematic in complex landscapes, such as valley-like regions that are quite common within the loss functions of deep learning models (Zhuang et al. 2020). In such regions, the true direction of optimization often has a small gradient, making it difficult for traditional optimizers to identify and efficiently optimize in these directions. Traditional optimizers tend to oscillate along valley sides, where gradients are larger, leading to slow and inefficient convergence. We refer to this phenomenon as the *Valley Dilemma*, which occurs when optimization techniques fail to converge effectively due to the challenging landscape as shown in Figure 1.

[*] These authors contributed equally.
[†] Corresponding Author

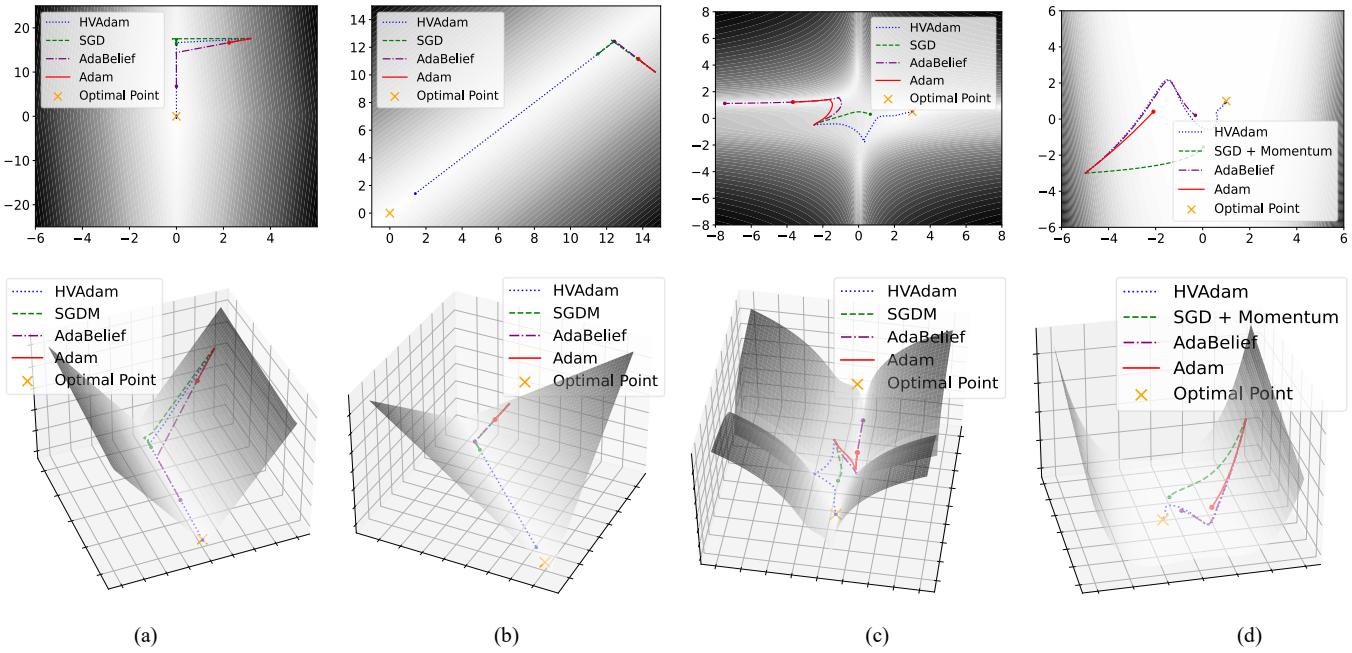


Figure 2: Trajectories of SGD, Adam, AdaBelief and HVAdam. The functions are f_1, f_2, f_3 and f_4 from (a) to (d). The functions are mentioned in the supplementary material's Sec.E. HVAdam reaches the optimal point (marked as orange cross in 2D and 3D plots) the fastest in all cases.

To further demonstrate the *Valley Dilemma*, we illustrate its effect on optimizers using two more common loss functions (e.g., (c) and (d) in Figure 2). In valley-like regions, traditional optimizers face several key challenges: **1. Identifying Update Direction:** In valley-like regions, the true optimal direction (along the valley floor) often has a very small gradient, making it difficult for traditional optimizers to detect and follow this direction, as shown in Figure 1. **2. Changing Landscapes:** As training progresses, the direction of optimization can change, as shown in Figure 2(b)(c), especially in complex and dynamic landscapes. **3. Adjusting Learning Rate:** Since this stable update direction incorporates trend information, the learning rate needs to be adjusted accordingly. To help convergence in other directions, we should design a more reasonable learning rate adjustment strategy based on the information of the trend for them.

To address the aforementioned challenges, we propose the hidden-vector-based adaptive gradient method (HVAdam), an adaptive optimizer that considers the full dimensions of the parameters. With the help of the *hidden vector*, HVAdam can determine the stable update direction, which enables faster convergence along the descending trajectory of the loss function through an increased learning rate compared to the traditional zigzag approach. Additionally, HVAdam employs a restart strategy to adapt the change of the hidden vector, namely the changing direction of the landscapes. Lastly, HVAdam incorporates information from all dimensions to adjust the learning rate for each dimension using a new preconditioning matrix based on the hidden vector. Contributions of our paper can be summarized as follows:

- We propose HVAdam, which constructs a vector that approximates the invariant components within the gradients, namely the *hidden vector*, to more effectively guide parameter updates as a solution to the *Valley Dilemma*.
- We demonstrate HVAdam's convergence properties under online convex and non-convex stochastic optimization, emphasizing its efficacy and robustness.
- We empirically evaluate the performance of HVAdam, demonstrating its significant improvements across image classification, NLP, and GANs tasks.

Background and Motivation

Notations

- $f(\theta) \in \mathbb{R}, \theta \in \mathbb{R}^d$: $f(\theta)$ is the scalar-valued function to minimize, θ is the parameter in $\mathbb{R}^d$ to be optimal.
- $\prod_{\mathcal{F},S}(y) = \text{argmin}_{x \in \mathcal{F}} ||S^{1/2}(x-y)||$: The projection of y onto convex feasible set $\mathcal{F}$.
- $g_t \in \mathbb{R}^d$: The gradient at step t.
- $m_t \in \mathbb{R}^d$: The exponential moving average (EMA) of g_t.
- $v_t \in \mathbb{R}^d$: The hidden vector calculated by v_{t-1} and m_t.
- $h_t \in \mathbb{R}$: The EMA of $\|m_t\|^2$.
- $p_t \in \mathbb{R}^d$: $p_t = (g_t - v_{t-1})^2$. p_t is an intermediate variable.
- $\eta_t \in \mathbb{R}^d$: The factor measure the size of p_t.
- $a_t, s_t \in \mathbb{R}^d$: The EMA of g_t^2 and $\eta_t p_t$.
- $\alpha_1, \alpha_2, \gamma \in \mathbb{R}$: α_1 is the unadjusted learning rate for m_t; α_2 is the unadjusted learning rate for v_t ; γ is a constant to limit the value of η_t, which is usually set as 0. These are hyperparameters.
- $\delta_t \in \mathbb{R}$: The factor used to adjust α_{2t}.

- $\epsilon \in \mathbb{R}$: The hyperparameter ϵ is a small number, used for avoiding division by 0.
- $lr(\delta_{t_2}, \widehat{\delta_{t_2}}) \in \mathbb{R} \times \mathbb{R} \to \mathbb{R}$: $\begin{cases} 10^{\delta_{t_2} \cdot 6 - 3}, & \text{if } \widehat{\delta_{t_2}} \geq 0.1 \\ 0, & \text{otherwise} \end{cases}$. The function can be set as other more suitable choices; we will not change it in the rest of the paper unless otherwise specified.
- $\beta_1, \beta_2 \in \mathbb{R}$: The hyperparameter for EMA, $0 \leq \beta_1, \beta_2 < 1$, typically set as 0.9 and 0.999.

Adaptive Moment Estimation (Adam)

Algorithm 1: Adam Optimizer

Input: α_1, β_1 , β_2, ϵ
Initialize θ_0, $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$

while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $a_t \leftarrow \beta_2 a_{t-1} + (1 - \beta_2) g_t^2$
  Bias Correction
    $\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$, $\widehat{a_t} \leftarrow \frac{a_t}{1-\beta_2^t}$
  Update
    $\theta_t \leftarrow \prod_{\mathcal{F}, \sqrt{\widehat{a_t}}} \left(\theta_{t-1} - \frac{\alpha_1 \widehat{m_t}}{\sqrt{\widehat{a_t}} + \epsilon} \right)$
end while

Adam integrates the merits of the adaptive gradient algorithm (AdaGrad) and root mean square propagation (RMSProp) by adaptively adjusting the learning rates based on estimates of the first and second moments of the gradients, making it exceptionally suitable for large-scale and complex machine learning problems. The algorithm is summarized in Algorithm 1, and all operations are element-wise.

In Algorithm. 1, β_1 and β_2 are hyperparameters that represent the exponential decay rates of the moving averages of the past gradients and squared gradients, respectively. The learning rate is denoted by α, and ϵ is a small constant added to the denominator to ensure numerical stability.

Adam adapts the learning rate for each parameter. Its straightforward implementation has made it a popular choice in the field of deep learning, particularly when working with large datasets or high-dimensional spaces.

Problems and Motivation

Traditional Adam and its variants do not effectively solve the valley dilemma. Adam adjusts the learning rate for each parameter. In regions with small partial derivatives or saddle points, the gradient $g_{t,i}$ at step t is small, leading to a small $a_{t,i}$, since it is the EMA of $g_{t,i}^2$. As $\sqrt{a_{t,i}}$ is in the denominator, Adam takes a relatively large step in the $\theta_{t,i}$ direction due to this small denominator. This adjustment strategy is effective in escaping saddle points or regions with small partial derivatives because it enables larger updates in such scenarios, which is critical for maintaining the momentum of the optimizer (Xie et al. 2022). However, for the valley dilemma case, all related $g_{t,i}$ can be large, so the corresponding $a_{t,i}$ will also be large. This results in small update step sizes for all parameters, including in directions where acceleration is needed for effective updating. Therefore, this direction is "hidden" from Adam. Adam adjusts the learning rates for each parameter separately, which allows it to perform well when dealing with problems where parameter updates are primarily aligned with the coordinate axes. However, this approach is less effective for problems that require significant updates in non-axis-aligned directions. This limitation highlights that Adam's learning rate adjustment strategy is most effective for problems that are "close to axis-aligned", as discussed in (Balles and Hennig 2018). Most other first-order adaptive optimizers face similar challenges. Therefore, in the context of the non-axis-aligned valley dilemma, the critical challenge is to identify this "hidden" direction, which we refer to as the *hidden vector*.

We provide specific examples to clarify our intuitive explanations regarding the valley dilemma and hidden vector. As shown in Figure 2, panels (a) and (b) depict two typical valley functions. The hidden vector, corresponding to the gradient at the bottom of the valley, represents the intersection line of two planes. The HVadam converges most rapidly towards the direction of the hidden vector and achieves the fastest convergence. Figure 2(a) illustrates a typical coordinate-aligned valley problem, where AdaBelief also converges quickly. However, for the non-axis-aligned valley problem represented in Figure 2(b), Adam's convergence speed is relatively slow. We have further generalized the concept of the hidden vector to functions where the hidden vector frequently changes using a restart strategy. Furthermore, we have verified the convergence performance of HVAdam in more general cases, as shown in Figure 2(c)(d), where HVAdam continues to demonstrate good convergence. These examples provide insight into the local behavior of optimizers in deep learning. Their behavior reflects common patterns seen in deep networks, such as ReLU activation, neuron connections, cross-entropy loss, and smooth activations (Zhuang et al. 2020). Further details on the analysis of these examples are available in Sec.E of the supplementary material.

HVAdam

Step	1	2	3	4	5
g_x	5	-3	5	-3	5
g_y	-3	5	-3	5	-3
$\widehat{m_x}$	5	0.7895	2.3432	0.7895	1.8177
$\widehat{m_y}$	-3	-1.2105	-0.3432	1.2105	0.1823
v_x	5	1	1	1	1
v_y	-3	1	1	1	1
v_x^*	1	1	1	1	1
v_y^*	1	1	1	1	1

Table 1: Consider $f(x, y) = 4|x-y| + |x+y|$. Optimization process for the example function. Our algorithm uses only two steps to get the hidden vector v^*.

We propose HVAdam, a first-order, full-dimension op-

timizer designed to address the non-axis-aligned valley dilemma. It not only solves the valley dilemma but also proves effective for general deep learning optimization. Specifically, we obtain a hidden vector using gradient projection, which represents the stable gradient trend of the loss function. By employing the restart strategy, we extend this approach to situations where the hidden vector may change over time. Finally, we enhance HVAdam's effectiveness through a hidden-vector-based preconditioning matrix adjustment strategy, where the hidden vector is used to adjust the learning rate. The HVAdam algorithm is summarized in Algorithm 2. All operations are element-wise, except for $\|\cdot\|$ and $\langle\cdot,\cdot\rangle$. The proof of the optimizer's convergence is shown in the supplementary material's Sec.C and Sec.D.

Algorithm 2: HVAdam Optimizer

Input: $\alpha_1, \beta_1, \beta_2, \epsilon, \gamma$
Initialize $\theta_0, \alpha_1 \leftarrow \alpha_2, m_0 \leftarrow 0, s_0 \leftarrow 0, v_0 \leftarrow 0, t \leftarrow 0, t_2 \leftarrow -1, \delta_0 \leftarrow 0$

1: **while** θ_t not converged **do**
2: $t \leftarrow t+1$
3: $t_2 \leftarrow t_2+1$
4: $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
5: $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
6: $p_t \leftarrow (g_t - v_{t-1})^2$
7: $\eta_t \leftarrow \frac{p_t}{(g_t - m_t)^2 + \gamma p_t + \epsilon}$
8: $s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2)\eta_t p_t + \epsilon$
9: **Bias Correction**
10: $\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}, \widehat{s_t} \leftarrow \frac{s_t}{1-\beta_2^t}$
11: **if** t_2 **not** 0 **then**
12: **if** $v_{t-1} = \widehat{m_t}$ **then**
13: $k_t = 0$
14: **else**
15: $k_t \leftarrow \frac{\langle v_{t_2-1} - \widehat{m_t}, v_{t_2-1}\rangle}{\|v_{t_2-1} - \widehat{m_t}\|^2}$
16: **end if**
17: $v_{t_2} \leftarrow k_t \widehat{m_t} + (1-k_t) v_{t_2-1}$
18: $\delta_{t_2} \leftarrow \beta_2 \delta_{t_2-1} + (1-\beta_2)\cos\langle v_{t_2}, \widehat{m_t}\rangle$
19: $\widehat{\delta_{t_2}} \leftarrow \frac{\delta_{t_2}}{1-\beta_2^{t_2}}$
20: $b_t \leftarrow lr(\delta_{t_2}, \widehat{\delta_{t_2}})$
21: **if** $b_t = 0$ **then**
22: $t_2 = -1$
23: **end if**
24: **else**
25: $v_0 \leftarrow \widehat{m_t}, \delta_0 \leftarrow 0$
26: **end if**
27: **Update**
28: $\theta_t \leftarrow \prod_{\mathcal{F}, \sqrt{\widehat{s_t}}} \left(\theta_{t-1} - \frac{\alpha_1 \widehat{m_t}}{\sqrt{\widehat{s_t}+\epsilon}} - \alpha_2 b_t v_t\right)$
29: **end while**

Hidden Vector

$$k_t := \begin{cases} \frac{\langle v_{t-1} - \widehat{m_t}, v_{t-1}\rangle}{\|v_{t-1} - \widehat{m_t}\|^2}, & \text{if } v_{t-1} \neq \widehat{m_t} \\ 0, & \text{otherwise} \end{cases}, \quad (1)$$

$$v_t := k_t \widehat{m_t} + (1-k_t) v_{t-1}, \quad (2)$$

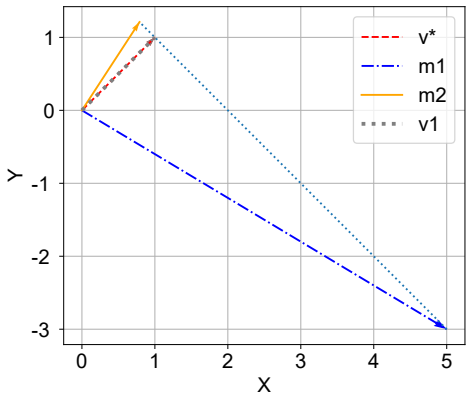


Figure 3: Consider $f(x,y) = 4|x-y| + |x+y|$. The figure shows how we make v_t approximate v^* for the function.

To obtain the hidden vector of the loss function, we analyze the relationship between the hidden vector of the bivariate function $f(x,y) = 4|x-y| + |x+y|$ in its valley region and its corresponding vectors v_t and $\widehat{m_t}$. The process is illustrated in Table 1 and Figure 3. We observe that the height of the triangle formed by the edges v_t and $\widehat{m_t}$ can be used to update v_t, leading to its convergence to the hidden vector v^*. The update process can be formulated as Eq. (1) and Eq. (2). We extend the algorithm to higher dimensions and prove its convergence in Sec.B of the Supplementary Material. The update process of the hidden vector is detailed in line 12 ~ 17 of Algorithm 2.

Restart Strategy Now we can calculate v_t through Eq. (1) and Eq. (2), however, v_t is a value that gradually converges to v^* over time. Therefore, it is necessary to measure the current convergence rate of v_t. Then, we update the parameters in the direction of v_t according to this convergence rate. Considering that the moving average of m_t can reflect the region trend of the loss function, we use the cosine similarity between v_t and $\widehat{m_t}$ to measure the convergence rate of v_t. Although $\widehat{m_t}$ can roughly reflect the current region trend, it is unstable and therefore cannot replace v_t. To make the results more stable, we use the moving average of cosine similarity as the index, which is the 18th line of Algorithm 2, that is

$$\delta_{t_2} := \beta_2 \delta_{t_2-1} + (1-\beta_2)\cos\langle v_{t_2}, \widehat{m_t}\rangle, \quad (3)$$

Furthermore, $\widehat{m_t}$ is adaptive to any changes between different local trends, while v_t cannot automatically adjust itself. Whenever the difference between $\widehat{m_t}$ and v_t becomes too large, it indicates that the trend of the region is changing, which requires the reinitiation of the calculations v_t at time t. Considering that the initialized values of v_t and $\widehat{m_t}$ do not represent an accurate estimate of the current trend, we introduce the unbiased estimate of δ_{t_2} as a criterion to determine whether a restart is needed,

$$\widehat{\delta_{t_2}} := \frac{\delta_{t_2}}{1-\beta_2^{t_2}}. \quad (4)$$

A larger δ_{t_2} means that the direction of v_t is closer to the direction of m_t, and vice versa. Therefore, a larger step in the direction of v_t can be taken, while a small δ_t indicates less confidence. When δ_t is extremely small, it implies that the hidden vector v^* has changed, so we will restart the calculation of v_t.

Finally, we obtain the step size b_t by δ_{t_2} which represents the convergence rate of v_t. $lr(\cdot)$ should be an increasing function capable of covering a wide range of magnitudes. After some experiments, we empirically select Eq. (5), following the same process as in (Luo et al. 2019). Whether to restart, we use $\widehat{\delta_{t_2}}$ as the criterion. When $\widehat{\delta_{t_2}}$is less than the threshold of 0.1, it indicates a significant deviation between v_t and $\widehat{m_t}$. In such cases, we recompute v_t starting from the current step t. The value of 0.1 here is selected empirically.

$$b_t = lr(\delta_{t_2}, \widehat{\delta_{t_2}}) := \begin{cases} 10^{\delta_{t_2} \cdot 6 - 3}, & \text{if } \widehat{\delta_{t_2}} \geq 0.1 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

The pseudocode for the restart strategy corresponds to lines 18 to 23 of Algorithm 2, where t_2 is used to indicate whether a restart is needed.

Hidden-vector-based preconditioning matrix adjustment strategy A crucial element of adaptive optimizers is the preconditioning matrix, which improves the information about the gradient and controls the step size in each direction of the gradient (Yue et al. 2023). In order to adjust the learning rate based on the stable trend information obtained from the hidden vector v_t, we measure the difference between the gradients at the current position and the trend in the current region,

$$p_t := (g_t - v_{t-1})^2. \tag{6}$$

This difference indicates the magnitude of the noise in each direction of the coordinate axis. The magnitude of noise determines the step size of parameter updates in this direction; large noise corresponds to small steps, while small noise corresponds to large steps. The p_t only represents the absolute magnitude of the noise. For different orders of magnitude of g_t, it is necessary to incorporate the relative magnitude of p_t on each dimension into the design of the learning rate adjustment strategy. Therefore, we introduce Eq. (7) to extract the relative magnitude factor of the noise in each dimension.

$$\eta_t := \frac{p_t}{(g_t - m_t)^2 + \gamma p_t + \epsilon} \tag{7}$$

We obtain the relative magnitude of the difference between g_t and v_t by comparing it to the difference between g_t and m_t. And in order to reduce the impact of abnormal data that make the denominator too small, we add γp_t. The ϵ is used to avoid a zero denominator.

Finally, the relative magnitude of noise $\eta_t p_t$ constitute the adjustment factor of the learning rate, formalized as

$$s_t := \beta_2 s_{t-1} + (1 - \beta_2)\eta_t p_t + \epsilon. \tag{8}$$

The hidden vector-based preconditioning matrix adjustment strategy is in the line 6˜8 of Algorithm 2. If p_t is large, it means that there is a significant difference between the projection of the gradient and v_t on the parameters. In this case, as the denominator, $\sqrt{s_t}$ is the EMA of $\eta_t p_t$, making α_1 small which makes updating more "cautious". If p_t is small, this means that we should accelerate the update so the small $\sqrt{s_t}$ makes α_1 large. For the denominators of Adam and AdaBelief, their value ranges are narrow. Adam's $\sqrt{a_t}$ is in $(0, \max|g|)$ and AdaBelief's is in $(0, \max|2g_t|)$. And the range of the denominator determines the adjustment range of α_1. For HVAdam, we multiply p_t by η_t. We use $(g_t - m_t)^2$ to measure p_t. If the latter is larger than the former, η_t is decreased. Otherwise, η_t is increased. The formula shows that the value range of η_t is $(0, \frac{1}{\gamma})$, which can make the range wider so that an optimal learning rate can be reached.

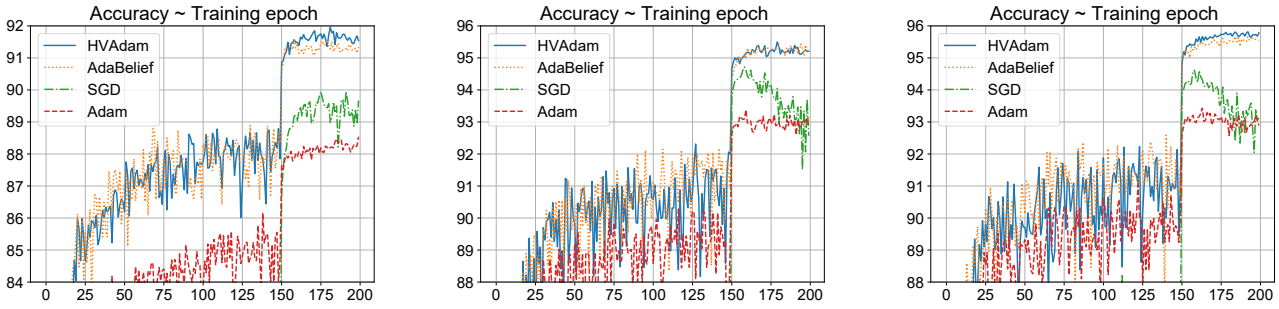
Validation on Tasks in Deep Learning

We compare HVAdam with 13 baseline optimizers in various experiments, including SGD (Sutskever et al. 2013), Adam, AdamW (Loshchilov and Hutter 2017), Yogi (Zaheer et al. 2018), AdaBound, RAdam (Liu et al. 2019), Fromage (Bernstein et al. 2020), RMSProp, SWA(Izmailov et al. 2018), Lookahead(Zhang et al. 2019), AdaBelief, Adai (Xie et al. 2022) and Lookaournd(Zhang et al. 2023), and the The experiments include: (a) image classification in the CIFAR-10 dataset and CIFAR-100 dataset (Krizhevsky, Hinton et al. 2009) with VGG (Simonyan and Zisserman 2014), ResNet (He et al. 2016) and DenseNet (Huang et al. 2017); (b) natural language processing tasks with LSTM (Ma et al. 2015) on Penn TreeBank dataset (Marcus, Santorini, and Marcinkiewicz 1993) and Transformer on IWSLT14 dataset; (c) WGAN (Arjovsky, Chintala, and Bottou 2017), WGAN-GP (Gulrajani et al. 2017) and Spectral-Norm GAN (SN-GAN) (Miyato et al. 2018) on CIFAR-10. The hyperparameter settings and searching are shown in supplementary material.

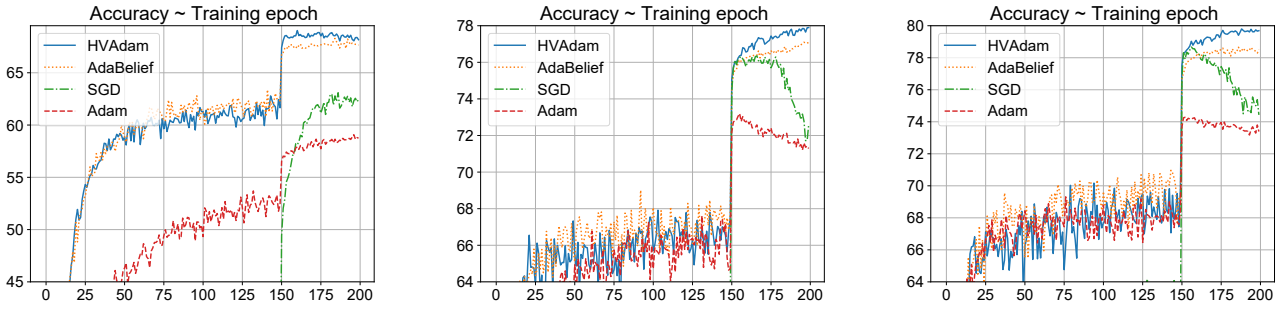
Experiments for Image Classification

CNNs on image classification The experiments are conducted on CIFAR-10 with VGG11, ResNet34, and DenseNet121 on CIFAR-10 using the *official implementation* of AdaBelief. The accuracy of other optimizers is obtained from (Buvanesh and Panwar 2021). As Figure 4 shows, HVAdam achieves a fast convergence comparable to other adaptive methods. When the training accuracy of several optimizers is close to 100%, HVAdam achieves the highest test accuracy and outperforms other optimizers. The results show that HVAdam has both fast convergence and high generalization performance.

We then train a ResNet50 on ImageNet (Deng et al. 2009) and report the accuracy on the validation set in Table 2. The experiment is conducted using the *official implementation* of Lookaround. For other optimizers, we report the best result in the literature. Due to the heavy computational burden, we were unable to perform an extensive hyperparameter search. But our optimizer still outperforms other adaptive methods and achieves comparable accuracy Lookaround (77.22 vs 77.32), which closes the generalization gap between adaptive methods and non-adaptive methods. Experiments validate the fast convergence and great generalization performance of HVAdam.



(a) VGG11 on CIFAR-10 (b) ResNet34 on CIFAR-10 (c) DenseNet121 on CIFAR-10



(d) VGG11 on CIFAR-100 (e) ResNet34 on CIFAR-100 (f) DenseNet121 on CIFAR-100

Figure 4: Test accuracies with three models using different optimizers on CIFAR-10 and CIFAR-100.

SGDM†	Adam†	Adai†	SWA‡	Lookahead‡	Lookaround‡	HVAdam
76.49	72.87	76.80	76.78	76.52	**77.32**	77.22

Table 2: Top-1 accuracy of ResNet50 on ImageNet. †is reported in (Xie et al. 2022), ‡is reported in (Zhang et al. 2023).

Visual Transformer on image classification Besides validating with the classical model CNNs, we also validate the performance of HVAdam with Visual Transformer (ViT)(Dosovitskiy et al. 2021). As Table 3 shows, HVAdam outperforms all other optimizers, which means that HVAdam performs well on both classical and advanced models.

	Adam	SWA	Lookahead	Lookaround	HVAdam
CIFAR10	98.34	98.47	98.51	98.71	**99.00**
CIFAR100	91.55	91.32	91.76	92.21	**92.38**

Table 3: The test set accuracy under optimizers using ViT-B/16. The results of other optimizers are taken from (Zhang et al. 2023).

Experiments for Natural Language Processing

LSTMs on language modeling We experiment with 1-layer, 2-layer, and 3-layer LSTM models on the Penn TreeBank dataset. The results of the experiments are shown in Figure 5. Except for HVAdam, the score data for the other optimizers is provided by (Buvanesh and Panwar 2021). For all LSTM models, AdaBelief achieves the lowest perplexity or the best performance. The experiments resonate with both the fast convergence and the excellent accuracy of HVAdam.

Experiments for Image Generation

Generative adversarial networks (GANs) on image generation Stability is also important for optimizers. And, as mentioned in (Salimans et al. 2016), mode collapse and numerical instability can easily impact GAN training. So, training GANs can reflect the stability of optimizers. For example, SGD often fails when training GANs, whereas Adam can effectively train GANs. To assess the robustness of HVAdam, we performed experiments with WGAN, WGAN-GP, and SN-GAN using the CIFAR-10 dataset. The results were generated using the code from the *official implementation* of AdaBelief and compared with the findings in (Buvanesh and Panwar 2021), where hyperparameters were explored more extensively. We perform 5 runs of experiments, and The comparison results on WGAN, WGAN-GP, and SN-GAN are reported in Figure 6, and Table 4 . And we also add the great results of expriments on diffusion model (Nichol and Dhariwal 2021) in the supplementary material. We can see that HVAdam gets the lowest FID (Heusel et al. 2017) scores with all GANs, where the lower the FID is, the better the quality and diversity of the generated images. Fur-

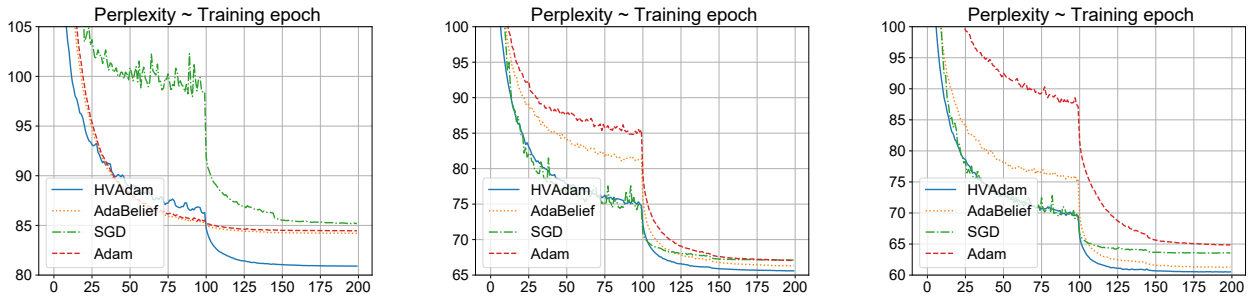
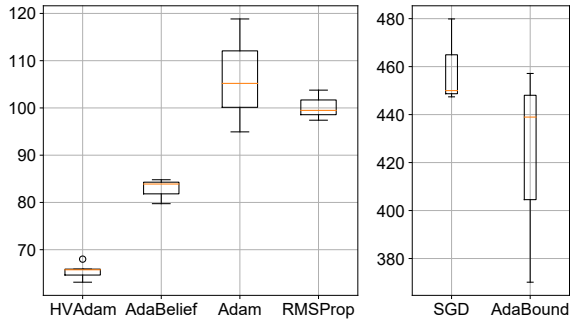


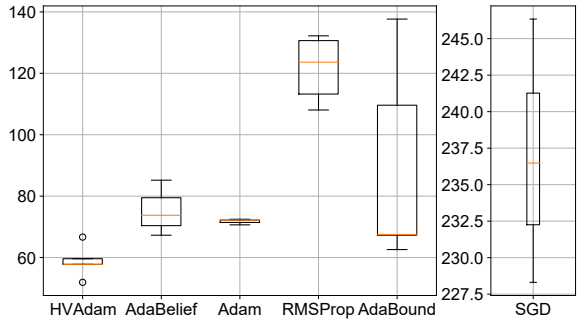
Figure 5: The perplexity on Penn Treebank for 1,2,3-layer LSTM from left to right. **Lower** is better.

HVAdam	AdaBelief	RAdam	RMSProp	Adam	Fromage	Yogi	SGD	AdaBound
12.72±0.21	12.98±0.22	13.10±0.20	12.86±0.08	13.01±0.15	46.31±0.86	14.16±0.05	48.94±2.88	16.84±0.10

Table 4: FID values ($[\mu \pm \sigma]$) of a SN-GAN with ResNet generator on CIFAR-10. A lower FID value is better.



(a) FID scores of WGAN.



(b) FID scores of WGAN-GP.

Figure 6: FID scores of different optimizers on WGAN and WGAN-GP (using the vanilla CNN generator on CIFAR-10). Lower FID indicates better performance. For every model, the successful and failed optimizers are displayed on the left and right sides, with different ranges of y values.

thermore, HVAdam's FID score with WGAN is even better than the other optimizers' FID scores with WGAN-GP. So, the stability of HVAdam is fully validated.

Ablation Study

We perform the ablation study to demonstrate that every step of improvement plays important roles. We use three methods to train a 1-layer-LSTM. (a) Use Adam (HVAdam^0). (b) Use Adam which we introduce v_t (HVAdam^1). (c) After introducing v_t, we can change Adam's adjustment strategy of learning rate by using v_t to calculate s_t (HVAdam^2). (d) Based on HVAdam^2, we introduce the restart strategy (HVAdam). As Table. 5 shows, all the steps can help the optimizer perform better.

	HVAdam^0	HVAdam^1	HVAdam^2	HVAdam
valid ppl	85.04	84.54	83.42	83.31

Table 5: The valid perplexity in ablation study.

Conclusion

We propose the HVAdam optimizer, which obtains the trend of the loss function through a hidden vector and restart strategy and utilizes the trend to update the learning rate, thereby accelerating the model's convergence speed. We validate HVAdam's advantages with four simple but representative functions and prove its convergence in both convex and non-convex cases. Experimental results show that HVAdam outperforms almost all other optimizers in tasks that cover a comprehensive category of deep learning tasks. Although our optimizer achieves excellent results and faster convergence speed in these experiments, the algorithm's calculation process is relatively complex and incurs extra memory overhead. We recognize that these factors may limit the algorithm's practicality in memory-constrained environments. In future work, we plan to optimize the algorithm's computational efficiency and reduce its memory requirements to enhance its applicability.

Acknowledgements

This project was supported by the Key Research and Development Project of Hubei Province under grant number 2022BCA057. The numerical calculations in this paper have been done on the supercomputing system in the Supercomputing Center of Wuhan University.

References

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein generative adversarial networks. In *International conference on machine learning*, 214–223. PMLR.

Balles, L.; and Hennig, P. 2018. Dissecting adam: The sign, magnitude and variance of stochastic gradients. In *International Conference on Machine Learning*, 404–413. PMLR.

Bernstein, J.; Vahdat, A.; Yue, Y.; and Liu, M.-Y. 2020. On the distance between two neural networks and the stability of learning. *Advances in Neural Information Processing Systems*, 33: 21370–21381.

Buvanesh, A.; and Panwar, M. 2021. [Re] AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients. In *ML Reproducibility Challenge 2021 (Fall Edition)*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *ICLR*.

Graves, A. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; and Courville, A. C. 2017. Improved training of wasserstein gans. In *Advances in neural information processing systems*, 5767–5777.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Heusel, M.; Ramsauer, H.; Unterthiner, T.; Nessler, B.; and Hochreiter, S. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, 6626–6637.

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4700–4708.

Izmailov, P.; Podoprikhin, D. A.; Garipov, T.; Vetrov, D.; and Wilson, A. G. 2018. Averaging Weights Leads to Wider Optima and Better Generalization. *Conference on Uncertainty in Artificial Intelligence*.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images.

Liu, L.; Jiang, H.; He, P.; Chen, W.; Liu, X.; Gao, J.; and Han, J. 2019. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*.

Loshchilov, I.; and Hutter, F. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Luo, L.; Xiong, Y.; Liu, Y.; and Sun, X. 2019. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*.

Ma, X.; Tao, Z.; Wang, Y.; Yu, H.; and Wang, Y. 2015. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*, 54: 187–197.

Marcus, M.; Santorini, B.; and Marcinkiewicz, M. A. 1993. Building a large annotated corpus of English: The Penn Treebank.

Miyato, T.; Kataoka, T.; Koyama, M.; and Yoshida, Y. 2018. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*.

Nichol, A. Q.; and Dhariwal, P. 2021. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, 8162–8171. PMLR.

Robbins, H.; and Monro, S. 1951. A stochastic approximation method. *The annals of mathematical statistics*, 400–407.

Salimans, T.; Goodfellow, I.; Zaremba, W.; Cheung, V.; Radford, A.; and Chen, X. 2016. Improved techniques for training gans. In *Advances in neural information processing systems*, 2234–2242.

Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sutskever, I.; Martens, J.; Dahl, G.; and Hinton, G. 2013. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, 1139–1147.

Xie, Z.; Wang, X.; Zhang, H.; Sato, I.; and Sugiyama, M. 2022. Adaptive inertia: Disentangling the effects of adaptive learning rate and momentum. In *International Conference on Machine Learning*, 24430–24459. PMLR.

Yue, Y.; Ye, Z.; Jiang, J.; Liu, Y.; and Zhang, K. 2023. AGD: an Auto-switchable Optimizer using Stepwise Gradient Difference for Preconditioning Matrix. *Advances in Neural Information Processing Systems*, 36: 45812–45832.

Zaheer, M.; Reddi, S.; Sachan, D.; Kale, S.; and Kumar, S. 2018. Adaptive methods for nonconvex optimization. In *Advances in neural information processing systems*, 9793–9803.

Zhang, J.; Liu, S.; Song, J.; Zhu, T.; Xu, Z.; and Song, M. 2023. Lookaround Optimizer: k steps around, 1 step average. In *Advances in Neural Information Processing Systems*.

Zhang, M.; Lucas, J.; Ba, J.; and Hinton, G. E. 2019. Lookahead Optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems*, 9593–9604.

Zhuang, J.; Tang, T.; Ding, Y.; Tatikonda, S. C.; Dvornek, N.; Papademetris, X.; and Duncan, J. 2020. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Advances in neural information processing systems*, 33: 18795–18806.