OPT2024: 16th Annual Workshop on Optimization for Machine Learning

Understanding Adam Requires Better Rotation Dependent Assumptions

Tianyue H. Zhang[1,2,†] TIANYUE.ZHANG@MILA.QUEBEC
Lucas Maes[1,2,†] LUCAS.MAES@MILA.QUEBEC
Alexia Jolicoeur-Martineau[3]
Ioannis Mitliagkas[1,2,5]
Damien Scieur[2,3]
Simon Lacoste-Julien[1,2,3,4]
Charles Guille-Escuret[1,2]
[1]Mila, Quebec AI Institute [2]Université de Montréal [3]Samsung SAIL Montreal
[4]Canada CIFAR AI Chair [5]Archimedes Unit, Athena Research Center
†Equal contributions

Abstract

Despite its widespread adoption, Adam's advantage over Stochastic Gradient Descent (SGD) lacks a comprehensive theoretical explanation. This paper investigates Adam's sensitivity to rotations of the parameter space. We demonstrate that Adam's performance in training transformers degrades under random rotations of the parameter space, indicating a crucial sensitivity to the choice of basis. This reveals that conventional rotation-invariant assumptions are insufficient to capture Adam's advantages theoretically. To better understand the rotation-dependent properties that benefit Adam, we also identify structured rotations that preserve its empirical performance. We then examine the rotation-dependent assumptions in the literature, evaluating their adequacy in explaining Adam's behaviour across various rotation types. This work highlights the need for new, rotation-dependent theoretical frameworks to understand Adam's empirical success in modern machine learning fully.

1. INTRODUCTION

Large Language Models (LLMs) have demonstrated remarkable capabilities as their scale grows [3, 20]. However, this unprecedented growth in model scale has led to a proportional increase in the economic [10, 41, 43] and environmental [30, 31] costs associated with their training.

Despite this clear motivation, Adaptive Moment Estimation (Adam) [22] has persisted as the go-to optimizer for language models, with only minor modifications such as AdamW [29] becoming widely adopted since Adam's inception. This success has prompted extensive research to provide theoretical justification for Adam's performance. While the original convergence proof for Adam was later found to be flawed [40], recent studies have proposed rigorous convergence proofs under plausible assumptions [4, 8, 26].

However, these proofs do not elucidate Adam's advantages over SGD when training transformer models [44]. Numerous works attempted to explain Adam's superiority, employing diverse assumptions and analytical frameworks [24, 36, 50, 52]. The heterogeneity of these approaches leads to a lack of consensus on theoretical explanations most accurately capture the fundamental mechanisms underlying Adam's improved performance. For instance, Zhang et al. [48] suggests it stems from enhanced robustness to heavy-tailed noise, while Kunstner et al. [23] argues it plays no role.

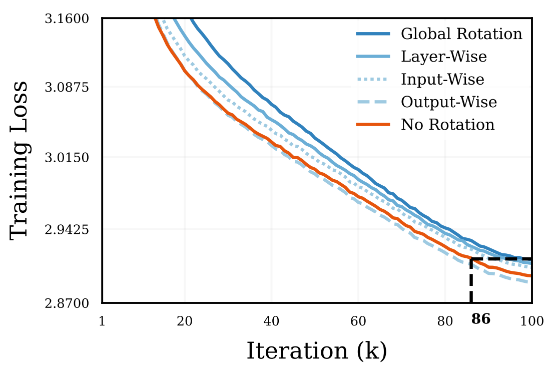

Figure 1: GPT2 (124M)

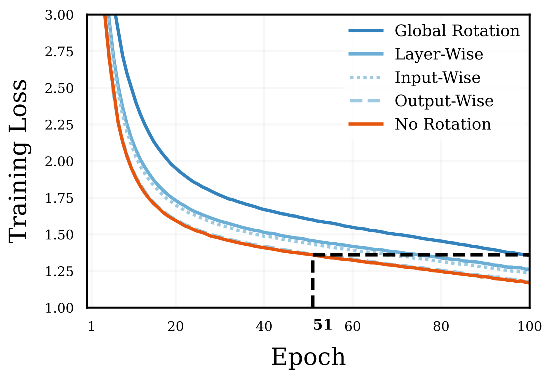

Figure 2: ViT/S (22M)

Figure 3: Adam's performance degrades under certain random rotations of the parameter space, demonstrating its dependence on the standard basis. (a) For GPT2, global rotations lead to a 16% slowdown in training. (b) ViT experiences a more dramatic 96% slowdown under global rotations. Performance is preserved under output-wise rotations but progressively worsens with input-wise, layer-wise, and global rotations, revealing Adam's increasing sensitivity to broader scopes.

This study focuses on a fundamental distinction between Adam and SGD: Adam's dependency on the coordinate system. SGD is rotation-equivariant, i.e., if the loss landscape is rotated, the resulting optimization trajectories from SGD will be the same up to that rotation. In contrast, Adam produces substantially different trajectories. Our experimental investigation reveals that Adam's performance when training transformers empirically degrades when the objective function undergoes random rotations (Figure 3). This result challenges the adequacy of existing theoretical frameworks used to analyze Adam's performance. Indeed, Appendix F shows that most assumptions employed in the literature are rotation-invariant, hence they cannot fully capture Adam's empirical advantages.

To deepen our understanding of the relationship between basis orientation and Adam's performance, we address two fundamental questions:

Q1. How do various types of rotations influence Adam's performance?

We investigate **Q1** by conducting experiments in Section 3, examining Adam's convergence when rotating specific regions of the parameter space. We also identify some rotations that preserve or enhance Adam's performance. These findings provide a more nuanced picture of Adam's adaptive behaviour and which properties of the basis are most consequential. Finally, a few rotation-dependent assumptions do exist in the literature. This naturally raises the question:

Q2. Do existing rotation-dependent assumptions adequately capture Adam's behaviour under rotations?

Section 4 examines three assumptions in this context: L_∞ bounded gradients, Hessian block-diagonality, and L_∞-smoothness [47]. Our analysis reveals that while none of these conditions fully capture Adam's behaviour under rotations, L_∞-smoothness demonstrates some promising characteristics. This highlights the need for more refined theoretical frameworks to model Adam's performance. These findings pave the way for future research to develop more nuanced, rotation-aware assumptions that better align with Adam's empirical behaviour.

We summarize related work in Appendix A.

2. PRELIMINARIES

2.1. Notations and Settings

Let $f : \mathbb{R}^d \to \mathbb{R}$ be the loss of a neural network with d trainable parameters. Stochastic optimization algorithms approximate $\arg\min_{w\in\mathbb{R}^d} f(w)$ by only accessing independent stochastic functions f_B that depend on a stochastic minibatch B following some data distribution $\mathcal{D}$ such that $\forall w \in \mathbb{R}^d, \mathbb{E}_{B\sim\mathcal{D}}[f_B(w)] = f(w)$. Our study examines the optimization process under **rotations of the parameter space**. More formally, let $SO(d)$ be the set of rotation matrices,

$$SO(d) = \left\{\mathbf{R}\in\mathbb{R}^{d\times d}\colon \mathbf{R}^\top\mathbf{R} = \mathbf{R}\mathbf{R}^\top = \mathbf{I}, \det(\mathbf{R}) = 1\right\}. \tag{1}$$

Instead of directly optimizing f, we consider its rotated counterpart $f^{(\mathbf{R})} : w \to f(\mathbf{R}^\top w),\ \mathbf{R} \in SO(d)$. This transformation rotates the coordinate system while preserving the geometry of the optimization landscape. Unless specified otherwise, we use the popular AdamW variant of Adam (see Appendix E).

2.2. Rotational Equivariance of SGD

We say that an optimizer is **rotation equivariant** if its trajectories are equally rotated after a rotation of the parameter space.

Definition 2.1 (Rotational equivariance) *Consider an optimization algorithm $\mathcal{A}$ applied to the function f, generating iterates $w_{t+1} = \mathcal{A}(\{w_i\}_{i=0\ldots t}, f, t)$. we say that the optimization algorithm is* ***rotation equivariant*** *if it satisfies, $\forall \mathbf{R} \in SO(d)$,*

$$\mathbf{R}w_{t+1} = \mathcal{A}(\{\mathbf{R}w_i\}_{i=0\ldots t}, f^{(\mathbf{R})}, t).$$

Proposition 2.2 *Stochastic Gradient Descent with momentum is rotation-equivariant. Proof: See Appendix B.*

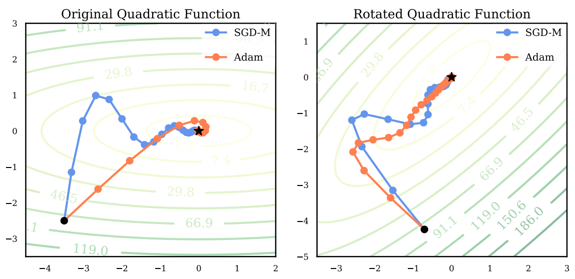

Figure 4: (Left) Trajectories of SGD-momentum and Adam on a quadratic function with eigenvectors aligned with coordinates. (Right) Trajectories on the same function after rotation. Unlike Adam, SGD-momentum exhibits rotation equivariance, maintaining its trajectory relative to the objective function in both scenarios.

In contrast, as illustrated in Figure 4, Adam is not a rotation equivariant. This dependence stems from its element-wise division, see Algorithm 1, step 8 in Appendix E.

2.3. Training Neural Networks in Rotated Parameter Spaces

A crucial aspect of our study is the empirical evaluation of Adam's performance under rotations of the parameter space. Our approach, described in Figure 5, maintains the weights w_t in the standard basis while performing Adam's optimization in the rotated space. This method allows us to leverage existing neural network frameworks while examining Adam's behaviour under rotation.

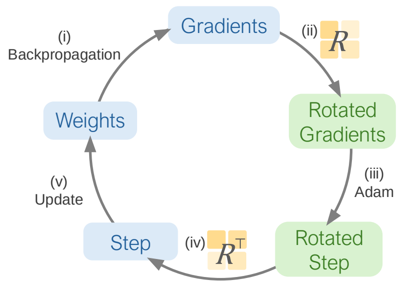

Figure 5: Methodology to train neural networks under rotations of the parameter space. (i) Forward and backward passes in the standard space. (ii) The gradients are rotated with $\mathbf{R}$. (iii) Adam receives the rotated gradients and produces an update $\Delta w^{(\mathbf{R})}$ in the rotated space. (iv) $\Delta w^{(\mathbf{R})}$ is rotated back to the original space using $\mathbf{R}^\top$. (v) The parameters are updated with $\mathbf{R}^\top \Delta w^{(\mathbf{R})}$.

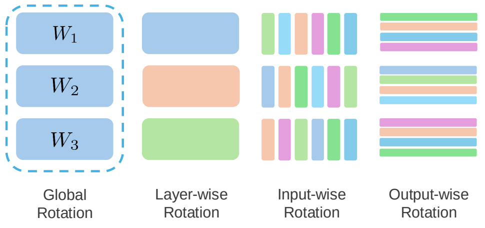

Figure 6: Illustration of different rotation scopes for a model with weights $\mathcal{W} \triangleq \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3\}$. Global rotation rotates the entire parameter space at once, layer-wise only performs rotations within each layer subspace, and input-wise (resp. output-wise) rotates within the weights originating from the same input neuron (resp. leading to the same output neuron).

It is computationally intractable to operate with full $d \times d$ rotation matrices due to the size of modern neural networks. We employ a composite approach that combines block-diagonal rotations with strategic permutations to circumvent this limitation while preserving the essential characteristics of uniformly sampled rotations. This method effectively emulates the statistical properties of full-scale rotations. Details of our sampling process and ablation studies are provided in Appendix C.

3. ROTATIONS' INFLUENCE ON ADAM'S EFFICIENCY

We investigate how random rotations of varying granularity in neural network parameter space, namely *Global, Layer-wise, Output-wise and Input-wise* as shown in Figure 6, affect Adam's performance and explore the relationship between basis orientation and Adam's behavior.

Experimental setting. We conduct experiments across three diverse and representative applications in modern deep learning: *GPT2-OpenWebText*, *ViT-Imagenet*, and *ResNet50-ImageNet*, covering both language and vision tasks as well as Transformer and ResNet architectures. Technical details such as hyperparameters are provided in Appendix D.

Results. We make several key observations. (i) Adam's performance degrades under global rotations across all settings, confirming that the standard basis possesses advantageous properties. (ii) The performance further degrades with broader rotation scopes. Layer-wise rotations, which preserve some basis structure, consistently outperform global rotations, highlighting the importance of local coordinate alignment. (iii) ResNet exhibits minimal performance degradation under rotations. This reduced sensitivity suggests Adam obtains limited benefit from the standard basis structure in

ResNets, explaining its historically inferior performance in training these networks. (iv) Output-wise rotations show no significant degradation across all settings, with GPT2 even slightly improving. This suggests that Adam's adaptivity within output neurons is minimal, supporting recent approaches to reduce redundancy in Adam's second moments [51].

4. ADEQUACY OF EXISTING ASSUMPTIONS

While rotation-invariant assumptions dominate optimization literature, some frameworks incorporate rotation-dependent properties. This section examines whether these existing assumptions adequately capture Adam's rotation dependency: **(i)** it must be realistic in practical settings, and **(ii)** it should align with Adam's performance. Specifically, the assumption should hold (or have favourable constants) under rotations with better performance and break down (or have unfavourable constants) under rotations with worse performance. Results presented in this section are from GPT2-OpenWebText.

4.1. L_∞ bounded gradients

Kingma and Ba [22], Reddi et al. [39] assume a bound on the L_∞ norm of stochastic gradients,

$$\forall w \in \mathbb{R}^d, \ \|\nabla f_B(w)\|_\infty \leq C \quad \text{almost surely.} \tag{2}$$

The constant C depends on the basis as the L_∞ norm is not preserved under rotations. To evaluate this assumption's relevance to Adam's performance, we compute the empirical bound on the rotated gradients: $\tilde{C}(\mathbf{R}) := \max_{B_i} \|\nabla f_{B_i}^{(\mathbf{R})}(w_{\mathbf{R}})\|_\infty$, where $w_{\mathbf{R}}$ denotes the last checkpoint obtained by running Adam under rotation $\mathbf{R}$. The maximum is over 1000 stochastic minibatches B_i. Table 1 reveals that $\tilde{C}$ significantly decreases under random global rotations, predicting better performance for Adam. Hence, L_∞ gradient bound fails to capture the beneficial properties of the basis for Adam.

	No Rotation	Global
$\tilde{C}$	0.883	0.009

Table 1: Empirical L_∞ gradient bound $\tilde{C}$ over 1000 stochastic gradients at the end checkpoint.

4.2. Block-diagonality of the Hessian

A common hypothesis in understanding the behaviour of Adam is that the Hessian can be well approximated by a block-diagonal matrix [50]. This property can indeed be related to the performance of Adam and would be tied to the standard base, as random rotations are likely to break the block-diagonality. Moreover, it would explain why rotating only within appropriate parameter blocks does not degrade performance, as this property would consequently be preserved. We investigate this hypothesis by sampling rows of the Hessian following the procedure described in Appendix D.5.2.

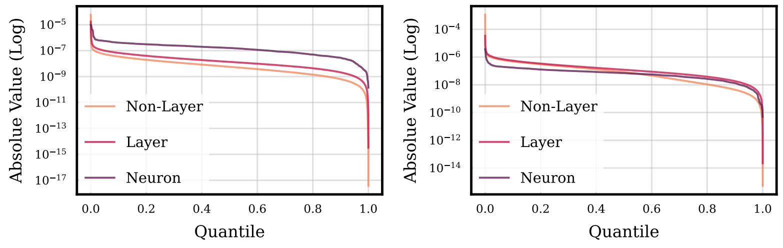

Figure 7: Hessian row estimate of 2nd attention layer. Left: No rotation. Right: Global rotation.

δw direction	Random	Update direction
Neuron	2.86e-05	-4.60e-10
Layer	-8.71e-06	1.30e-08
Non-layer	**1.48e-04**	**2.02e-07**

Table 2: Gradient contribution of layer 2 attention block in random directions (left column) or the update direction (right column).

Although Figure 7 does indicate that the absolute values of the Hessian are of orders of magnitude larger within diagonal blocks, a significant caveat is that the size of these blocks is, in turn, several orders of magnitude smaller than the entire parameter space. Consequently, a natural question is whether the values outside the block can be neglected. In Table 2, we show that while the diagonal blocks contain larger values, their limited size compared to the full parameter space means that off-diagonal elements collectively play a crucial role in shaping the loss landscape's geometry, challenging the strict block-diagonal Hessian assumption in theoretical analyses.

4.3. L_∞-smoothness and $(1,1)$-norm

	No Rotation	Global	Output
$(1,1)$ norm	19.87	25.00	139.60

Table 3: Estimated (1,1)-norm of the Hessian and final accuracy for no rotation, Global and output-wise rotations.

L_∞-smoothness was recently shown to guarantee the convergence of Adam and presented as a potential key property of the basis in [47]. We first remind its definition: A function f is C-smooth wrt. $\|\cdot\|_\infty$ if $\|\nabla f(x) - \nabla f(y)\|_1 \leq C\|x-y\|_\infty \; \forall \, x, y \in \mathbb{R}^d$. Given the challenges in directly estimating the L_∞-smoothness constant, [47] proposed using the (1,1)-norm of the Hessian as a surrogate measure. This norm is defined as $\|H\|_{(1,1)} := \sum_{m=1}^{M} \sum_{n=1}^{N} |H_{mn}|$, where H_{mn} represents the element at the m-th row and n-th column of the Hessian matrix. Notably, they observed a degradation in their estimate of $\|H\|_{(1,1)}$ under global random rotations. However, it remains unclear whether this degradation is a universal phenomenon for all rotations of the parameter space or if it specifically correlates with Adam's performance. We investigate this with the same methodology. Table 3 illustrates the variations in $\|H\|_{(1,1)}$ under no rotation, global, and output-wise rotations. Under global rotations, we confirm the findings of [47] that the $(1,1)$-norm also degrades. Unfortunately, we also observe a degradation in the $(1,1)$-norm for output-wise rotation where the performance is preserved in practice. We conclude although $(1,1)$-norm possesses some potential, it may not fully encapsulate all factors influencing Adam's behaviour in its current form.

5. CONCLUSION AND LIMITATIONS

We investigate Adam's sensitivity to rotations, uncovering critical insights into its optimization dynamics by demonstrating how certain rotations possess advantageous properties. Our study reveals that Adam's performance is intricately tied to basis choice—a relationship current rotation-invariant theoretical frameworks inadequately capture. By highlighting the limitations of existing assumptions, we underscore the need for basis-dependent theoretical tools. While we do not propose a definitive property fully explaining Adam's performance, we provide a detailed blueprint for the missing theoretical framework, outlining key characteristics and offering novel insights into the relationship between Adam and the standard base. These findings are expected to spark new research directions, potentially leading to more robust optimization algorithms and a deeper understanding of fundamental principles in deep learning optimization.

ACKNOWLEDGEMENTS

This research was partially supported by the Canada CIFAR AI Chair program (Mila) and Samsung Electronics Co., Ltd. Simon Lacoste-Julien is a CIFAR Associate Fellow in the Learning in Machines & Brains program. We also acknowledge that this research was enabled in part by computing resources, software, and technical assistance provided by Mila and the Digital Research Alliance of Canada. Ioannis Mitliagkas acknowledges support by an NSERC Discovery grant (RGPIN-2019-06512). We would like to thank Adam Ibrahim for his helpful comments and insights, and Ayoub Echchahed, Pedram Khorsandi, Alan Milligan, Ryan d'Orazio and Vitória Barin Pacela for their valuable feedback.

References

[1] Yossi Arjevani, Yair Carmon, John C. Duchi, Dylan J. Foster, Nathan Srebro, and Blake E. Woodworth. Lower bounds for non-convex stochastic optimization. *Mathematical Programming*, 199:165–214, 2019.

[2] Lucas Beyer, Xiaohua Zhai, and Alexander Kolesnikov. Better plain vit baselines for imagenet-1k. *arXiv:2205.01580*, 2022.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Neurips*, 2020.

[4] Xiangyi Chen, Sijia Liu, Ruoyu Sun, and Mingyi Hong. On the convergence of a class of adam-type algorithms for non-convex optimization. In *ICLR*, 2019.

[5] Michael Crawshaw, Mingrui Liu, Francesco Orabona, Wei Zhang, and Zhenxun Zhuang. Robustness to unbounded smoothness of generalized signsgd. In *Neurips*, 2022.

[6] Mathieu Dagréou, Pierre Ablin, Samuel Vaiter, and Thomas Moreau. How to compute hessian-vector products?, 2024. URL https://iclr-blogposts.github.io/2024/blog/bench-hvp/.

[7] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Neurips*, 2022.

[8] Alexandre Défossez, Leon Bottou, Francis Bach, and Nicolas Usunier. A simple convergence proof of adam and adagrad. *Transactions on Machine Learning Research*, 2022.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[10] Haiwei Dong and Shuang Xie. Large language models (llms): Deployment, tokenomics and sustainability. *arXiv preprint arXiv:2405.17147*, 2024.

[11] Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

[12] Alicia Golden, Samuel Hsia, Fei Sun, Bilge Acun, Basil Hosmer, Yejin Lee, Zachary DeVito, Jeff Johnson, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. Is flash attention stable? *arXiv:2405.02803*, 2024.

[13] Baptiste Goujaud, Adrien Taylor, and Aymeric Dieuleveut. Optimal first-order methods for convex functions with a quadratic upper bound. *arXiv:2205.15033*, 2022.

[14] Charles Guille-Escuret, Baptiste Goujaud, Manuela Girotti, and Ioannis Mitliagkas. A study of condition numbers for first-order optimization. In *AISTATS*, 2020.

[15] Charles Guille-Escuret, Adam Ibrahim, Baptiste Goujaud, and Ioannis Mitliagkas. Gradient descent is optimal under lower restricted secant inequality and upper error bound. In *Neurips*, 2022.

[16] Charles Guille-Escuret, Hiroki Naganuma, Kilian Fatras, and Ioannis Mitliagkas. No wrong turns: The simple geometry of neural networks optimization paths. In *ICML*, 2024.

[17] Zhishuai Guo, Yi Xu, Wotao Yin, Rong Jin, and Tianbao Yang. A novel convergence analysis for algorithms of the adam family and beyond, 2022.

[18] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2015.

[19] Florian Hübler, Junchi YANG, Xiang Li, and Niao He. Parameter-agnostic optimization under relaxed smoothness. In *OPT 2023: Optimization for Machine Learning (Neurips Workshop)*, 2023.

[20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[21] Andrej Karpathy. NanoGPT, 2022. URL https://github.com/karpathy/nanoGPT.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[23] Frederik Kunstner, Jacques Chen, Jonathan Wilder Lavington, and Mark Schmidt. Noise is not the main factor behind the gap between sgd and adam on transformers, but sign descent might be. In *ICLR*, 2023.

[24] Frederik Kunstner, Robin Yadav, Alan Milligan, Mark Schmidt, and Alberto Bietti. Heavy-tailed class imbalance and why adam outperforms gradient descent on language models, 2024.

[25] Haochuan Li, Jian Qian, Yi Tian, Alexander Rakhlin, and Ali Jadbabaie. Convex and non-convex optimization under generalized smoothness. In *Neurips*, 2023.

[26] Haochuan Li, Alexander Rakhlin, and Ali Jadbabaie. Convergence of adam under relaxed assumptions. In *Neurips*, 2023.

[27] Huan Li and Zhouchen Lin. On the $o(\frac{\sqrt{d}}{T^{1/4}})$ convergence rate of rmsprop and its momentum extension measured by ℓ_1 norm, 2024.

[28] Zhaoqi Li, Yu Ma, Catalina Vajiac, and Yunkai Zhang. Exploration of numerical precision in deep neural networks, 2018.

[29] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR 2019,*, 2019.

[30] Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Estimating the carbon footprint of bloom, a 176b parameter language model. *Journal of Machine Learning Research*, 24(253):1–15, 2023.

[31] Sasha Luccioni, Victor Schmidt, Alexandre Lacoste, and Thomas Dandres. Quantifying the carbon emissions of machine learning. In *NeurIPS 2019 Workshop on Tackling Climate Change with Machine Learning*, 2019.

[32] Zhi-Quan Luo and Paul Tseng. Error bounds and convergence analysis of feasible descent methods: a general approach. *Annals of Operations Research*, 46(1):157–178, 1993.

[33] Francesco Mezzadri. How to generate random matrices from the classical compact groups. *Notices of the American Mathematical Society*, 54:592 – 604, 2007.

[34] Yurii Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, 269:543–547, 1983.

[35] Maris Ozols. How to generate a random unitary matrix. In *technical report*, 2009.

[36] Yan Pan and Yuanzhi Li. Toward understanding why adam converges faster than SGD for transformers. In *OPT 2022: Optimization for Machine Learning (NeurIPS Workshop)*, 2022.

[37] Boris T. Polyak. Gradient methods for the minimisation of functionals. *USSR Computational Mathematics and Mathematical Physics*, 3(4):864 – 878, 1963.

[38] B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[39] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *ICLR*, 2018.

[40] David Martınez Rubio. Convergence analysis of an adaptive method of gradient descent. *University of Oxford, Oxford, M. Sc. thesis*, 2017.

[41] Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training nlp models: A concise overview. *arXiv preprint arXiv:2004.08900*, 2020.

[42] Yuxin Sun, Dong Lao, Ganesh Sundaramoorthi, and Anthony Yezzi. Surprising instabilities in training deep networks and a theoretical analysis. In *Neurips*, 2022.

[43] Gaël Varoquaux, Alexandra Sasha Luccioni, and Meredith Whittaker. Hype, sustainability, and the price of the bigger-is-better paradigm in ai. *arXiv:2409.14160*, 2024.

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neurips*, 2017.

[45] Bohan Wang, Jingwen Fu, Huishuai Zhang, Nanning Zheng, and Wei Chen. Closing the gap between the upper bound and lower bound of adam's iteration complexity. In *Neurips*, 2023.

[46] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *Neurips*, 2018.

[47] Shuo Xie, Mohamad Amin Mohamadi, and Zhiyuan Li. Adam exploits ℓ_∞-geometry of loss landscape via coordinate-wise adaptivity. In *High-dimensional Learning Dynamics: The Emergence of Structure and Reasoning (ICML Workshop)*, 2024.

[48] Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank Reddi, Sanjiv Kumar, and Suvrit Sra. Why are adaptive methods good for attention models? In *Neurips*, 2020.

[49] Yushun Zhang, Congliang Chen, Naichen Shi, Ruoyu Sun, and Zhi-Quan Luo. Adam can converge without any modification on update rules. In *Neurips*, 2022.

[50] Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhi-Quan Luo. Why transformers need adam: A hessian perspective, 2024.

[51] Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. Adam-mini: Use fewer learning rates to gain more, 2024.

[52] Dongruo Zhou, Jinghui Chen, Yuan Cao, Ziyan Yang, and Quanquan Gu. On the convergence of adaptive gradient methods for nonconvex optimization. *Transactions on Machine Learning Research*, 2024. Featured Certification.

[53] Fangyu Zou, Li Shen, Zequn Jie, Weizhong Zhang, and Wei Liu. A sufficient condition for convergences of adam and rmsprop. In *CVPR*, June 2019.

Appendix A. RELATED WORK

Adam under rotations. We consider the concurrent work [47] to be the closest related study, showing that Adam converges more slowly with a randomly rotated loss landscape. They provide convergence analysis based on L_∞ geometry, demonstrating that this yields a better empirical smoothness constant for GPT-2 models. While their work offers valuable theoretical insights, our study takes a more experimental stance. We aim to paint a comprehensive picture of Adam's behavior under a spectrum of rotations, from random to structured transformations, and evaluate how existing rotation invariant assumptions correlate with Adam performance.

Understanding Adam. Our work casts light on the critical interactions between Adam and the coordinate system, contributing to a growing body of research on Adam's behavior and convergence. Recent works have attributed Adam's success to the heterogeneous block-diagonal structure of its Hessian [50], though we find this assumption to be unrealistic. Others have improved convergence guarantees: [8] and [17] offered simplified and novel derivations, [49] argued that vanilla Adam converges without modification, [52] provided a general convergence analysis for adaptive methods in non-convex settings, and [26] proposed a convergence proof for Adam without relying on globally bounded gradients. [25] developed a convergence analysis based on generalized smoothness conditions, and [19] proposed parameter-agnostic convergence results under these relaxed conditions. Finally, lower bounds for non-convex optimization were established by [1], with [45] addressing the gap between upper and lower bounds for Adam's iteration complexity.

Adam's advantages over SGD. Prior works have also attempted to justify Adam's advantages over SGD. [48, 52] suggest SGD suffers more from heavy-tailed noise, with Adam converging faster when gradients are sparse. However, [23] found noise reduction through larger batch sizes benefits Adam but not SGD. Additionally, [24] ties Adam's advantage over SGD in language models to heavy-tailed class imbalance, and [36] to the concept of directional sharpness.

Appendix B. PROOFS

Proof Let A_{SGD} denote SGD[1] with momentum, defined by the following update rule:

$$A_{SGD}(\{w_i\}_{i=0,\dots,t}, f, t, B_t, \eta) = w_t - \eta \nabla f_{B_t}(w_t)$$

Where $w_t \in \mathbb{R}^d$ denotes the iterate at step t, f is the objective function, B_t the minibatch sampled at step t, and $\eta > 0$ the learning rate.

Let $(w_t)_{t\geq 0}$ be a sequence of iterates obtained by applying $\mathcal{A}_{SGD-M}$ recursively, starting in some $w_0 \in \mathbb{R}^d$ with some fixed η, , $(B_t)_{t\geq 0}$, and f.

From the chain rule, we have $\nabla f^{(R)}_{B_t}(w_t) = R\nabla f_{B_t}(R^\top w_t)$, hence

$$\begin{aligned}
\mathcal{A}(\{Rw_i\}_{i=0,\dots,t}, f^{(R)}, t, B_t, \eta) &= Rw_t - \eta \nabla f^{(R)}_{B_t}(Rw_t) \\
&= Rw_t - \eta R \nabla f_{B_t}(R^\top R w_t) \\
&= Rw_t - \eta R \nabla f_{B_t}(w_t) \\
&= R\mathcal{A}(\{w_i\}_{i=0,\dots,t}, f, t, B_t, \eta) = Rw_{t+1}.
\end{aligned}$$

1. For the sake of simplicity we are not considering momentum here, but the results similarly hold for heavy ball [38] and Nesterov Accelerated Gradient [34].

■

Appendix C. SAMPLING RANDOM ROTATIONS IN HIGH DIMENSION

This section explains our method of sampling random rotations for high-dimensional spaces and the implementation details.

C.1. High-Dimensional Rotations

Even small modern machine learning models typically have millions of parameters. Consequently, storing a $d \times d$ rotation matrix is often intractable, let alone performing the dot product required to rotate the gradient vector. To address this issue, we sample a $n \times n$ rotation matrix $\mathbf{R}_n$ with $n \ll d$ uniformly (in the sense of the Haar measure) from the special orthogonal group $SO(n)$, and a random permutation π of $0, \dots, d-1$. For now, we assume $\frac{d}{n} \in \mathbb{N}$, see appendix C.3 for a general case. To rotate a gradient g, we compute:

$$\begin{aligned} g^{(\mathbf{R}_n, \pi)} &:= \pi^{-1} \circ \left(\left[\bigoplus_{i=1}^{d/n} R_n \right] (\pi \circ g) \right), & (3) \\ &= \pi^{-1} \circ \begin{bmatrix} R_n & & & & \\ & R_n & & 0 & \\ & & R_n & & \\ & 0 & & \ddots & \\ & & & & R_n \end{bmatrix} (\pi \circ g), & (4) \end{aligned}$$

where $\bigoplus$ denotes the direct sum operation, producing a block-diagonal matrix with d/n blocks R_n. This procedure effectively computes a rotation by blocks of size n picked from a random partition of indices, constituting a valid rotation.

Intuitively, if n is sufficiently large, we expect this procedure to approximate well the effect of random rotations sampled uniformly from $SO(d)$, due to the law of large numbers homogenizing geometric properties across coordinates. To confirm this intuition, we perform an ablation study in Figure 8, finding that the impact on Adam's performance saturates well below our operational values.

Our approximation reduces the memory cost from $O(d^2)$ to $O(n^2 + d)$, and the computational cost from $O(d^2)$ to $O(nd)$. Since batch matrix multiplications required for the rotation can be performed efficiently on modern GPUs, the final overhead of applying rotations is extremely small.

C.2. Reflections and Sampling From The Haar Measure

To sample R_n uniformly from $SO(n)$ with respect to the Haar measure, we employ the QR decomposition trick [33, 35], which samples from the Haar measure μ of the orthogonal group $O(n)$. Let us consider the projection $\pi : O(n) \to SO(n)$, such that $\pi(\mathbf{R})$ is $\mathbf{R}$ when $\mathbf{R} \in SO(n)$, and $\pi(\mathbf{R})$ simply multiplies the first column of $\mathbf{R}$ by -1 when $\mathbf{R} \in O(n) \setminus SO(n)$. The push forward of μ by π is the Haar measure on $SO(n)$. Since Adam is reflection equivariant, rotating with $\pi(\mathbf{R})$ and with $\mathbf{R}$ will lead to identical performance for any $\mathbf{R} \in O(n)$. Thus we can omit to apply π, and simply sample from μ using the QR decomposition method.

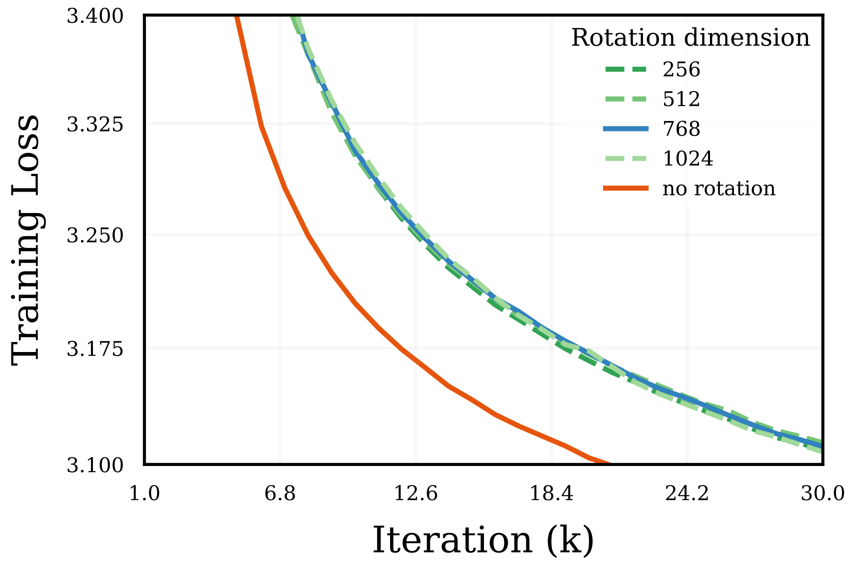

Figure 8: Training loss of GPT-2 when training with different rotation dimension n. The loss of performance is consistent across n at our range.

Similarly, Adam is permutation equivariant, thus we omit to apply the inverse permutation before providing the rotated gradients to Adam, and to apply the permutation before rotating the update, as removing these two steps do not affect performances.

C.3. Rotation Residual

Based on the type of rotation and the chosen dimension n, the number of blocks may not divide evenly, i.e., $\frac{d}{n} \notin \mathbb{N}$. To address this issue, we introduce an additional rotation matrix, which we refer to as the *residual* matrix, to complete the missing dimensions. More formally, let d represent the dimensionality of the parameter space, and let n denote the block dimensions of the rotation. We define $b \triangleq \lfloor \frac{d}{n} \rfloor$ as the number of complete blocks. The **residual** matrix $\mathcal{R}$ is then sampled from $SO(p)$, where $p \triangleq d - nb$. Therefore, eq. (3) becomes

$$g^{(\mathbf{R}_n, \mathcal{R}, \pi)} := \pi^{-1} \circ \left([B \oplus \mathcal{R}] \, (\pi \circ g) \right), \tag{5}$$

$$\tag{6}$$

$$= \pi^{-1} \circ \begin{bmatrix} B & \\ & \mathcal{R} \end{bmatrix} (\pi \circ g) \,, \tag{7}$$

$$= \pi^{-1} \circ \begin{bmatrix} R_n & & & & \\ & R_n & & 0 & \\ & & \ddots & & \\ & 0 & & R_n & \\ & & & & \mathcal{R} \end{bmatrix} (\pi \circ g) \,. \tag{8}$$

where $B = \bigoplus_{i=1}^{b} R_n$.

C.4. Overall Validation and Impact of Flash Attention

Given the sensitivity of neural network training to numerical precision [28, 42, 46], it is crucial to ensure that rounding errors from applying rotations to the gradient do not significantly confound the impact of the change of basis. In particular:

- We apply rotations in single precision.
- We refrain from using FlashAttention [7], which was found to increase numeric deviations [12].

In Figure 9, we present the training loss when training GPT-2 with SGD without rotations, with global random rotations using flash attention, and with global random rotations without flash attention. In particular, we confirm two important observations:

- Without flash attention (the setting we use for our experiments) the performances of SGD under global random rotation and under no rotations are identical. This validates that our experimental setting is behaving as expected.
- When we use flash attention with rotations, we observe a slight difference in performance. This is due to flash attention amplifying numerical errors from the application of the rotation. Interestingly, likely due to a slight regularization effect, it slows down performance at first but actually provides a small improvement to the loss at the end of training.

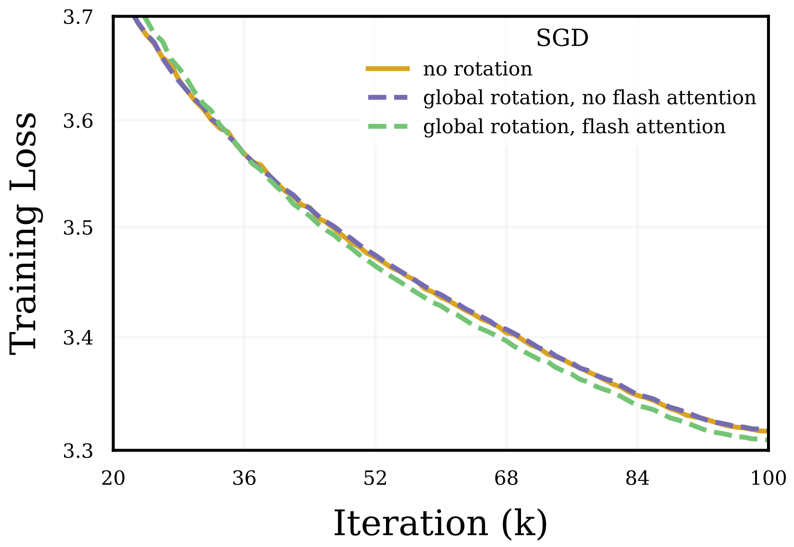

Figure 9: SGD performance when applying global random rotations, with and without flash attention.

Appendix D. EXPERIMENTAL DETAILS AND ADDITIONAL RESULTS

D.1. Rotation implementation details

- **Global Rotation:** We apply a single rotation matrix $\mathbf{R}$ to the entire parameter space, enabling us to assess Adam's sensitivity to global changes in the coordinate system. This involves applying $\mathbf{R}$ to the concatenated vector of all weights, represented as $\text{vec}(\mathbf{W}_1 \mid \mathbf{W}_2 \mid \mathbf{W}_3)$, as shown in Figure 5.

- **Layer-wise Rotation:** We partition the parameter space by layers, applying the same rotation matrix $\mathbf{R}$ independently to each layer. This preserves the layer structure of the basis by preventing the shuffling of coordinates corresponding to different layers. Specifically, we rotate each layer's weights represented as $\text{vec}(\mathbf{W}_i)$ for $1 \leq i \leq 3$.
- **Neuron-wise Rotation:** This method is similar to layer-wise rotation but focuses on partitioning weights at the neuron level. We explore two variants:
  1. **Output-wise:** In this case, we rotate weights $\mathbf{W}_{ij}$ that lead to the same output neuron. The rotation matrix $\mathbf{R}$ is applied to the output vectors $\mathbf{W}_{ij}^\top$.
  2. **Input-wise:** This approach rotates weights $\mathbf{W}_{ij}$ that originate from the same input neuron, applying $\mathbf{R}$ to the input vectors $\mathbf{W}_{ij}$.

Rotation in Transformers. By default, many implementations store the query, key, and value parameters within a single linear layer. Thus, we split them to treat them as separate layers, reflecting the fundamental differences in how their parameters are involved in forward computations. Additionally, PyTorch stores parameters as tensors in the shape `(output_dim, input_dim)`, but embeddings are stored as lookup tables in the shape `(input_dim, output_dim)`. For output neuron and input neuron rotations to behave intuitively, we thus transpose embedding layers before and after rotations.

D.2. GPT2-OpenWebText

Language Modeling: We trained a GPT-2 model with 124M parameters on the OpenWebText dataset [11] using a configuration designed for efficient pretraining. The model architecture includes 12 layers, 12 attention heads, and a 768-dimensional embedding space, with no bias in LayerNorm or Linear layers. We employed the AdamW optimizer with a peak learning rate of 6e-4, $\beta_1 = 0.9$, $\beta_2 = 0.95$, and a weight decay of 1e-1, applying gradient clipping of 1.0. Training ran for 100,000 iterations (or 30,000 for some smaller ablations), with learning cosine rate decay starting after a 2,000-iteration warm-up, decaying to a minimum of 6e-5. We used batch size of 12 with gradient accumulation steps simulating an effective batch size of 40. All experiments were performed on four A100 80GB GPUs, leveraging mixed precision. Unless otherwise specified, all optimizer hyperparameters were shared across experiments and set to the default values specified in [21].

D.3. ViT/S - ImageNet

Image Classification (Transformer): We trained a Vision Transformer (ViT) model on the ImageNet-1K dataset [9] using the SimpleViT architecture [2]. The model consists of 12 layers, 6 attention heads, a hidden dimension of 384, and an MLP dimension of 1536, with a patch size of 16 and input image size of 224. The AdamW optimizer was employed with a learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e - 8$, and a weight decay of 0.1. We used a cosine learning rate schedule with 5 warm-up epochs. The training was conducted for 100 epochs with a batch size of 1024. All experiments were performed with mixed precision.

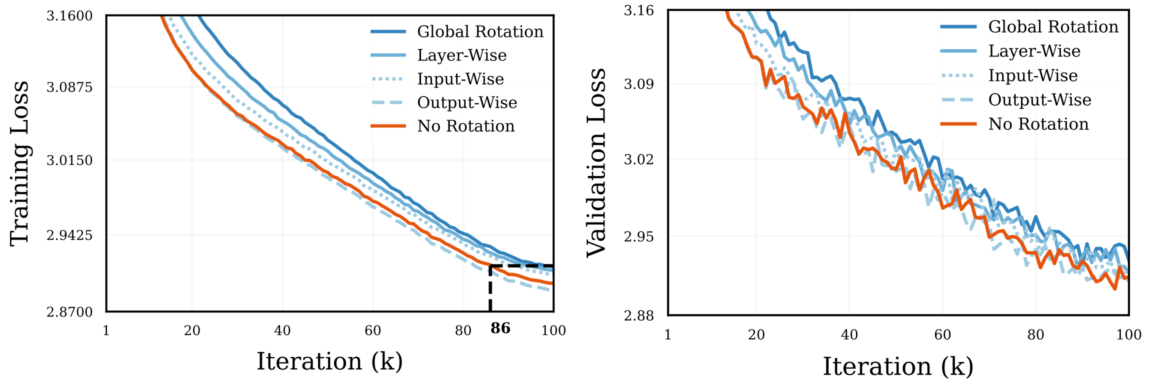

Figure 10: GPT2 (124M) - OpenWebText training loss and validation loss

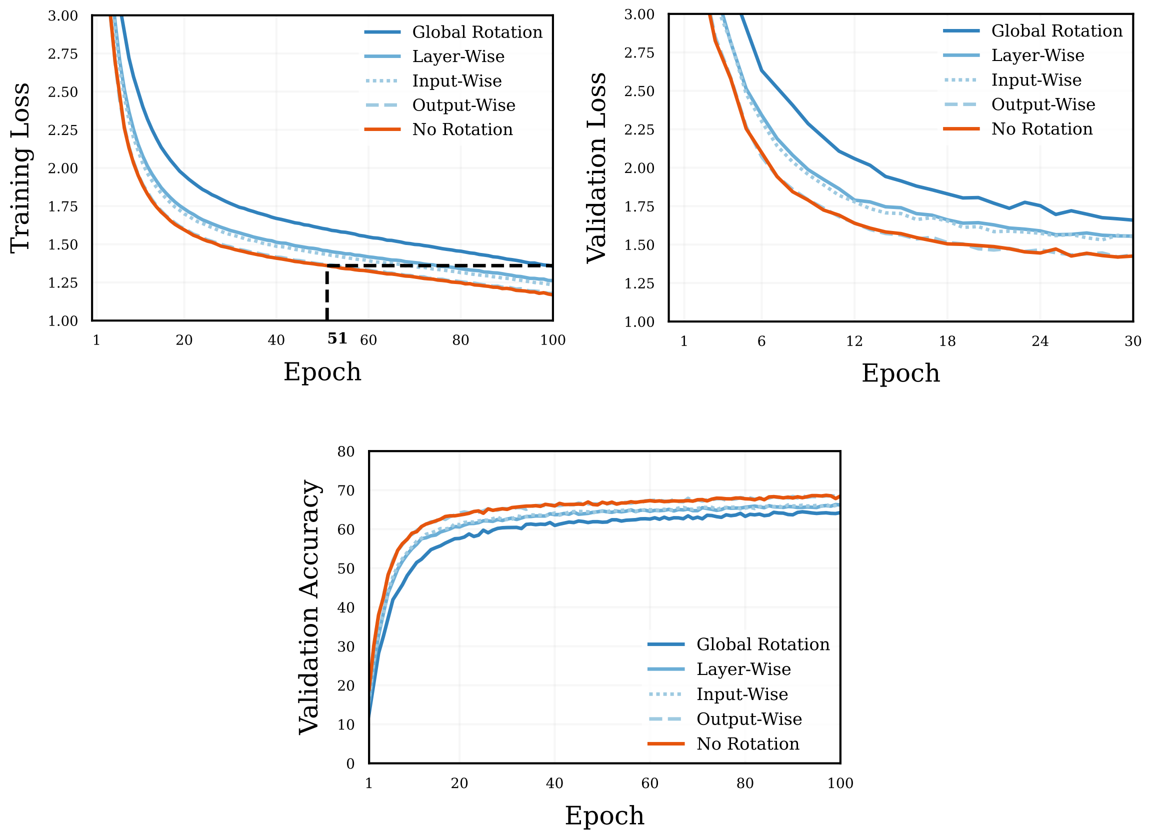

Figure 11: SimpleViT - Imagenet training loss, validation loss and top-1 validation accuracy

D.4. Resnet50 - ImageNet

Image Classification (ResNet): We trained a ResNet-50 model [18] on the ImageNet-1K dataset [9] using the AdamW optimizer. The optimizer was configured with a learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e - 8$, and a weight decay of 0.0001. We employed a cosine learning rate schedule with 5 warm-up epochs. The training ran for 100 epochs with a batch size of 256.

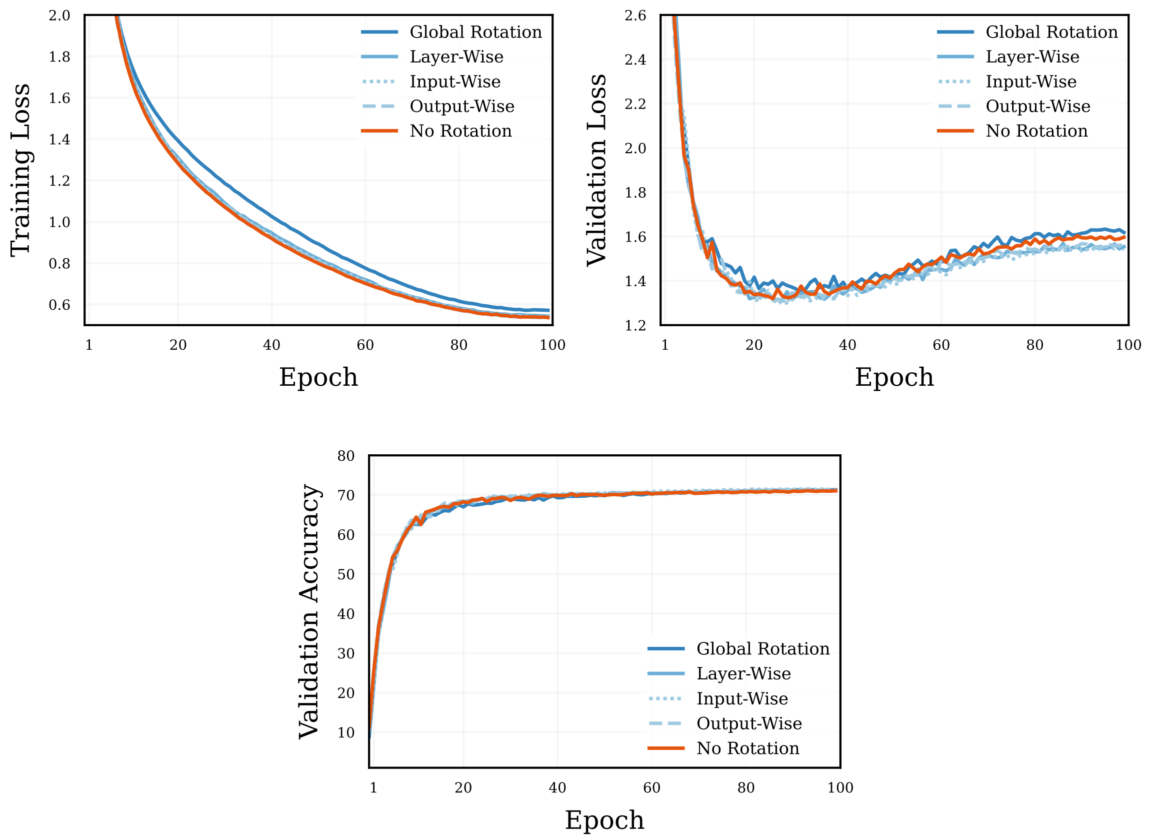

Figure 12: ResNet50 - Imagenet training loss, validation loss and top-1 validation accuracy

D.5. Empirically Testing Existing Assumptions

D.5.1. l_∞-BOUNDED GRADIENT

Using the most recent saved checkpoint, we estimate the gradient bound C for GPT-2 (124M). To do this, we track the maximum absolute value encountered in the gradient over a fixed number of iterations (in our case, 1000).

D.5.2. HESSIAN ROW SAMPLING

Consider a neural network f with parameters $w \in \mathbb{R}^d$ and Hessian matrix $\mathbf{H} \in \mathbb{R}^{d \times d}$ at w. For large-scale models like GPT2 (125M parameters), computing the full Hessian is computationally

```
Algorithm 2: Gradient Bound Estimation
Input: $T$: total number of iterations (1000), Saved checkpoint: $\theta$
1 Initialize $C \leftarrow 0$;
2 for $t \leftarrow 1$ to $T$ do
3     Compute gradient $g_t \leftarrow \nabla_\theta f_t(\theta)$;
4     Compute $C' \leftarrow \|g_t\|_\infty$;
5     Update $C \leftarrow \max(C, C')$;
6 end
7 return $C$ ;                      // Return the estimated gradient bound
```

intractable. We estimate it by sampling rows using the Hessian Vector Product (HVP) method with a one-hot vector v with $v_i = 1$ for row i, implementing the approach from [6]:

$$\nabla^2 f(\theta)v = \lim_{\epsilon \to 0} \frac{1}{\epsilon}[\nabla f(\theta + \epsilon v) - \nabla f(\theta)] = \nabla[\langle \nabla f(\cdot), v \rangle](\theta)$$

We sampled 8 rows per training checkpoint, uniformly distributed across the network to maximize diversity. The sampled rows for each run were as follows:

Layer	Computation Step	Row ID
Transformer Layer 2 (Attention Block)	Attention Computation	53,970,691
Transformer Layer 2 (Attention Block)	Attention Projection	55,680,281
Transformer Layer 2 (MLP Block)	Feedforward Computation	57,847,212
Transformer Layer 2 (MLP Block)	Feedforward Projection	58,790,300
Transformer Layer 8 (Attention Block)	Attention Computation	96,585,666
Transformer Layer 8 (Attention Block)	Attention Projection	97,929,821
Transformer Layer 8 (MLP Block)	Feedforward Computation	99,845,651
Transformer Layer 8 (MLP Block)	Feedforward Projection	102,950,588

Table 4: Hessian sampled from GPT-2: Transformer Layers, Computation Step, and corresponding Row ID.

Our objective in analyzing gradient contributions is to identify which components of the Hessian matrix contribute most significantly to the gradient. Let H denote the Hessian matrix of GPT-2, and let H_i represent the i-th row of this matrix. We categorize the parameters associated with H_i into three distinct groups:

Neuron parameters: The parameters that directly influence the i-th neuron of GPT-2, denoted as θ_{neuron}.

Layer parameters: The parameters of the layer containing the i-th parameters, excluding those of the neuron, denoted as θ_{layer}.

Non-Layer parameters:: The parameters that do not belong to the aforementioned categories, denoted as $\theta_{\text{non-layer}}$.

To compute the gradient contribution from each of these parameter groups, we first define the training direction in the parameter space as $d = w_{t+1} - w_t$. We then decompose d into contributions from the three groups mentioned above, as well as apply the same decomposition to the Hessian row H_i:

$$d = [d_{\text{neuron}}, d_{\text{layer}}, d_{\text{non-layer}}]$$
$$H_i = [H_{i,\text{neuron}}, H_{i,\text{layer}}, H_{i,\text{non-layer}}]$$

Next, we compute the inner product of d with the Hessian row H_i for each group respectively:

$$C_{\text{neuron}} = \langle d_{\text{neuron}}, H_{i,\text{neuron}} \rangle$$
$$C_{\text{layer}} = \langle d_{\text{layer}}, H_{i,\text{layer}} \rangle$$
$$C_{\text{non-layer}} = \langle d_{\text{non-layer}}, H_{i,\text{non-layer}} \rangle$$

For a comprehensive evaluation, we also compute the total contribution of the i-th row to the gradient by considering the inner product between the entire Hessian row and the direction d:

$$C_{\text{total}} = \langle d, H_i \rangle$$

This allows for a fair comparison of the contributions from each parameter group to the overall gradient.

D.5.3. HESSIAN BLOCK-DIAGONALITY

Row ID	No Rotation				Global Rotation			
	Neuron	Layer	Non-layer	Total	Neuron	Layer	Non-layer	Total
53970691	-4.60e-10	1.30e-08	**2.02e-07**	2.15e-07	-1.92e-10	4.78e-07	**2.36e-05**	2.40e-05
55680281	-9.42e-08	-6.45e-08	**2.80e-06**	2.64e-06	1.11e-09	-1.07e-07	**-7.18e-06**	-7.28e-06
57847212	2.64e-09	1.28e-07	**6.53e-06**	6.66e-06	1.42e-10	-1.07e-06	**-4.11e-06**	-5.19e-06
58790300	-8.08e-07	-2.24e-06	**-2.33e-06**	-5.38e-06	1.76e-08	-2.43e-06	**2.82e-06**	4.14e-07
96585666	7.46e-11	-5.03e-08	**-2.16e-07**	2.67e-07	1.01e-11	-2.66e-08	**-5.89e-06**	-5.91e-06
97929821	5.72e-09	1.84e-08	**4.47e-06**	4.49e-06	1.15e-09	-1.96e-08	**-2.58e-06**	-2.60e-06
99845651	-5.03e-08	-5.65e-07	**-2.15e-06**	-2.77e-06	2.77e-10	-1.49e-07	**-2.31e-05**	-2.33e-05
102950588	-7.17e-08	-4.42e-07	**-5.09e-06**	-5.61e-06	1.64e-09	-2.38e-08	**1.24e-05**	1.24e-05

Table 5: Expected gradient contribution for sampled weights for no rotation and global rotation. Observe that the non-layer gradients consistently contribute magnitudes larger than gradients in layer and neuron, and is often similar to the total contribution, showing the limitation of attributing most of the contribution to elements within the block-diagonal structure of the Hessian.

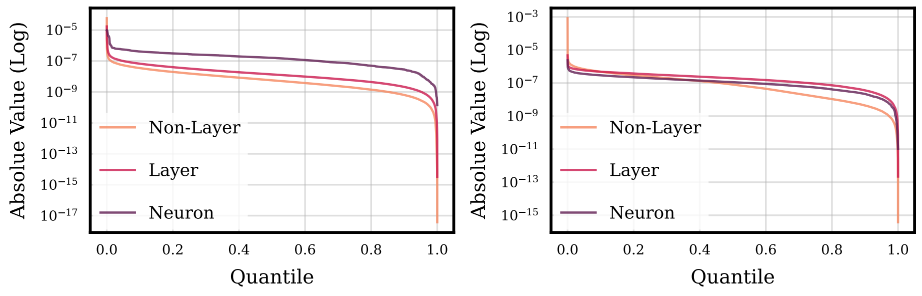

Figure 13: Hessian row estimate of layer 2 Attention block attention computation (Row 53,970,691). Left: No rotation. Right: Global rotation.

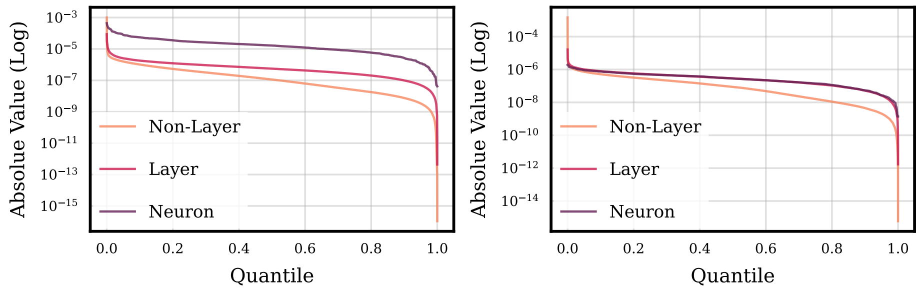

Figure 14: Hessian row estimate of layer 2 Attention block attention projection (Row 55,680,281). Left: No rotation. Right: Global rotation.

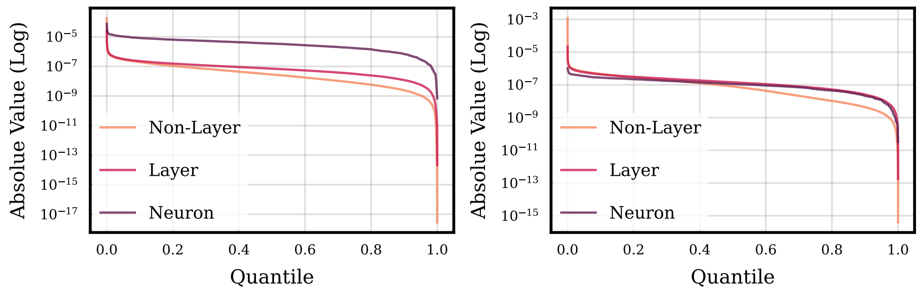

Figure 15: Hessian row estimate of layer 2 MLP block feedforward computation (Row 57,847,212). Left: No rotation. Right: Global rotation.

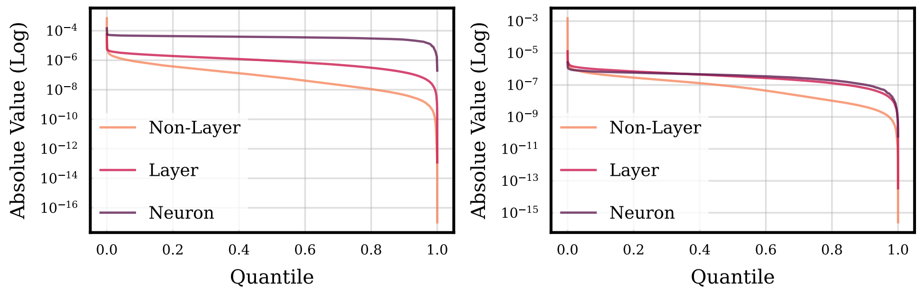

Figure 16: Hessian row estimate of layer 2 MLP block feedforward projection (Row 58,790,300). Left: No rotation. Right: Global rotation.

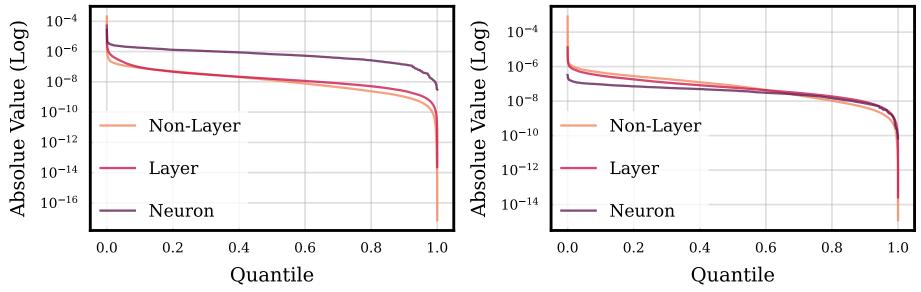

Figure 17: Hessian row estimate of layer 8 Attention block attention computation (Row 96,585,666). Left: No rotation. Right: Global rotation.

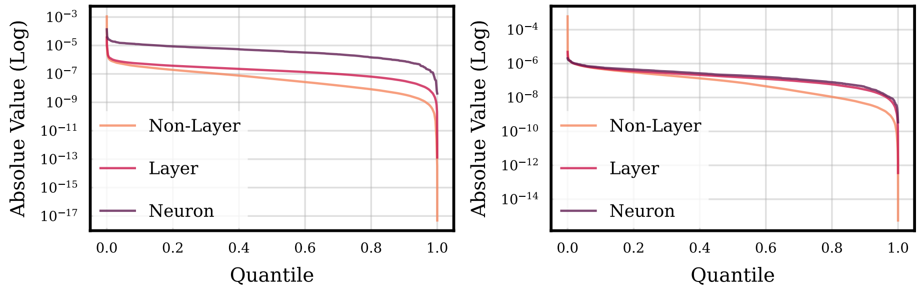

Figure 18: Hessian row estimate of layer 8 Attention block attention projection (Row 97,929,821). Left: No rotation. Right: Global rotation.

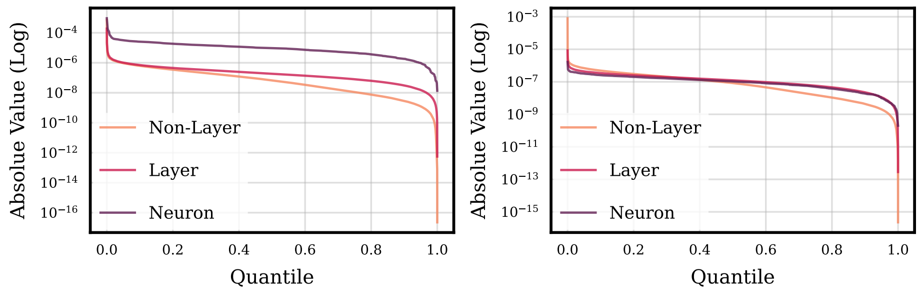

Figure 19: Hessian row estimate of layer 8 MLP block feedforward computation (Row 99,845,651). Left: No rotation. Right: Global rotation.

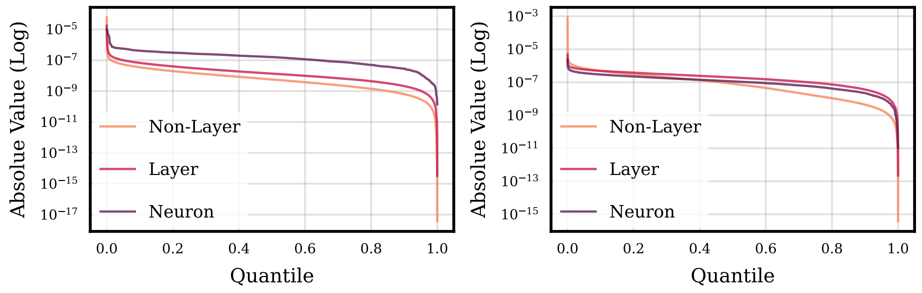

Figure 20: Hessian row estimate of layer 8 MLP block feedforward projection (Row 102,950,588). Left: No rotation. Right: Global rotation.

From Figures 13 to 20, we consistently observe a distinct structure in non-rotated Hessian rows. Specifically, for a given weight corresponding to a particular row index, the largest values tend to appear in the neighbouring parameters associated with the neurons containing that weight. When these neurons are removed, the highest values in the Hessian row shift to the layer containing the weight. This suggests a potential block-diagonal structure in the Hessian of Transformer architectures. However, this does not fully explain Adam's advantage, as gradients outside the layer contribute substantially to the change direction and cannot be overlooked.

Appendix E. ADAMW ALGORITHM

We remind here the AdamW algorithm (pseudocode) in Algorithm 1, and provide a rotated version in Algorithm 2.

Algorithm 1 AdamW Optimization Algorithm

```
Require: $\alpha$: stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: exponential decay rates for moment estimates
Require: $\lambda$: weight decay coefficient
Require: $\epsilon$: small constant for numerical stability
Require: $f(\theta)$: stochastic objective function with parameters $\theta$
 1: Initialize $\theta_0$, $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$ while $\theta_t$ not converged do
 2:
        end
        $t \leftarrow t + 1$
 3: $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$    ▷ Get gradients w.r.t. stochastic objective at timestep $t$
 4: $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$    ▷ Update biased first moment estimate
 5: $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$    ▷ Update biased second raw moment estimate
 6: $\hat{m}_t \leftarrow m_t/(1 - \beta_1^t)$    ▷ Compute bias-corrected first moment estimate
 7: $\hat{v}_t \leftarrow v_t/(1 - \beta_2^t)$    ▷ Compute bias-corrected second raw moment estimate
 8: $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon) - \alpha \cdot \lambda \cdot \theta_{t-1}$    ▷ Update parameters
 9: return $\theta_t$    ▷ Return the final parameters
```

Algorithm 2 AdamW Optimization Algorithm with Rotation

```
Require: $\alpha$: stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: exponential decay rates for moment estimates
Require: $\lambda$: weight decay coefficient
Require: $\epsilon$: small constant for numerical stability
Require: $f(\theta)$: stochastic objective function with parameters $\theta$
 1: Initialize $\theta_0$, $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$ while $\theta_t$ not converged do
 2:
        end
        $t \leftarrow t + 1$
 3: $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$    ▷ Get gradients w.r.t. stochastic objective at timestep $t$
 4: $\tilde{g}_t = R \cdot g_t$    ▷ Apply rotation to gradients
 5: $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \tilde{g}_t$    ▷ Update biased first moment estimate
 6: $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot \tilde{g}_t{}_t^2$    ▷ Update biased second raw moment estimate
 7: $\hat{m}_t \leftarrow m_t/(1 - \beta_1^t)$    ▷ Compute bias-corrected first moment estimate
 8: $\hat{v}_t \leftarrow v_t/(1 - \beta_2^t)$    ▷ Compute bias-corrected second raw moment estimate
 9: $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot R^{-1} \cdot (\hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)) - \alpha \cdot \lambda \cdot \theta_{t-1}$    ▷ Update parameters
10: return $\theta_t$    ▷ Return the final parameters
```

Appendix F. COMMON ASSUMPTIONS IN FIRST-ORDER OPTIMIZATION THEORY

We present a non-exhaustive summary of common assumptions used in theoretical works for first-order optimization, see Table 6. For each assumption, we indicate whether it is rotation invariant.

Assumption	**Rotation-Invariant**
(Strong-) Convexity	✓
Polyak-Lojasiewicz [37]	✓
Star-(Strong)-Convexity [14]	✓
Quadratic Growth [13]	✓
L-Smoothness (L_2 norm) [8, 52]	✓
Gradient Growth Condition [49]	✓
Bounded Expected Gradient Squared Norm [53]	✓
(L_0, L_1)-Smoothness [26]	✓
Restricted Secant Inequality [15]	✓
Error Bound [16, 32]	✓
L-smoothness (L_∞ norm)[17]	✗
Coordinate-wise (L_0, L_1)-Smoothness [5]	✗
Coordinate-wise “Affine” Variance Noise [27]	✗
Bounded Gradient (L_∞) [39]	✗

Table 6: Common assumptions involved in first-order optimization algorithm, indicating whether they are rotation-invariant. Rotation-dependent assumptions are comparatively rare in the literature.