Improving Adaptive Moment Optimization via Preconditioner Diagonalization

Son Nguyen **Bo Liu** **Lizhang Chen** **Qiang Liu**
The University of Texas at Austin
{sonnv77,bliu,lzchen,lqiang}@utexas.edu

Abstract

Modern deep learning heavily relies on adaptive optimization methods like Adam and its variants, celebrated for their robustness against model scale and ease of hyperparameter tuning. However, the gradient statistics employed by these methods often do not leverage sufficient gradient covariance information, leading to suboptimal updates in certain directions of the parameter space and potentially slower convergence. In this work, we keep track of such covariance statistics in the form of a structured preconditioner matrix. Unlike other works, our approach does not apply direct approximations to estimate this matrix. We instead *implement an invertible transformation that maps the preconditioner matrix into a new space where it becomes approximately diagonal*. This enables a diagonal approximation of the preconditioner matrix in the transformed space, offering several computational advantages. Empirical results show that our approach can substantially enhance the convergence speed of modern adaptive optimizers. Notably, for large language models like LLaMA, we can achieve a 2× speedup in sample efficiency compared to Adam. In addition, our method can also be integrated with memory-efficient optimizers to manage computational overhead.

1. INTRODUCTION

In the realm of deep learning optimization, finding efficient and reliable solutions to complex problems has become a key challenge. As model scales and datasets continue to expand, such optimization problems demand extensive training time and substantial computational resources to achieve breakthrough performance.

Standard first-order methods, such as stochastic gradient descent (SGD) and its variants, have emerged as canonical tools for training large-scale deep networks. These methods are straightforward to implement using modern automatic differentiation frameworks and are easily adaptable to non-conventional training setups (Konečnỳ et al., 2016; Li et al., 2020; Finn et al., 2017). However, despite their strong theoretical grounding (Şimşekli et al., 2019; Zhou et al., 2020; Smith et al., 2021; Tian et al., 2023; Allen-Zhu et al., 2019), first-order methods typically require meticulous tuning of hyperparameters to ensure the optimization process can converge to the desired local optima (Zhang et al., 2022). In practice, these methods often struggle when navigating highly non-convex loss surfaces, a common characteristic of deep learning models. Pathological features like saddle points, flat regions, and sharp valleys in the loss landscape can significantly hinder convergence, leading to inefficient training.

To tackle these challenges, optimization techniques have evolved to incorporate curvature geometry or second-order information, providing more adaptive and efficient updates. A classic family of such algorithms is preconditioned gradient methods, in which the gradient is premultiplied by a matrix called a preconditioner before each optimization step. Classic algorithms in this family include Newton methods (Bonnans et al., 2006) and Natural Gradient (Martens, 2020), which employ the inverses of local Hessian and Fisher Information Matrix as preconditioners, respectively. Although preconditioning methods typically exhibit much faster convergence than first-order approaches, their practical application is limited by the size of most real-world problems, as they demand quadratic storage and cubic computation time for each gradient update (Fletcher, 2000; Bonnans et al., 2006).

In addition to preconditioning, adaptive moment estimation is another highly influential line of work in

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

deep learning optimization. These methods, including AdaGrad (Duchi et al., 2011), Adam (Kingma and Ba, 2014), AMSGrad (Reddi et al., 2019), and Adafactor (Shazeer and Stern, 2018) dynamically adapt the learning rate of each parameter throughout the optimization process by leveraging cumulative second-order gradient statistics. While their theoretical foundations are not yet fully explored, these algorithms have demonstrated more robust empirical results than SGD in various domains and often exhibit better convergence behaviors in practice (Zhang et al., 2020; Kunstner et al., 2024; Zhang et al., 2024).

Our approach. In this work, we explore adaptive moment-based algorithms from the perspective of preconditioning methods. We are driven by the understanding that the second-moment estimates in these algorithms can be derived from the diagonal approximation of a structured preconditioner. We propose to improve this approximation by applying an invertible transformation that maps the preconditioner into a new space where it becomes approximately diagonal. In this transformed space, the diagonal elements of the preconditioner can be accumulated to estimate second-order statistics better. Since the transformation is invertible, we can formulate the update on the original parameter space by doing a simple projection back.

Our key contributions are outlined as follows:

1. Our approach is designed to be both straightforward and versatile, facilitating easy integration into existing adaptive moment-based optimizers such as RMSprop, Adam, and its variants.

2. We establish a convergence guarantee for the general framework without requiring typical strong assumptions. This guarantee is significant because it is broadly applicable to a wide range of adaptive optimizers, ensuring reliable performances in diverse scenarios.

3. We can effortlessly adapt our method to memory-efficient optimizers, enabling practical training of large models while preserving strict convergence guarantees.

4. Empirical results show that our proposed methods can substantially enhance the convergence speed and efficiency of adaptive moment-based optimization baselines. Particularly in pretraining large-scale models like LLaMA, our approach can achieve a speedup of 1.5x to 2x compared to the Adam, but with manageable computational overhead.

Notations: For any matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m\times n}$, $\sqrt{\mathbf{A}}$, $\mathbf{A}^2$, and $\mathbf{A}/\mathbf{B}$ denote element-wise square root, square, and division. $\mathbf{A}^\top$ is the transpose, $\langle \mathbf{A}, \mathbf{B}\rangle = \mathrm{tr}(\mathbf{A}^\top\mathbf{B})$ is the matrix inner product, and $\mathbf{A}\otimes\mathbf{B}$ is the Kronecker product. Let $\mathrm{diag}(\cdot)$ and $\mathrm{vec}(\cdot)$ denote the diagonal and vectorization operators of a matrix, respectively.

2. PRELIMINARIES AND BACKGROUND

We consider an unconstrained, continuous optimization problem $\min_{\mathbf{W}\in\mathbb{R}^d} \mathcal{L}(\mathbf{W};\mathbf{X})$, with $\mathbf{X}$ denotes observations, $\mathcal{L}:\mathbb{R}^d\to\mathbb{R}$ is a proper differentiable and lower bounded objective function.

2.1. Preconditioned Gradient Descent

The iterative scheme of preconditioned gradient descent can be expressed as follows:

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_t \mathbf{C}(t)\nabla\mathcal{L}(\mathbf{W}_t;\mathbf{X}), \tag{1}$$

where the matrix $\mathbf{C}(t)$ is referred to as preconditioner. When $\mathbf{C}(t)$ is set to the identity matrix, the update simplifies to ordinary gradient descent. To capture curvature informativeness, systematic designs of $\mathbf{C}(t)$ have been developed using local numerical approximations. Classic algorithms in this category, including Newton methods and Natural Gradient, utilize the inverse of Hessian and Fisher Information Matrix, respectively, as preconditioners. These methods offer a built-in mechanism for curvature awareness, promoting larger updates in directions associated with small Hessian eigenvalues to swiftly navigate flat regions while limiting movement in directions with large Hessian eigenvalues to avoid sharp valleys. However, for large-scale models, further approximations to the preconditioners are necessary to ensure their practicality. Various techniques have been proposed for this purpose, such as Quasi-Newton methods (Fletcher, 2000), Gaussian-Newton estimators (Botev et al., 2017; Martens, 2020; Liu et al., 2023), K-FAC (Martens and Grosse, 2015; Grosse and Martens, 2016).

2.2. Adaptive Moment Estimation

These methods, such as AdaGrad and Adam dynamically adjust learning rates for each parameter by incorporating a form of gradient-based statistics. Specifically with Adam, it estimates both first and second-order moments by maintaining exponential moving averages (EMA) on the mean and uncentered variance of gradient information across iterations. Let $\mathbf{G}_\tau \triangleq \nabla\mathcal{L}(\mathbf{W}_\tau;\mathbf{X})$ denote the gradient of loss function at iteration τ, the update rules are defined as:

$$\mathbf{M}_t = \hat\beta_{1t}\mathbf{M}_{t-1} + (1-\hat\beta_{1t})\mathbf{G}_t \triangleq \underset{\tau\in[t]}{\mathrm{EMA}}\left[\mathbf{G}_\tau\right]$$

$$\mathbf{V}_t = \hat\beta_{2t}\mathbf{V}_{t-1} + (1-\hat\beta_{2t})\mathbf{G}_t^2 \triangleq \underset{\tau\in[t]}{\mathrm{EMA}}\left[\mathbf{G}_\tau^2\right]$$

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_t \frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon}$$

where $\hat\beta_{1t}, \hat\beta_{2t}$ are the decay moment coefficients, ϵ is the smoothing tolerance constant to avoid numerical

instability. In principle, the first moment amplifies the gradient in directions that are consistently the same sign and dampens the gradient in directions that are reversing sign. Meanwhile, the second moment captures the curvature by adjusting the step size based on gradient magnitude: smaller steps in steep-gradient regions to avoid overshooting and larger steps in shallow-gradient regions for faster convergence.

2.3. Adaptive Moment Estimation via Diagonal Preconditioning Approximation

Let us examine the matrix case where $\mathbf{W}$ represents a weight parameter with dimensions $m \times n$. We analyze a preconditioner $\mathbf{C}_t$ exploiting the second-order moment of accumulated gradients in the following *inverse* form:

$$\mathbf{C}_t = \left[\mathbb{E}_{p(\mathbf{X})}[\mathrm{vec}(\mathbf{G}_t)\mathrm{vec}(\mathbf{G}_t)^\top] + \epsilon \mathbf{I}_{mn}\right]^{-1/2} \quad (2)$$

We have $\mathbf{C}_t$ as a positive definite matrix of size $mn \times mn$, which is quadratic to the size of model parameter $\mathbf{W}$. An analytical formulation of this quality is often intractable in practice. However, under the assumption of stationary gradient distribution, we can approximate the expectation by leveraging mini-batch sampling in conjunction with the exponential moving average technique. We then obtain an empirical preconditioner defined by:

$$\mathbf{C}_t = \left[\underset{\tau\in[t]}{\mathrm{EMA}}\left[\mathrm{vec}(\mathbf{G}_\tau)\mathrm{vec}(\mathbf{G}_\tau)^\top\right] + \epsilon \mathbf{I}_{mn}\right]^{-1/2}, \quad (3)$$

where $\mathbf{G}_\tau := \frac{1}{|\mathcal{B}_\tau|}\sum_{\mathbf{X}\in\mathcal{B}_\tau} \nabla\mathcal{L}(\mathbf{W}_\tau; \mathbf{X})$ is the minibatch gradient at training step τ. This empirical preconditioner closely resembles the full matrix version of AdaGrad (Duchi et al., 2011), but instead of using a cumulative sum, we apply an exponential moving average (EMA).

Directly calculating and storing the matrix $\mathbf{C}_t$ is computationally expensive, particularly with modern network architectures, since it requires inverting a very large matrix. A practical way to alleviate this bottleneck is by using diagonal approximation:

$$\begin{aligned}\mathbf{C}_t^{(d)} &= \left[\underset{\tau\in[t]}{\mathrm{EMA}}\,\mathrm{diag}\left(\mathrm{vec}(\mathbf{G}_\tau)\mathrm{vec}(\mathbf{G}_\tau)^\top\right) + \epsilon \mathbf{I}_{mn}\right]^{-1/2} \\ &= \mathrm{diag}^{-1/2}\left(\underset{\tau\in[t]}{\mathrm{EMA}}\left[\mathrm{vec}(\mathbf{G}_\tau^2) + \epsilon\right]\right). \end{aligned} \quad (4)$$

The diagonal preconditioner $\mathbf{C}_t^{(d)}$ represents the inverse root square of the second-order gradient accumulator, which is widely adopted as the adaptive moment estimation in optimizers such as AdaGrad, RMSprop, Adam, and variants. Implementing this diagonal approximation offers advantages in both computational efficiency and memory usage. Amari et al. (2019) also demonstrate that the off-diagonal components of $\mathbf{C}_t$ are smaller than the diagonal components by a factor of $1/\sqrt{N}$, where N is the number of elements in the matrix. This insight contributes to understanding the practical success of optimizers like AdaGrad, Adam, and others. However, by omitting the off-diagonal elements, the algorithm does not incorporate gradient correlations, which can be particularly useful in accelerating optimization (Martens and Grosse, 2015; Gupta et al., 2018; Liu et al., 2023).

Algorithm 1 AdaDiag and AdaDiag++ for matrix parameter $\mathbf{W}$ of size $m \times n$, $m \le n$

Inputs: moment coefficients β_1, β_2, smoothing term $\epsilon = 10^{-8}$, regularization constant λ
Initialization: weight parameters $\mathbf{W}_0 \in \mathbb{R}^{m\times n}$, initial moments $\mathbf{M}_0, \mathbf{V}_0 \leftarrow 0$
repeat
  $t \leftarrow t+1$
  $\mathbf{G}_t = \nabla\mathcal{L}(\mathbf{W}_t; \mathcal{B}_t)$
  if $t \bmod T = 0$ **then**
    $\mathbf{P}_t$, _, $\mathbf{Q}_t^\top$ = `torch.linalg.svd`($\mathbf{G}_t$, `full_matrices=True`)
  else
    $\mathbf{P}_t, \mathbf{Q}_t^\top \leftarrow \mathbf{P}_{t-1}, \mathbf{Q}_{t-1}^\top$
  end if
  $\widetilde{\mathbf{G}}_t = \mathbf{P}_t^\top \mathbf{G}_t$ $\widetilde{\mathbf{G}}_t = \mathbf{P}_t^\top \mathbf{G}_t \mathbf{Q}_t$
  $\mathbf{M}_t = \hat{\beta}_{1t}\mathbf{M}_{t-1} + (1-\hat{\beta}_{1t})\widetilde{\mathbf{G}}_t$
  $\mathbf{V}_t = \hat{\beta}_{2t}\mathbf{V}_{t-1} + (1-\hat{\beta}_{2t})\widetilde{\mathbf{G}}_t^2$
  $\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_t\left(\mathbf{P}_t\frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon} + \lambda\mathbf{W}_t\right)$
  $\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_t\left(\mathbf{P}_t\frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon}\mathbf{Q}_t^\top + \lambda\mathbf{W}_t\right)$
until *stopping criterion is met*
return optimized parameter $\mathbf{W}_t$

3. PRECONDITIONER DIAGONALIZATION WITH GRADIENT PROJECTION

To leverage the off-diagonal components of the preconditioner matrix, one can implement structural approximations, like Gaussian-Newton estimators (Botev et al., 2017; Martens, 2020; Liu et al., 2023), or Kronecker factorization (Martens and Grosse, 2015; Gupta et al., 2018). In this section, we approach the problem from a different perspective of preconditioner diagonalization. Specifically, we will rationalize the diagonal approximation assumption by applying an implicit orthogonal transformation on the preconditioner matrix $\mathbf{C}_t$. Intuitively, this technique will rotate the gradients to align with coordinate axes partially, ultimately caus-

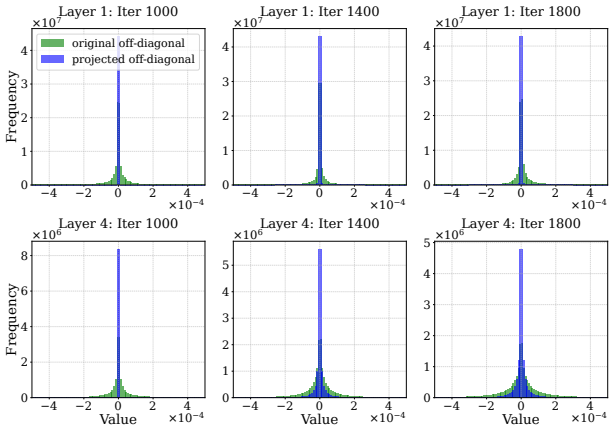

Figure 1: Histograms of off-diagonal elements $\mathcal{C}(\mathbf{G}_\tau)$ (original) and $\mathcal{C}(\widetilde{\mathbf{G}}_\tau)$ (*two-sided* projection), corresponding to the two first layers of ResNet50 trained on ImageNet1k. In this experiment, we set the frequency $T = 500$ and plot histograms at iterations with and without SVD applied.

ing the matrix $\mathbf{C}_t$ to become approximately diagonal. Moreover, we will show that this transformation is invertible via a network reparameterization, leading to a simple update on the original parameter space.

Recall $\mathbf{G}_\tau$ is a matrix of size $m \times n$, with $m \leq n$. Define $\mathcal{C}(\mathbf{G}_\tau) \triangleq \text{vec}(\mathbf{G}_\tau)\text{vec}(\mathbf{G}_\tau)^\top$, then:

$$\mathbf{C}_t = \left[\underset{\tau\in[t]}{\text{EMA}}\left[\mathcal{C}(\mathbf{G}_\tau)\right] + \epsilon\mathbf{I}_{mn}\right]^{-1/2}. \tag{5}$$

Let's start by drawing some intuitions through the diagonalization of matrix $\mathcal{C}(\mathbf{G}_\tau)$. Given the special formula of $\mathcal{C}(\mathbf{G}_\tau)$, we can perform a straightforward approach using Singular Value Decomposition (SVD) on the gradient $\mathbf{G}_\tau$. Suppose we have $\mathbf{G}_\tau = \mathbf{P}_\tau\mathbf{\Sigma}_\tau\mathbf{Q}_\tau^\top$, in which $\mathbf{P}_\tau, \mathbf{Q}_\tau$ are orthogonal matrices of size $m \times m, n \times n$, respectively, and $\mathbf{\Sigma}_\tau$ is a diagonal matrix of size $m \times n$. Substituting this representation into $\mathcal{C}(\mathbf{G}_\tau)$ gives us:

$$\begin{aligned}\mathcal{C}(\mathbf{G}_\tau) &= \text{vec}(\mathbf{P}_\tau\mathbf{\Sigma}_\tau\mathbf{Q}_\tau^\top)\text{vec}(\mathbf{P}_\tau\mathbf{\Sigma}_\tau\mathbf{Q}_\tau^\top)^\top \\ &= (\mathbf{Q}_\tau \otimes \mathbf{P}_\tau)\text{vec}(\mathbf{\Sigma}_\tau)\text{vec}(\mathbf{\Sigma}_\tau)^\top(\mathbf{Q}_\tau \otimes \mathbf{P}_\tau)^\top\end{aligned}$$

Since $\mathbf{\Sigma}_\tau$ is a diagonal matrix, we have $\text{vec}(\mathbf{\Sigma}_\tau)\text{vec}(\mathbf{\Sigma}_\tau)^\top$ is almost diagonal (off-diagonal elements are mostly zero). Moreover, the matrix $\mathbf{Q}_\tau \otimes \mathbf{P}_\tau$ satisfies $(\mathbf{Q}_\tau \otimes \mathbf{P}_\tau)(\mathbf{Q}_\tau \otimes \mathbf{P}_\tau)^\top = (\mathbf{Q}_\tau\mathbf{Q}_\tau^\top) \otimes (\mathbf{P}_\tau\mathbf{P}_\tau^\top) = \mathbf{I}_{mn}$, so we can consider $\mathbf{Q}_\tau \otimes \mathbf{P}_\tau$ as an orthogonal diagonalizing matrix with:

$$(\mathbf{Q}_\tau \otimes \mathbf{P}_\tau)^{-1}\mathcal{C}(\mathbf{G}_\tau)(\mathbf{Q}_\tau \otimes \mathbf{P}_\tau) = \text{vec}(\mathbf{\Sigma}_\tau)\text{vec}(\mathbf{\Sigma}_\tau)^\top.$$

Alternatively, the diagonalization process above can be equivalently derived from $\mathcal{C}(\widetilde{\mathbf{G}}_\tau)$, with $\widetilde{\mathbf{G}}_\tau \triangleq \mathbf{P}_\tau^\top\mathbf{G}_\tau\mathbf{Q}_\tau = \mathbf{\Sigma}_\tau$. This rotation aligns the gradient $\widetilde{\mathbf{G}}_t$ with coordinate axes and consequently induces a roughly diagonal structure on $\mathcal{C}(\widetilde{\mathbf{G}}_\tau)$.

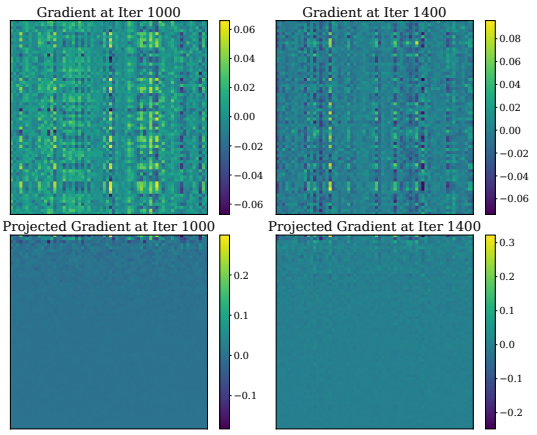

Figure 2: Sparsity of *one-sided* projection.

3.1. Periodic Subspace Projection

The analysis presented above applies only to the iteration in which SVD is implemented. However, it is impractical to utilize SVD at every iteration of the training procedure due to computational overhead. Fortunately, there are recent works on low-rank gradient projection (Gur-Ari et al., 2018; Gooneratne et al., 2020; Zhao et al., 2024; Liang et al., 2024) indicating that the optimization process usually acts on low-dimensional subspaces. GaLore (Zhao et al., 2024) exploits this concept by showing that the training trajectory can be divided into continual subspaces, from which the gradients within each subspace can be governed by a common spanning basis. GaLore deployed this idea by periodically applying SVD on the gradients to extract projection matrices. Mathematically, during each period of length T, say $[\kappa T, (\kappa+1)T]$, the gradient $\mathbf{G}_\tau$ can be decomposed as:

$$\mathbf{G}_\tau \approx \mathbf{P}_\kappa\mathbf{\Sigma}_\tau\mathbf{Q}_\kappa^\top \quad \forall\tau \in [\kappa T, (\kappa+1)T]$$

where $\mathbf{P}_\kappa, \_, \mathbf{Q}_\kappa^\top \leftarrow \text{SVD}(\mathbf{G}_{\kappa T})$ are kept the same throughout the period.

We can adapt this assumption to our framework. In this way, we can expect that our projected gradients in each period are still approximately diagonal, namely:

$$\widetilde{\mathbf{G}}_\tau \triangleq \mathbf{P}_\kappa^\top\mathbf{G}_\tau\mathbf{Q}_\kappa \approx (\mathbf{P}_\kappa^\top\mathbf{P}_\kappa)\mathbf{\Sigma}_\tau(\mathbf{Q}_\kappa^\top\mathbf{Q}_\kappa) = \mathbf{\Sigma}_\tau.$$

As a result, we can achieve a desired diagonal structure on $\mathcal{C}(\widetilde{\mathbf{G}}_\tau), \forall\tau \in [\kappa T, (\kappa+1)T]$.

Why the full matrices $\mathbf{P}_\kappa, \mathbf{Q}_\kappa$ matter: connection and difference with Galore. It is essential to note that, since GaLore's focus is on memory efficiency, they just implemented truncated SVD to capture the top-K representation of the gradient matrices. This assumption is reasonable when considering that gradients are low-rank matrices. However, using truncated SVD involves careful tweaking of the K values and

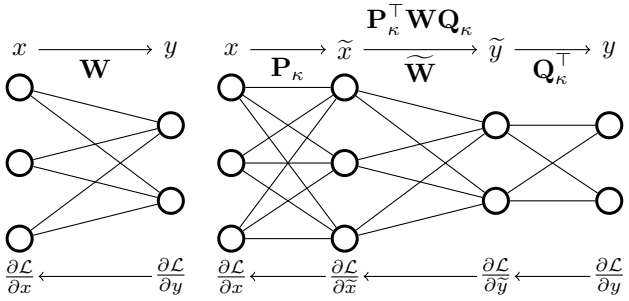

Figure 3: Illustration of network reparameterization induced by full-rank gradient projection.

more importantly, the gradient projection step has to be executed on the smaller dimension of the matrix.[1] On the other hand, our framework would require sophisticated modifications in GaLore, of which $\mathbf{P}_\kappa, \mathbf{Q}_\kappa$ need to be the full matrices or K is at least the effective rank. This guarantees that $\mathbf{P}_\kappa, \mathbf{Q}_\kappa$ can form complete bases for the rows and columns of the gradient matrices in each period. To make the algorithm effortless, we propose to adopt the full matrices instead of tunning effective rank K.

One-sided projection. Instead of using *two-sided* projection as described so far, we can opt for a simpler version involving just *one-sided* projection, namely $\widetilde{\mathbf{G}}_\tau \triangleq \mathbf{P}_\tau^\top \mathbf{G}_\tau = \mathbf{\Sigma}_\tau \mathbf{Q}_\tau^\top$, then we have:

$$\begin{aligned}\mathcal{C}(\widetilde{\mathbf{G}}_\tau) &\triangleq \mathrm{vec}(\widetilde{\mathbf{G}}_\tau)\mathrm{vec}(\widetilde{\mathbf{G}}_\tau)^\top = \mathrm{vec}(\mathbf{\Sigma}_\tau \mathbf{Q}_\tau^\top)\mathrm{vec}(\mathbf{\Sigma}_\tau \mathbf{Q}_\tau^\top)^\top \\ &= (\mathbf{Q}_\tau \otimes \mathbf{I}_m)\mathrm{vec}(\mathbf{\Sigma}_\tau)\mathrm{vec}(\mathbf{\Sigma}_\tau)^\top (\mathbf{Q}_\tau \otimes \mathbf{I}_m)^\top.\end{aligned}$$

The projected gradient $\widetilde{\mathbf{G}}_\tau = \mathbf{\Sigma}_\tau \mathbf{Q}_\tau^\top$ inherently exhibits a sparse structure as illustrated in Fig. 2. This is because $\mathbf{\Sigma}_\tau$ is a diagonal matrix and the smallest singular values on the diagonal will zero out the magnitude of corresponding rows on $\mathbf{Q}_\tau$. In Fig. 1 and Fig. 12, we further show the histograms of off-diagonal elements of $\mathcal{C}(\mathbf{G}_\tau)$ and $\mathcal{C}(\widetilde{\mathbf{G}}_\tau)$ (both *two-sided* and *one-sided*), corresponding to the two first layers of ResNet50 trained on ImageNet1k. We can observe notable differences in sparsity patterns of off-diagonal elements of $\mathcal{C}(\widetilde{\mathbf{G}}_\tau)$ compared to the original $\mathcal{C}(\mathbf{G}_\tau)$ over iterations. The matrix $\mathcal{C}(\widetilde{\mathbf{G}}_\tau)$ is roughly diagonal, enabling a more accurate diagonal approximation for the preconditioner matrix $\widetilde{\mathbf{C}}_t$.

[1] Let's omit subscripts τ, κ for simplicity. Consider $\mathbf{G}_{m\times n} \approx \mathbf{P}_{m\times K}\mathbf{\Sigma}_{K\times K}\mathbf{Q}_{n\times K}^\top$ as the truncated SVD of gradient matrix $\mathbf{G}_{m\times n}$ with $m \le n$. The GaLore algorithm performs a gradient projection as $\widetilde{\mathbf{G}} = \mathbf{P}_{m\times K}^\top \mathbf{G}$, which maps n vectors of size m from $\mathbf{G}$ onto the subspace spanned by K vectors of size m from $\mathbf{P}_{m\times K}$. Since $K \le m \le n$, this operator is more effective than projecting in larger dimensions, i.e. $\widetilde{\mathbf{G}} = \mathbf{G}\mathbf{Q}_{n\times K}$.

3.2. Gradient Projection Implies Network Reparameterization

In the previous sections, we demonstrated that we can rotate the gradients $\mathbf{G}_\tau$ to $\widetilde{\mathbf{G}}_\tau$ in such a way that the matrix $\mathcal{C}(\widetilde{\mathbf{G}}_\tau)$ is approximately diagonal. Consequently, it induces a diagonal approximation of $\widetilde{\mathbf{C}}_t$ as follows:

$$\begin{aligned}\widetilde{\mathbf{C}}_t^{(d)} &= \left[\underset{\tau\in[t]}{\mathrm{EMA}}\, \mathrm{diag}\left[\mathrm{vec}(\widetilde{\mathbf{G}}_\tau)\mathrm{vec}(\widetilde{\mathbf{G}}_\tau)^\top\right] + \epsilon \mathbf{I}_{mn}\right]^{-1/2} \\ &= \mathrm{diag}^{-1/2}\left[\underset{\tau\in[t]}{\mathrm{EMA}}\left[\mathrm{vec}(\widetilde{\mathbf{G}}_\tau^2) + \epsilon\right]\right], \qquad (6)\end{aligned}$$

The preconditioned gradient at iteration t then becomes:

$$\widetilde{\mathbf{C}}_t^{(d)}\mathrm{vec}(\widetilde{\mathbf{G}}_t) = \frac{\mathrm{vec}(\widetilde{\mathbf{G}}_t)}{\sqrt{\mathrm{EMA}_{\tau\in[t]}\left[\mathrm{vec}(\widetilde{\mathbf{G}}_\tau^2) + \epsilon\right]}} \qquad (7)$$

However, this quantity cannot be directly applied to update the weight parameters, as the gradient projection implicitly imposes a reparameterization on the weight space. In Fig. 3, we illustrate how the update in Eq. (7) can be utilized to learn a rotated network rather than the original one. This rotated network introduces two auxiliary layers defined by $\mathbf{P}_\kappa, \mathbf{Q}_\kappa^\top$, and reparameterizes the original weight parameters as $\widetilde{\mathbf{W}} = \mathbf{P}_\kappa^\top \mathbf{W} \mathbf{Q}_\kappa$. While this transformation does not alter the forward pass of the original network, it does lead to a corresponding gradient, represented as:

$$\begin{aligned}\nabla_{\widetilde{\mathbf{W}}}\mathcal{L}(\widetilde{\mathbf{W}}; \mathbf{X}) &= \mathbf{P}_\kappa^\top \nabla_{\mathbf{W}}\mathcal{L}(\widetilde{\mathbf{W}}; \mathbf{X})\mathbf{Q}_\kappa \\ &\underset{\text{same forward response}}{=} \mathbf{P}_\kappa^\top \nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W}; \mathbf{X})\mathbf{Q}_\kappa,\end{aligned}$$

which is equivalent to our *two-sided* gradient projection. Therefore, we can derive the preconditioned gradient descent for the rotated network using the following update:

$$\widetilde{\mathbf{W}}_{t+1} = \widetilde{\mathbf{W}}_{t+1} - \eta_t \frac{\widetilde{\mathbf{G}}_t}{\sqrt{\mathrm{EMA}_{\tau\in[t]}\left[\widetilde{\mathbf{G}}_\tau^2 + \epsilon\right]}} \qquad (8)$$

of which, we dropped the vectorization to ensure dimensional compatibility. Since $\mathbf{P}_\kappa$ and $\mathbf{Q}_\kappa$ are full-rank orthogonal matrices, the reparameterization is invertible and thus the update on the original parameter can be obtained by a projection back step as:

$$\mathbf{W}_{t+1} = \mathbf{W}_{t+1} - \eta_t \mathbf{P}_\kappa \frac{\widetilde{\mathbf{G}}_t}{\sqrt{\mathrm{EMA}_{\tau\in[t]}\left[\widetilde{\mathbf{G}}_\tau^2 + \epsilon\right]}} \mathbf{Q}_\kappa^\top, \qquad (9)$$

for $t \in [\kappa T, (\kappa+1)T]$.

3.3. Final Algorithm and Related Works

We provide the details of our proposal in Algorithm 1, which encompasses both *two-sided* and *one-sided* rotation versions. We also employ an exponential moving average of the projected gradients $\widetilde{\mathbf{G}}$ to derive the first-order moment estimation $\mathbf{M}_t$. This accumulation is performed before applying preconditioning. It should be noted that our implementation requires `torch.linalg.svd`($\mathbf{G}_t$, `full_matrices=True`) to extract the full projection matrices $\mathbf{P}_t, \mathbf{Q}_t$ in each period.

Given the flexibility of our framework, we can adapt it for other adaptive optimizers. In Appendix A, we provided variants of the algorithm tailored for memory-efficient optimizers such as Adafactor (Shazeer and Stern, 2018), Hfac (Nguyen et al., 2024), along with empirical results evaluating their performances. In connection with other existing algorithms, several prior works on optimization are relevant to our framework. George et al. (2018) and Liu et al. (2018) proposed utilizing the eigenbasis of the Fisher Information Matrix to construct diagonal preconditioning approximations within the natural gradient or online Laplace approximation families. Similarly, Anil et al. (2020) extends this idea by leveraging the eigendecomposition of Shampoo's preconditioners as a basis for the diagonal transformations.

Our method, in contrast, focuses on diagonalizing the preconditioner matrix within the generalized family of adaptive moment-based optimization algorithms, which includes Adam, Adafactor as specific cases. While primarily inspired by the critical idea of gradient projection in GaLore, we explore the full-rank projection case and thus move beyond GaLore's main focus on memory efficiency. We also acknowledge the concurrent work by SOAP (Vyas et al., 2024), which obtains the projection matrices $\mathbf{P}_t$ and $\mathbf{Q}_t$ by performing eigendecomposition on the accumulators of $\mathbf{G}_t\mathbf{G}_t^\top$ and $\mathbf{G}_t^\top\mathbf{G}_t$ (referred to as Shampoo preconditioners), respectively. Essentially, the eigenvector matrix retrieved from the eigendecomposition of $\mathbf{G}_t\mathbf{G}_t^\top$ (and $\mathbf{G}_t^\top\mathbf{G}_t$) corresponds to the left (and right) singular matrix of $\mathbf{G}_t$. Our proposal is therefore effectively equivalent to SOAP without accumulations, resulting in enhanced memory efficiency. From a practical standpoint, our algorithms substantially outperform Adam with only a manageable overhead.

3.4. Computational Overhead Analysis

For a rigorous runtime analysis, we estimate the additional computational cost introduced by AdaDiag and AdaDiag++ relative to Adam for a single linear layer of size $m \times n$, where $m \leq n$. The extra cost mainly arises from the full SVD computation (with `full_matrices=True`) and the associated projection operations.

For AdaDiag, the additional cost is mn^2/T FLOPs for computing $\mathbf{P}$, together with $4m^2n$ FLOPs for the projection and back-projection steps. In the square case, this overhead is upper bounded by approximately $5m^2n$ matrix-multiplication FLOPs. For AdaDiag++, the overhead is $(m^2n + n^3)/T$ FLOPs for computing both $\mathbf{P}$ and $\mathbf{Q}$, plus $4(m^2n + mn^2)$ FLOPs for the corresponding projection operations, which is at most about $10mn^2$ FLOPs in the square case.

For a standard linear transformation, the baseline training cost per step, including both the forward and backward passes, is approximately $6Bmn$ FLOPs, where B denotes the number of input tokens in the batch. Therefore, the relative overhead is bounded by $5m/6B$ for AdaDiag and $10n/6B$ for AdaDiag++. As a concrete example, consider the LLaMA-350M setting with layer dimensions $(m, n) = (1024, 4096)$ and batch size $B = 130{,}000$ tokens. Substituting these values yields an estimated overhead of 0.65% for AdaDiag, and 5.2% for AdaDiag++.

These estimates indicate that, under practical LLM training regimes, the additional FLOP cost introduced by AdaDiag and AdaDiag++ remains modest, staying well below 10% in this setting. Moreover, since larger-scale training typically uses even larger batch sizes, the relative overhead becomes smaller in practice.

4. CONVERGENCE ANALYSIS

In comparison to optimizers such as AdaGrad, Adam, and variants, the algorithm 1 introduces additional projection matrices $\{\mathbf{P}_t, \mathbf{Q}_t\}$, resulting in complex interactions between the model parameter $\mathbf{W}_t$ and optimization states $\mathbf{S}_t = \{\mathbf{M}_t, \mathbf{V}_t\}$. It is important to note that the second-order momentum $\mathbf{V}_t$ is the key factor in our framework. Without this quantity, the proposed algorithm would degenerate to the standard gradient descent with momentum, eliminating any potential improvements. This implies that the algorithms cannot be reduced to a simpler form for analyzing convergence guarantees. Moreover, applying SVD periodically to produce projection matrices also causes intricate behaviors in the dynamic subspace of optimization trajectory. It is unclear whether the momentum estimates accumulated across previous subspaces would be consistent with each other and advantageous for updates conducted in subsequent subspaces. As such, we need a general and robust theoretical framework to encompass a wide range of adaptive moment-based optimizers, including memory-efficient ones like Adafactor (Shazeer and Stern, 2018) and Hfac (Nguyen et al., 2024).

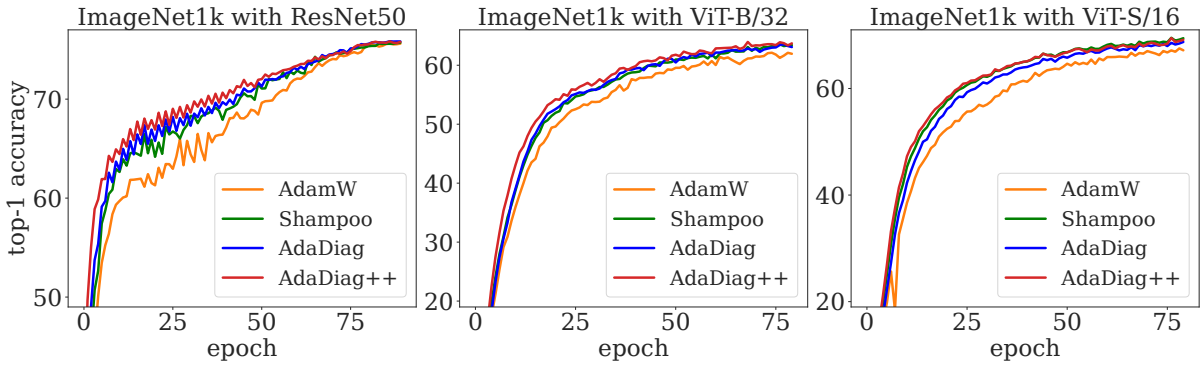

Figure 4: Top-1 Accuracy of optimizers in pretraining ResNet50, ViT-B/32, and ViT-S/16 from scratch on the ImageNet1k. For better ViT's visualization, we crop the learning curve up to epoch 80.

Inspired by the recent works on Online Subspace Descent (Liang et al., 2024) and Hfac, we leverage the Hamiltonian descent framework to tackle these challenges. Essentially, this framework investigates continuous-time ODE forms of optimizers in the limit of infinitesimal step size. In this setting, the optimizers will minimize an associated Hamiltonian function $\mathcal{H}(.)$, which is an augmented version of the original objective $\mathcal{L}(.)$. For example, we can derive a continuous-time form for Adam optimizer as:

$$\text{(Adam-ODE)}: \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t = -\mathbf{M}_t/(\sqrt{\mathbf{V}_t}+\epsilon),$$
$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t = \mathbf{G}_t - \mathbf{M}_t, \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{V}_t = \mathbf{G}_t^2 - \mathbf{V}_t,$$

which yields a Hamiltonian functions defined by: $\mathcal{H}(\mathbf{W},\mathbf{M},\mathbf{V}) = \mathcal{L}(\mathbf{W}) + \frac{1}{2}\left\langle \mathbf{M}/(\sqrt{\mathbf{V}}+\epsilon), \mathbf{M}\right\rangle$.

Proposition 1. *Using this general approach, we formulate continuous-time forms for AdaDiag and AdaDiag++:*

$$(\textit{AdaDiag}): \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t = -\mathbf{P}_t\frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon},$$
$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t = \mathbf{P}_t^\top\mathbf{G}_t - \mathbf{M}_t, \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{V}_t = (\mathbf{P}_t^\top\mathbf{G}_t)^2 - \mathbf{V}_t$$
$$(\textit{AdaDiag++}): \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t = -\mathbf{P}_t\frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon}\mathbf{Q}_t^\top,$$
$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t = \mathbf{P}_t^\top\mathbf{G}_t\mathbf{Q}_t - \mathbf{M}_t, \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{V}_t = (\mathbf{P}_t^\top\mathbf{G}_t\mathbf{Q}_t)^2 - \mathbf{V}_t$$

Both yield the same Hamiltonian function: $\mathcal{H}(\mathbf{W},\mathbf{M},\mathbf{V}) = \mathcal{L}(\mathbf{W}) + \frac{1}{2}\left\langle \mathbf{M}/(\sqrt{\mathbf{V}}+\epsilon), \mathbf{M}\right\rangle$.

Convergence to Local Optima. The key properties is that the function $\mathcal{H}(.)$ is monotonically non-decreasing along its ODE trajectory, namely $\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}(\mathbf{W}_t,\mathbf{S}_t) \leq 0, \forall t$. By LaSalle's Invariance principle, the set of accumulation points $(\mathbf{W}_t,\mathbf{S}_t)$ must be contained in $\mathcal{I}$, where $\mathcal{I}$ = {the union of complete trajectories satisfying $\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}(\mathbf{W}_t,\mathbf{S}_t) = 0$}. The points in limit set $\mathcal{I}$ should satisfy $\mathbf{P}_t^\top\nabla\mathcal{L}(\mathbf{W}_t) \equiv 0$ for AdaDiag or $\mathbf{P}_t^\top\nabla\mathcal{L}(\mathbf{W}_t)\mathbf{Q}_t \equiv 0$ for AdaDiag++, respectively. Since $\mathbf{P}_t, \mathbf{Q}_t$ are full-rank orthogonal matrices, we must have $\nabla\mathcal{L}(\mathbf{W}_t) \equiv 0$, which indicates that all trajectories will converge to local optimal points. Detailed analysis is provided in Appendix E.

Table 1: Comparison of Adam and AdaDiag on pretraining ResNets and ViTs architectures with ImageNet1k dataset. Top-1 accuracy on the validation set is reported.

Models	ResNet50	ViT-S/16	ViT-B/32
AdamW	75.61	78.35	72.20
Shampoo	75.71	**79.58**	**73.47**
AdaDiag	75.85	**79.18**	**73.39**
AdaDiag++	75.86	**79.14**	**73.24**

5. EXPERIMENTS

In this section, we conduct several experiments on image classification and language modeling tasks to verify the efficiency of our algorithms. We further demonstrate in Appendix A that our general framework can be effectively applied to enhance memory-efficient optimizers such as Adafactor and Hfac.

5.1. Image Classification

We first evaluated the optimization algorithms, including AdamW, AdaDiag, and AdaDiag++, by pretraining the ImageNet1k dataset from scratch using ResNets and Vision Transformers (ViTs) architectures. The images underwent Inception-style cropping (Szegedy et al., 2016) and random horizontal flipping during pre-

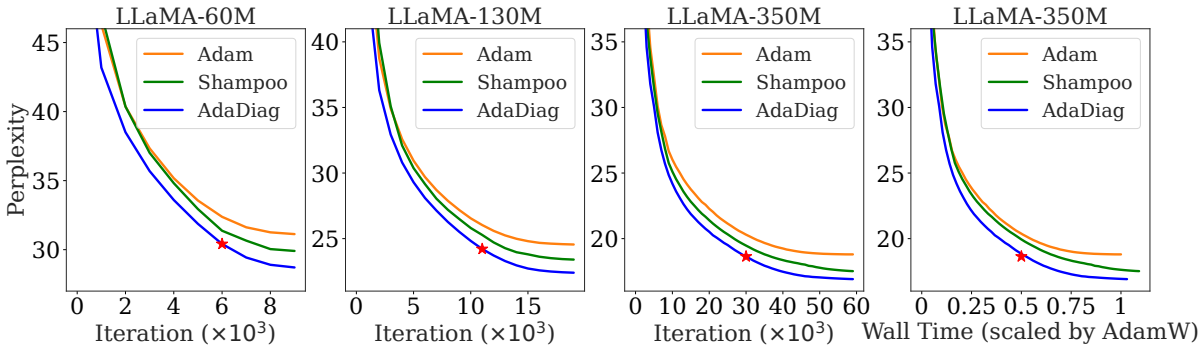

Figure 5: Training progression for pre-training LLaMA models on C4 dataset.

processing. We trained ResNet50 for 90 epochs with a batch size of 1024, utilizing a cosine learning rate decay scheduler. For ViTs, we conducted training over 300 epochs with a batch size of 4096, using a learning rate schedule that included a 10,000-step warmup followed by linear decay. Additionally, we employed strong data augmentations, such as RandAugment (2,15) (Cubuk et al., 2019) and mixup (0.5) (Zhang et al., 2017), to further improve the performance of the ViTs. Details on hyperparameter tuning are presented in Appendix D.

As shown in Fig. 4, AdaDiag and AdaDiag++ exhibit substantial improvements in convergence speed compared to the baseline AdamW. For the ViT-B/32 and ViT-S/16 models, we focus on the first third of the training phase so that we can observe notable accelerations across the three models. For the full performances, we refer to the results in Fig. 11 in Appendix B. The final results at convergence are provided in Tab. 1. By more accurate approximations of the preconditioner matrix, we expect that AdaDiag++ can navigate the complex curvature more efficiently and thus provide better convergence properties, even compared to AdaDiag. The results appear to support this argument. It's important to mention that these two algorithms would perform similarly after converging at some point. We hypothesize that the optimization trajectory will eventually converge to a stable region where the gradients reside in a very low-dimensional subspace. The precondition approximations, at this stage, become less critical as the optimization process focuses on fine-tuning within this reduced space.

5.2. Language Modeling

We apply the algorithms to pre-train LLaMA-based large language model, with RMSNorm and SwiGLU activations (Zhang and Sennrich, 2019; Shazeer, 2020; Touvron et al., 2023), on the C4 dataset (Raffel et al., 2020). We measured the perplexity of the models on the validation set throughout training to assess convergence properties and final model performance. Specifically, we trained LLaMA models of sizes 60M, 130M, 350M, and 1B for 10K, 20K, 60K, 100K steps, respectively. The learning rate schedule included a warmup phase during the first 10% of the training steps, followed by cosine annealing that decayed the learning rate to 10% of its initial value. All models used a maximum sequence length of 256 and a batch size of 512.

Table 2: Comparison of Adam and AdaDiag on pre-training LLaMA models with the C4 dataset. Validation perplexity is reported for models with 60M/130M/350M/1B parameters trained for 1.1B/2.6B/7.8B/13.1B training tokens.

Optimizer	Validation Perplexity			
	60M	130M	350M	1B
Adam	31.12	24.55	18.79	15.92
Shampoo	29.91	23.40	17.52	15.36
AdaDiag	**28.71**	**22.40**	**16.91**	**14.11**

The results are provided in Fig. 5 and Tab. 2. Our optimizer AdaDiag consistently outperforms Adam on various sizes of LLaMA models. In particular, AdaDiag achieves 1.8x-2x speed-up compared to Adam, matching the same perplexity with half fewer steps. Due to resource constraints, we were unable to conduct experiments on more billion-parameter models. However, we are confident that similar results can be reliably achieved with larger-scale models.

5.3. Ablation Study

We present an ablation study in Fig. 9 (Appendix B) to demonstrate the role of full-rank SVD. Essentially, we can incrementally raise the rank of the truncated SVD employed in GaLore until we achieve the effective rank. While this can in principle improve the performance of the algorithm, it requires additional tuning effort as discussed in Section 3.1.

For update frequency T, we illustrate its impact on our algorithm in Fig. 10 (Appendix B). We found that

the values $T = 200, 500$ consistently yielded the best results across all our experiments. These period lengths are also practical in terms of computational cost, as they make the overhead induced by SVD negligible in our algorithm, as evidenced by the wall clock time measurements displayed in Fig. 5. In contrast, more frequent updates ($T = 2, 10$) degrade the performance significantly. This might stem from the algorithm having to excessively adapt to the instability introduced by minibatch training. A reasonable frequency should strike a balance between memorization and adaptation throughout the training process. Notably, in cases where the projection matrix was fixed ($T = 10k, 20k$), we also observed improved results over Adam on language tasks. This finding suggests potential for future works to exploit the optimal subspace for applying moment estimates.

6. DISCUSSION

In this work, we proposed an efficient approach to improve adaptive moment-based optimizers by introducing a preconditioner diagonalization strategy. By leveraging an invertible transformation, we were able to enhance the reliability of the diagonal approximation used for the preconditioner matrix, resulting in a more effective estimation of second-order statistics. Our empirical evaluations demonstrated significant improvements in both convergence speed and final model performance across several standard tasks. Furthermore, our work also underscores the significance of employing structural preconditioning techniques to improve existing adaptive learning rate optimizers. Devising new preconditioners or exploring network reparameterization present promising approaches to this problem. In addition, it is vital to establish theoretically sound frameworks to better understand adaptive moment-based optimizers in dynamic subspaces. This would enable us to identify the optimal subspace where the moment estimates can be applied most effectively.

Acknowledgements

The research is conducted in Statistics & AI group at UT Austin, which receives supports in part from NSF CAREER1846421, Office of Navy Research, and NSF AI Institute for Foundations of Machine Learning (IFML).

References

Allen-Zhu, Z., Li, Y., and Song, Z. (2019). A convergence theory for deep learning via over-parameterization. In *International conference on machine learning*, pages 242–252. PMLR.

Amari, S.-i., Karakida, R., and Oizumi, M. (2019). Fisher information and natural gradient learning in random deep networks. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 694–702. PMLR.

Anil, R., Gupta, V., Koren, T., Regan, K., and Singer, Y. (2020). Scalable second order optimization for deep learning. *arXiv preprint arXiv:2002.09018*.

Bonnans, J.-F., Gilbert, J. C., Lemaréchal, C., and Sagastizábal, C. A. (2006). *Numerical optimization: theoretical and practical aspects*. Springer Science & Business Media.

Botev, A., Ritter, H., and Barber, D. (2017). Practical gauss-newton optimisation for deep learning. In *International Conference on Machine Learning*, pages 557–565. PMLR.

Cao, Y., Li, X., and Song, Z. (2024). Grams: Gradient descent with adaptive momentum scaling. *arXiv preprint arXiv:2412.17107*.

Chen, L., Liu, B., Liang, K., and Liu, Q. (2023). Lion secretly solves constrained optimization: As lyapunov predicts. *arXiv preprint arXiv:2310.05898*.

Chen, X., Liang, C., Huang, D., Real, E., Wang, K., Pham, H., Dong, X., Luong, T., Hsieh, C.-J., Lu, Y., et al. (2024). Symbolic discovery of optimization algorithms. *Advances in Neural Information Processing Systems*, 36.

Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. (2019). Randaugment: Practical automated data augmentation with a reduced search space. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3008–3017.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

Fletcher, R. (2000). *Practical methods of optimization*. John Wiley & Sons.

Gao, X., Gürbüzbalaban, M., and Zhu, L. (2022). Global convergence of stochastic gradient hamiltonian monte carlo for nonconvex stochastic optimization: Nonasymptotic performance bounds and momentum-based acceleration. *Operations Research*, 70(5):2931–2947.

George, T., Laurent, C., Bouthillier, X., Ballas, N., and Vincent, P. (2018). Fast approximate natural gradient descent in a kronecker factored eigenbasis.

Advances in Neural Information Processing Systems, 31.

Gooneratne, M., Sim, K. C., Zadrazil, P., Kabel, A., Beaufays, F., and Motta, G. (2020). Low-rank gradient approximation for memory-efficient on-device training of deep neural network. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3017–3021. IEEE.

Grosse, R. and Martens, J. (2016). A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582. PMLR.

Gupta, V., Koren, T., and Singer, Y. (2018). Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pages 1842–1850. PMLR.

Gur-Ari, G., Roberts, D. A., and Dyer, E. (2018). Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*.

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

Konečnỳ, J., McMahan, H. B., Ramage, D., and Richtárik, P. (2016). Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*.

Kunstner, F., Yadav, R., Milligan, A., Schmidt, M., and Bietti, A. (2024). Heavy-tailed class imbalance and why adam outperforms gradient descent on language models. *arXiv preprint arXiv:2402.19449*.

Li, T., Sahu, A. K., Talwalkar, A., and Smith, V. (2020). Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 37(3):50–60.

Liang, K., Liu, B., Chen, L., and Liu, Q. (2024). Memory-efficient llm training with online subspace descent. *arXiv preprint arXiv:2408.12857*.

Liu, H., Li, Z., Hall, D., Liang, P., and Ma, T. (2023). Sophia: A scalable stochastic second-order optimizer for language model pre-training. *arXiv preprint arXiv:2305.14342*.

Liu, X., Masana, M., Herranz, L., Van de Weijer, J., Lopez, A. M., and Bagdanov, A. D. (2018). Rotate your networks: Better weight consolidation and less catastrophic forgetting. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2262–2268. IEEE.

Maddison, C. J., Paulin, D., Teh, Y. W., O'Donoghue, B., and Doucet, A. (2018). Hamiltonian descent methods. *arXiv preprint arXiv:1809.05042*.

Martens, J. (2020). New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76.

Martens, J. and Grosse, R. (2015). Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR.

Nguyen, S., Chen, L., Liu, B., and Liu, Q. (2024). H-fac: Memory-efficient optimization with factorized hamiltonian descent. *arXiv preprint arXiv:2406.09958*.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Reddi, S. J., Kale, S., and Kumar, S. (2019). On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*.

Shazeer, N. (2020). Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Shazeer, N. and Stern, M. (2018). Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR.

Şimşekli, U., Gürbüzbalaban, M., Nguyen, T. H., Richard, G., and Sagun, L. (2019). On the heavy-tailed theory of stochastic gradient descent for deep neural networks. *arXiv preprint arXiv:1912.00018*.

Smith, S. L., Dherin, B., Barrett, D. G., and De, S. (2021). On the origin of implicit regularization in stochastic gradient descent. *arXiv preprint arXiv:2101.12176*.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

Tian, Y., Zhang, Y., and Zhang, H. (2023). Recent advances in stochastic gradient descent in deep learning. *Mathematics*, 11(3):682.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Vyas, N., Morwani, D., Zhao, R., Shapira, I., Brandfonbrener, D., Janson, L., and Kakade, S. (2024). Soap: Improving and stabilizing shampoo using adam. *arXiv preprint arXiv:2409.11321*.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018). Glue: A multi-task bench-

mark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Zhang, B. and Sennrich, R. (2019). Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32.

Zhang, H., Cissé, M., Dauphin, Y., and Lopez-Paz, D. (2017). mixup: Beyond empirical risk minimization. *ArXiv*, abs/1710.09412.

Zhang, J., Karimireddy, S. P., Veit, A., Kim, S., Reddi, S., Kumar, S., and Sra, S. (2020). Why are adaptive methods good for attention models? *Advances in Neural Information Processing Systems*, 33:15383–15393.

Zhang, Y., Chen, C., Ding, T., Li, Z., Sun, R., and Luo, Z.-Q. (2024). Why transformers need adam: A hessian perspective. *arXiv preprint arXiv:2402.16788*.

Zhang, Y., Chen, C., Shi, N., Sun, R., and Luo, Z.-Q. (2022). Adam can converge without any modification on update rules. *Advances in neural information processing systems*, 35:28386–28399.

Zhao, J., Zhang, Z., Chen, B., Wang, Z., Anandkumar, A., and Tian, Y. (2024). Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*.

Zhou, P., Feng, J., Ma, C., Xiong, C., Hoi, S. C. H., et al. (2020). Towards theoretically understanding why sgd generalizes better than adam in deep learning. *Advances in Neural Information Processing Systems*, 33:21285–21296.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Improving Adaptive Moment Optimization via Preconditioner Diagonalization: Supplementary Materials

A. Improving computational and memory efficiency

Although our proposal can greatly enhance the performance, it comes with an inevitable trade-off in algorithmic complexity. While the total computational overhead induced by periodic SVD is negligible (less than 10%), the memory usage caused by the full-rank projection may limit the applicability of the algorithms. To address this challenge, we propose to apply the general framework to memory-efficiency optimization methods such as Adafactor (Shazeer and Stern, 2018), Hfac (Nguyen et al., 2024). Adafactor and Hfac have demonstrated results comparable to Adam on various tasks (Shazeer and Stern, 2018; Chen et al., 2024; Nguyen et al., 2024). Therefore, a simple integration with our method could potentially surpass Adam's performance while maintaining a similar complexity.

A.1. Adafactor optimizer

Algorithm 2 AdafacDiag for matrix parameter $\mathbf{W}$ of size $m \times n$, $m \leq n$. When omitting the SVD step and setting the projection matrix $\mathbf{P}$ to the identity, we recover the Adafactor optimizer.

Inputs: moment decay coefficients β_1, β_2, smoothing term $\epsilon = 10^{-30}$, regularization constant λ
Initialization: weight parameters $\mathbf{W}_1 \in \mathbb{R}^{m\times n}$, initial moments $\mathbf{M}_0, \mathbf{r}_0, \mathbf{s}_0 \leftarrow 0$
repeat
  $t \leftarrow t+1$
  $\mathbf{G}_t = \nabla\mathcal{L}(\mathbf{W}_t; \mathcal{B}_t)$
  if $t \bmod T = 0$ **then**
    $\mathbf{P}_t,\ \_,\ \mathbf{Q}_t^\top$ = `torch.linalg.svd`($\mathbf{G}_t$, `full_matrices=True`)
  else
    $\mathbf{P}_t, \mathbf{Q}_t^\top \leftarrow \mathbf{P}_{t-1}, \mathbf{Q}_{t-1}^\top$
  end if
  $\widetilde{\mathbf{G}}_t = \mathbf{P}_t^\top \mathbf{G}_t$
  $\mathbf{r}_t = \hat{\beta}_{2t}\mathbf{r}_{t-1} + (1-\hat{\beta}_{2t})\left[(\widetilde{\mathbf{G}}_t)^2 + \epsilon\right]\mathbf{1}_n$
  $\mathbf{s}_t = \hat{\beta}_{2t}\mathbf{s}_{t-1} + (1-\hat{\beta}_{2t})\left[(\widetilde{\mathbf{G}}_t^\top)^2 + \epsilon\right]\mathbf{1}_m$
  $\mathbf{V}_t = \mathbf{r}_t\mathbf{s}_t^\top / (\mathbf{1}_m^\top \mathbf{r}_t)$
  $\mathbf{M}_t = \hat{\beta}_{1t}\mathbf{M}_{t-1} + (1-\hat{\beta}_{1t})\text{clip}\left(\widetilde{\mathbf{G}}_t / \left(\sqrt{\mathbf{V}_t}\right)\right)$
  $\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_t\left(\mathbf{P}_t\mathbf{M}_t + \lambda\mathbf{W}_t\right)$
until *stopping criterion is met*
return optimized parameter $\mathbf{W}_t$

Adafactor (Shazeer and Stern, 2018) proposed an efficient rank-1 parameterization for the second moment $\mathbf{V}$, which is widely adopted in adaptive optimization methods like RMSprop, Adam, and its variants. The factorization was derived by minimizing the total elementwise I-divergence subject to componentwise non-negative constraints:

$$\underset{\mathbf{r}\in\mathbb{R}^m, \mathbf{s}\in\mathbb{R}^n}{\text{minimize}} \sum_{i=1}^{m}\sum_{j=1}^{n} d(V_{ij}, r_i s_j)$$

in which $r_i \geq 0, s_j \geq 0$ and $d(p,q) = p\log\frac{p}{q} - p + q$.

Solving this problem results in a closed-form solution denoted by $\mathbf{r} = \mathbf{V}\mathbf{1}_n, \mathbf{s} = \mathbf{V}^\top\mathbf{1}_m / \mathbf{r}^\top\mathbf{1}_n$. Intuitively, Adafactor tracks the moving averages of the row and column sums of squared gradients throughout iterations, yielding factored second-moment estimators $\mathbf{r}_t$ and $\mathbf{s}_t$. It then reconstructs a low-rank parameterization of the

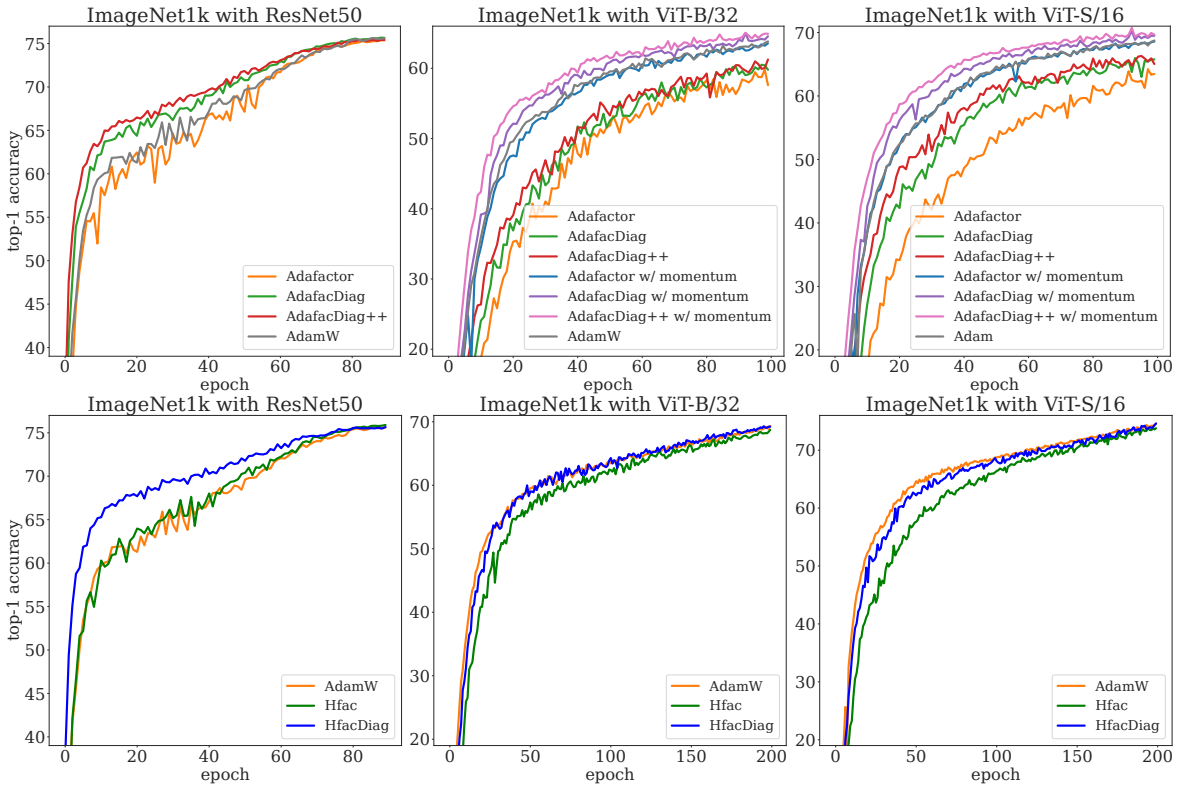

Figure 6: Top-1 Accuracy of memory-efficient optimizers in pre-training ResNet50, ViT-B/32, and ViT-S/16 from scratch on the ImageNet1k.

Table 3: Comparison of Adam, Adafactor & AdafacDiag, Hfac & HfacDiag on pre-training ResNets and ViTs architectures with ImageNet1k dataset. Top-1 accuracy on the validation set is reported. The bold values indicate noticeble improvements over the corresponding Adam results.

Models	ResNet50	ViT-S/16	ViT-B/32
AdamW	75.61	78.35	72.20
Adafactor	75.37	77.14	71.42
AdafacDiag	75.68	78.45	72.54
AdafacDiag++	75.60	78.56	72.78
Adafactor w/ momentum	-	78.44	72.31
AdafacDiag w/ momentum	-	**78.90**	**73.24**
AdafacDiag++ w/ momentum	-	**78.66**	**73.28**
Hfac	75.90	77.20	71.87
HfacDiag	75.78	78.20	72.76

second-order momentum using a normalized outer product $\mathbf{r}_t\mathbf{s}_t^\top/(\mathbf{1}_m^\top\mathbf{r}_t)$. This method is computationally efficient and scalable, as it directly offers analytical formulations without requiring further approximations.

Incorporating Adafactor into our framework offers significant computational and memory efficiency benefits. This can be evident in Tab. 4, from which AdafacDiag demonstrates lower complexity in optimization states when compared to Adam. To further assess the effectiveness of this integration, we carried out experiments similar to those described in the main text. As shown in Fig. 6 (top row) and Tab. 3, AdafacDiag (on ResNet50) and AdafacDiag with momentum (on ViTs) can outperform Adam by noticeable margins. The experiments on LLaMA-based models using the C4 dataset also delivered consistent results, presented in Fig. 7 and Tab. 5. These advantages highlight the potential of utilizing these algorithms in a wide range of real-world tasks, particularly in large-scale applications.

Table 4: Memory requirements for different optimizers, with weight parameter of size $m \times n, m \leq n$. To estimate practical memory costs, we calculate the memory usage of optimization states, specifically the moment estimates, for each optimizer applied to various LLaMA-based models, using the BF16 format.

Optimizers	Weights	Gradient	Optim. States	60M	130M	350M	1.3B
Adam	mn	mn	$2mn$	0.23G	0.52G	1.44G	5.23G
AdaDiag	mn	mn	$m^2 + 2mn$	0.26G	0.62G	1.78G	6.61G
AdaDiag++	mn	mn	$m^2 + n^2 + 2mn$	0.36G	0.97G	3.03G	11.59G
Adafactor w/ momentum	mn	mn	$mn + m + n$	0.18G	0.36G	0.85G	2.87G
Adafactor w/o momentum	mn	mn	$m + n$	0.06G	0.10G	0.13G	0.26G
AdafacDiag w/ momentum	mn	mn	$m^2 + mn + m + n$	0.21G	0.45G	1.19G	4.25G
AdafacDiag w/o momentum	mn	mn	$m^2 + m + n$	0.09G	0.19G	0.47G	1.63G
Hfac	mn	mn	$2(m + n)$	0.13G	0.19G	0.26G	0.52G
HfacDiag	mn	mn	$m^2 + 2(m + n)$	**0.16G**	**0.29G**	**0.60G**	**1.89G**
GaLore $rank - r = m/4$	mn	mn	$mr + 2nr$	0.13G	0.28G	0.54G	1.78G

A.2. Hfac optimizer

Algorithm 3 HfacDiag for matrix parameters $\mathbf{W}$ of size $m \times n$, $m \leq n$. When omitting the SVD step and setting the projection matrix $\mathbf{P}$ to the identity, we recover the Hfac optimizer.

Inputs: moment decay coefficients β_1, β_2, smoothing term ϵ, and regularization constant λ
Initialization: weight parameters $\mathbf{W}_1 \in \mathbb{R}^{m \times n}$, initial factored moments $\mathbf{u}_0, \mathbf{v}_0, \mathbf{r}_0, \mathbf{s}_0 \leftarrow 0$
repeat
  $\mathbf{G}_t = \nabla \mathcal{L}(\mathbf{W}_t; \mathcal{B}_t)$
  if t mod $T = 0$ **then**
    $\mathbf{P}_t$, _, $\mathbf{Q}_t^\top$ = `torch.linalg.svd`($\mathbf{G}_t$, `full_matrices=True`)
  else
    $\mathbf{P}_t, \mathbf{Q}_t^\top \leftarrow \mathbf{P}_{t-1}, \mathbf{Q}_{t-1}^\top$
  end if
  $\widetilde{\mathbf{G}}_t = \mathbf{P}_t^\top \mathbf{G}_t$
  $\mathbf{u}_t = \hat{\beta}_{1t}\mathbf{u}_{t-1} + (1 - \hat{\beta}_{1t})\widetilde{\mathbf{G}}_t \mathbf{1}_n / n$
  $\mathbf{v}_t = \hat{\beta}_{1t}\mathbf{v}_{t-1} + (1 - \hat{\beta}_{1t})\widetilde{\mathbf{G}}_t^\top \mathbf{1}_m / m$
  $\mathbf{r}_t = \hat{\beta}_{2t}\mathbf{r}_{t-1} + (1 - \hat{\beta}_{2t})\big[(\widetilde{\mathbf{G}}_t)^2 + \epsilon\big]\mathbf{1}_n$
  $\mathbf{s}_t = \hat{\beta}_{2t}\mathbf{s}_{t-1} + (1 - \hat{\beta}_{2t})\big[(\widetilde{\mathbf{G}}_t^\top)^2 + \epsilon\big]\mathbf{1}_m$
  $\widehat{\mathbf{V}}_t = \mathbf{r}_t \mathbf{s}_t^\top / (\mathbf{1}_m^\top \mathbf{r}_t)$
  $\phi_{term} = \hat{\beta}_{1t}\big(\mathbf{u}_t \mathbf{1}_n^\top - \widetilde{\mathbf{G}}_t \mathbf{1}_n \mathbf{1}_n^\top / n\big) / \sqrt{\mathbf{r}_t \mathbf{1}_n^\top / n}$
  $\psi_{term} = \hat{\beta}_{1t}\big(\mathbf{1}_m \mathbf{v}_t^\top - \mathbf{1}_m \mathbf{1}_m^\top \widetilde{\mathbf{G}}_t / m\big) / \sqrt{\mathbf{1}_m \mathbf{s}_t^\top / m}$
  $\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_t \left[\mathbf{P}_t \left(0.5(\phi_{term} + \psi_{term}) + \text{clip}\Big(\widetilde{\mathbf{G}}_t / \sqrt{\widehat{\mathbf{V}}_t}\Big)\right) + \lambda \mathbf{W}_t\right]$
until *stopping criterion is met*

Hfac (Nguyen et al., 2024) advances the memory-efficient optimizers by further decomposing the first moment into a rank-1 parameterization. This procedure involves projecting the full gradient onto rank-one spaces defined by column means and row means, then exponentially accumulating these statistics throughout the training process. Hfac can reduce the memory cost to a sublinear level, comparable to that of vanilla SGD without momentum, and still delivers favorable and competitive results across various architectures and datasets.

Specific updates are outlined in Algorithm 3. Intuitively, Adafactor can utilize a full moment $\mathbf{M}$, whereas Hfac decomposes $\mathbf{M}$ into vectors $\mathbf{u}$ and $\mathbf{v}$, leading to different normalizing mechanisms. While Adafactor normalizes the first-moment $\mathbf{M}$ using the second-moment approximation $\mathbf{r}\mathbf{s}^\top / \mathbf{1}_m^\top \mathbf{r}$, Hfac scales the factored components of the first moment, $\mathbf{u}\mathbf{1}_n^\top$ and $\mathbf{1}_m \mathbf{v}^\top$, by their respective factored second-moment estimators, namely $\mathbf{r}_t \mathbf{1}_n^\top / n$ and $1_m \mathbf{s}_t^\top / m$.

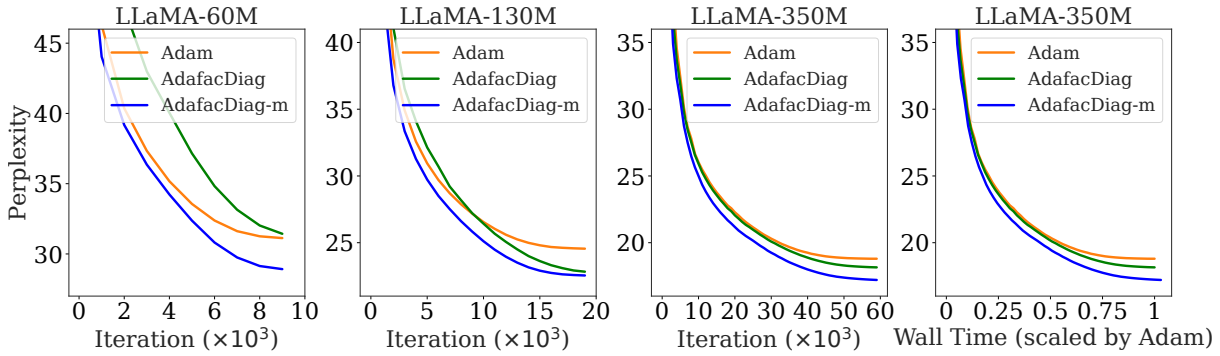

Figure 7: Training progression of Adam and AdafacDiag for pre-training LLaMA models on C4 dataset.

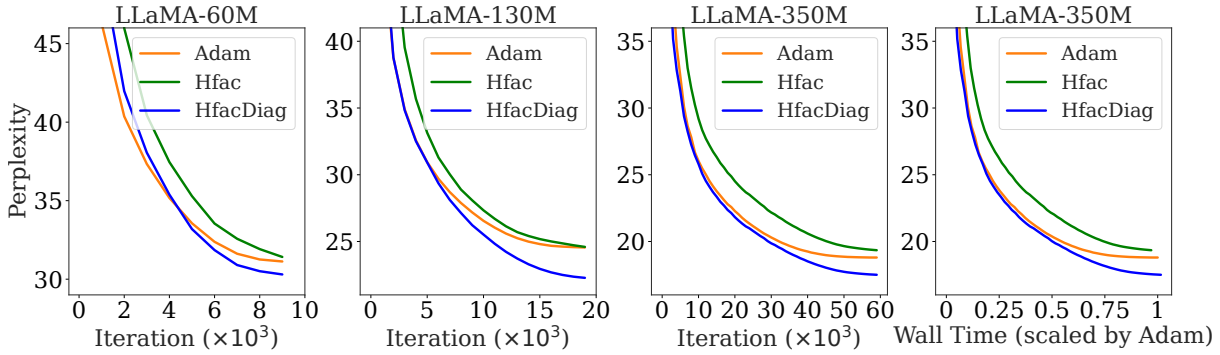

Figure 8: Training progression of Adam, Hfac, and HfacDiag for pre-training LLaMA models on C4 dataset.

Table 5: Comparison of Adam and AdafacDiag on pre-training LLaMA models with the C4 dataset. Validation perplexity is reported for models with 60M/130M/350M/1B parameters trained for 1.1B/2.6B/7.8B/13.1B training tokens. The bold values indicate improvements over the corresponding Adam results.

Optimizer	**Validation Perplexity**			
	60M / 1.1B	130M / 2.6B	350M / 7.8B	1B / 13.1B
Adam (Kingma and Ba, 2014)	31.12	24.55	18.79	15.92
AdafacDiag (ours)	31.43	**22.82**	**18.15**	**14.80**
AdafacDiag w/ momentum (ours)	**28.91**	**22.54**	**17.21**	**14.32**
Hfac (Nguyen et al., 2024)	31.41	24.59	19.34	16.39
HfacDiag (ours)	**30.30**	**22.27**	**17.50**	**15.12**

Although Hfac employs the same derivation for second-moment factorization as in Adafactor, their parameter update schemes are fundamentally different. Adafactor, similar to Adam, updates parameters using the signal-to-noise ratio $\mathbf{M}/\sqrt{\mathbf{V}}$. On the other hand, Hfac adopts a momentum RMSprop-like approach, where parameter updates act as accumulators of normalized gradients. Both the momentum and the current gradient in Hfac are rescaled by their corresponding cumulative second-moment information.

Building on the competitive results of Hfac reported in Nguyen et al. (2024), we can expect that a naive combination with our preconditioning technique can further boost its performance. We refer to this integrated method as HfacDiag. To validate this hypothesis, we have evaluated HfacDiag and presented the results in Fig. 6 (bottom row) and Fig. 8. On the image classification task, HfacDiag performs on par with Adam and even significantly surpasses Adam on ResNet50. For the language modeling task, HfacDiag consistently outperforms Adam across all tested models, showcasing its robustness and reliability. Notably, HfacDiag maintains a reasonable computational cost and memory footprint, as demonstrated in Tab. 4 and Fig. 8. Specifically with memory usage, HfacDiag achieves the same efficiency as the low-rank GaLore while delivering substantial improvements in practical performance. This highlights HfacDiag as a compelling optimizer for large-scale training workloads.

B. Additional Figures

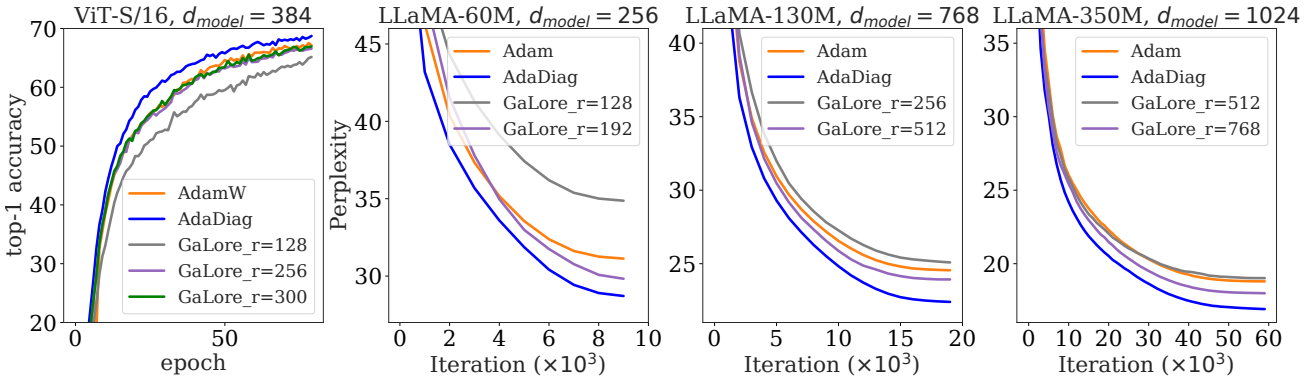

Figure 9: An ablation study on why the full-rank SVD matters.

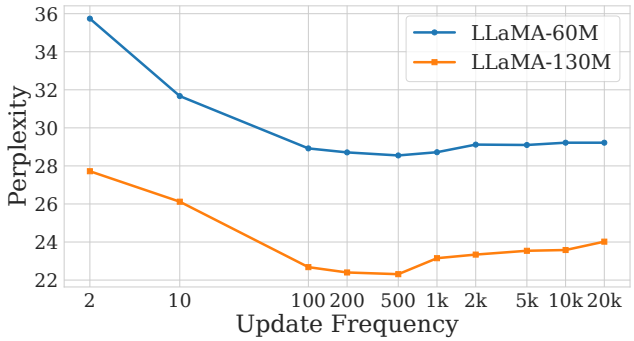

Figure 10: Performances of AdaDiag on LLaMA-60M and LLaMA-130M with varying update frequencies T.

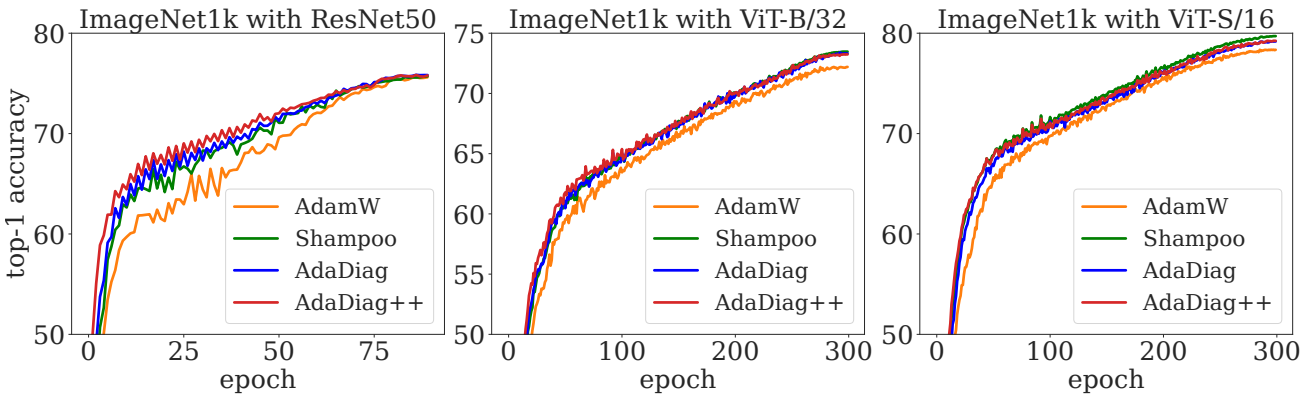

Figure 11: Top-1 Accuracy of optimizers in pre-training (to the end) ResNet50, ViT-B/32, and ViT-S/16 from scratch on the ImageNet1k.

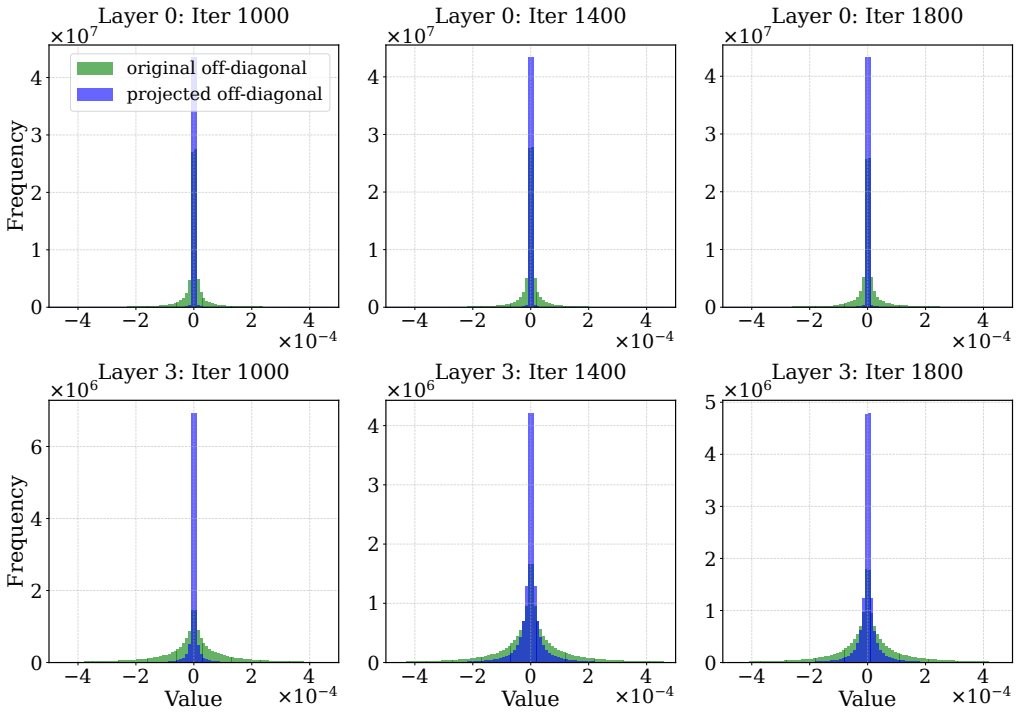

Figure 12: Histograms of off-diagonal elements $\mathcal{C}(\mathbf{G}_\tau)$ (original) and $\mathcal{C}(\widetilde{\mathbf{G}}_\tau)$ (*one-sided* projection, AdaDiag), corresponding to the two first layers of ResNet50 trained on ImageNet1k. In this experiment, we set the frequency $T = 500$ and plot histograms at iterations with and without SVD applied.

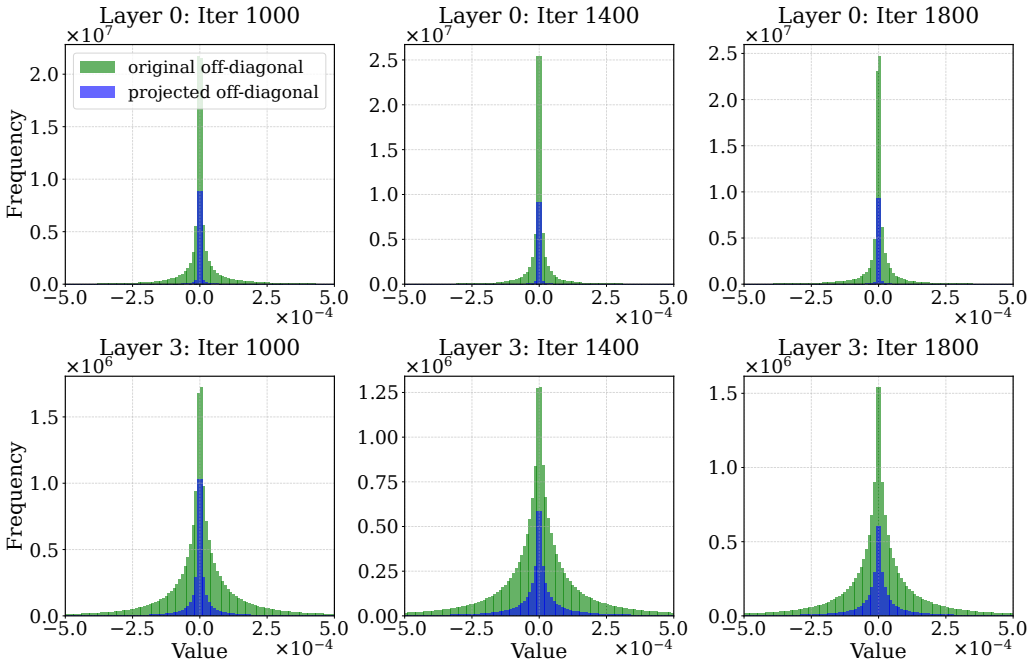

Figure 13: Histograms of off-diagonal elements $\mathcal{C}(\mathbf{G}_\tau)$ (original) and $\mathcal{C}(\widetilde{\mathbf{G}}_\tau)$ (**GaLore**), corresponding to the two first layers of ResNet50 trained on ImageNet1k. Compared to the full-rank case, the low-rank GaLore projection ($r = \min\{m, n\}/2$) does not exhibit a discernible sparsity structure in the off-diagonal elements. GaLore even creates off-diagonal elements with larger magnitudes. Note it was truncated for better visualization.

Table 6: Comparison of optimizers on the GLUE benchmark using pre-trained RoBERTa-Base. We report the average score across all tasks. Higher is better.

	CoLA	STS-B	MRPC	RTE	SST-2	MNLI	QNLI	QQP	AVG (↑)
Adam	60.82	90.94	92.49	78.33	93.57	87.39	92.60	**92.04**	86.02
AdaDiag (ours)	**63.39**	**91.70**	91.45	**80.09**	93.80	**88.22**	**93.44**	91.85	**86.74**
AdafacDiag (ours)	62.55	90.80	92.87	79.00	**94.26**	87.00	92.75	91.83	86.38
HfacDiag (ours)	62.59	90.25	**93.30**	76.17	94.03	87.20	92.45	91.79	85.97

C. Fine-Tuning Experiments

To further evaluate the effectiveness of the proposed optimizers, we perform downstream assessments on the GLUE benchmark (Wang et al., 2018), a standard suite for testing pretrained language models. GLUE includes nine natural language understanding tasks, covering sentence similarity, text classification, entailment, and question answering, along with a diagnostic set for fine-grained linguistic analysis. We focus on eight tasks, including CoLA, STS-B, MRPC, RTE, SST-2, MNLI, QNLI, and QQP, which do not require additional task-specific fine-tuning for evaluation. We report the downstream results of models pretrained with Adam, AdaDiag, AdafacDiag, and HfacDiag on these tasks, alongside their average scores as overall performance measures.

As shown in Tab. 6, models trained with AdaDiag and AdafacDiag achieve stronger downstream performance compared to Adam, with AdaDiag providing the best overall average. HfacDiag delivers comparable results to Adam while requiring substantially less memory, underscoring its efficiency advantage.

D. Hyperparameter Settings

We conducted experiments in distributed setup on 8 V100 GPUs, using Accelerate library from Hugging Face.

For all optimizers, we used decay moment coefficients $(\beta_1, \beta_2) = (0.9, 0.999)$ along with the bias-correction steps, the smoothing constant $\epsilon = 10^{-8}$. For all experiments, we adopt learning rate warmup for the first 10% of the training steps, and apply cosine annealing for the learning rate schedule, decaying to 10% of the initial learning rate. Specifically with AdaDiag, AdaDiag++, we use the SVD frequency $T = 500, 200$ for image and language tasks, respectively.

For the image classification task, we opted for recommended configurations from prior works, using $(lr, \lambda) = (0.001, 0.1)$ and $(0.0003, 0.1)$ for all ResNets and ViTs experiments, respectively.

For LLaMA pre-training, we tuned the learning rate over the set {0.003, 0.001, 0.0003, 0.0001} and selected the optimal value based on validation perplexity. We use a max sequence length of 256 for all models, with a batch size of 131K tokens. The specific settings are summarized in the Tab. 7.

Table 7: Training configuration for LLaMA models.

Models	# tokens	# training steps	# warmup steps	lr
60M	1.3B	10K	1K	0.003
130M	2.6B	20K	2K	0.001
350M	7.8B	60K	6K	0.001
1B	13.1B	100K	10K	0.001

For the fine-tuning task, we use the pre-trained RoBERTa-Base model on the GLUE benchmark with the Hugging Face implementation[2]. The model was trained with a maximum sequence length of 512 for 30 epochs, using a batch size of 16 for all tasks except CoLA, where the batch size was set to 32. We searched the learning rate over {1e-5, 2e-5, 3e-5} and chose the optimal value according to validation metrics.

[2] `https://huggingface.co/transformers/model_doc/roberta.html`

E. Hamiltonian function analyses

We first formalize a unified view of adaptive moment-based optimizers in the update scheme as:

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \phi_t(\mathbf{S}_t) \qquad \mathbf{S}_t = \psi_t(\mathbf{S}_{t-1}, \nabla\mathcal{L}(\mathbf{W}_t)) \tag{10}$$

where $\mathbf{S}_t$ is the optimization state, and ϕ_t, ψ_t are some mapping functions. One powerful approach to studying the dynamic behavior of these optimizers is to examine their continuous-time ODE forms in the limit of infinitesimal step size (Maddison et al., 2018; Gao et al., 2022; Chen et al., 2023; Cao et al., 2024). It provides insights into the asymptotic convergence of the algorithms, abstracting away the choices of step size, discretization, and accumulation errors.

We observe that the update 10 can be discretized from the following continuous-time form:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t &= \partial_{\mathbf{S}}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t) - \Phi(\partial_{\mathbf{W}}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t)) \\ \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{S}_t &= -\partial_{\mathbf{W}}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t) - \Psi(\partial_{\mathbf{S}}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t))\end{aligned} \tag{11}$$

where $\mathcal{H}(.)$ is a Hamiltonian function that satisfies:

$$\min_{\mathbf{S}} \mathcal{H}(\mathbf{W}, \mathbf{S}) = \mathcal{L}(\mathbf{W}) \quad \forall \mathbf{W},$$

meaning that minimizing $\mathcal{H}(\mathbf{W}, \mathbf{S})$ will reduce to minimizing the original objective $\mathcal{L}(\mathbf{W})$. Additionally, Φ, Ψ are two monotonic mapping satisfying:

$$\|\mathbf{A}\|_\Phi^2 = \langle \mathbf{A}, \Phi(\mathbf{A})\rangle \geq 0, \qquad \|\mathbf{A}\|_\Psi^2 = \langle \mathbf{A}, \Psi(\mathbf{A})\rangle \geq 0, \quad \forall \mathbf{A}$$

The key properties is that $\mathcal{H}(\mathbf{W}, \mathbf{S})$ is monotonically non-decreasing along the trajectory 11:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t) &= \left\langle \partial_{\mathbf{W}}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t), \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t \right\rangle + \left\langle \partial_{\mathbf{S}}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t), \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{S}_t \right\rangle \\ &= -\left\langle \partial_{\mathbf{W}}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t), \Phi(\partial_{\mathbf{W}}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t))\right\rangle - \left\langle \partial_{\mathbf{S}}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t), \Psi(\partial_{\mathbf{S}}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t))\right\rangle \\ &= -\|\partial_{\mathbf{W}}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t)\|_\Phi^2 - \|\partial_{\mathbf{S}}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t)\|_\Psi^2 \leq 0,\end{aligned}$$

where we cancel out the cross terms $\langle \partial_{\mathbf{W}}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t), \partial_{\mathbf{S}}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t)\rangle$ to get the second equation.

This powerful framework covers a wide range of optimizers, including moment-based algorithms as follows:

Momentum SGD:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t = -\mathbf{M}_t, \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t = \alpha(\mathbf{G}_t - \mathbf{M}_t), \quad \mathbf{G}_t = \nabla\mathcal{L}(\mathbf{W}_t)$$

with the Hamiltonian function is defined by: $\mathcal{H}(\mathbf{W}, \mathbf{M}) = \mathcal{L}(\mathbf{W}) + \frac{1}{2\alpha}\langle \mathbf{M}, \mathbf{M}\rangle$.

Adam (Kingma and Ba, 2014):

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t = -\frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t} + \epsilon}, \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t = \mathbf{G}_t - \mathbf{M}_t, \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{V}_t = \mathbf{G}_t^2 - \mathbf{V}_t, \quad \mathbf{G}_t = \nabla\mathcal{L}(\mathbf{W}_t)$$

with the Hamiltonian function is defined by: $\mathcal{H}(\mathbf{W}, \mathbf{M}, \mathbf{V}) = \mathcal{L}(\mathbf{W}) + \frac{1}{2}\left\langle \frac{\mathbf{M}}{\sqrt{\mathbf{V}} + \epsilon}, \mathbf{M}\right\rangle$.

Adafactor (Shazeer and Stern, 2018):

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t = -\frac{\mathbf{M}_t}{\sqrt{\mathbf{r}_t\mathbf{s}_t^\top/\mathbf{1}_m^\top\mathbf{r}_t}}, \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t = \mathbf{G}_t - \alpha\mathbf{M}_t, \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{r}_t = (\mathbf{G}_t)^2\mathbf{1}_n - \alpha\mathbf{r}_t, \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{s}_t = (\mathbf{G}_t^\top)^2\mathbf{1}_m - \alpha\mathbf{s}_t$$

with the Hamiltonian function is defined by: $\mathcal{H}(\mathbf{W}, \mathbf{M}, \mathbf{r}, \mathbf{s}) = f(\mathbf{W}) + \frac{1}{2}\left\langle \frac{\mathbf{M}}{\sqrt{\mathbf{r}\mathbf{s}^\top/\mathbf{1}_m^\top\mathbf{r}}}, \mathbf{M}\right\rangle$.

Hfac (Nguyen et al., 2024):

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t = -\frac{\mathbf{G}_t}{\sqrt{\mathbf{r}_t\mathbf{s}_t^\top/\mathbf{1}_m^\top\mathbf{r}_t}} - \frac{1}{2}\left[\frac{\mathbf{u}_t\mathbf{1}_n^\top - \mathbf{G}_t\mathbf{1}_n\mathbf{1}_n^\top/n}{\sqrt{\mathbf{r}_t\mathbf{1}_n^\top}} + \frac{\mathbf{1}_m\mathbf{v}_t^\top - \mathbf{1}_m\mathbf{1}_m^\top\mathbf{G}_t/m}{\sqrt{\mathbf{1}_m\mathbf{s}_t^\top}}\right]$$
$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{u}_t = \mathbf{G}_t\mathbf{1}_n/n - \alpha\mathbf{u}_t, \qquad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}_t = \mathbf{G}_t^\top\mathbf{1}_m/m - \alpha\mathbf{v}_t, \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{r}_t = (\mathbf{G}_t)^2\mathbf{1}_n - \alpha\mathbf{r}_t, \qquad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{s}_t = (\mathbf{G}_t^\top)^2\mathbf{1}_m - \alpha\mathbf{s}_t$$

with the Hamiltonian function is defined by: $\mathcal{H}(\mathbf{W},\mathbf{M},\mathbf{r},\mathbf{s}) = f(\mathbf{W}) + \frac{1}{4}\left\langle \frac{\mathbf{u}\mathbf{1}_n^\top}{\sqrt{\mathbf{r}\mathbf{1}_n^\top}}, \mathbf{u}\mathbf{1}_n^\top \right\rangle + \frac{1}{4}\left\langle \frac{\mathbf{1}_m\mathbf{v}^\top}{\sqrt{\mathbf{1}_m\mathbf{s}^\top}}, \mathbf{1}_m\mathbf{v}^\top \right\rangle$.

Proof: Let's take Adam optimizer as a specific example, we have the derivative:

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}(\mathbf{W}_t,\mathbf{M}_t,\mathbf{V}_t)\\
&= \left\langle \mathbf{G}_t, \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t \right\rangle + \left\langle \frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon}, \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t \right\rangle - \frac{1}{4}\left\langle \frac{\mathbf{M}_t^2}{\sqrt{\mathbf{V}_t}\odot\left(\sqrt{\mathbf{V}_t}+\epsilon\right)^2}, \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{V}_t \right\rangle\\
&= \left\langle \mathbf{G}_t, -\frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon} \right\rangle + \left\langle \frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon}, \mathbf{G}_t - \mathbf{M}_t \right\rangle - \frac{1}{4}\left\langle \frac{\mathbf{M}_t^2}{\sqrt{\mathbf{V}_t}\odot\left(\sqrt{\mathbf{V}_t}+\epsilon\right)^2}, \mathbf{G}_t^2 - \mathbf{V}_t \right\rangle\\
&= -\left\langle \frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon}, \mathbf{M}_t \right\rangle + \frac{1}{4}\left\langle \frac{\mathbf{M}_t^2}{\left(\sqrt{\mathbf{V}_t}+\epsilon\right)^2}, \sqrt{\mathbf{V}_t} \right\rangle - \frac{1}{4}\left\langle \frac{\mathbf{M}_t^2}{\sqrt{\mathbf{V}_t}\odot\left(\sqrt{\mathbf{V}_t}+\epsilon\right)^2}, \mathbf{G}_t^2 \right\rangle\\
&= -\left\langle \frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon}, \mathbf{M}_t \right\rangle + \frac{1}{4}\left\langle \frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon}\odot\frac{\sqrt{\mathbf{V}_t}}{\sqrt{\mathbf{V}_t}+\epsilon}, \mathbf{M}_t \right\rangle - \frac{1}{4}\left\langle \frac{\mathbf{M}_t^2}{\sqrt{\mathbf{V}_t}\odot\left(\sqrt{\mathbf{V}_t}+\epsilon\right)^2}, \mathbf{G}_t^2 \right\rangle\\
&\le -\frac{1}{4}\left\langle \frac{\mathbf{M}_t^2}{\sqrt{\mathbf{V}_t}\odot\left(\sqrt{\mathbf{V}_t}+\epsilon\right)^2}, \mathbf{G}_t^2 \right\rangle \le 0.
\end{aligned}$$

A similar analysis can be applied to the AdaDiag and AdaDiag++. In AdaDiag's case, we have the continuous form:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t = -\mathbf{P}_t\frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon}, \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t = \mathbf{P}_t^\top\mathbf{G}_t - \mathbf{M}_t, \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{V}_t = (\mathbf{P}_t^\top\mathbf{G}_t)^2 - \mathbf{V}_t, \tag{12}$$

yielding the Hamiltonian function $\mathcal{H}(\mathbf{W},\mathbf{M},\mathbf{V}) = \mathcal{L}(\mathbf{W}) + \frac{1}{2}\left\langle \frac{\mathbf{M}}{\sqrt{\mathbf{V}}+\epsilon}, \mathbf{M} \right\rangle$, for which:

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}(\mathbf{W}_t,\mathbf{M}_t,\mathbf{V}_t)\\
&= \left\langle \mathbf{G}_t, \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t \right\rangle + \left\langle \frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon}, \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t \right\rangle - \frac{1}{4}\left\langle \frac{\mathbf{M}_t^2}{\sqrt{\mathbf{V}_t}\odot\left(\sqrt{\mathbf{V}_t}+\epsilon\right)^2}, \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{V}_t \right\rangle\\
&= \left\langle \mathbf{G}_t, -\mathbf{P}_t\frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon} \right\rangle + \left\langle \frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon}, \mathbf{P}_t^\top\mathbf{G}_t - \mathbf{M}_t \right\rangle - \frac{1}{4}\left\langle \frac{\mathbf{M}_t^2}{\sqrt{\mathbf{V}_t}\odot\left(\sqrt{\mathbf{V}_t}+\epsilon\right)^2}, (\mathbf{P}_t^\top\mathbf{G}_t)^2 - \mathbf{V}_t \right\rangle\\
&= -\left\langle \frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon}, \mathbf{M}_t \right\rangle + \frac{1}{4}\left\langle \frac{\mathbf{M}_t^2}{\left(\sqrt{\mathbf{V}_t}+\epsilon\right)^2}, \sqrt{\mathbf{V}_t} \right\rangle - \frac{1}{4}\left\langle \frac{\mathbf{M}_t^2}{\sqrt{\mathbf{V}_t}\odot\left(\sqrt{\mathbf{V}_t}+\epsilon\right)^2}, (\mathbf{P}_t^\top\mathbf{G}_t)^2 \right\rangle\\
&\le -\frac{1}{4}\left\langle \frac{\mathbf{M}_t^2}{\sqrt{\mathbf{V}_t}\odot\left(\sqrt{\mathbf{V}_t}+\epsilon\right)^2}, (\mathbf{P}_t^\top\mathbf{G}_t)^2 \right\rangle \le 0.
\end{aligned}$$

This result implies that the points in the limit set:

$$\mathcal{I} = \{\text{the union of complete trajectories satisfying } \frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}(\mathbf{W}_t,\mathbf{M}_t,\mathbf{V}_t) = 0\}$$

must satisfy $\mathbf{P}_t^\top \mathbf{G}_t \equiv 0$. Since $\mathbf{P}_t$ is a full-rank orthogonal matrix, we have $\mathbf{G}_t \equiv 0$, indicating that the optimization algorithm converges to a local optimum.

For AdafacDiag and HfacDiag, the proofs can be derived in a similar manner by incorporating the analyses of Adafactor and Hfac provided in Nguyen et al. (2024). It is worth mentioning that our algorithms can also be regarded as a special case of Online Subspace Descent with Generalized Linear Projection (Liang et al., 2024). In their work, however, to provide the convergence guarantee for arbitrary update rules of the projection matrix $\mathbf{P}_t$, the authors need a mild condition to avoid the degenerate case of $\mathbf{P}_t\mathbf{G}_t = 0$ while $\mathbf{G}_t = 0$ in the invariant set of the system. In our analysis, this assumption is not required when $\mathbf{P}_t$ is a full-rank orthogonal matrix. □