Memory-Efficient Optimization with Factorized Hamiltonian Descent

Son Nguyen **Lizhang Chen** **Bo Liu** **Qiang Liu**

The University of Texas at Austin
{sonnv77,lzchen,bliu,lqiang}@utexas.edu

Abstract

Modern deep learning heavily depends on adaptive optimizers such as Adam and its variants, which are renowned for their capacity to handle model scaling and streamline hyperparameter tuning. However, these algorithms typically experience high memory overhead caused by the accumulation of optimization states, leading to a critical challenge in training large-scale network models. In this study, we introduce a novel adaptive optimizer, *H-Fac*, which incorporates a memory-efficient factorization approach to address this challenge. By employing a rank-1 parameterization for both momentum and scaling parameter estimators, *H-Fac* reduces memory costs to a sublinear level while maintaining competitive performance across a wide range of architectures. We develop our algorithms based on principles derived from Hamiltonian dynamics, providing robust theoretical underpinnings in optimization dynamics and convergence guarantees. These optimization algorithms are designed to be both straightforward and adaptable, facilitating easy implementation in diverse settings.

1. INTRODUCTION

Optimization algorithms play an indisputable role in the remarkable development of AI, especially in the realm of modern deep learning. In recent years, the emergence of breakthroughs in architectural innovation (Bommasani et al., 2021) and their practical applications, has further promoted the necessity for embracing efficient training paradigms, which encompass optimization algorithms striking a balance between performance and manageable memory costs.

Stochastic gradient descent (SGD) is widely regarded as the standard algorithm for training deep learning models, supported by extensive theoretical foundations (Şimşekli et al., 2019; Smith et al., 2021; Tian et al., 2023; Zhou et al., 2020). However, it requires thorough tuning of hyperparameters and frequently exhibits undesirable convergence rates when applied to many contemporary architectures (Devlin et al., 2018; Vaswani et al., 2017; Zhang et al., 2020). Meanwhile, adaptive moment-based methods such as Adam (Kingma and Ba, 2014), AdaGrad (Duchi et al., 2011), AMSGrad (Reddi et al., 2019), and variants, can adjust the learning rate for each parameter throughout the optimization process by utilizing a cumulative second moment of the gradient statistics. Although the theoretical aspects have not yet been fully exploited, these methods show empirical performance that surpasses SGD across multiple domains and often provides better convergence properties in practice (Zhang et al., 2020; Loshchilov and Hutter, 2017), making them highly appealing for large-scale applications. However, maintaining momentum and per-coordinate scaling parameter accumulators in these algorithms will significantly increase memory overhead. This barrier typically restricts model size, reduces the number of examples per mini-batch, or limits communication in decentralized training (Konečnỳ et al., 2016; Li et al., 2020; Liu et al., 2024), thereby negatively affecting convergence and accuracy.

Several memory-efficient optimizers have been devised to address this problem. In particular, Adafactor (Shazeer and Stern, 2018) is a highly efficient algorithm that was designed for low memory usage and greater scalability. It achieves substantial memory savings by employing a non-negative matrix factorization approach to decompose the second-moment accumulator into two distinct rank-1 factors. This technique not only minimizes memory footprint but also enhances computational efficiency. However, to avoid performance degradation, Adafactor still needs to maintain

Proceedings of the 28[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

the first-moment estimator, particularly when applied to large-scale models. This could potentially limit its applicability in memory-constrained environments. Additionally, Adafactor does not yet have a solid theoretical foundation, leaving its underlying methodologies and effectiveness without formal justification.

In this work, we propose to approach the aforementioned challenges through the powerful tool of Hamiltonian dynamics. Motivated by some inspirational works (Maddison et al., 2018; Chen et al., 2023), we explore how Hamiltonian principles in modeling dynamic systems can inform and improve optimization strategies, especially for memory efficiency purposes. Our contributions can be summarized in the following:

- We demonstrate that Adafactor can be deduced from an ordinary differential equation (ODE) that solves a minimization problem with constrained factors. This novel perspective allows us to derive other algorithms with the same principles.
- We propose a new class of efficient optimization algorithms that embraces both momentum-based and adaptive methods. By employing rank-1 parameterization for momentum and scaling parameter, our optimizers can offer sublinear memory costs, comparable to that of vanilla SGD without momentum. To the best of our knowledge, our proposal is the first endeavor to exploit Hamiltonian dynamics in developing a class of memory-efficient optimization algorithms that utilize factorized gradient statistic estimators.
- Distinct from existing optimization techniques for memory efficiency, our algorithm can offer clear insights into optimization dynamics and convergence guarantees, which are naturally inherited from the fundamental theory of Hamiltonian mechanics. This guarantee is significant because it can be broadly applied to a wide range of proposed optimizers, without relying on strong assumptions about underlying models or objective functions.
- Empirical results show that our optimizers can achieve favorable and comparable results across various architectures, including ResNets, Vision Transformers, and language models.

Organization. The rest of the paper is organized as follows: The next section will discuss related works on memory-efficient optimization. We provide in section 3 a brief overview of gradient-based optimization and the Hamiltonian descent framework. Section 4 presents our main methodological contributions. The experimental results are shown in section 5. Finally, in section 6, we will discuss the potential of our proposal for future works. The proofs, experimental settings, and additional results are left in Supplementary Materials.

Notations: We denote model parameters by a matrix of size $m \times n$. Let $\mathbf{1}_m$ and $\mathbf{1}_n$ be vectors of ones with dimensions m and n, respectively. For any matrices $\mathbf{A}, \mathbf{B}$ of size $m \times n$, we use $\sqrt{\mathbf{A}}$ for element-wise square root, $\mathbf{A}^2$ for element-wise square, and $\mathbf{A}/\mathbf{B}$ to denote element-wise division. $\mathbf{A}^\top$ stands for the transpose matrix, $\mathrm{trace}(\mathbf{A})$ denotes the trace of *square* matrix $\mathbf{A}$, $\langle \mathbf{A}, \mathbf{B} \rangle = \mathrm{trace}(\mathbf{A}^\top \mathbf{B})$ represents the inner products of matrices, and the function $\mathrm{RMS}(\mathbf{A}) = \sqrt{\frac{1}{m \times n} \sum_{i,j} A_{ij}^2}$ represents root-mean-square calculation.

2. RELATED WORKS

Numerous techniques have been developed to address memory overhead in optimization algorithms, each targeting specific aspects to enhance efficiency.

Adafactor (Shazeer and Stern, 2018) minimizes the memory cost to a sublinear level by factorizing the second-moment estimator using a row-column outer product. Luo et al. (Luo et al., 2023) introduced a confidence-guided strategy to mitigate erroneous updates and reduce instability in Adafactor training, while still retaining the same memory footprint. SM3 (Anil et al., 2019) is another efficient algorithm, which organizes the parameter space into sets and simplifies the maintenance of second-moment statistics by computing the maximum squared gradient for each set.

There are several methods that focus on the low-rank structure of the gradient rather than the moment statistics (Gooneratne et al., 2020; Zhao et al., 2024). For instance, GaLore (Zhao et al., 2024) periodically applies Singular Value Decomposition (SVD) to project the full gradients onto a lower-dimensional subspace, and subsequently utilize these projected gradients in adaptive optimization processes. Sketchy (Feinberg et al., 2024) leverages the Frequent Directions (FD) sketch technique to maintain a low-rank approximation of the gradient covariances matrix. Additionally, there is a line of work adapting quantization to reduce the memory cost of optimizer states (Dettmers et al., 2021, 2024; Li et al., 2024).

For second-order (Hessian-based) methods, the memory constraints primarily arise from computing the inverse Hessian matrix. Some successful works have also resolved this bottleneck using Gauss-Newton decomposition (Liu et al., 2023) or Kronecker-based factorization (Mozaffari et al., 2024).

3. BACKGROUND

3.1. Gradient-based Optimization

We consider an unconstrained, continuous optimization problem $\min_{\mathbf{W}\in\mathbb{R}^{m\times n}} f(\mathbf{W})$, with a proper differentiable and lower bounded objective $f:\mathbb{R}^{m\times n}\to\mathbb{R}$. In deep learning, gradient-based optimization algorithms have been the de facto choice for solving such problems, with first-order methods being widely adopted due to their scalability and effectiveness. Formally, we can unify these algorithms in a generalized form as:

$$\mathbf{W}_{t+1}=\mathbf{W}_t+\varphi_t(\mathbf{S}_t)\quad \mathbf{S}_t=\gamma_t(\mathbf{S}_{t-1},\nabla f(\mathbf{W}_t)) \quad (1)$$

where $\mathbf{S}_t$ is the optimization state, and φ_t,γ_t are some mapping functions. Principled designs of $\mathbf{S}_t$ rely on first-order gradient statistics, leading to popular algorithms including Momentum Gradient Descent (GDm (Polyak, 1964)), Sign Momentum (Signum Bernstein et al. (2018), Lion (Chen et al., 2024)), and adaptive optimizers like RMSprop (Tieleman, 2012), Adam (Kingma and Ba, 2014) and variants.

Specifically with Adam, the optimization states consist of estimates for the mean and uncentered variance of gradient information. These moment estimators are computed by applying exponential moving averages (EMA) on both gradients and their squares across training iterations. The update rules are:

$$\begin{aligned}\mathbf{M}_t&=\hat\beta_{1t}\mathbf{M}_{t-1}+(1-\hat\beta_{1t})\nabla f(\mathbf{W}_t)\\ \mathbf{V}_t&=\hat\beta_{2t}\mathbf{V}_{t-1}+(1-\hat\beta_{2t})\nabla f(\mathbf{W}_t)^2\\ \mathbf{W}_{t+1}&=\mathbf{W}_t-\eta_t\mathbf{M}_t/(\sqrt{\mathbf{V}_t}+\epsilon)\end{aligned} \quad (2)$$

where the decay moment coefficients $\hat\beta_{1,2t}=\beta_{1,2}(1-\beta_{1,2}^{t-1})/(1-\beta_{1,2}^t)$ is defined as equivalent to the bias correction step. By leveraging the second moment $\mathbf{V}_t$, Adam can dynamically adjust the learning rate for each parameter, making it highly effective at navigating non-convex loss landscapes. However, this advantage comes with the trade-off of increased memory overhead.

Adafactor optimizer. Shazeer and Stern (2018) address Adam's memory cost by employing an efficient rank-1 parameterization for the scaling factor, $\mathbf{V}\approx\mathbf{r}\mathbf{s}^\top$. The updates of vectors r and s were derived by minimizing the total elementwise I-divergence subject to componentwise non-negative constraints:

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

in which $r_i\ge 0, s_j\ge 0$ and $d(p,q)=p\log\frac{p}{q}-p+q$. Solving this problem results in a closed-form solution denoted by $\mathbf{r}=\mathbf{V}\mathbf{1}_m, \mathbf{s}=\mathbf{V}^\top\mathbf{1}_n/\mathbf{r}^\top\mathbf{1}_m$. Subsequently,

Algorithm 1 Adafactor-m for matrix parameters.

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 moments $\mathbf{M}_0,\mathbf{r}_0,\mathbf{s}_0\leftarrow 0$
for $t=1$ to T **do**
$\mathbf{G}_t=\nabla f_t(\mathbf{W}_t)$
$\mathbf{M}_t=\hat\beta_{1t}\mathbf{M}_{t-1}+(1-\hat\beta_{1t})\mathbf{G}_t$
$\mathbf{r}_t=\hat\beta_{2t}\mathbf{r}_{t-1}+(1-\hat\beta_{2t})\big[(\mathbf{G}_t)^2+\epsilon\big]\mathbf{1}_n$
$\mathbf{s}_t=\hat\beta_{2t}\mathbf{s}_{t-1}+(1-\hat\beta_{2t})\big[(\mathbf{G}_t^\top)^2+\epsilon\big]\mathbf{1}_m$
$\widehat{\mathbf{V}}_t=\mathbf{r}_t\mathbf{s}_t^\top/(1_m^\top\mathbf{r}_t)$
$\mathbf{W}_{t+1}=\mathbf{W}_t-\eta_t\left(\text{clip}\left(\mathbf{M}_t/\sqrt{\widehat{\mathbf{V}}_t}\right)+\lambda\mathbf{W}_t\right)$
end for

the iterative process of Adafactor optimizer can be outlined as in Algorithm 1. In the model parameter update, we have $\text{clip}(\mathbf{U})=\mathbf{U}/\max(1,\text{RMS}(\mathbf{U})/d)$ with RMS(.) refers to the root-mean-square function and d is the threshold value, meaning that we cap the norm of the actual update rather than just the gradient. This technique aims to eliminate the larger-than-desired updates and also stabilize the training process with slow decay. Briefly, Adafactor tracks the moving averages of the row and column sums of squared gradients over iterations, resulting in estimates of factored second moment $\mathbf{r}_t$ and $\mathbf{s}_t$. A normalized outer product $\mathbf{r}_t\mathbf{s}_t^\top/(\mathbf{1}_m^\top\mathbf{r}_t)$ is then used to reconstruct a low-rank parameterization of the second moment $\widehat{\mathbf{V}}_t$.

3.2. Hamiltonian Descent

One powerful approach to studying the dynamic behavior of the gradient-based optimization methods is to examine their continuous-time forms in the limit of infinitesimal step size (Maddison et al., 2018; Gao et al., 2022; Chen et al., 2023). In this regime, we can derive insights into the asymptotic convergence of the algorithms, abstracting away the choices of step size, discretization, and accumulation errors.

Let's examine the case of Momentum Gradient Descent (GDm) for simplicity. The update procedure of GDm is based on the following iterative scheme:

$$\mathbf{W}_{t+1}=\mathbf{W}_t-\eta_t\mathbf{M}_t,\ \mathbf{M}_t=\beta\mathbf{M}_{t-1}+\nabla f(\mathbf{W}_t) \quad (3)$$

In case we employ *full-batch* gradients, the GDm scheme becomes deterministic and can be viewed as a discretization of a *continuous-time system* as follows:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t=-\mathbf{M}_t,\qquad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t=\nabla f(\mathbf{W}_t)-\gamma\mathbf{M}_t \quad (4)$$

where $\mathbf{M}_t$ and γ are in analogy to the *velocity* and *friction* in classical mechanics. In principle, this ordinary

differential equation (ODE) defines a trajectory of the particle $\mathbf{W}_t$ and its velocity $\mathbf{M}_t$, which drives the system characterized by the *total energy* or Hamiltonian function,

$$\mathcal{H}(\mathbf{W}, \mathbf{M}) \triangleq f(\mathbf{W}) + \langle \mathbf{M}, \mathbf{M} \rangle / 2 = f(\mathbf{W}) + \|\mathbf{M}\|_F^2/2,$$

towards stationary points. The Hamiltonian $\mathcal{H}(.)$ is an augmented function of the original objective $f(.)$ and it satisfies $\min_{\mathbf{M}} \mathcal{H}(\mathbf{W}, \mathbf{M}) = \mathcal{L}(\mathbf{W}) \, \forall \mathbf{W}$, meaning that minimizing $\mathcal{H}(\mathbf{W}, \mathbf{M})$ will reduce to minimizing $f(\mathbf{W})$.

Convergence to Local Optima. We can show that the Hamiltonian $\mathcal{H}_t$ is monotonically decreasing along the ODE trajectory:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}_t &= \mathrm{trace}(\nabla^\top f(\mathbf{W}_t)\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t) + \mathrm{trace}(\mathbf{M}_t^\top \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t) \\ &= -\gamma \mathrm{trace}(\mathbf{M}_t^\top \mathbf{M}_t) = -\gamma \|\mathbf{M}_t\|_F^2 \leq 0.\end{aligned}$$

with $\|.\|_F$ is the Frobenius norm and $\mathcal{H}_t \triangleq \mathcal{H}(\mathbf{W}_t, \mathbf{M}_t)$. By LaSalle's Invariance principle, the set of accumulation points $(\mathbf{W}_t, \mathbf{M}_t)$ must be contained in $\mathcal{I}$, where $\mathcal{I} = \{$the union of complete trajectories satisfying $\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}(\mathbf{W}_t, \mathbf{M}_t) = 0, \forall t\}$. The points in the limit set $\mathcal{I}$ should satisfy $\mathbf{M}_t \equiv 0$, leading to $\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t \equiv 0$ and $\nabla f(\mathbf{W}_t) \equiv 0$. This indicates that all trajectories will converge to local optimal points.

From this perspective, momentum-based optimizers can be generalized using the following ODE:

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

where $\mathbf{G}_t \triangleq \nabla f(\mathbf{W}_t)$, $\mathcal{K}(\mathbf{X})$ is any convex function whose minimum is achieved at $\mathbf{X} = 0$, and $\nabla \mathcal{K}(.)$ is a monotonic operator. This system yields a Hamiltonian:

$$\mathcal{H}(\mathbf{W}, \mathbf{M}) = f(\mathbf{W}) + \frac{1}{\beta}\mathcal{K}(\beta \mathbf{M}).$$

We can see that this framework naturally recovers the two classical first-order methods, namely Gradient Descent and Sign Gradient Descent, by choosing $\mathcal{K}(\mathbf{X}) = \|\mathbf{X}\|_2^2/2$ and $\mathcal{K}(\mathbf{X}) = \|\mathbf{X}\|_1$, respectively.

The Hamiltonian interpretation is meaningful because its underlying logic suggests that a proper optimizer should be guaranteed to converge to the local optima of loss function, at least when the step sizes are sufficiently small. This powerful tool can facilitate us to theoretically establish a broader class of optimizers with guarantees, as well as providing complementary insights, improving or extending existing algorithms. In the next section, we will reinforce this perspective by extending the Hamiltonian framework to more modern algorithms, especially by providing new memory-efficient optimizers based on this principle.

4. FACTORIZED HAMILTONIAN DESCENT

In this section, we first interpret Adafactor optimizer from the perspective of Hamiltonian descent. We then introduce a general approach to factorize the momentum in first-order optimization methods. We will specifically deliver a factorization version for the sign-based momentum update and name this optimizer as *sign*FSGD. Based on further insights, we propose a novel adaptive factorized optimization algorithm, named *H-Fac*. This is our main algorithmic contribution to this paper.

4.1. Adafactor optimizer as Hamiltonian

The factorization technique used in Adafactor is computationally efficient and scalable as it offers analytical formulations without the need for additional approximations. However, directly applying a non-negative decomposition approach like that often appears heuristic. It fails to provide any insights into the optimization dynamics and convergence guarantees when we reparameterize the second-moment estimator using such a low-rank representation. Compared to simple optimizers like momentum gradient descent, Adafactor involves complex interactions between model parameter $\mathbf{W}_t$ and optimization states $\{\mathbf{M}_t, \mathbf{r}_t, \mathbf{s}_t\}$. This could hinder the analysis of convergence or other theoretical properties.

Interestingly, we demonstrate that the iterative procedure in Adafactor-m algorithm can be discretized from an ordinary differential equation (ODE) as follows:

$$\begin{aligned}&\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t = -\frac{\mathbf{M}_t}{\sqrt{\mathbf{r}_t \mathbf{s}_t^\top / \mathbf{1}_m^\top \mathbf{r}_t}}, \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t = \mathbf{G}_t - \alpha \mathbf{M}_t \\ &\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{r}_t = (\mathbf{G}_t)^2 \mathbf{1}_n - \alpha \mathbf{r}_t, \qquad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{s}_t = (\mathbf{G}_t^\top)^2 \mathbf{1}_m - \alpha \mathbf{s}_t \\ &\qquad\qquad\qquad\qquad\qquad\qquad \text{// Adafactor-m (ODE)}\end{aligned} \tag{5}$$

which solves a minimization problem with respect to the Hamiltonian function described by:

$$\mathcal{H}(\mathbf{W}, \mathbf{M}, \mathbf{r}, \mathbf{s}) = f(\mathbf{W}) + \frac{1}{2}\left\langle \frac{\mathbf{M}}{\sqrt{\mathbf{r}\mathbf{s}^\top / \mathbf{1}_m^\top \mathbf{r}}}, \mathbf{M} \right\rangle.$$

Proposition 1. *A key property is that the function $\mathcal{H}$ monotonically decreases along the ODE trajectory, that is,* $\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}(\mathbf{W}_t, \mathbf{M}_t, \mathbf{r}_t, \mathbf{s}_t) \leq 0$.

A detailed proof is provided in Appendix A.2.

Proposition 1 is a natural extension of the traditional Hamiltonian Descent presented in Section 3, where the regularization term in the Hamiltonian function

$\mathcal{H}(\mathbf{W}, \mathbf{M}, \mathbf{r}, \mathbf{s})$ still plays a role as kinetic energy. The key difference is that the momentum $\mathbf{M}$ is now normalized by the scaling parameter $\mathbf{r}\mathbf{s}^\top/(1_m^\top \mathbf{r})$, making the term equivalent to the kinetic energy with non-uniform mass values. In the next sections, we will broaden the framework to encompass a wider range of optimizers, including both momentum-based and adaptive algorithms. By doing so, we aim to integrate various memory-efficient optimization techniques within a unified framework of Hamiltonian dynamics.

4.2. Factorized first-moment estimation

Factoring the second moment is a straightforward yet intuitive idea to handle memory overhead. This approach offers an advantage in that it allows for the tolerance of some information loss, as our focus is solely on the magnitude of the scaling parameter. Despite this, most algorithms still exhibit an inherent drawback of requiring a full momentum to avoid performance degradation. Dealing with the first moment of gradient information, however, poses more challenging problems, as we must consider both the magnitude and update directions.

Factorization via rank-1 parameterization. We propose here a sublinear memory optimizer that factorizes the first-moment estimator. The algorithm is specifically derived from an ordinary differential equation outlined as (recall that $\mathbf{W} \in \mathbb{R}^{m\times n}$ is a matrix):

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t &= -\nabla\phi\big(\beta\hat{\mathbf{u}}_t\mathbf{1}_n^\top + \mathbf{G}_t\big) - \nabla\psi\big(\beta\mathbf{1}_m\hat{\mathbf{v}}_t^\top + \mathbf{G}_t\big)\\
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{u}_t &= \mathbf{G}_t\mathbf{1}_n/n - \alpha\mathbf{u}_t, \quad \hat{\mathbf{u}}_t = \mathbf{u}_t - \mathbf{G}_t\mathbf{1}_n/n \qquad (6)\\
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}_t &= \mathbf{G}_t^\top\mathbf{1}_m/m - \alpha\mathbf{v}_t, \quad \hat{\mathbf{v}}_t = \mathbf{v}_t - \mathbf{G}_t^\top\mathbf{1}_m/m\\
&\qquad\qquad \text{// Factorized first moment (ODE)}
\end{aligned}$$

where α, β are two positive parameters; ϕ and ψ are any convex functions, such that $\nabla\phi(\cdot), \nabla\psi(\cdot)$ are monotonic operators. These functions play the same role as $\mathcal{K}(.)$ in the general framework described in Section 3.2. Therefore, the simplest option to consider is that $\phi(\mathbf{W}) = \psi(\mathbf{W}) = \|\mathbf{W}\|_2^2/2$, resulting their gradients $\nabla\phi(\mathbf{W}) = \nabla\psi(\mathbf{W}) = \mathbf{W}$. This system yields the Hamiltonian function:

$$\mathcal{H}(\mathbf{W}, \mathbf{u}, \mathbf{v}) = f(\mathbf{W}) + \frac{\beta}{2}\left\langle \mathbf{u}\mathbf{1}_n^\top, \mathbf{u}\mathbf{1}_n^\top\right\rangle + \frac{\beta}{2}\left\langle \mathbf{1}_m\mathbf{v}^\top, \mathbf{1}_m\mathbf{v}^\top\right\rangle.$$

A detailed analysis is provided in Appendix A.1.

Back to our framework, a number of remarks are in order:

1) Compared with algorithms that maintain a full rank momentum matrix $\mathbf{M} \in \mathbb{R}^{m\times n}$ as defined in (4), this algorithm employs two rank-one momentum vectors $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^n$, which significantly reduces the memory cost from $O(mn)$ to $O(m+n)$. In the algorithm, we use the simple unit vectors $\mathbf{1}_m, \mathbf{1}_n$ to first down project the full gradient $\nabla f(\mathbf{W})$ into the rank-one spaces of column means and row means, and exponentially accumulate these statistics across training steps. We then reconstruct the rank-one momentum $\hat{\mathbf{u}}, \hat{\mathbf{v}}$ to update the model parameters. The $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$ are centralized variants of $\mathbf{u}$ and $\mathbf{v}$.

Algorithm 2 *sign*FSGD for matrix parameters.

Inputs: moment coefficients $\beta = 0.9$, and regularization constant λ
Initialization: weight parameters $\mathbf{W}_1 \in \mathbb{R}^{m\times n}$, initial moment factors $\mathbf{u}_0, \mathbf{v}_0 \leftarrow 0$
for $t = 1$ to T **do**
$\mathbf{G}_t = \nabla f_t(\mathbf{W}_t)$
$\mathbf{u}_t = \beta\mathbf{u}_{t-1} + (1-\beta)\mathbf{G}_t\mathbf{1}_n/n$
$\mathbf{v}_t = \beta\mathbf{v}_{t-1} + (1-\beta)\mathbf{G}_t^\top\mathbf{1}_m/m$
$\hat{\mathbf{u}}_t = \mathbf{u}_t - \mathbf{G}_t\mathbf{1}_n/n$
$\hat{\mathbf{v}}_t = \mathbf{v}_t - \mathbf{G}_t^\top\mathbf{1}_m/m$
$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_t\Big(\mathrm{sign}(\beta\hat{\mathbf{u}}_t\mathbf{1}_n^\top + \mathbf{G}_t) + \mathrm{sign}(\beta\mathbf{1}_m\hat{\mathbf{v}}_t^\top + \mathbf{G}_t) + \lambda\mathbf{W}_t\Big)$
end for

2) Discretizing ODE (6) using the Euler method results in the following iterative scheme:

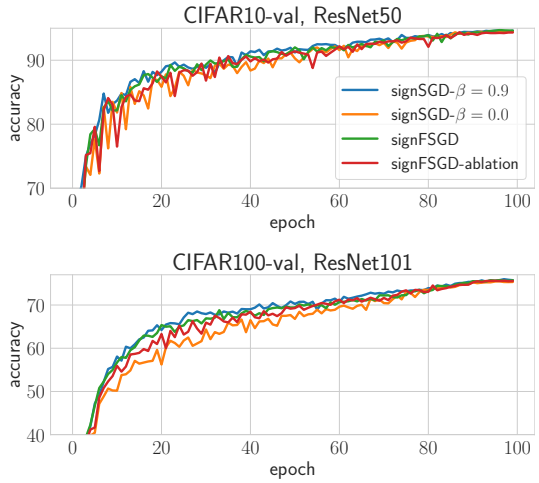
$$\begin{aligned}
\mathbf{u}_t &= \hat{\beta}_{1t}\mathbf{u}_{t-1} + (1-\hat{\beta}_{1t})\mathbf{G}_t\mathbf{1}_n/n, \quad \hat{\mathbf{u}}_t = \mathbf{u}_t - \mathbf{G}_t\mathbf{1}_n/n\\
\mathbf{v}_t &= \hat{\beta}_{1t}\mathbf{v}_{t-1} + (1-\hat{\beta}_{1t})\mathbf{G}_t^\top\mathbf{1}_m/m, \quad \hat{\mathbf{v}}_t = \mathbf{v}_t - \mathbf{G}_t^\top\mathbf{1}_m/m\\
\mathbf{W}_t &= \mathbf{W}_{t-1} - \eta_t\Big[\nabla\phi\left(\hat{\beta}_{1t}\hat{\mathbf{u}}_t\mathbf{1}_n^\top + \mathbf{G}_t\right)\\
&\qquad + \nabla\psi\left(\hat{\beta}_{1t}\mathbf{1}_m\hat{\mathbf{v}}_t^\top + \mathbf{G}_t\right)\Big]
\end{aligned}$$

In discrete-time analysis, we centralize the column-means and row-means statistics $\mathbf{u}_t, \mathbf{v}_t$ by corrected terms $\mathbf{G}_t\mathbf{1}_n/n$ and $\mathbf{G}_t^\top\mathbf{1}_m/m$, respectively. These corrected terms are essential as they can guarantee the updates inside $\nabla\phi(.)$ and $\nabla\psi(.)$ will converge to the true first moment $\mathbb{E}[\mathbf{G}_t]$. Specifically, by unfolding the factored moment $\mathbf{u}_t$ (and $\mathbf{v}_t$, similarly), we can get an accumulator of column-wise mean of gradients as:

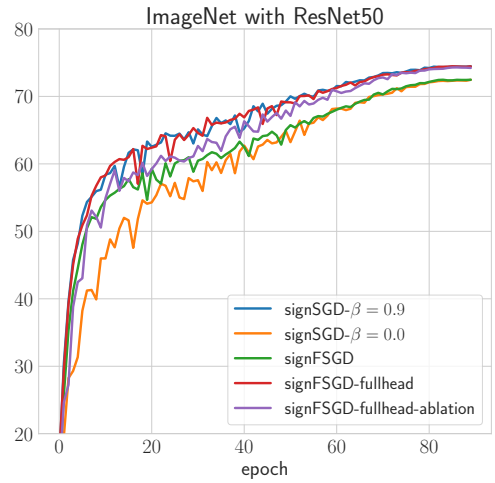
$$\mathbf{u}_t = \sum_{\tau=1}^{t}(1-\hat{\beta}_{1\tau})\prod_{\kappa=\tau+1}^{t}\hat{\beta}_{1\kappa}\mathbf{G}_\tau\mathbf{1}_n.$$

By the assumption that the gradient distribution is stationary, taking expectations gives us:

$$\begin{aligned}
\mathbb{E}[\mathbf{u}_t] &= \sum_{\tau=1}^{t}(1-\hat{\beta}_{1\tau})\prod_{\kappa=\tau+1}^{t}\hat{\beta}_{1\kappa}\mathbb{E}[\mathbf{G}_\tau\mathbf{1}_n]\\
&= \sum_{\tau=1}^{t}(1-\hat{\beta}_{1\tau})\prod_{\kappa=\tau+1}^{t}\hat{\beta}_{1\kappa}\mathbb{E}[\mathbf{G}_t\mathbf{1}_n].
\end{aligned}$$



(a) Validation accuracy on CIFAR datasets



(b) Validation accuracy on ImageNet1K

Figure 1: A comparison of optimizer performance on ResNet architectures. For *sign*SGD, β denotes the momentum coefficient. For *sign*FSDG, "*ablation*" means the version without corrected terms, "*fullhead*" means the version using full momentum for the MLP head layer.

Futhermore, we can prove by induction that $\sum_{\tau=1}^{t}(1-\hat{\beta}_{1\tau})\prod_{\kappa=\tau+1}^{t}\hat{\beta}_{1\kappa}=1$, leading to $\mathbb{E}[\mathbf{u}_t]=\mathbb{E}[\mathbf{G}_t)\mathbf{1}_n]$ and hence the expected value of centralized moment factor $\mathbb{E}[\hat{\mathbf{u}}_t]$ will approximate the true first moment $\mathbb{E}[\mathbf{G}_t]$.

3) When the momentum coefficient is deactivated ($\beta_1 = 0$), the update terms inside $\nabla\phi(.)$ and $\nabla\psi(.)$ degenerate to $\mathbf{G}_t$. In this scenario, the algorithm simplifies to standard gradient descent:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t = -\nabla\phi(\mathbf{G}_t) - \nabla\psi(\mathbf{G}_t).$$

To empirically examine the prospects of our general framework and especially the necessity of the corrected terms, we conducted a basic image classification experiment on the CIFAR10 and CIFAR100 datasets using ResNet50 and ResNet101, respectively. We choose $\phi(.)$ and $\psi(.)$ to be the L_1-norm, namely $\phi(\mathbf{W}) = \psi(\mathbf{W}) = \|\mathbf{W}\|_1$, resulting in their gradients $\nabla\phi(.)$ and $\nabla\psi(.)$ being *sign*(.) functions. In this case, we refer to our algorithm as *sign*FSGD 2, which incorporates a rank-1 parameterization to factorize the momentum in *sign*SGD optimizer (Bernstein et al., 2018). We will evaluate the effectiveness of *sign*FSGD with and without the corrected terms, compare its performance to that of *sign*SGD with momentum enabled or disabled. Figure 1a shows that *sign*FSGD can yield comparable results to *sign*SGD with momentum. However, without the corrected terms, our algorithm is generally less stable and only gains relative improvements compared to *sign*SGD without momentum. Thus, simply accumulating row means and column means of gradient information across training steps is insufficient to accelerate the optimization process.

Motivated by the recent development of Lion optimizer (Chen et al., 2024), we adopted an efficient technique called double$-\beta$ scheme to further improve the performance of *sign*FSGD on ResNet models. We refer to this new algorithm as LionFactor 5, and provide some discussions in Appendix B.

4.3. Fully factorized moment estimations

Inadequacy of first-moment factorization. For ResNet models, mini-batch gradients are typically quite small and well concentrated around zero mean as shown in Figure 2. Consequently, accumulating the momentum through column means and row means of the gradient information will significantly flatten the magnitude. This issue could potentially reduce the impact of momentum on the optimization updates, especially with large-scale settings such as ImageNet and modern deep learning architectures.

In Figure 1b, we present the performances of *sign*SGD and *sign*FSGD when training ResNet50 from scratch on ImageNet1K. Notably, our method does not yield substantial enhancements compared to *sign*SGD without momentum as observed on CIFAR10, CIFAR100 datasets. However, when employing a full momentum instead of the factorized one for the MLP head layer, our *sign*FSGD can perform comparably to *sign*SGD with momentum. We witness the same behavior when applying the algorithm to larger ResNet architectures.

This observation reinforces our previous remark regarding the moderate impact of our factorized first moment on scale models. Although a few minor customizations, such as omitting the factorization of the MLP head

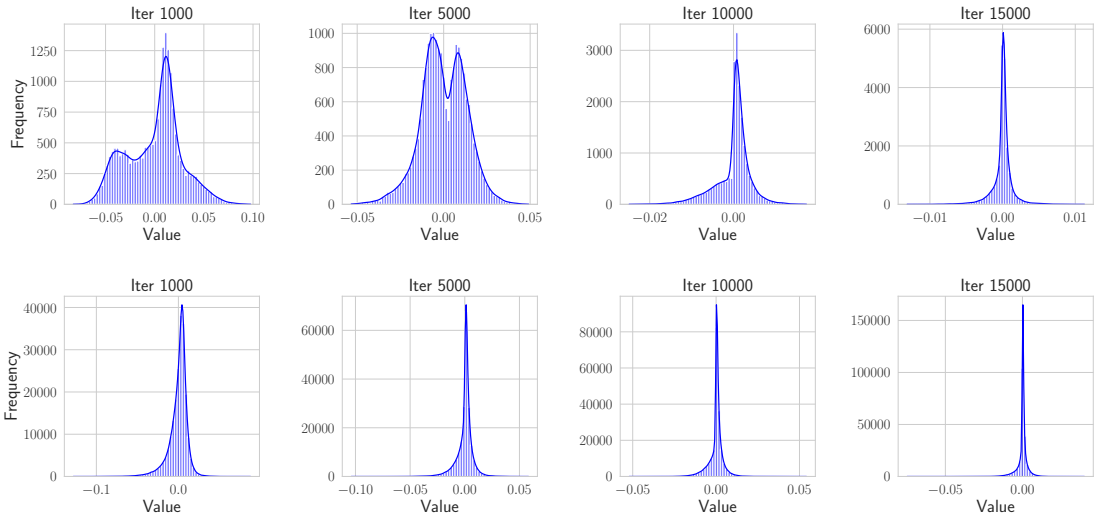


Figure 2: Histograms illustrating the gradients of MLP head layers in ResNet50 (top) and ResNet101 (bottom) trained on CIFAR10 and CIFAR100, respectively.

layer like above, could overcome the issue on ResNets, our initial expectation leaned towards a more general-purpose optimization algorithm. We specifically aim to extend the scalability of our algorithms to encompass modern convolution-free architectures.

Incorporating second-moment estimator. A natural idea to deal with the magnitude issue of factorized first-moment estimator is to normalize them by factored second moments. Indeed, by integrating our factorization approach for the first-moment estimator into Adafactor optimizer, we can establish a unified method where both moment estimators are fully factorized. In principle, we aim to conceptualize our algorithms within Hamiltonian frameworks, but deriving an objective function for such a simplistic integration might be challenging. We instead come up with a novel algorithm characterized by the following ODE:

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

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{u}_t = \mathbf{G}_t\mathbf{1}_n/n - \alpha\mathbf{u}_t, \qquad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}_t = \mathbf{G}_t^\top\mathbf{1}_m/m - \alpha\mathbf{v}_t$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{r}_t = (\mathbf{G}_t)^2\mathbf{1}_n - \alpha\mathbf{r}_t, \qquad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{s}_t = (\mathbf{G}_t^\top)^2\mathbf{1}_m - \alpha\mathbf{s}_t \qquad \text{// } \textit{H-Fac} \text{ (ODE)}$$

which yields the following Hamiltonian function:

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

Algorithm 3 *H-Fac* for matrix parameters.

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 $\mathbf{u}_0, \mathbf{v}_0, \mathbf{r}_0, \mathbf{s}_0 \leftarrow 0$
for $t = 1$ to T **do**
$\mathbf{G}_t = \nabla f_t(\mathbf{W}_t)$
$\mathbf{u}_t = \hat{\beta}_{1t}\mathbf{u}_{t-1} + (1-\hat{\beta}_{1t})\mathbf{G}_t\mathbf{1}_n/n$
$\mathbf{v}_t = \hat{\beta}_{1t}\mathbf{v}_{t-1} + (1-\hat{\beta}_{1t})\mathbf{G}_t^\top\mathbf{1}_m/m$
$\mathbf{r}_t = \hat{\beta}_{2t}\mathbf{r}_{t-1} + (1-\hat{\beta}_{2t})\big[(\mathbf{G}_t)^2 + \epsilon\big]\mathbf{1}_n$
$\mathbf{s}_t = \hat{\beta}_{2t}\mathbf{s}_{t-1} + (1-\hat{\beta}_{2t})\big[(\mathbf{G}_t^\top)^2 + \epsilon\big]\mathbf{1}_m$
$\widehat{\mathbf{V}}_t = \mathbf{r}_t\mathbf{s}_t^\top/(\mathbf{1}_m^\top\mathbf{r}_t)$
$\phi_{term} = \hat{\beta}_{1t}\big(\mathbf{u}_t\mathbf{1}_n^\top - \mathbf{G}_t\mathbf{1}_n\mathbf{1}_n^\top/n\big)/\sqrt{\mathbf{r}_t\mathbf{1}_n^\top/n}$
$\psi_{term} = \hat{\beta}_{1t}\big(\mathbf{1}_m\mathbf{v}_t^\top - \mathbf{1}_m\mathbf{1}_m^\top\mathbf{G}_t/m\big)/\sqrt{\mathbf{1}_m\mathbf{s}_t^\top/m}$
$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_t\Big(0.5(\phi_{term} + \psi_{term}) + \mathrm{clip}\left(\mathbf{G}_t/\sqrt{\widehat{\mathbf{V}}_t}\right) + \lambda\mathbf{W}_t\Big)$
end for

Intuitively, Adafactor-m uses a full momentum $\mathbf{M}$, whereas H-Fac decomposes $\mathbf{M}$ into vectors $\mathbf{u}$ and $\mathbf{v}$, leading to different normalizations in the kinetic energy. While Adafactor normalizes the first-moment $\mathbf{M}$ using the second-moment approximation $\mathbf{r}\mathbf{s}^\top/\mathbf{1}_m^\top\mathbf{r}$, H-Fac 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 $1_m\mathbf{s}_t^\top/m$. Although H-Fac employs the same derivation for second-moment factorization as in Adafactor-m, their parameter update mechanisms are fundamentally different. Adafactor-m, similar to Adam, updates parameters using the signal-to-noise ratio $\mathbf{M}/\sqrt{\mathbf{V}}$. On the other

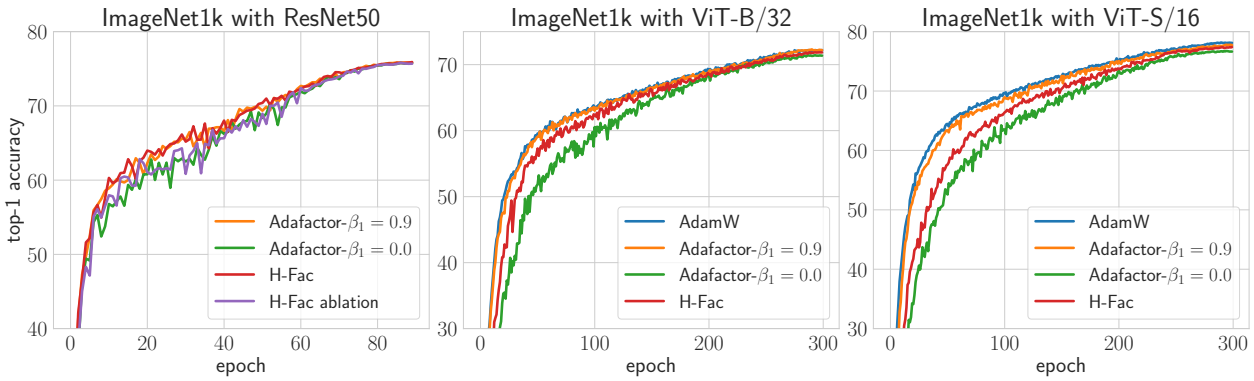


Figure 3: Top-1 Accuracy of optimizers in training ResNet50, ViT-B/32, and ViT-S/16 from scratch on the ImageNet1K. For *H-Fac*, "ablation" means the version without corrected terms.

hand, H-Fac adopts a momentum RMSprop-like approach, where parameter updates act as accumulators of normalized gradients. Both the momentum and the current gradient in H-Fac are rescaled by their corresponding cumulative second-moment information.

The discrete-time equivalent is presented in Algorithm 3. Specifically, our update includes a combination of three key elements: (i) the normalized momentum factorization $0.5 * (\phi_{term} + \psi_{term})$, in which the factorized first moment in section 4.2 is normalized by the row means and the column means of second-moment estimators; (ii) a clipping of normalized gradient $\text{clip}\big(\mathbf{G}_t/\sqrt{\hat{\mathbf{V}}_t}\big)$ which is inherited from Adafactor update; and (iii) a decouple weight decay $\lambda \mathbf{W}_t$ for enhancing the generalization performance of adaptive optimizers (Loshchilov and Hutter, 2017).

5. EXPERIMENTS

In this section, we conduct several experiments to demonstrate the effectiveness of our *H-Fac* algorithm. Unlike the earlier proposal of *sign*FSGD, we do not customize the algorithm for any specific layers, but perform factorization on the entire architectures.

5.1. Image Classification

Setup. We evaluated the optimization algorithms described in this paper on pre-training ResNet50 and Vision Transformers (ViTs) from scratch using ImageNet1K dataset, following previous works (Dosovitskiy et al., 2020; He et al., 2016). The images are preprocessed by Inception-style cropping (Szegedy et al., 2016) and random horizontal. We train ResNet50 for 90 epochs, using a batch size of 1024, with cosine learning rate decay scheduler. For ViTs, we train them for 300 epochs, using a batch size of 4096, with a learning rate schedule of 10,000 steps warmup followed by linear decay. We also adopted strong data augmentations, including RandAugment(2,15) (Cubuk et al., 2019) and mixup (0.5) (Zhang et al., 2017), to boost ViTs performances. Hyperparameters tuning of learning rate, weight decay, and dropout rate is given in Appendix D.

Evaluation. The results are shown in Figure 3 and Table 1. On ResNet50, we can notice consistent patterns with those in Figure 1b. While our algorithm, *H-Fac*, can achieve similar memory efficiency as Adafactor without momentum, it demonstrates more stable training and delivers highly competitive performance compared to Adafactor with momentum. The performance degradation is also clearly evident in the ablation study, consolidating the significance of the corrected terms in our algorithms. On ViT models, despite performance gaps in the early stages, *H-Fac* progressively reaches the level of AdamW and Adafactor with momentum as training advances. Furthermore, the factorized momentum in our algorithm actually accelerates the optimization process, as evidenced by substantial improvements over Adafactor without momentum.

5.2. Language Modeling

Setup. We apply the algorithms to pre-train LLaMA-based large language model models (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 and 130M for 10K and 20K, 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 use a maximum sequence length of 256 and a batch size of 512.

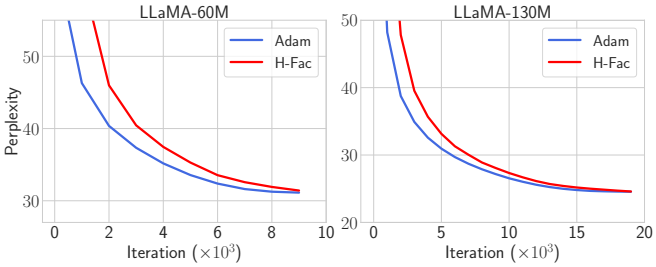


Figure 4: Training progression for pre-training LLaMA models on C4 dataset. Lower is better.

Evaluation. As shown in Figure 4, H-Fac demonstrates competitive performances, closely tracking Adam throughout training. At convergence, H-Fac achieves perplexities of 31.41 and 24.59 for the 60M and 130M models, respectively, which are comparable to Adam's values of 31.12 and 24.55.

6. CONCLUSION

In this study, we represent one of the first attempts to unify and devise adaptive optimization methods through the lens of Hamiltonian dynamics framework. Our approach enables the flexible design of a new class of memory-efficient optimizers, all while preserving theoretical convergence guarantees. For future works, we propose extending our algorithms to rank-k approximations, aiming to bridge the gap between these full-rank baselines. Another potential direction is to explore optimal projected subspaces, where the row and column means of gradient information can generate better-adapted momentum. For practical use, our current optimizers show promise in applications where ResNets are advantageous because of their efficient training nature. For example, we can adapt our algorithms to the federated learning optimization problem, so that the factored moment estimators can be efficiently communicated to accelerate the convergence. Other applications would also be of interest.

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

Anil, R., Gupta, V., Koren, T., and Singer, Y. (2019). Memory efficient adaptive optimization. *Advances in Neural Information Processing Systems*, 32.

Bernstein, J., Wang, Y.-X., Azizzadenesheli, K., and Anandkumar, A. (2018). signsgd: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*, pages 560–569. PMLR.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

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.

Dettmers, T., Lewis, M., Shleifer, S., and Zettlemoyer, L. (2021). 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2024). Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

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

Feinberg, V., Chen, X., Sun, Y. J., Anil, R., and Hazan, E. (2024). Sketchy: Memory-efficient adaptive regularization with frequent directions. *Advances in Neural Information Processing Systems*, 36.

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.

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.

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*, pages 770–778.

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*.

Li, B., Chen, J., and Zhu, J. (2024). Memory efficient optimizers with 4-bit states. *Advances in Neural Information Processing Systems*, 36.

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.

Liu, B., Wu, L., Chen, L., Liang, K., Zhu, J., Liang, C., Krishnamoorthi, R., and Liu, Q. (2024). Communication efficient distributed training with distributed lion. *arXiv preprint arXiv:2404.00438*.

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*.

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

Luo, Y., Ren, X., Zheng, Z., Jiang, Z., Jiang, X., and You, Y. (2023). Came: Confidence-guided adaptive memory efficient optimization. *arXiv preprint arXiv:2307.02047*.

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

Mozaffari, M., Li, S., Zhang, Z., and Mehri Dehnavi, M. (2024). Mkor: Momentum-enabled kronecker-factor-based optimizer using rank-1 updates. *Advances in Neural Information Processing Systems*, 36.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17.

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.

Tieleman, T. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26.

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*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

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.

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. [No]

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]

Memory-Efficient Optimization with Factorized Hamiltonian Descent: Supplementary Materials

A. HAMILTONIAN FUNCTION ANALYSES

A.1. Factorized momentum-based algorithm

Recall our general framework for factorized momentum-based optimization methods, which is characterized by the following ODE:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t &= -\nabla\phi\big(\beta\hat{\mathbf{u}}_t\mathbf{1}_n^\top + \nabla f(\mathbf{W}_t)\big) - \nabla\psi\big(\beta\mathbf{1}_m\hat{\mathbf{v}}_t^\top + \nabla f(\mathbf{W}_t)\big)\\
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{u}_t &= \nabla f(\mathbf{W}_t)\mathbf{1}_n/n - \alpha\mathbf{u}_t, \qquad \hat{\mathbf{u}}_t = \mathbf{u}_t - \nabla f(\mathbf{W}_t)\mathbf{1}_n/n\\
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}_t &= \nabla^\top f(\mathbf{W}_t)\mathbf{1}_m/m - \alpha\mathbf{v}_t, \qquad \hat{\mathbf{v}}_t = \mathbf{v}_t - \nabla^\top f(\mathbf{W}_t)\mathbf{1}_m/m.
\end{aligned}$$

Note that $\mathbf{W}_t$ is the weight matrix of size $m \times n$, $\mathbf{u}_t$ and $\mathbf{v}_t$ are column vectors of size m and n, respectively. For a canonical example with $\phi(\mathbf{X}) = \psi(\mathbf{X}) = \|\mathbf{X}\|_2^2/2$, we have $\nabla\phi(\mathbf{X}) = \nabla\psi(\mathbf{X}) = \mathbf{X}$, for all matrices $\mathbf{X}$. We will show that the following Hamiltonian function:

$$\mathcal{H}(\mathbf{W},\mathbf{u},\mathbf{v}) = f(\mathbf{W}) + \frac{\beta}{2}\left\langle \mathbf{u}\mathbf{1}_n^\top, \mathbf{u}\mathbf{1}_n^\top\right\rangle + \frac{\beta}{2}\left\langle \mathbf{1}_m\mathbf{v}^\top, \mathbf{1}_m\mathbf{v}^\top\right\rangle = f(\mathbf{W}) + \frac{\beta n}{2}\|\mathbf{u}\|_2^2 + \frac{\beta m}{2}\|\mathbf{v}\|_2^2,$$

monotonically decreases along the ODE trajectory above. Denote $\mathbf{G}_t = \nabla f(\mathbf{W}_t)$, we can explicitly express the derivative of the model parameter as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t = -\left(\beta\mathbf{u}_t\mathbf{1}_n^\top + \mathbf{G}_t - \frac{\beta}{n}\mathbf{G}_t\mathbf{1}_n\mathbf{1}_n^\top\right) - \left(\beta\mathbf{1}_m\mathbf{v}_t^\top + \mathbf{G}_t - \frac{\beta}{m}\mathbf{1}_m\mathbf{1}_m^\top\mathbf{G}_t\right),$$

then we can shorten the derivative of the function $\mathcal{H}(.)$ as:

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}(\mathbf{W}_t,\mathbf{u}_t,\mathbf{v}_t)\\
&\quad= \mathrm{trace}\left(\mathbf{G}_t^\top\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t\right) + \frac{\beta}{n}\mathbf{u}_t^\top\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{u}_t + \frac{\beta}{m}\mathbf{v}_t^\top\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}_t\\
&\quad= -\mathrm{trace}\left(\mathbf{G}_t^\top\left(\mathbf{G}_t - \frac{\beta}{n}\mathbf{G}_t\mathbf{1}_n\mathbf{1}_n^\top\right)\right) - \mathrm{trace}\left(\mathbf{G}_t^\top\left(\mathbf{G}_t - \frac{\beta}{m}\mathbf{1}_m\mathbf{1}_m^\top\mathbf{G}_t\right)\right) - \frac{\alpha\beta}{n}\|\mathbf{u}_t\|_2^2 - \frac{\alpha\beta}{m}\|\mathbf{v}_t\|_2^2\\
&\quad= -2\|\mathbf{G}_t\|_F^2 + \frac{\beta}{n}\|\mathbf{G}_t\mathbf{1}_n\|_2^2 + \frac{\beta}{m}\left\|\mathbf{G}_t^\top\mathbf{1}_m\right\|_2^2 - \frac{\alpha\beta}{n}\|\mathbf{u}_t\|_2^2 - \frac{\alpha\beta}{m}\|\mathbf{v}_t\|_2^2,
\end{aligned}$$

in which $\|.\|_F$ is the Frobenius norm. Since $\beta \in (0,1)$, applying C-S inequality, we have:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}(\mathbf{W}_t,\mathbf{u}_t,\mathbf{v}_t) \le -2\|\mathbf{G}_t\|_F^2 + \frac{1}{n}\|\mathbf{G}_t\mathbf{1}_n\|_2^2 + \frac{1}{m}\left\|\mathbf{G}_t^\top\mathbf{1}_m\right\|_2^2 \le 0. \qquad \square$$

A.2. Adafactor

Recall the ODE of Adafactor with first-order moment integration:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t &= -\frac{\mathbf{M}_t}{\sqrt{\mathbf{r}_t\mathbf{s}_t^\top/\mathbf{1}_m^\top\mathbf{r}_t}}, \qquad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t = \nabla f(\mathbf{W}_t) - \alpha\mathbf{M}_t\\
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{r}_t &= (\nabla f(\mathbf{W}_t))^2\mathbf{1}_n - \beta\mathbf{r}_t, \qquad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{s}_t = (\nabla^\top f(\mathbf{W}_t))^2\mathbf{1}_m - \beta\mathbf{s}_t
\end{aligned}$$

We will show that the Hamiltonian function is described by:

$$\mathcal{H}(\mathbf{W},\mathbf{M},\mathbf{r},\mathbf{s}) = f(\mathbf{W}) + \frac{1}{2}\left\langle \frac{\mathbf{M}}{\sqrt{\mathbf{r}\mathbf{s}^\top/\mathbf{1}_m^\top\mathbf{r}}}, \mathbf{M}\right\rangle = f(\mathbf{W}) + \underbrace{\frac{1}{2}\sum_{i=1,j=1}^{m,n}\frac{M_{ij}^2\sqrt{\sum_{i=1}^m r_i}}{\sqrt{r_i s_j}}}_{\mathcal{R}(\mathbf{M},\mathbf{r},\mathbf{s})},$$

is monotonically decreasing along the ODE trajectory when $\beta \le 2\alpha$, in which M_{ij} is the element (i,j)-th of the first-moment matrix $\mathbf{M}$, r_i and s_j represent the i-th and j-th elements of the column vectors $\mathbf{r}$ and $\mathbf{s}$, respectively. Denote $\mathbf{G}_t = \nabla f(\mathbf{W}_t)$ and $\mathcal{R}_t = \mathcal{R}(\mathbf{M}_t,\mathbf{r}_t,\mathbf{s}_t)$, then the derivative of $\mathcal{H}(.)$ can be expressed as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}(\mathbf{W}_t,\mathbf{M}_t,\mathbf{r}_t,\mathbf{s}_t) = \mathrm{trace}\left(\mathbf{G}_t^\top\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t\right) + \mathrm{trace}\left((\nabla_{\mathbf{M}}\mathcal{R}_t)^\top\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t\right) + (\nabla_{\mathbf{r}}\mathcal{R}_t)^\top\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{r}_t + (\nabla_{\mathbf{s}}\mathcal{R}_t)^\top\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{s}_t.$$

Analyzing each element specifically, we get:

$$\mathrm{trace}(\mathbf{G}_t^\top\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{W}_t) = -\mathrm{trace}\left(\mathbf{G}_t^\top\frac{\mathbf{M}_t\sqrt{\mathbf{1}_m^\top\mathbf{r}_t}}{\sqrt{\mathbf{r}_t\mathbf{s}_t^\top}}\right),$$

$$\mathrm{trace}\left((\nabla_{\mathbf{M}}\mathcal{R}_t)^\top\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{M}_t\right) = \mathrm{trace}\left((\mathbf{G}_t - \alpha\mathbf{M}_t)^\top\frac{\mathbf{M}_t\sqrt{\mathbf{1}_m^\top\mathbf{r}_t}}{\sqrt{\mathbf{r}_t\mathbf{s}_t^\top}}\right) = \mathrm{trace}\left(\mathbf{G}_t^\top\frac{\mathbf{M}_t\sqrt{\mathbf{1}_m^\top\mathbf{r}_t}}{\sqrt{\mathbf{r}_t\mathbf{s}_t^\top}}\right) - \alpha\,\mathrm{trace}\left(\mathbf{M}_t^\top\frac{\mathbf{M}_t\sqrt{\mathbf{1}_m^\top\mathbf{r}_t}}{\sqrt{\mathbf{r}_t\mathbf{s}_t^\top}}\right),$$

$$\begin{aligned}
(\nabla_{\mathbf{r}}\mathcal{R}_t)^\top\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{r}_t &= \sum_{i=1}^m(\partial_{r_i}\mathcal{R}_t)(\mathbf{1}_n^\top\mathbf{G}_{t,i}^2 - \beta r_{t,i}) && \text{// } \mathbf{G}_{t,i} \text{ is the } i\text{-th row of } \mathbf{G}_t, \text{ and } r_{t,i} \text{ is the } i\text{-th element of } \mathbf{r}_t\\
&= \sum_{i=1}^m\left(\frac{1}{2}\sum_{j=1}^n\frac{M_{t,ij}^2}{\sqrt{s_{t,j}}}\right)\frac{-\sum_{k\neq i}r_{t,k}}{2\sqrt{r_{t,i}^{3/2}}\sqrt{\sum_i r_{t,i}}}(\mathbf{1}_n^\top\mathbf{G}_{t,i}^2 - \beta r_{t,i})\\
&&& \text{// } M_{t,ij} \text{ is the } (i,j)\text{-th element of } \mathbf{M}_t, \text{ and } s_{t,j} \text{ is the } j\text{-th element of } \mathbf{s}_t\\
&\le \sum_{i=1}^m\left(\frac{\beta}{4}\sum_{j=1}^n\frac{M_{t,ij}^2}{\sqrt{s_{t,j}}}\right)\frac{\sum_{k\neq i}r_{t,k}}{\sqrt{r_{t,i}}\sqrt{\sum_i r_{t,i}}} && \text{// the multiplication by } \mathbf{1}_n^\top\mathbf{G}_{t,i}^2 \text{ is } \le 0\\
&\le \sum_{i=1}^m\left(\frac{\beta}{4}\sum_{j=1}^n\frac{M_{t,ij}^2}{\sqrt{s_{t,j}}}\right)\frac{\sqrt{\sum_i r_{t,i}}}{\sqrt{r_{t,i}}} && \text{// } \textstyle\sum_{k\neq i} r_{t,k} \le \sum_i r_{t,i}\\
&= \sum_{i=1}^m\left(\frac{\beta}{4}\sum_{j=1}^n\frac{M_{t,ij}^2\sqrt{\sum_i r_{t,i}}}{\sqrt{r_{t,i}s_{t,j}}}\right)\\
&= \frac{\beta}{4}\mathrm{trace}\left(\mathbf{M}_t^\top\frac{\mathbf{M}_t\sqrt{\mathbf{1}_m^\top\mathbf{r}_t}}{\sqrt{\mathbf{r}_t\mathbf{s}_t^\top}}\right),
\end{aligned}$$

$$\begin{aligned}
(\nabla_{\mathbf{s}}\mathcal{R}_t)^\top\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{s}_t &= \sum_{j=1}^n(\partial_{s_j}\mathcal{R}_t)(\mathbf{1}_m^\top\mathbf{G}_{t,:j}^2 - \beta s_{t,j}) && \text{// } \mathbf{G}_{t,:j} \text{ is the } j\text{-th column of } \mathbf{G}_t\\
&= \sum_{j=1}^n\left(\frac{1}{2}\sum_{i=1}^m\frac{M_{t,ij}^2\sqrt{\sum_i r_{t,i}}}{\sqrt{r_{t,i}}}\right)\frac{-1}{2\sqrt{s_{t,j}^{3/2}}}(\mathbf{1}_m^\top\mathbf{G}_{t,:j}^2 - \beta s_{t,j})\\
&\le \sum_{j=1}^n\left(\frac{\beta}{4}\sum_{i=1}^m\frac{M_{t,ij}^2\sqrt{\sum_i r_{t,i}}}{\sqrt{r_{t,i}s_{t,j}}}\right) && \text{// the multiplication by } \mathbf{1}_m^\top\mathbf{G}_{t,:j}^2 \text{ is } \le 0\\
&= \frac{\beta}{4}\mathrm{trace}\left(\mathbf{M}_t^\top\frac{\mathbf{M}_t\sqrt{\mathbf{1}_m^\top\mathbf{r}_t}}{\sqrt{\mathbf{r}_t\mathbf{s}_t^\top}}\right).
\end{aligned}$$

Canceling out similar quantities, we can rewrite the Hamiltonian derivative as follows:

$$\frac{\mathrm{d}}{\mathrm{dt}}\mathcal{H}(\mathbf{W}_t, \mathbf{M}_t, \mathbf{r}_t, \mathbf{s}_t) \leq \left(\frac{\beta}{2} - \alpha\right) \text{trace}\left(\mathbf{M}_t^\top \frac{\mathbf{M}_t\sqrt{\mathbf{1}_m^\top \mathbf{r}_t}}{\sqrt{\mathbf{r}_t\mathbf{s}_t^\top}}\right) = \left(\frac{\beta}{2} - \alpha\right)\sum_{i=1}^{m}\sum_{j=1}^{n}\frac{M_{t,ij}^2\sqrt{\sum_i r_{t,i}}}{\sqrt{r_{t,i}s_{t,j}}} \leq 0. \qquad \square$$

A.3. H-Fac

In this part, we will prove that the following ODE trajectory (with $\beta \leq 4\alpha$):

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{W}_t &= -\frac{1}{2}\left(\frac{\mathbf{u}_t\mathbf{1}_n^\top - \nabla f(\mathbf{W}_t)\mathbf{1}_n\mathbf{1}_n^\top/n}{\sqrt{\mathbf{r}_t\mathbf{1}_n^\top}} + \frac{\mathbf{1}_m\mathbf{v}_t^\top - \mathbf{1}_m\mathbf{1}_m^\top\nabla f(\mathbf{W}_t)/m}{\sqrt{\mathbf{1}_m\mathbf{s}_t^\top}}\right) - \frac{\nabla f(\mathbf{W}_t)}{\sqrt{\mathbf{r}_t\mathbf{s}_t^\top/\mathbf{1}_m^\top\mathbf{r}_t}} \\
\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{u}_t &= \nabla f(\mathbf{W}_t)\mathbf{1}_n/n - \alpha\mathbf{u}_t \\
\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{v}_t &= \nabla^\top f(\mathbf{W}_t)\mathbf{1}_m/m - \alpha\mathbf{v}_t \\
\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{r}_t &= (\nabla f(\mathbf{W}_t))^2\mathbf{1}_n - \beta\mathbf{r}_t \\
\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{s}_t &= (\nabla^\top f(\mathbf{W}_t))^2\mathbf{1}_m - \beta\mathbf{s}_t
\end{aligned}$$

descends the Hamiltonian function defined by:

$$\mathcal{H}(\mathbf{W}, \mathbf{u}, \mathbf{v}, \mathbf{r}, \mathbf{s}) := f(\mathbf{W}) + \frac{1}{4}\left\langle\frac{\mathbf{u}\mathbf{1}_n^\top}{\sqrt{\mathbf{r}\mathbf{1}_n^\top}}, \mathbf{u}\mathbf{1}_n^\top\right\rangle + \frac{1}{4}\left\langle\frac{\mathbf{1}_m\mathbf{v}^\top}{\sqrt{\mathbf{1}_m\mathbf{s}^\top}}, \mathbf{1}_m\mathbf{v}^\top\right\rangle = f(\mathbf{W}) + \underbrace{\frac{n}{4}\sum_{i=1}^{m}\frac{u_i^2}{\sqrt{r_i}} + \frac{m}{4}\sum_{j=1}^{n}\frac{v_j^2}{\sqrt{s_j}}}_{\mathcal{R}(\mathbf{u},\mathbf{v},\mathbf{r},\mathbf{s})}.$$

with u_i, r_i represent the i-th elements of the column vectors $\mathbf{u}, \mathbf{r}$, and v_j, s_j represent the j-th elements of the column vectors $\mathbf{v}, \mathbf{s}$. Denote $\mathbf{G}_t = \nabla f(\mathbf{W}_t)$ and $\mathcal{R}_t = \mathcal{R}(\mathbf{u}_t, \mathbf{v}_t, \mathbf{r}_t, \mathbf{s}_t)$, we have:

$$\frac{\mathrm{d}}{\mathrm{dt}}\mathcal{H}(\mathbf{W}_t, \mathbf{u}_t, \mathbf{v}_t, \mathbf{r}_t, \mathbf{s}_t) = \text{trace}\left(\mathbf{G}_t^\top\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{W}_t\right) + (\nabla_{\mathbf{u}}\mathcal{R}_t)^\top\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{u}_t + (\nabla_{\mathbf{v}}\mathcal{R}_t)^\top\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{v}_t + (\nabla_{\mathbf{r}}\mathcal{R}_t)^\top\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{r}_t + (\nabla_{\mathbf{s}}\mathcal{R}_t)^\top\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{s}_t.$$

Similar to the previous part, we can calculate each element specifically as follows:

$$\begin{aligned}
\text{trace}(\mathbf{G}_t^\top\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{W}_t) &= -\frac{1}{2}\text{trace}\left(\mathbf{G}_t^\top\frac{\mathbf{u}_t\mathbf{1}_n^\top}{\sqrt{\mathbf{r}_t\mathbf{1}_n^\top}}\right) - \frac{1}{2}\text{trace}\left(\mathbf{G}_t^\top\frac{\mathbf{1}_m\mathbf{v}_t^\top}{\sqrt{\mathbf{1}_m\mathbf{s}_t^\top}}\right) \\
&\quad + \text{trace}\left(\mathbf{G}_t^\top\frac{\mathbf{G}_t\mathbf{1}_n\mathbf{1}_n^\top}{2n\sqrt{\mathbf{r}_t\mathbf{1}_n^\top}}\right) + \text{trace}\left(\mathbf{G}_t^\top\frac{\mathbf{1}_m\mathbf{1}_m^\top\mathbf{G}_t}{2m\sqrt{\mathbf{1}_m\mathbf{s}_t^\top}}\right) - \text{trace}\left(\mathbf{G}_t^\top\frac{\mathbf{G}_t\sqrt{\mathbf{1}_m^\top\mathbf{r}_t}}{\sqrt{\mathbf{r}_t\mathbf{s}_t^\top}}\right), \\
(\nabla_{\mathbf{u}}\mathcal{R}_t)^\top\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{u}_t &= \frac{n}{2}\left(\frac{\mathbf{u}_t}{\sqrt{\mathbf{r}_t}}\right)^\top(\mathbf{G}_t\mathbf{1}_n/n - \alpha\mathbf{u}_t) = \frac{1}{2}\left(\frac{\mathbf{u}_t}{\sqrt{\mathbf{r}_t}}\right)^\top\mathbf{G}_t\mathbf{1}_n - \frac{n\alpha}{2}\left(\frac{\mathbf{u}_t}{\sqrt{\mathbf{r}_t}}\right)^\top\mathbf{u}_t, \\
(\nabla_{\mathbf{v}}\mathcal{R}_t)^\top\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{v}_t &= \frac{m}{2}\left(\frac{\mathbf{v}_t}{\sqrt{\mathbf{s}_t}}\right)^\top\left(\mathbf{G}_t^\top\mathbf{1}_m/m - \alpha\mathbf{v}_t\right) = \frac{1}{2}\left(\frac{\mathbf{v}_t}{\sqrt{\mathbf{s}_t}}\right)^\top\mathbf{G}_t^\top\mathbf{1}_m - \frac{m\alpha}{2}\left(\frac{\mathbf{v}_t}{\sqrt{\mathbf{s}_t}}\right)^\top\mathbf{v}_t, \\
(\nabla_{\mathbf{r}}\mathcal{R}_t)^\top\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{r}_t &= \left(-\frac{n}{8}\frac{\mathbf{u}_t^2}{\sqrt{\mathbf{r}_t^{3/2}}}\right)^\top(\mathbf{G}_t^2\mathbf{1}_n - \beta\mathbf{r}_t) \leq \frac{n\beta}{8}\left(\frac{\mathbf{u}_t^2}{\sqrt{\mathbf{r}_t^{3/2}}}\right)^\top\mathbf{r}_t, \quad \textit{// the multiplication by } \mathbf{G}_t^2\mathbf{1}_n \textit{ is} \leq 0 \\
(\nabla_{\mathbf{s}}\mathcal{R}_t)^\top\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{s}_t &= \left(-\frac{m}{8}\frac{\mathbf{v}_t^2}{\sqrt{\mathbf{s}_t^{3/2}}}\right)^\top\left((\mathbf{G}_t^2)^\top\mathbf{1}_m - \beta\mathbf{s}_t\right) \leq \frac{m\beta}{8}\left(\frac{\mathbf{v}_t^2}{\sqrt{\mathbf{s}_t^{3/2}}}\right)^\top\mathbf{s}_t.
\end{aligned}$$

$$\textit{// the multiplication by } (\mathbf{G}_t^2)^\top\mathbf{1}_m \textit{ is} \leq 0$$

Canceling out crossing terms:

$$\text{trace}\left(\mathbf{G}_t^\top \frac{\mathbf{u}_t\mathbf{1}_n^\top}{\sqrt{\mathbf{r}_t\mathbf{1}_n^\top}}\right) = \left(\frac{\mathbf{u}_t}{\sqrt{\mathbf{r}_t}}\right)^\top \mathbf{G}_t\mathbf{1}_n, \qquad \text{trace}\left(\mathbf{G}_t^\top \frac{\mathbf{1}_m\mathbf{v}_t^\top}{\sqrt{\mathbf{1}_m\mathbf{s}_t^\top}}\right) = \left(\frac{\mathbf{v}_t}{\sqrt{\mathbf{s}_t}}\right)^\top \mathbf{G}_t^\top\mathbf{1}_m$$

$$\left(\frac{\mathbf{u}_t}{\sqrt{\mathbf{r}_t}}\right)^\top \mathbf{u}_t = \left(\frac{\mathbf{u}_t^2}{\sqrt{\mathbf{r}_t^{3/2}}}\right)^\top \mathbf{r}_t, \qquad \left(\frac{\mathbf{v}_t}{\sqrt{\mathbf{s}_t}}\right)^\top \mathbf{v}_t = \left(\frac{\mathbf{v}_t^2}{\sqrt{\mathbf{s}_t^{3/2}}}\right)^\top \mathbf{s}_t$$

and with a note that $\beta \le 4\alpha$, we can rewrite the Hamiltonian derivative as follows:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}_t &\le \text{trace}\left(\mathbf{G}_t^\top \frac{\mathbf{G}_t\mathbf{1}_n\mathbf{1}_n^\top}{2n\sqrt{\mathbf{r}_t\mathbf{1}_n^\top}}\right) + \text{trace}\left(\mathbf{G}_t^\top \frac{\mathbf{1}_m\mathbf{1}_m^\top\mathbf{G}_t}{2m\sqrt{\mathbf{1}_m\mathbf{s}_t^\top}}\right) - \text{trace}\left(\mathbf{G}_t^\top \frac{\mathbf{G}_t\sqrt{\mathbf{1}_m^\top\mathbf{r}_t}}{\sqrt{\mathbf{r}_t\mathbf{s}_t^\top}}\right)\\ &= \text{trace}\left(\frac{1}{n}(\mathbf{G}_t\mathbf{1}_n)^2\frac{1}{2\sqrt{\mathbf{r}_t^\top}}\right) + \text{trace}\left(\frac{1}{m}(\mathbf{G}_t^\top\mathbf{1}_m)^2\frac{1}{2\sqrt{\mathbf{s}_t^\top}}\right) - \text{trace}\left(\mathbf{G}_t^2\frac{\sqrt{\mathbf{1}_m^\top\mathbf{r}_t}}{\sqrt{\mathbf{s}_t\mathbf{r}_t^\top}}\right).\end{aligned}$$

Applying C-S inequality gives us $\frac{1}{n}(\mathbf{G}_t\mathbf{1}_n)^2 \le \mathbf{G}_t^2\mathbf{1}_n$ and $\frac{1}{m}(\mathbf{G}_t^\top\mathbf{1}_m)^2 \le (\mathbf{G}_t^\top)^2\mathbf{1}_m$, therefore:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}_t &\le \text{trace}\left(\mathbf{G}_t^2\mathbf{1}_n\frac{1}{2\sqrt{\mathbf{r}_t^\top}}\right) + \text{trace}\left((\mathbf{G}_t^\top)^2\mathbf{1}_m\frac{1}{2\sqrt{\mathbf{s}_t^\top}}\right) - \text{trace}\left(\mathbf{G}_t^2\frac{\sqrt{\mathbf{1}_m^\top\mathbf{r}_t}}{\sqrt{\mathbf{s}_t\mathbf{r}_t^\top}}\right)\\ &= \frac{1}{2}\text{trace}\Bigg[\mathbf{G}_t^2\Bigg(\underbrace{\mathbf{1}_n\frac{1}{\sqrt{\mathbf{r}_t^\top}} + \frac{1}{\sqrt{\mathbf{s}_t}}\mathbf{1}_m^\top - 2\frac{\sqrt{\mathbf{1}_m^\top\mathbf{r}_t}}{\sqrt{\mathbf{s}_t\mathbf{r}_t^\top}}}_{\mathcal{F}}\Bigg)\Bigg].\end{aligned}$$

By zero initialization, we note that the moving averages of row sums and column sums are symmetric, in other words, $\mathbf{1}_m^\top\mathbf{r}_t$ and $\mathbf{1}_n^\top\mathbf{s}_t$ both represent the moving average of the sum of all squared gradient entries. Denote this quantity by $\mathcal{S}$, we have $\mathcal{S} = \mathbf{1}_m^\top\mathbf{r}_t = \mathbf{1}_n^\top\mathbf{s}_t \ge (r_{t,i} + s_{t,j})/2$ for all i, j, then consider each element of matrix $\mathcal{F}$:

$$\frac{1}{\sqrt{r_{t,i}}} + \frac{1}{\sqrt{s_{t,j}}} - 2\frac{\sqrt{\mathcal{S}}}{\sqrt{r_{t,i}s_{t,j}}} \le \frac{1}{\sqrt{r_{t,i}}} + \frac{1}{\sqrt{s_{t,j}}} - \frac{\sqrt{2(r_{t,i}+s_{t,j})}}{\sqrt{r_{t,i}s_{t,j}}} \le 0,$$

by C-S inequality. Therefore $\mathcal{F}$ is a negative matrix, and as a result, $\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}(\mathbf{W}_t, \mathbf{u}_t, \mathbf{v}_t, \mathbf{r}_t, \mathbf{s}_t) \le 0$. □

Convergence to local optimal. The above optimization algorithms all ensure that their Hamiltonian $\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t)$ are monotonically decreasing along the corresponding ODE trajectories, where $\mathbf{S}_t$ represents the optimization states in general. Consequently, by applying LaSalle's Invariance principle, the set of accumulation points $(\mathbf{W}_t, \mathbf{S}_t)$ must lie within the set $\mathcal{I}$, where $\mathcal{I} = \{\text{the union of complete trajectories satisfying } \frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}(\mathbf{W}_t, \mathbf{S}_t) = 0, \forall t\}$. Based on the preceding inequality transformations, it is evident that the points in the limit set $\mathcal{I}$ must satisfy $\nabla f(\mathbf{W}_t) \equiv 0$. This implies that those trajectories will converge to local optima.

B. FACTORIZED LION OPTIMIZER

Recently, a new optimization named Lion (Evolved Sign Momentum) (Chen et al., 2024) was discovered by an evolutionary search algorithm applied to a symbolically represented program space. Lion has been shown to achieve at least comparable performance to AdamW on a wide range of tasks while reducing memory cost and training time. Notably, Lion can be formulated as an iterative update procedure:

$$\begin{aligned}\mathbf{M}_t &= \beta_2\mathbf{M}_{t-1} + (1-\beta_2)\nabla f(\mathbf{W}_t)\\ \mathbf{W}_{t+1} &= \mathbf{W}_t - \eta_t\left(\text{sign}\left(\beta_1\mathbf{M}_{t-1} + (1-\beta_1)\nabla f(\mathbf{W}_t)\right) + \lambda\mathbf{W}_t\right) \qquad \text{// Lion}\end{aligned}$$

Algorithm 4 Lion.

Inputs: *double*-moment coefficients $\beta_1 = 0.9$, $\beta_2 = 0.99$, and regularization constant λ
Initialization: weight parameters $\mathbf{W}_1 \in \mathbb{R}^{m\times n}$, initial momentum $\mathbf{M}_0 \leftarrow 0$
for $t = 1$ to T **do**
  $\mathbf{G}_t = \nabla f_t(\mathbf{W}_t)$
  update model parrameters
  $\mathbf{C}_t = \beta_1\mathbf{M}_{t-1} + (1-\beta_1)\mathbf{G}_t$
  $\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_t\left(\text{sign}(\mathbf{C}_t) + \lambda\mathbf{W}_t\right)$
  update exponential moment averages
  $\mathbf{M}_t = \beta_2\mathbf{M}_{t-1} + (1-\beta_2)\mathbf{G}_t$
end for

Algorithm 5 Lionfactor for matrix parameter, with factored first-order moments.

Inputs: *double*-moment coefficients $\beta_1 = 0.9$, $\beta_2 = 0.99$, and regularization constant λ
Initialization: weight parameters $\mathbf{W}_1 \in \mathbb{R}^{m\times n}$, initial moment factors $\mathbf{u}_0, \mathbf{v}_0 \leftarrow 0$
for $t = 1$ to T **do**
  $\mathbf{G}_t = \nabla f_t(\mathbf{W}_t)$
  Update model parameters:
  $\hat{\mathbf{u}}_t = \beta_1\mathbf{u}_{t-1} + (1-\beta_1)\mathbf{G}_t\mathbf{1}_n/n - \mathbf{G}_t\mathbf{1}_n/n$
  $\hat{\mathbf{v}}_t = \beta_1\mathbf{v}_{t-1} + (1-\beta_1)\mathbf{G}_t^\top\mathbf{1}_m/m - \mathbf{G}_t^\top\mathbf{1}_m/m$
  $\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_t\left(\text{sign}\left(\hat{\mathbf{u}}_t\mathbf{1}_n^\top + \mathbf{G}_t\right)\right.$
  $\left. + \text{sign}(\mathbf{1}_m\hat{\mathbf{v}}_t^\top + \mathbf{G}_t) + \lambda\mathbf{W}_t\right)$
  Update exponential moment averages:
  $\mathbf{u}_t = \beta_2\mathbf{u}_{t-1} + (1-\beta_2)\mathbf{G}_t\mathbf{1}_n/n$
  $\mathbf{v}_t = \beta_2\mathbf{v}_{t-1} + (1-\beta_2)\mathbf{G}_t^\top\mathbf{1}_m/m$
end for

We can see that when $\beta_2 = \beta_1$, Lion will resemble *sign*SGD with momentum (Bernstein et al., 2018). However, Lion used a double$-\beta$ scheme with default values $\beta_1 = 0.9, \beta_2 = 0.99$. Intuitively, this allows Lion to remember longer the gradient history accumulated by the momentum, meanwhile assign a higher weight to the current gradient. Comprehensive experimental results show that Lion converges faster and usually generalizes better than AdamW, but with greater memory efficiency as it only keeps track of the momentum.

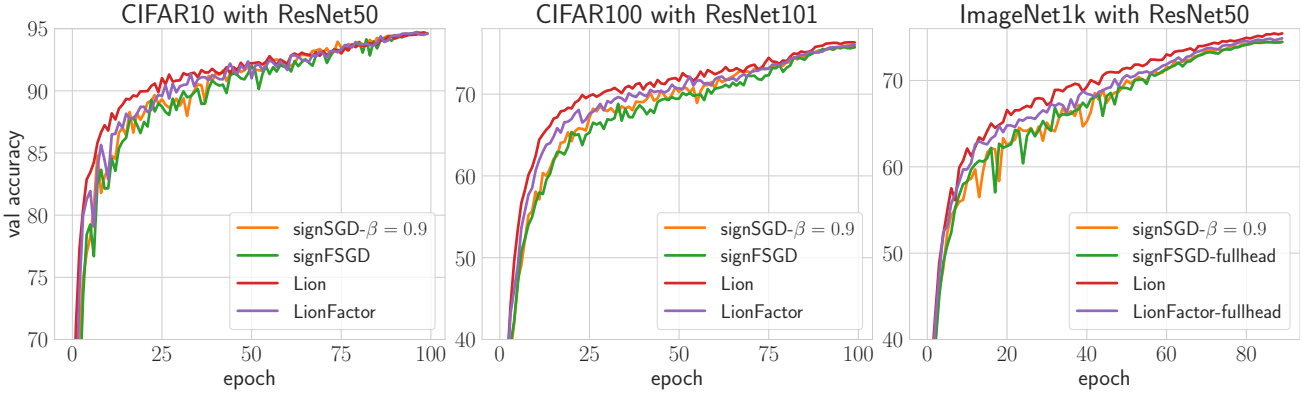


Figure 5: Performance of sign-based optimizers on ResNet architectures. "fullhead" means the version using full momentum for the MLP head layer.

We can similarly apply the double$-\beta$ scheme to our *sign*FGSD, and obtain a new algorithm that we call LionFactor 5. We conducted several experiments to evaluate the performance of LionFactor on ResNet models. The results are shown in Figure 5. Interestingly, LionFactor performs significantly better than *sign*FSGD, even *sign*SGD with momentum, in terms of both convergence rate and accuracy. Although LionFactor still shares the same drawbacks with *sign*FSGD when applied to models such as ViTs, it makes a lot of sense to explore more efficient algorithms to factorize the momentum in Lion optimizer. We leave it for future work.

C. NUMERICAL RESULTS

Table 1: ImageNet1K top-1 accuracy results for various optimizers on different models.

Models	***sign*SGD**		***sign*FSGD**	***sign*FSGD**	AdamW	Adafactor		H-Fac
	$m = 0.9$	$m = 0.0$		*fullhead*		$m = 0.9$	$m = 0.0$	
ResNet50	74.47	72.36	72.47	74.45	75.61	75.85	75.79	**75.90**
ViT-B/32					72.20	**72.31**	71.36	71.87
ViT-S/16					**78.35**	77.81	76.74	77.20

Table 2: 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
sign-GD w/ momentum	mn	mn	mn	0.11G	0.26G	0.72G	2.62G
Adam	mn	mn	$2mn$	0.23G	0.52G	1.44G	5.23G
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
Hfac **(ours)**	mn	mn	$2(m + n)$	**0.13G**	**0.19G**	**0.26G**	**0.52G**

We provide in Table 2 the memory cost of optimizers implemented in this paper. Our proposed optimizers, *sign*FSGD and H-Fac, offer sublinear memory costs, which are comparable to those of conventional gradient descent (without momentum). In our implementation, the rank-1 parameterization enables in-place vector operators that do not require additional memory.

D. HYPERPARAMETER SETTINGS

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

For the image classification task, we opted for recommended configurations of learning rate (lr), weight decay (λ), and dropout rate (dr) from prior research. The detailed settings are given in Table 3.

Table 3: Hyperparameters for the experiments of pre-training ResNet50 and ViTs on ImagNet1k

Optimizers	β_1	β_2	ϵ	**ResNet50**		**ViT-B/32, ViT-S/16**				
				lr	λ	lr	λ	dr	RandAug	Mixup
*sign*SGD	0.9, 0.0	-	-	0.0003	1.0	-	-	-	-	-
*sign*FSGD	0.9	-	-	0.0003	1.0	-	-	-	-	-
Lion	0.9	0.99	-	0.0003	1.0	-	-	-	-	-
Lionfactor	0.9	0.99	-	0.0003	1.0	-	-	-	-	-
Adam	0.9	0.999	1e-8	0.001	0.1	0.003	0.1	0.0	2, 15	0.5
Adafactor	0.9, 0.0	0.999	1e-30	0.001	0.1	0.003	0.1	0.0	2, 15	0.5
H-Fac	0.9	0.999	1e-30	0.001	0.1	0.003	0.1	0.0	2, 15	0.5

For language modeling, 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. The specific settings are summarized in the Table 4.

Table 4: Training configuration for LLaMA models

LLaMA models	**Tokens**	**Training Steps**	**Warmup Steps**	**Learning Rate**
60M	1.3B	10K	1K	0.003
130M	2.6B	20K	2K	0.001