Why Transformers Need Adam: A Hessian Perspective

Yushun Zhang [1 2] **Congliang Chen** [1 2] **Tian Ding** [2] **Ziniu Li** [1 2] **Ruoyu Sun** [1 2] **Zhi-Quan Luo** [1 2]

Abstract

SGD performs worse than Adam by a significant margin on Transformers, but the reason remains unclear. In this work, we provide an explanation through the lens of Hessian: (i) Transformers are "heterogeneous": the Hessian spectrum across parameter blocks vary dramatically, a phenomenon we call "block heterogeneity"; (ii) Heterogeneity hampers SGD: SGD performs worse than Adam on problems with block heterogeneity. To validate (i) and (ii), we check various Transformers, CNNs, MLPs, and quadratic problems, and find that SGD can perform on par with Adam on problems without block heterogeneity, but performs worse than Adam when the heterogeneity exists. Our initial theoretical analysis indicates that SGD performs worse because it applies one single learning rate to all blocks, which cannot handle the heterogeneity among blocks. This limitation could be ameliorated if we use coordinate-wise learning rates, as designed in Adam. Our code is available at https://github.com/zyushun/hessian-spectrum.

1. Introduction

Transformers (Vaswani et al., 2017) have become a major workhorse behind AI development (e.g., (Achiam et al., 2023)). However, the understanding of Transformer training remains limited. For instance, Transformer training largely relies on the Adam optimizer (Kingma & Ba, 2014; Loshchilov & Hutter, 2017). In contrast, stochastic gradient descent with momentum (SGD) (We introduce the update rules of Adam(W) and SGD in Appendix E.1), the de-facto optimizer for convolution neural networks (CNNs) (LeCun et al., 1998), performs significantly worse than Adam on Transformers (e.g., Figure 2). Yet, the reasons behind this performance gap remain unclear. Understanding why SGD performs worse than Adam on Transformers is an intriguing question. **First,** from a theoretical perspective, this can help us better understand the training of Transformers and more generally, neural networks. **Second,** from a computational perspective, the understanding may inspire the design of better algorithms for training neural networks.

In this work, we explore why SGD largely underperforms Adam on Transformers through the lens of Hessian. We start by investigating the *full* Hessian spectrum of Transformers, i.e., the full eigenvalue density of Hessian (see Figure 7). By theory, the full Hessian spectrum largely determines the behavior of gradient-based methods (Nesterov, 2013; Goh, 2017; Sun, 2019; Goujaud et al., 2022), so we suspect it may also help explain SGD's unsatisfactory performance. Using tools from numerical linear algebra (Bai et al., 1996), we empirically compare the full spectra of CNNs (where SGD is on par with Adam) and those of Transformers (where SGD largely lags behind Adam). **Due to the limited space, we present the full spectra in Figure 7 in Appendix A.** Unfortunately, as shown in Figure 7, the spectra for CNNs and Transformers are often largely *similar* despite the different optimizer behaviors. As such, we have *not* identified critical features in the full Hessian spectra associated with the gap between Adam and SGD on Transformers. To reveal the cause, a more fine-grained investigation into the Hessian is needed.

What would cause SGD to perform significantly worse than Adam on Transformers, but not on CNNs? By dissecting the structures of CNNs and Transformers, we notice that CNNs are constructed by the repetitive stacking of *similar* parameter blocks (convolution layers), while Transformers involve the non-sequential stacking of *disparate* parameter blocks (e.g. Query, Key, Value, Output projection blocks in attention and MLP layers). We hypothesize that these architectural differences might lead to different optimization properties. Intuitively, disparate parameter blocks contribute differently to the overall loss. So each block might benefit from a specialized treatment by optimizers, a flexibility offered by Adam but not by SGD. This observation motivates us to investigate the Hessian spectrum of each parameter block, which we refer to as the blockwise Hessian spectrum.

By inspecting the blockwise Hessian spectrum, we discover a possible explanation for why SGD is worse: the "heterogeneity" inherent in Transformers. We provide both empiri-

*Equal contribution [1]The Chinese University of Hong Kong, Shenzhen, China [2]Shenzhen Research Institute of Big Data. Correspondence to: Ruoyu Sun <sunruoyu@cuhk.edu.cn>.

Proceedings of the 41^{th} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

cal and theoretical evidence to support this explanation. Our contributions can be summarized as follows:

- **Why SGD underperforms Adam on Transformers.** We explain why SGD is worse than Adam on Transformers by examining the blockwise Hessian spectrum. First, we identify a phenomenon called "block heterogeneity", which refers to the large differences in the Hessian spectra across parameter blocks. This block heterogeneity is observed in all examined Transformers but not in CNNs. Second, we verify that block heterogeneity hinders SGD. Across various Transformers, CNNs, and MLPs, we show that SGD consistently performs worse than Adam on problems with block heterogeneity but can perform similarly to Adam, otherwise.
- **Theoretical results on quadratic models.** We construct convex quadratic problems with and without block heterogeneity and find that gradient descent (GD) largely underperforms Adam on problems with block heterogeneity, but can perform comparably otherwise. Our theoretical analysis shows that GD can be slower than Adam on quadratic problems with block heterogeneity. We point out GD is slower than Adam because it uses a single learning rate for all blocks. The deficiency can be mitigated by assigning different learning rates across blocks, as Adam does.

We emphasize that we do *not* claim block heterogeneity is the only cause for the performance gap between Adam and SGD, but just that it is at least one important cause. We verify, both empirically and theoretically, that SGD underperforms Adam when block heterogeneity is present.

2. Main Results

2.1. Problem Settings

Notations. We denote the training loss as $\mathcal{L}(w)$, where $w \in \mathbb{R}^d$ is the neural network parameters. We denote the gradient and Hessian of the training loss w.r.t. neural network parameters as $\nabla\mathcal{L}(w) \in \mathbb{R}^d$ and $\nabla^2\mathcal{L}(w) \in \mathbb{R}^{d\times d}$, respectively. We use $[d]$ to denote the index set $\{1, 2, \cdots, d\}$. Given an arbitrary partition $\{\mathcal{D}_l\}_{l=1}^L$ over $[d]$ with $d_l \triangleq |\mathcal{D}_l|$, we can split w into L parameter blocks $\{w_l\}_{l=1}^L$, where $w_l = \mathbb{R}^{d_l}$ consists of parameters with indexes in the l-th block $\mathcal{D}_l$. We denote $[\nabla^2\mathcal{L}(w)]_l \in \mathbb{R}^{d_l\times d_l}$ as the Hessian of l-th parameter-block w_l, where $[\nabla^2\mathcal{L}(w)]_{l,i,j} = \frac{\partial^2}{\partial_{w_{l,i}}\partial_{w_{l,j}}}\mathcal{L}(w_l)$. Note that $[\nabla^2\mathcal{L}(w)]_l$ is the l-th principal block sub-matrix of $\nabla^2\mathcal{L}(w)$.

Setups. Hessian of large-scale NNs are intractable to compute and store. In this work, we apply a numerical tool called Stochastic Lanczos Quadrature method (SLQ) (Bai et al., 1996) to approximate the Hessian spectrum. A detailed introduction to SLQ is provided in Appendix E.2. All experimental setups in this section is shown in Appendix F.

We focus primarily on the following models/tasks.

- **CNNs.** We study ResNet18 (11M) and VGG16 (138M) on ImageNet (He et al., 2016; Simonyan & Zisserman, 2014). On these tasks, SGD performs on par with Adam. See Figure 11 in Appendix D for the evidence.
- **Transformers.** We study Transformer with various scales and modalities, including GPT2 (125M) on OpenWebText (Radford et al., 2019); ViT-base (86M) on ImageNet (Dosovitskiy et al., 2020); BERT (40M) on Cornell Movie-Dialogs Corpus (Devlin et al., 2018); GPT2-nano[1] (11M) on English corpus. On these tasks, SGD performs significantly worse than Adam. See Figure 12 in Appendix D for the evidence.

For each model, we estimated **(1)** the full Hessian spectrum $\nabla^2\mathcal{L}(w)$, and **(2)** the blockwise Hessian spectrum $[\nabla^2\mathcal{L}(w)]_l, l \in [L]$. For the latter, we split w according to the default partition in PyTorch implementation, e.g., Embedding layer, Query in each attention layer, Key in each attention layer, Value in each attention layer, etc. Note that the term "block" differs from the term "layer". For instance, Query and Key can reside in the same layer but are different parameter blocks.

2.2. Full Hessian Spectrum Is Not Informative Enough

We study the full Hessian spectrum of Transformers for two reasons. First, as stated in Section 1, the Hessian spectrum significantly influences the behavior of gradient methods (Nesterov, 2013). Second, previous research shows that the Hessian spectrum provides insights into neural network phenomena, like BatchNorm's effect on training speed (Ghorbani et al., 2019). Therefore, we hypothesize that the Hessian spectrum may also explain why SGD largely lags behind Adam on Transformers.

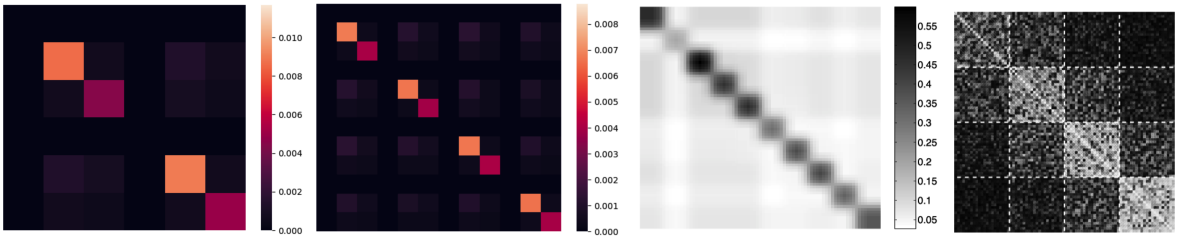
We compare the full Hessian spectra of CNNs (where SGD performs similarly to Adam) and those of Transformers (where SGD underperforms Adam). Due to the limited space, these results are relegated in Figure 7 in Appendix A. Unfortunately, we find that the full Hessian spectrum alone may not suffice to explain why SGD is worse than Adam on Transformers. We provide more detailed explanation in Appendix A.

2.3. Main Findings Through Blockwise Hessian Spectra

What other factors could cause SGD to perform significantly worse than Adam on Transformers but not on CNNs? We identify some critical features that **have been overlooked** in the full Hessian spectrum analysis above.

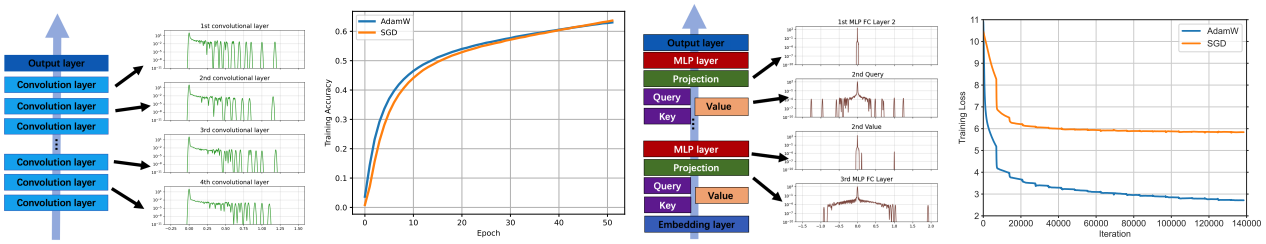
1. **Hessian structure.** Existing literatures showed that the Hessians of MLPs are close to *block-diagonal matrices* (Collobert, 2004; Roux et al., 2007; Martens & Grosse,

[1] https://github.com/karpathy/nanoGPT/



(a) 2-layer Transformer (b) 4-layer Transformer (c) MLP from (Collobert, 2004) (d) MLP from (Martens & Grosse, 2015)

Figure 1: (a,b): The initial Hessian of small Transformers. We take the absolute value of each entry to highlight non-zero entries (including negatives) and then report the average value in each block. We observe a near-block-diagonal structure. (c, d): The block-diagonal structure is also reported in the Hessian of MLPs. More similar results are re-stated in Appendix D.



(a) VGG16 (b) VGG16 (c) BERT (d) BERT

Figure 2: **(a) (c)**: The blockwise Hessian spectra of VGG16 (CNN) and BERT (Transformer) at initialization. The x-axis records the eigenvalues and the y-axis records the frequency in the log scale. We sample 4 blocks in each model. The plotted spectra are normalized by their 10th largest eigenvalues. The spectra are similar among blocks for VGG and differ significantly across blocks for BERT. **(b) (d)** Adam v.s. SGD for training VGG16 and BERT.

2015). We restate their findings in Appendix D. Further, (Collobert, 2004, Section 7) theoretically attributes the block diagonal structure to (i) the layer-by-layer design in NNs and (ii) the Cross-Entropy loss. Following this line of work, we also observe a near block-diagonal Hessian in small Transformers in Figure 1, where the variables in each principal block correspond to the parameters of each block in the Transformer. These results suggest that near block-diagonal Hessian might be common in NNs.

2. **Build-up rules of Transformers.** As shown in Figure 2, CNNs are constructed by the repetitive stacking of *similar* parameter blocks (convolution layers). In contrast, Transformers consist of *disparate* parameter blocks, e.g. Query, Key, Value in attention, and MLP layers. Further, these blocks are stacked in a non-sequential manner. Due to the different designs of these blocks, they may have different properties for optimization, which can further affect the optimizer behavior.

Combining these together, we hypothesize that the blockwise Hessian spectrum, i.e., the spectrum of principal blocks of Hessian $[\nabla^2 \mathcal{L}(w)]_l, l \in [L]$, might provide additional insights. **What extra information can be contained in the blockwise spectrum but not in the full spectrum?** By definition, the blockwise Hessians form the principal block sub-matrix of the full Hessian. We note that Transformers are observed to have a near block-diagonal Hessian. For block diagonal matrices, blockwise spectra encode the location of eigenvalues, i.e., which block an eigenvalue (of the full matrix) resides in. In contrast, the full spectrum overlooks this location information. In the following, we study the blockwise Hessian spectra of various models and show that they indeed carry more information than the full spectrum for distinguishing CNNs and Transformers.

We here demonstrate the shape of blockwise spectra in VGG16 (He et al., 2016) (CNN) and BERT (Devlin et al., 2018) (Transformer). We sample four blocks for each model and present the spectra in Figure 2. In BERT, the Hessian spectra of embedding, attention, and MLP blocks are largely *different*. In contrast, in ResNet, the spectra of convolution layers are *similar*. We further verify this observation for the rest of the parameter blocks. We calculate the Jensen-Shannon (JS) distance between two eigenvalue densities of all possible block pairs and show the results in Figure 3.

The results in Figure 3 showed a new phenomenon: for all Transformers we checked, the blockwise Hessian spectra are largely *different* across blocks. In the following, we

refer to this phenomenon as "**block heterogeneity**". In contrast, the blockwise Hessian spectra of CNNs are *similar* and the block heterogeneity is *not* observed. We refer to this phenomenon as "**block homogeneity**". These results indicate that block heterogeneity is informative in distinguishing CNNs and Transformers. Intuitively, the block-homogeneity in CNNs comes from the repetitively similar convolution layers, while the block-heterogeneity in Transformers stems from the non-sequential stacking disparate layers such as Query, Value, and MLPs. In the following, we will show that the block heterogeneity is strongly correlated with the performance gap between SGD and Adam on Transformers.

2.4. SGD Performs Worse than Adam on Various Tasks with Block Heterogeneity

Figure 2 and 3 have shown that **(1)** SGD is worse than Adam on Transformers. **(2)** Transformers have block-heterogeneity. Now we further link block heterogeneity to SGD's unsatisfactory performance on **non-Transformer** models. This would directly establish a connection between "block heterogeneity" and "why SGD is worse than Adam", without going through Transformers or attention blocks as an intermediary. We consider one man-made example and one real-world example.

Example 1: A man-made MLP. We consider a 4-layer MLP on MNIST and change the degree of heterogeneity by scaling each layer by constant c. Figure 4 shows SGD gradually performs worse than Adam as heterogeneity grows.

Example 2: MLP-mixer. We consider MLP-mixer (Tolstikhin et al., 2021), a famous all-MLP architecture that outperforms CNNs and ViTs on some vision tasks. The results are shown in the third column in Figure 6. We find that the initial Hessian of MLP-mixer has block heterogeneity and SGD lags behind Adam on this architecture.

2.5. Reduced Block Heterogeneity in Pre-trained Transformers

We remark that different Transformers exhibit different levels of block heterogeneity. Although all examined Transformers show strong block heterogeneity, we find that this heterogeneity can be mitigated, resulting in less performance deterioration for SGD. As illustrated in Figure 5, pre-trained GPT2 on SFT tasks can exhibit less block heterogeneity compared to pre-training GPT2 from scratch (Figure 3 (f)). In this case, although SGD is still slower than Adam, it achieves a similar loss at convergence. Compared with training GPT2 from scratch (Figure 12 (d) in Appendix D), the performance gap between SGD and Adam is significantly narrowed down. These findings suggest that the heterogeneity induced by architectural design can be alleviated by selecting "good" weights. This partly explains why simpler methods like SGD and even its zeroth-order version can still be effective for fine-tuning language models, albeit with slower convergence (Lv et al., 2023; Malladi et al., 2023).

2.6. Implication on Choosing SGD or Adam

We have shown that SGD can largely underperform Adam on various architectures. This leads to an intriguing question: **Can we predict the incompetence of SGD before the training begins**?

Our findings can bring up an empirical guidance: we can compute the blockwise spectrum of initial Hessian, and then decide whether to use Adam or SGD. Such a method could be useful in scenarios in training large models that are not mainstream Transformers or CNNs, e.g., Mamba (Gu & Dao, 2023). In these cases, there is not much prior experience in choosing optimizers.

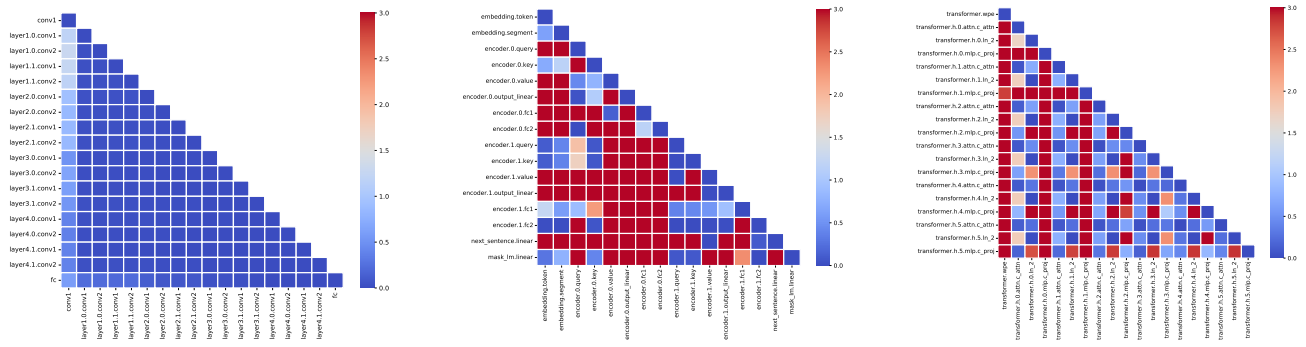
It would be intriguing to decide whether SGD is suitable for the task before the training is launched. One might argue that a simple trial is enough: try both SGD and Adam; if Adam is remarkably better, then pick Adam; if Adam and SGD are similar, then pick SGD. Nevertheless, this simple approach may not be easy for large models. First, for large models, it may take days to know whether one run of an algorithm is good or not. Second, it requires tuning hyperparameters at least a few times to get a reasonably good judgment, making the cost of the trial even higher.

We here propose a quantitative metric that can predict the incompetence of SGD before the training. With the help of this metric, we can save much expense on the trial and error for SGD. The metric is simply the averaged JS distance among blockwise Hessian spectra at initialization, i.e., the averaged value in the heatmap of Figure 3. We denote it as JS^0. We present JS^0 of various models in Table 1. Note that JS^0 establishes a quantitative difference between the loss landscape of Transformers and CNNs. Further, JS^0 is independent of optimizers and can be checked before training.

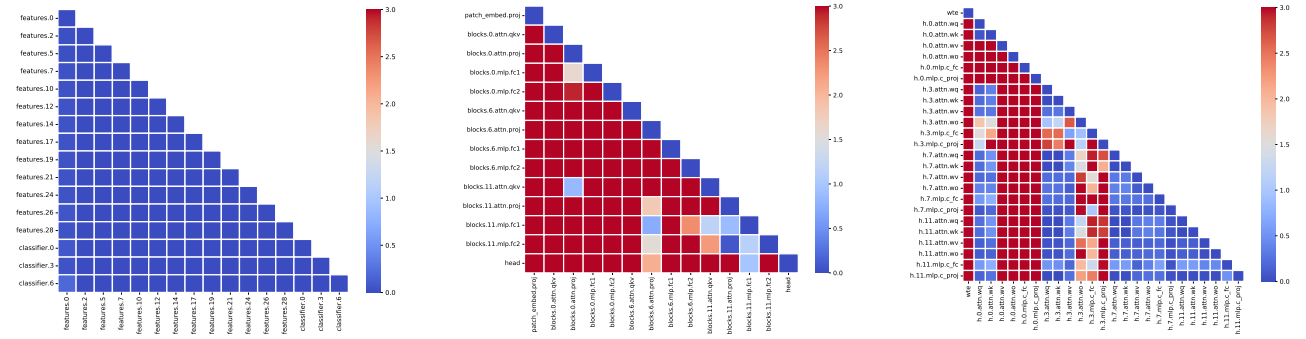
To validate the effectiveness of the quantitative metric JS^0, we summarize JS^0 of different models and the corresponding SGD performance in Figure 6. We find that the performance gap between SGD and Adam becomes greater as JS^0 increases. Thus, JS^0 can serve as a potential indicator to predict whether SGD may underperform Adam.

Table 1: JS^0 denotes the average JS distance between the initial Hessian spectra of each pair of parameter blocks. A larger JS^0 suggests that the task is more difficult for SGD.

Model	ResNet18	VGG16	MLP-mixer
JS^0	0.10	0.09	34.90
Model	BERT	GPT2	ViT-base
JS^0	53.38	83.23	286.41



(a) ResNet18 (b) BERT (c) GPT2-nano



(d) VGG16 (e) ViT-base (f) GPT2

Figure 3: The JS distance among blockwise Hessian spectra at initialization. We find that the JS distance of blockwise spectra in CNNs is significantly smaller than that in Transformers.

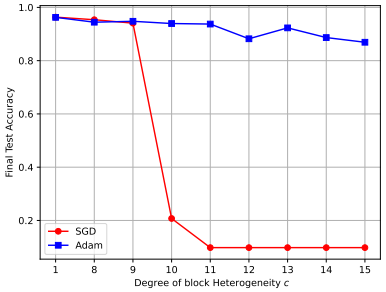
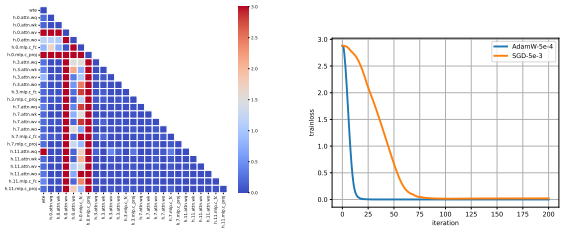


Figure 4: SGD v.s. Adam on a man-made MLP with different degrees of heterogeneity c. Each point records the best-converged test accuracy under the learning rate grid search. SGD performs worse as heterogeneity grows.



(a) GPT2 (pre-trained) (b) SGD v.s. Adam on fine-tuning GPT2 (pre-trained)

Figure 5: (a) The JS distance among blockwise Hessian spectra for GPT2 (pre-trained) when fine-tuning on Alpaca Eval. (b) SGD could reach similar loss as Adam.

3. Case Study of Quadratic Models and Preliminary Theory

We further study quadratic functions with block diagonal Hessian, with or without block heterogeneity. Note that insights on quadratic models could be important for understanding realistic NNs, as mentioned by researchers such as LeCun et al. (2002) and OpenAI team (Kaplan et al., 2020).

We compare the performance of GD and Adam on Hessian with and without block heterogeneity. Initial theoretical results on these quadratic models will be provided. Due to the limited space, we relegate these results to Appendix B.

4. Conclusion

In this work, we explore why SGD largely underperforms Adam on Transformers. we establish a phenomenon called block heterogeneity in Hessian and link it to the performance gap between Adam and SGD. Initial theory is provided to support the claim.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint*

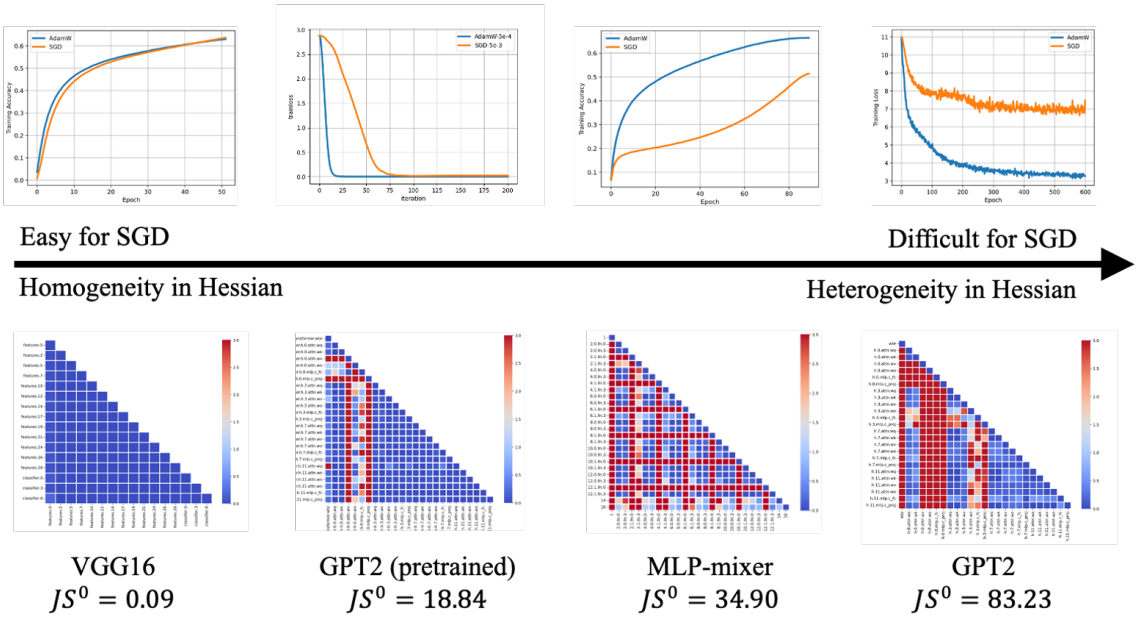


Figure 6: Comparison of JS^0 and the performance of SGD on different models. We find the performance gap between SGD and Adam becomes greater as JS^0 increases.

arXiv:2303.08774, 2023.

Adams, R. P., Pennington, J., Johnson, M. J., Smith, J., Ovadia, Y., Patton, B., and Saunderson, J. Estimating the spectral density of large implicit matrices. *arXiv preprint arXiv:1802.03451*, 2018.

Ahn, K., Cheng, X., Song, M., Yun, C., Jadbabaie, A., and Sra, S. Linear attention is (maybe) all you need (to understand transformer optimization). *arXiv preprint arXiv:2310.01082*, 2023.

Avron, H. and Toledo, S. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM (JACM)*, 58(2):1–34, 2011.

Bachlechner, T., Majumder, B. P., Mao, H., Cottrell, G., and McAuley, J. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in Artificial Intelligence*, pp. 1352–1361. PMLR, 2021.

Bai, Z. and Golub, G. H. Bounds for the trace of the inverse and the determinant of symmetric positive definite matrices. *Annals of Numerical Mathematics*, 4:29–38, 1996.

Bai, Z., Fahey, G., and Golub, G. Some large-scale matrix computation problems. *Journal of Computational and Applied Mathematics*, 74(1-2):71–89, 1996.

Beck, A. and Tetruashvili, L. On the convergence of block coordinate descent type methods. *SIAM journal on Optimization*, 23(4):2037–2060, 2013.

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

Bock, S. and Weiß, M. Non-convergence and limit cycles in the adam optimizer. In *Artificial Neural Networks and Machine Learning–ICANN 2019: Deep Learning: 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17–19, 2019, Proceedings, Part II 28*, pp. 232–243. Springer, 2019.

Brezinski, C. A direct proof of the christoffel-darboux identity and its equivalence to the recurrence relationship. *Journal of Computational and Applied Mathematics*, 32 (1-2):17–25, 1990.

Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J., Sagun, L., and Zecchina, R. Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018, 2019.

Chen, C., Shen, L., Zou, F., and Liu, W. Towards practical adam: Non-convexity, convergence theory, and mini-batch acceleration. *The Journal of Machine Learning Research*, 23(1):10411–10457, 2022.

Chen, J., Kunstner, F., and Schmidt, M. Heavy-tailed noise does not explain the gap between sgd and adam on transformers. In *13th Annual Workshop on Optimization for Machine Learning*, 2021.

Chen, M. X., Firat, O., Bapna, A., Johnson, M., Macherey, W., Foster, G., Jones, L., Parmar, N., Schuster, M., Chen,

Z., et al. The best of both worlds: Combining recent advances in neural machine translation. *arXiv preprint arXiv:1804.09849*, 2018.

Chen, X., Liu, S., Sun, R., and Hong, M. On the convergence of a class of adam-type algorithms for non-convex optimization. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Collobert, R. Large scale machine learning. Technical report, Université de Paris VI, 2004.

Crawshaw, M., Liu, M., Orabona, F., Zhang, W., and Zhuang, Z. Robustness to unbounded smoothness of generalized signsgd. *Advances in Neural Information Processing Systems*, 35:9955–9968, 2022.

Cullum, J. K. and Willoughby, R. A. *Lanczos algorithms for large symmetric eigenvalue computations: Vol. I: Theory*. SIAM, 2002.

Da Silva, A. B. and Gazeau, M. A general system of differential equations to model first-order adaptive algorithms. *The Journal of Machine Learning Research*, 21(1):5072–5113, 2020.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Défossez, A., Bottou, L., Bach, F., and Usunier, N. A simple convergence proof of adam and adagrad. *Transactions on Machine Learning Research*, 2022.

Dehghani, M., Djolonga, J., Mustafa, B., Padlewski, P., Heek, J., Gilmer, J., Steiner, A. P., Caron, M., Geirhos, R., Alabdulmohsin, I., et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pp. 7480–7512. PMLR, 2023.

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

Dong, Y., Cordonnier, J.-B., and Loukas, A. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *International Conference on Machine Learning*, pp. 2793–2803. PMLR, 2021.

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

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

Epperson, J. F. An introduction to numerical methods and analysis. 2013.

Forsythe, G. E. and Straus, E. G. On best conditioned matrices. *Proceedings of the American Mathematical Society*, 6(3):340–345, 1955.

Gadat, S. and Gavra, I. Asymptotic study of stochastic adaptive algorithms in non-convex landscape. *The Journal of Machine Learning Research*, 23(1):10357–10410, 2022.

Ghorbani, B., Krishnan, S., and Xiao, Y. An investigation into neural net optimization via hessian eigenvalue density. In *International Conference on Machine Learning*, pp. 2232–2241. PMLR, 2019.

Goh, G. Why momentum really works. *Distill*, 2017. doi: 10.23915/distill.00006. URL http://distill.pub/2017/momentum.

Golub, G. H. and Meurant, G. *Matrices, moments and quadrature with applications*, volume 30. Princeton University Press, 2009.

Golub, G. H. and Strakoš, Z. Estimates in quadratic formulas. *Numerical Algorithms*, 8:241–268, 1994.

Golub, G. H. and Welsch, J. H. Calculation of gauss quadrature rules. *Mathematics of computation*, 23(106):221–230, 1969.

Goujaud, B., Scieur, D., Dieuleveut, A., Taylor, A. B., and Pedregosa, F. Super-acceleration with cyclical step-sizes. In *International Conference on Artificial Intelligence and Statistics*, pp. 3028–3065. PMLR, 2022.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

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

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

Huang, X. S., Perez, F., Ba, J., and Volkovs, M. Improving transformer optimization through better initialization. In *International Conference on Machine Learning*, pp. 4475–4483. PMLR, 2020.

Hutchinson, M. F. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

Jiang, K., Malik, D., and Li, Y. How does adaptive optimization impact local neural network geometry? *Advances in Neural Information Processing Systems*, 36, 2023.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

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

Kunstner, F., Chen, J., Lavington, J. W., and Schmidt, M. Noise is not the main factor behind the gap between sgd and adam on transformers, but sign descent might be. *arXiv preprint arXiv:2304.13960*, 2023.

Lanczos, C. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. 1950.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–50. Springer, 2002.

Li, H., Rakhlin, A., and Jadbabaie, A. Convergence of adam under relaxed assumptions. *Advances in Neural Information Processing Systems*, 36, 2023.

Liao, Z. and Mahoney, M. W. Hessian eigenspectra of more realistic nonlinear models. *Advances in Neural Information Processing Systems*, 34:20104–20117, 2021.

Lin, L., Saad, Y., and Yang, C. Approximating spectral densities of large matrices. *SIAM review*, 58(1):34–65, 2016.

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

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

Liu, L., Liu, X., Gao, J., Chen, W., and Han, J. Understanding the difficulty of training transformers. *arXiv preprint arXiv:2004.08249*, 2020.

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

Luo, L., Xiong, Y., Liu, Y., and Sun, X. Adaptive gradient methods with dynamic bound of learning rate. In *International Conference on Learning Representations*, 2018.

Lv, K., Yang, Y., Liu, T., Gao, Q., Guo, Q., and Qiu, X. Full parameter fine-tuning for large language models with limited resources. *arXiv preprint arXiv:2306.09782*, 2023.

Malladi, S., Gao, T., Nichani, E., Damian, A., Lee, J. D., Chen, D., and Arora, S. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36:53038–53075, 2023.

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

Merrill, W., Ramanujan, V., Goldberg, Y., Schwartz, R., and Smith, N. Effects of parameter norm growth during transformer training: Inductive bias from gradient descent. *arXiv preprint arXiv:2010.09697*, 2020.

Molybog, I., Albert, P., Chen, M., DeVito, Z., Esiobu, D., Goyal, N., Koura, P. S., Narang, S., Poulton, A., Silva, R., et al. A theory on adam instability in large-scale machine learning. *arXiv preprint arXiv:2304.09871*, 2023.

Nesterov, Y. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

Nguyen, T. Q. and Salazar, J. Transformers without tears: Improving the normalization of self-attention. *arXiv preprint arXiv:1910.05895*, 2019.

Noci, L., Anagnostidis, S., Biggio, L., Orvieto, A., Singh, S. P., and Lucchi, A. Signal propagation in transformers: Theoretical perspectives and the role of rank collapse. *Advances in Neural Information Processing Systems*, 35: 27198–27211, 2022.

Pan, Y. and Li, Y. Toward understanding why adam converges faster than sgd for transformers. *arXiv preprint arXiv:2306.00204*, 2023.

Papyan, V. The full spectrum of deepnet hessians at scale: Dynamics with sgd training and sample size. *arXiv preprint arXiv:1811.07062*, 2018.

Papyan, V. Measurements of three-level hierarchical structure in the outliers in the spectrum of deepnet hessians. *arXiv preprint arXiv:1901.08244*, 2019.

Papyan, V. Traces of class/cross-class structure pervade deep learning spectra. *The Journal of Machine Learning Research*, 21(1):10197–10260, 2020.

Pearlmutter, B. A. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Reddi, S. J., Kale, S., and Kumar, S. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

Roux, N., Manzagol, P.-A., and Bengio, Y. Topmoumoute online natural gradient algorithm. *Advances in neural information processing systems*, 20, 2007.

Saad, Y. *Numerical methods for large eigenvalue problems: revised edition*. SIAM, 2011.

Sagun, L., Bottou, L., and LeCun, Y. Eigenvalues of the hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

Sagun, L., Evci, U., Guney, V. U., Dauphin, Y., and Bottou, L. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

Sankar, A. R., Khasbage, Y., Vigneswaran, R., and Balasubramanian, V. N. A deeper look at the hessian eigenspectrum of deep neural networks and its applications to regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 9481–9488, 2021.

Shi, N., Li, D., Hong, M., and Sun, R. Rmsprop converges with proper hyper-parameter. In *International Conference on Learning Representations*, 2020.

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

Sun, R. Optimization for deep learning: theory and algorithms. *arXiv preprint arXiv:1912.08957*, 2019.

Sun, R. and Ye, Y. Worst-case complexity of cyclic coordinate descent: O (nˆ 2) o (n 2) gap with randomized version. *Mathematical Programming*, 185:487–520, 2021.

Tolstikhin, I. O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34:24261–24272, 2021.

Ubaru, S., Chen, J., and Saad, Y. Fast estimation of tr(f(a)) via stochastic lanczos quadrature. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1075–1099, 2017.

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

Wang, B., Zhang, Y., Zhang, H., Meng, Q., Ma, Z.-M., Liu, T.-Y., and Chen, W. Provable adaptivity in adam. *arXiv preprint arXiv:2208.09900*, 2022a.

Wang, B., Fu, J., Zhang, H., Zheng, N., and Chen, W. Closing the gap between the upper bound and lower bound of adam's iteration complexity. *Advances in Neural Information Processing Systems*, 36, 2023a.

Wang, B., Zhang, H., Ma, Z., and Chen, W. Convergence of adagrad for non-convex objectives: Simple proofs and relaxed assumptions. In *The Thirty Sixth Annual Conference on Learning Theory*, pp. 161–190. PMLR, 2023b.

Wang, H., Ma, S., Dong, L., Huang, S., Zhang, D., and Wei, F. Deepnet: Scaling transformers to 1,000 layers. *arXiv preprint arXiv:2203.00555*, 2022b.

Wang, Q., Li, B., Xiao, T., Zhu, J., Li, C., Wong, D. F., and Chao, L. S. Learning deep transformer models for machine translation. *arXiv preprint arXiv:1906.01787*, 2019.

Wikipedia. Gaussian quadrature — Wikipedia, the free encyclopedia, 2023. URL https://en.wikipedia.org/w/index.php?title=Gaussian_quadrature&oldid=1191539517. [Online; accessed 20-January-2024].

Wortsman, M., Liu, P. J., Xiao, L., Everett, K., Alemi, A., Adlam, B., Co-Reyes, J. D., Gur, I., Kumar, A., Novak, R., et al. Small-scale proxies for large-scale transformer training instabilities. *arXiv preprint arXiv:2309.14322*, 2023.

Wu, Y., Zhu, X., Wu, C., Wang, A., and Ge, R. Dissecting hessian: Understanding common structure of hessian in neural networks. *arXiv preprint arXiv:2010.04261*, 2020.

Xiao, T., Singh, M., Mintun, E., Darrell, T., Dollár, P., and Girshick, R. Early convolutions help transformers see better. *Advances in neural information processing systems*, 34:30392–30400, 2021.

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

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pp. 10524–10533. PMLR, 2020.

Yang, A., Xiao, B., Wang, B., Zhang, B., Bian, C., Yin, C., Lv, C., Pan, D., Wang, D., Yan, D., et al. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*, 2023.

Yang, G., Hu, E. J., Babuschkin, I., Sidor, S., Liu, X., Farhi, D., Ryder, N., Pachocki, J., Chen, W., and Gao, J. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

Yao, Z., Gholami, A., Lei, Q., Keutzer, K., and Mahoney, M. W. Hessian-based analysis of large batch training and robustness to adversaries. *Advances in Neural Information Processing Systems*, 31, 2018.

Yao, Z., Gholami, A., Keutzer, K., and Mahoney, M. W. Pyhessian: Neural networks through the lens of the hessian. In *2020 IEEE international conference on big data (Big data)*, pp. 581–590. IEEE, 2020.

Zaheer, M., Reddi, S., Sachan, D., Kale, S., and Kumar, S. Adaptive methods for nonconvex optimization. *Advances in neural information processing systems*, 31, 2018.

Zeng, A., Liu, X., Du, Z., Wang, Z., Lai, H., Ding, M., Yang, Z., Xu, Y., Zheng, W., Xia, X., et al. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*, 2022.

Zhai, S., Likhomanenko, T., Littwin, E., Busbridge, D., Ramapuram, J., Zhang, Y., Gu, J., and Susskind, J. M. Stabilizing transformer training by preventing attention entropy collapse. In *International Conference on Machine Learning*, pp. 40770–40803. PMLR, 2023.

Zhang, B., Titov, I., and Sennrich, R. Improving deep transformer with depth-scaled initialization and merged attention. *arXiv preprint arXiv:1908.11365*, 2019a.

Zhang, G., Li, L., Nado, Z., Martens, J., Sachdeva, S., Dahl, G., Shallue, C., and Grosse, R. B. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. *Advances in neural information processing systems*, 32, 2019b.

Zhang, J., He, T., Sra, S., and Jadbabaie, A. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arXiv preprint arXiv:1905.11881*, 2019c.

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

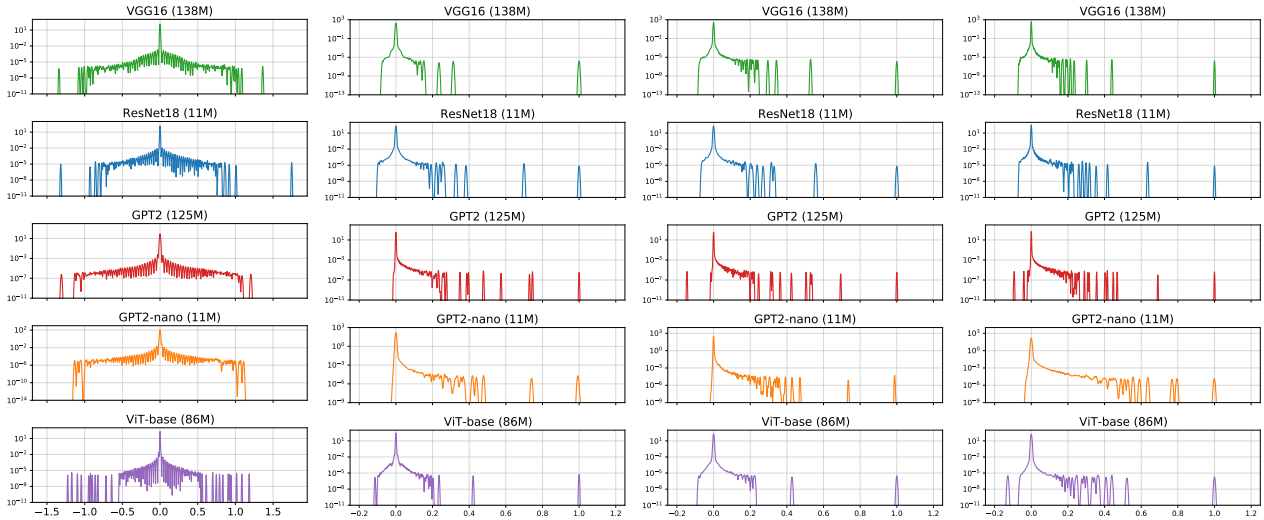
Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022a.

Zhang, Y., Chen, C., Shi, N., Sun, R., and Luo, Z.-Q. Adam can converge without any modification on update rules. *Advances in Neural Information Processing Systems*, 35: 28386–28399, 2022b.

Zhou, D., Chen, J., Cao, Y., Tang, Y., Yang, Z., and Gu, Q. On the convergence of adaptive gradient methods for nonconvex optimization. *arXiv preprint arXiv:1808.05671*, 2018.

Zou, F., Shen, L., Jie, Z., Zhang, W., and Liu, W. A sufficient condition for convergences of adam and rmsprop. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp. 11127–11135, 2019.

A. Full Hessian Spectra of Various Architectures



(a) Initialization (b) 25% steps (c) 50% steps (d) 100% steps

Figure 7: The full Hessian spectra of CNNs (VGG16 and ResNet18) and Transformers (GPT2, GPT2-nano, and ViT-base) at different training stages. The x-axis records the eigenvalues and the y-axis records the frequency in the log scale. To allow comparison in the same figure, the plotted spectra are normalized by their 10th largest eigenvalues. We find that the spectra on CNNs and Transformers are largely similar.

We find that the full Hessian spctra (Figure 7) is not informative enough to explain the gap between Adam and SGD on Transformers. We elaborate as follows. The primary information in the spectrum lies in its (A) dispersion, (B) shape, and (C) evolution during training. Regarding (A), we observe that the eigenvalues are dispersed similarly across different models, with no notably large outlier for Transformers. Thus, dispersion does not seem to be related to why SGD is worse than Adam. We further investigate (B) and (C). For all CNNs and Transformers in Figure 7, we observe similar phenomena: the spectrum's shape is approximately symmetrical around 0 at initialization. As training proceeds, the majority of negative eigenvalues disappear, and the shape evolves into a combination of a "bulk" and some "outliers". Since the spectral shape and evolution are quite similar for both Transformers and CNNs, they cannot explain why SGD is worse than Adam on Transformers, either. In summary, we have not identified any critical phenomena in the full Hessian spectra that can be linked to the performance gap between Adam and SGD on Transformers.

B. Case Study of Quadratic Models and Preliminary Theory

Now we study quadratic functions with block diagonal Hessian, with or without block heterogeneity. Note that insights on quadratic models could be important for understanding realistic NNs, as mentioned by researchers such as LeCun et al. (2002) and OpenAI team (Kaplan et al., 2020).

Setups and additional notations. We consider the following quadratic minimization.

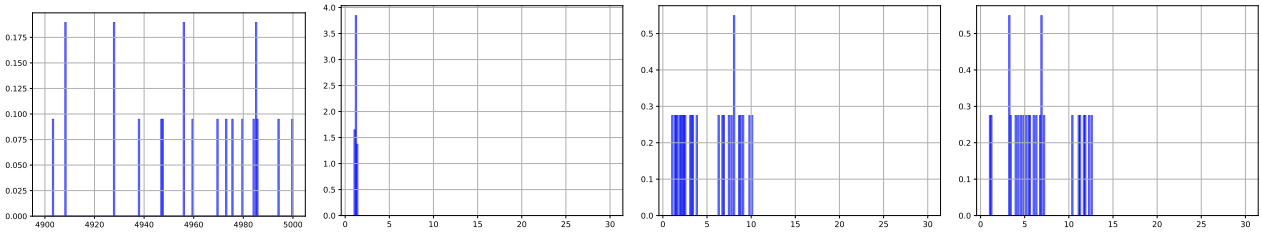
$$\min_{w\in\mathbb{R}^d} \mathcal{L}(w) = \frac{1}{2} w^T H w - h^T w,$$

where $H \in \mathbb{R}^{d\times d}$ is positive definite and $h \in \mathbb{R}^d$. We denote $\mathcal{L}^*$ as the minimum value of $\mathcal{L}(w)$. We set H as a block diagonal matrix: $H = \mathrm{diag}(H_1, \cdots, H_L)$, where $H_l \in \mathbb{R}^{d_l \times d_l}$ and $d = \sum_{l=1}^L d_l$. We use $w_l \in \mathbb{R}^{d_l}$ to denote the variable in the l-th block and $w = (w_1^T, \cdots, w_L^T)^T \in \mathbb{R}^d$. Similarly for $h_l \in \mathbb{R}^{d_l}$. Similarly, we use $[\nabla L(w)]_l \in \mathbb{R}^{d_l}$ to denote the gradient in the l-th block and denote $[\mathcal{L}(w)]_l = \frac{1}{2}(w_l^t)^T H_l w_l^t - h_l^T w_l$ as the objective function w.r.t. the l-th block. Note that $\mathcal{L}(w) = \sum_{l=1}^L [\mathcal{L}(w)]_l$. We denote $\lambda_1 \geq \lambda_2 \cdots \geq \lambda_d$ as the eigenvalues of H. Similarly for $\lambda_{l,1} \cdots \lambda_{l,d_l}$. We denote $\kappa = \frac{\lambda_1}{\lambda_d}$ and $\kappa_l = \frac{\lambda_{l,1}}{\lambda_{l,d_l}}$ as the condition number of H and H_l, respectively. We say an algorithm has complexity $\tilde{\mathcal{O}}(C)$ if it takes $\mathcal{O}(C\log(1/\epsilon))$ iterations to achieve error $\frac{\mathcal{L}(w)-\mathcal{L}^*}{\mathcal{L}(w^0)-\mathcal{L}^*} \leq \epsilon$, where w^0 is the initial point.

B.1. Experimental Observations

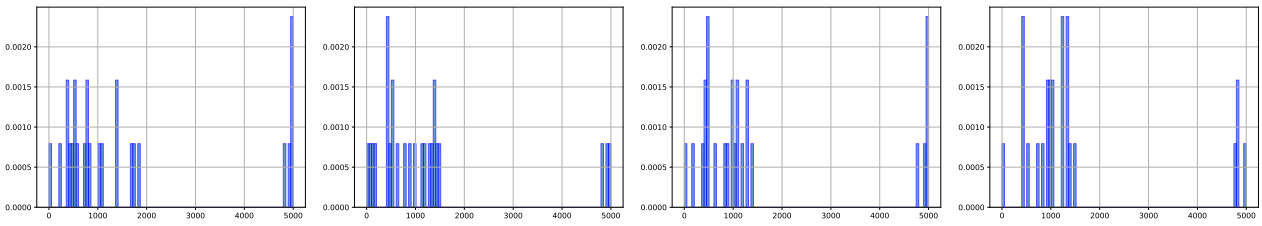
We consider four types of Hessian H as follows. For all cases, we set condition number = 5000.

- **Case 1: Hessian with Transformer-type spectra.** We choose $L = 4$ and $d_l = 25$. For $l \in [L]$, we construct $H_l = Q_l \Lambda_l Q_l^T$ where Q_l are matrices with i.i.d. standard Gassian entries and Λ_l are diagonal matrices. For the diagonal elements in Λ_l, we sample d_l numbers according to the spectrum of the embedding layer; 3rd Query, 3rd Value, 3rd MLP (`fc` layer) in GPT2. Shifting and proportional scaling are performed to ensure all elements in Λ_l lie in the interval $[1, 5000]$. This ensures strong convexity and controls the condition number of H equals 5000. The spectra of H_l are in Figure 8. We choose $h = 0$ for all cases.
- **Case 2: Hessian with CNN-type spectra.** We use the same setup as in **Case 1**. For the diagonal elements in Λ_l, we sample d_l numbers according to the spectrum of the 1st to 4th convolution layers in ResNet18. We then shift and scale Λ_l to the interval $[1, 5000]$ to ensure strong convexity and a condition number of 5000. The spectra of H_l are shown in Figure 9.
- **Case 3: Hessian with simplified heterogeneous spectra.** We choose $L = 3$ and $d_l = 3$. For $l \in [L]$, we construct $H_l = Q_l \Lambda_l Q_l^T$ where Q_l are independent standard Gassian random matrix and Λ_l are diagonal matrices. We set the diagonal elements of Λ_l as $\{1, 2, 3\}, \{99, 100, 101\}, \{4998, 4999, 5000\}$ for $l = 1, 2, 3$, respectively. The spectra of H_l are different due to their different supports. The condition number of Hessian H is 5000.
- **Case 4: Hessian with simplified homogeneous spectra.** We consider the same setup as **Case 3**. We set the diagonal elements of Λ_l as $\{1, 99, 4998\}, \{2, 100, 4999\}, \{3, 101, 5000\}$ for $l = 1, 2, 3$, respectively. The spectra of H_l are similar. The condition number is 5000.



(a) Spectrum of H_1 (b) Spectrum of H_2 (c) Spectrum of H_3 (d) Spectrum of H_4

Figure 8: Histogram of eigenvalues of each block in **Case 1** (the heterogeneous case). The eigenvalues in the four blocks are sampled from the spectrum of the embedding layer; 3rd Query, 3rd Value, 3rd MLP (`fc` layer) in GPT2, respectively. All the eigenvalues are shifted and proportionally scaled such that: the objective function is strong convex; the condition number of Hessian equals 5000; their relative ranges are preserved; and the block heterogeneity is preserved.



(a) Spectrum of H_1 (b) Spectrum of H_2 (c) Spectrum of H_3 (d) Spectrum of H_4

Figure 9: Histogram of eigenvalues of each block in **Case 2** (the homogeneous case). The eigenvalues in the four blocks are sampled from the spectrum of 1st to 4th convolution layers in ResNet18, respectively. All the eigenvalues are shifted and proportionally scaled such that: the objective function is strong convex; the condition number of Hessian equals 5000; their relative ranges are preserved; and the block homogeneity is preserved.

Now we study two types of optimizers: one that assigns a single learning rate for all blocks, and one that assign different learning rates across blocks.

- **Single-learning-rate optimizer.** We study gradient descent (GD).

$$w^{t+1} = w^t - \eta \nabla \mathcal{L}(w) = w^t - \eta(Hw^t - h) \tag{1}$$

We use the optimal learning rate $\eta = \frac{2}{\mu + L}$ (Nesterov, 2013). We use standard Gaussian initialization.

- **Coordinate-wise-learning-rate optimizer.** We study Adam with a constant learning rate and with no bias correction for simplicity (Algorithm 3). We set $\beta_1 = 0$ to erase the effect of momentum. **This helps us to focus on the effect of coordinate-wise learning rate** (or the effect of diagonal preconditioning) in Adam. We use $\epsilon = 0$. We consider $\beta_2 = 1$ and $\beta_2 = 0.99$, respectively. When $\beta_2 = 1$, Adam assigns coordinate-wise learning rates according to the initial gradient, but these learning rates are fixed along iteration. The update rule is as follows.

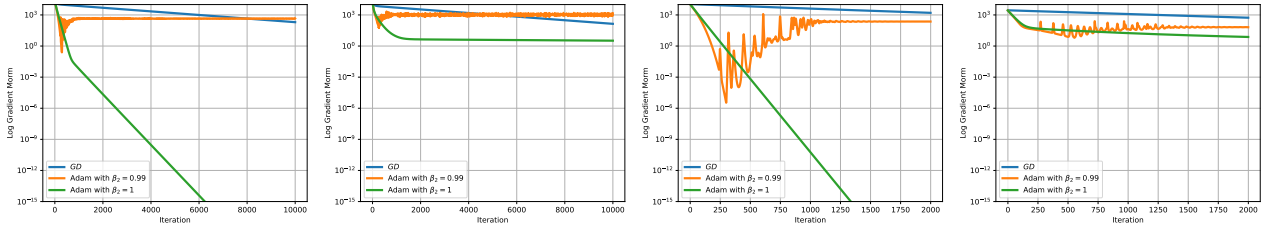
$$w^{t+1} = w^t - \eta (D^0_{Adam})^{-1} \nabla \mathcal{L}(w) = w^t - \eta (D^0_{Adam})^{-1}(Hw^t - h), \tag{2}$$

where $D^0_{Adam} = \operatorname{diag}(\nabla \mathcal{L}(w^0) \circ \nabla \mathcal{L}(w^0))^{\frac{1}{2}}$ and $\nabla \mathcal{L}(w^0) = Hw^0 - h$. When $\beta_2 < 1$, the coordinate-wise learning rates adaptively change along iteration. The update rule is as follows (note that $\nabla \mathcal{L}(w^k) = Hw^k - h$.).

$$w^{t+1} = w^t - \eta (D^t_{Adam})^{-1} \nabla \mathcal{L}(w) = w^t - \eta (D^t_{Adam})^{-1}(Hw^t - h), \quad \text{where} \tag{3}$$

$$D^t_{Adam} = \operatorname{diag}\left((1-\beta_2) \left(\sum_{k=1}^{t} \beta_2^{t-k} \nabla \mathcal{L}(w^k) \circ \nabla \mathcal{L}(w^k) \right) + \beta^t \operatorname{diag}(\nabla \mathcal{L}(w^0) \circ \nabla \mathcal{L}(w^0)) \right)^{\frac{1}{2}}$$

We grid search η and use the standard Gaussian initialization. We remark that when $\beta_2 < 1$, Adam would bounce among non-optimal points. This will be shown in Proposition 2.



(a) Hessian with GPT2 blockwise spectrum (b) Hessian with ResNet18 blockwise spectrum (c) Hessian with simplified heterogeneous blocks (d) Hessian with simplified homogeneous blocks

Figure 10: The performance of Adam and GD on homo/heterogeneous quadratic problems. The condition numbers of Hessian equal to 5000 for all four cases. When blocks are heterogeneous, GD largely lags behind Adam, and GD performs similarly to Adam if otherwise.

Summary of experimental observations. Figure 10 presents two phenomena. For Hessian with heterogeneous blocks (**Case 1 and 3**), GD largely lags behind Adam. For Hessian with homogeneous blocks (**Case 2 and 4**), GD is on par with Adam. We emphasize that all Hessians have the same condition number. Further, Hessian of **Case 3** and **4** share all the eigenvalues (not just the extreme ones). The gap between Adam and GD is due to the different blockwise spectra caused by the different locations of eigenvalues. We hypothesize that GD performs badly because it uses one single learning rate for all blocks, which cannot handle the heterogeneity among blocks. Such heterogeneity can be better handled using different learning rates across blocks, as designed in Adam.

B.2. Initial Theoretical Results

We now provide initial theoretical results to characterize how GD lags behind Adam in problems with heterogenous Hessian. Note that classical optimization theory depicts the rate of first-order methods by the condition number of the full Hessian κ. However, we point out that κ is not informative enough to describe the performance gap in Figure 10 since κ is the same in all four cases. To distinguish Adam and GD, we need to utilize more fine-grained quantities like blockwise spectra of sub-matrices.

Note that the blockwise spectrum is not common in the optimization area. The most related notion is perhaps "block Lipschitz constant" (Beck & Tetruashvili, 2013) for studying block coordinate descent (BCD) type methods, but it was not linked to the performance of SGD or Adam before. To our knowledge, we are not aware of any theory of Adam or GD built on the block diagonal structures or the blockwise spectra of Hessian. We now make an initial attempt in this direction. We first present the lower bound for GD.

Proposition 1. (Lower bound for GD.) Consider $\min_w \mathcal{L}(w) = \frac{1}{2} w^T H w - h^T w$ where $H \in \mathbb{R}^{d\times d}$ is positive definite and $h \in \mathbb{R}^d$. Let w_{GD}^t be the output of GD after t steps. There exists a block diagonal matrix H, h and an initial point w^0, s.t., for any η, we have:

$$\mathcal{L}(w_{GD}^{t+1}) - \mathcal{L}^* \geq \left(1 - \frac{2}{\kappa + 1}\right) \left(\mathcal{L}(w_{GD}^t) - \mathcal{L}^*\right) \tag{4}$$

where κ is the condition number of H.

Proposition 1 shows that GD has complexity $\tilde{\mathcal{O}}(\kappa)$ and such complexity is tight. Now we prove that Adam can achieves better complexity. This is because it chooses different learning rates for different block sub-matrix H_l via its diagonal preconditinoner D_{Adam}^0. We consider generic random initialization that covers commonly used distributions such as Gaussian, Uniform, etc.

Assumption 1. (Random initialization.) Assume the initialization w^0 is sampled from a continuous distribution, i.e., the probability measure (induced by w^0) of any zero-Lebesgue-measure set is 0.

Theorem 1. (Upper bound for Adam with $\beta_2 = 1$.) Consider the same setting as Proposition 1 and consider Adam with $\beta_1 = 0$ and $\beta_2 = 1$ as in (2). Assume the initialization satisfies Assumption 1. Let w_{Adam}^t be the output of Adam after t steps. Let $\eta = \min_{l\in[L]} \frac{1}{C_{l,1}}$. Then w.p.1., we have

$$\mathcal{L}(w_{Adam}^{t+1}) - \mathcal{L}^* \leq \max_{l\in[L]} \left(1 - \frac{1}{\kappa_{Adam,l}}\right) \left(\mathcal{L}(w_{Adam}^t) - \mathcal{L}^*\right) \tag{5}$$

where $\kappa_{Adam,l} = r\kappa_l$, κ_l is the condition number of H_l, constant r relates to w^0 defined as:

$$r = \frac{\max_{l\in[L]} C_{l,2}^2}{\min_{l\in[L]} C_{l,1}^2}, \text{ where } C_{l,1} = \min_{i\in[d_l]} \frac{|[\nabla \mathcal{L}(w^0)]_{l,i}|}{\lambda_{l,1}}, C_{l,2} = \max_{i\in[d_l]} \frac{|[\nabla \mathcal{L}(w^0)]_{l,i}|}{\lambda_{l,1}}. \tag{6}$$

The proofs of the above theorems are shown in Appendix G. Theorem 1 states that Adam (with $\beta_2 = 1$) has complexity $\tilde{O}\left(r \cdot \max_{l\in[L]} \kappa_l\right)$. We note that coefficient r depends on the ratio between initial gradient and the principal eigenvalue for each block, and smaller ratio would give faster convergence. We further remark that condition $\beta_2 = 1$ is necessary because any $\beta_2 < 1$ causes non-convergence issue (Bock & Weiß, 2019; Da Silva & Gazeau, 2020). We restate their results in Proposition 2. The non-convergence is also observed in Figure 10 (c), where we find that the iterates of Adam quickly converge to near-optimal solutions, and then bounce back. As such, $\beta_2 = 1$ is necessary for asymptotic analysis. The analysis for $\beta_2 = 1$ is still meaningful since it still shows the effect of Adam's preconditioner.

As shown in (Da Silva & Gazeau, 2020), the non-convergence is due to the constant learning rate. Reducing the learning rate reduces the gap between $\mathcal{L}(w_{Adam}^t)$ and $\mathcal{L}^*$, but does not remove it.

Proposition 2. (Non-convergence of constant-learning-rate Adam with $\beta_2 < 1$.) (Da Silva & Gazeau, 2020, Proposition 12, Figure 1) Consider $\min_{w\in\mathbb{R}} \mathcal{L}(w) = \frac{1}{2} w^2$. Consider Adam with $\beta_1 = 0$ and $\beta_2 < 1$ as in (3). Let w_{Adam}^t be the output of Adam after t steps. There exists a discrete limit cycle for (3) and $\liminf_{t\to\infty} \left(\mathcal{L}(w_{Adam}^t) - \mathcal{L}^*\right) > 0$.

We now compare the complexity of Adam and that of GD. By Theorem 1, Adam is faster than GD when $r \cdot \max_{l\in[L]} \kappa_l \leq \kappa$. In the quadratic model with heterogeneous blocks (**Case 3**), our simulation over 1000 trials shows that $r \leq 1000$ with probability $\geq \frac{2}{3}$ when using standard Gaussian random initialization. Since $\max_{l\in[L]} \kappa_l \approx 1$, we have $r \cdot \max_{l\in[L]} \kappa_l \leq 1000$, w.h.p., and is about $5\times$ smaller than $\kappa = 5000$. So Adam could be $5\times$ faster than GD, w.h.p.. This is indeed observed in Figure 10 where Adam outperforms GD by a significant margin. We summarize the complexity of GD and Adam in Table 2.

Table 2: The complexity of GD and Adam for minimizing a strongly convex quadratic function with block diagonal Hessian. The symbol ✗ means non-convergence. κ and κ_l denote the condition number of the full Hessian and the block submatrix, respectively. r is defined in (6).

Optimizer	GD	Adam with $\beta_1 = 0$ and $\beta_2 = 1$ (2)	Adam with $\beta_1 = 0$ and $\beta_2 < 1$ (3)
Complexity	$\tilde{O}(\kappa)$	$\tilde{O}\left(r \cdot \max_{l\in[L]} \kappa_l\right)$	✗

How to obtain a tighter complexity bound of Adam? It is valid to ask whether the complexity upper bound in Theorem 1 $\kappa_{Adam,l} = r\kappa_l$ can be tightened, e.g., improve the factor of r. We point out it would be difficult if there is no extra structure

on H_l. A key technical step is to bound the condition number of the preconditioned matrix $\kappa\left((D^0_{Adam,l})^{-1}H_l\right)$. Intuitively, a diagonal preconditioner of H_l is powerful when H_l itself has a near-diagonal structure, e.g., pure diagonal, tridiagonal or diagonal dominant (Forsythe & Straus, 1955). Unfortunately, it is unclear whether these structures hold in Transformers. Without any assumption on H_l, we find that the diagonal preconditioner of D^0_{Adam} could *increase* the condition number. For instance, when using standard Gaussian initialization, in **case 3**, we find $\kappa\left((D^0_{Adam,l})^{-1}H_l\right)$ equals $7.09\kappa_1$, $18.98\kappa_2$, $18.76\kappa_3$ for the 3 blocks, respectively (all averaged over 1000 trials). It would be interesting to explore if there are special structures of H_l in Transformers such that Adam preconditioner can reduce κ_l, rather than increase it. We leave it as a future direction.

Although Adam preconditioner might not always reduce the "local" condition number κ_l, the coefficient in the complexity is now independent of the "global" condition number κ. As argued above, such changes in coefficient could lead to considerable improvement over GD. Such improvement in complexity is attributed to the block diagonal structure in Hessian as well as its heterogeneous blockwise spectrum. To our knowledge, such improvement is not shown in the existing literature. In summary, our theory indicates that: for problems with block heterogeneity, the single-learning rate methods like GD can largely lag behind coordinate-wise learning rate methods like Adam.

C. Related Works

On the unsatisfactory performance of SGD on Transformers There is an active line of works that explores why SGD performs significantly worse than Adam on Transformers. One representative hypothesis is that SGD cannot handle the heavy-tailed stochastic noise in language tasks (Zhang et al., 2020). However, Chen et al. (2021); Kunstner et al. (2023) reported that the gap between Adam and SGD maintains even in the full-batch case with no stochasticity, so there might be other reasons. Further, SGD performs worse than Adam on Vision Transformers on ImageNet (See Figure 12. Also see (Xiao et al., 2021) for more evidence), so the data modality (e.g., language or vision tasks) might not be as crucial as the architecture. (Zhang et al., 2019c) showed that NLP tasks have "unbounded smoothness" issue and SGD with gradient clipping performs better than SGD in this case. Although clipping is an effective trick, we still observe a huge gap between clipped SGD and Adam [2], so there might be other reasons. Different from these works, we find SGD underperforms Adam because it uses one single learning rate for all blocks, which cannot handle the Hessian heterogeneity among blocks.

Understanding of Adam. There was once a long-standing debate on the possible divergence of Adam (Reddi et al., 2018). The convergence for the unmodified versions is later established in (Shi et al., 2020; Zhang et al., 2022b) for RMSprop and Adam. More convergence analyses of general adaptive gradient methods are listed later in this section. We here focus on the literature that explores the benefit of Adam. Xie et al. (2022) show that Adam can help avoid saddle points, which is an orthogonal direction to this work. Wang et al. (2022a); Crawshaw et al. (2022); Li et al. (2023) show that Adam and its variant outperform SGD under relaxed smoothness conditions, based on the intuition that Adam can adaptively change its learning rate along iteration (over time). We pointed out that the theory is not complete: even for quadratic functions where the smoothness is fixed, SGD sometimes performs largely worse than Adam (Figure 10). This indicates that the benefit of Adam is not merely due to its ability to adaptively change the learning rate (over time), and there are other reasons for Adam's success. We show that an important benefit of Adam is its ability to handle the heterogeneity across blocks (over space).

Recent works (Bernstein et al., 2018; Wu et al., 2020; Kunstner et al., 2023; Liu et al., 2023; Ahn et al., 2023) build a relation between Adam and the sign-based methods. Wu et al. (2020) further showed that sign-based methods can be effective when the Hessian is diagonal and satisfies several other properties. However, as put by the authors, it seems "unclear to what extent these properties hold for real problems". Pan & Li (2023) numerically found that the Adam can reduce the directional sharpness along trajectories, while its relation to fast convergence remains mysterious. A recent work (Jiang et al., 2023) point out that Adam biases the trajectories towards regions where Hessian has "uniform diagonal entries" while SGD cannot. The distribution of Hessian diagonal entries is also investigated in (Liu et al., 2023). The theory in (Jiang et al., 2023) implies that Adam is faster when the Hessian is diagonal. However, as argued above, it is unclear whether the diagonal Hessian structure commonly holds in real problems. In fact, we find the Hessian is closer to a block-diagonal (instead of pure diagonal) structure on some small Transformers. In these cases, blockwise eigenvalues carry more information than diagonal entries, providing extra details such as the location of eigenvalues. We find that these extra details are important for

[2] For all NLP tasks, clipping is performed immediately after backpropagation. So in Figure 12, SGD in NLP tasks essentially refers to clipped SGD.

distinguishing Adam and SGD.

Hessian Spectrum Analysis. There are several important attempts to explore the Hessian spectrum of MLPs and CNNs. Early works (Sagun et al., 2016; 2017; Chaudhari et al., 2019) found that the Hessian spectra of MLPs and CNNs consist of a "bulk" together with a few "outliers". Papyan (2020); Wu et al. (2020); Liao & Mahoney (2021) further characterized the bulks and outliers in theory. Papyan (2018; 2019) numerically built the relation between these "outliers" and the Gauss-Newton matrix. Sankar et al. (2021) numerically explored the relation between Hessian of CNNs and Gauss-Newton matrix in each layer. They further found that most CNN layers contribute similarly to the overall loss surface. We find that this result is restricted to CNNs and does not hold on Transformers due to the heterogeneity. Gur-Ari et al. (2018) showed that for MLPs and CNNs, gradient descent converges to a small subspace spanned by a few top eigenvectors of the Hessian. Yao et al. (2018); Zhang et al. (2019b) explored the relation between the Hessian spectrum of CNNs and some training phenomena such as the effect of batch sizes. Ghorbani et al. (2019); Yao et al. (2020) focused on explaining the effectiveness of techniques such as BatchNorm. Note that all these works are restricted to MLPs and CNNs, while we study the Hessian of Transformers (in addition to CNNs and MLPs) as well as its impacts on different optimizers.

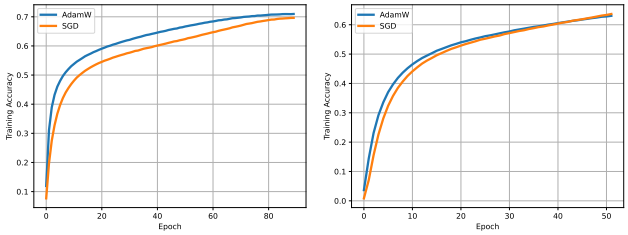
On the difficulties of Transformer training. Transformers are known to be difficult to train. Researchers have attributed the training difficulties to various phenomena in different components of Transformers, including: the logits divergence or the rank degeneracy in the outputs of attention layers (Dong et al., 2021; Noci et al., 2022; Wortsman et al., 2023; Zhai et al., 2023; Dehghani et al., 2023; Chowdhery et al., 2023); the growth of parameter norm in attention layers (Merrill et al., 2020); over-reliance on residue branches (Liu et al., 2020); and some negative impact of layer norm (Chen et al., 2018; Zhang et al., 2019a; Huang et al., 2020). These phenomena have a strong correlation with gradient vanishing or explosion in Transformers (Zhang et al., 2019a; Liu et al., 2020; Huang et al., 2020; Xiong et al., 2020; Noci et al., 2022; Wang et al., 2022b; Wortsman et al., 2023; Molybog et al., 2023), which leads to training difficulties.

Several solutions have been proposed. Liu et al. (2020) numerically observed that adaptive gradient methods can (partly) overcome gradient vanishing by giving "consistent update magnitude", while it seems unclear how consistent update magnitude would help optimization in principle. Researchers further develop training tricks such as warmup learning rate (Liu et al., 2019; Xiong et al., 2020), temperature scaling (Noci et al., 2022), better initialization (Zhang et al., 2019a; Huang et al., 2020; Wang et al., 2022b; Bachlechner et al., 2021; Yang et al., 2022), and variants of Layer Norm (Nguyen & Salazar, 2019; Wang et al., 2019; Xiong et al., 2020; Wang et al., 2022b; Dehghani et al., 2023). Recent researchers also suggest using z-loss regularization (Chowdhery et al., 2023; Yang et al., 2023) and tuning hyperparameters of Adam (Zhang et al., 2022b; Wortsman et al., 2023). All these tricks can help mitigate gradient explosion or vanishing. Nevertheless, training large-scale Transformers remains challenging (Zhang et al., 2022a; Zeng et al., 2022; Wortsman et al., 2023; Molybog et al., 2023; Chowdhery et al., 2023). Different from all aforementioned works, we investigate the training difficulties of Transformers through the eigenvalues of Hessian. We establish a strong correlation between "the blockwise Hessian spectra of Transformers" and "why SGD largely underperforms Adam on Transformers". We realize that our attempt is just a first step towards understanding Transformer training, and we believe there is rich information hidden in Hessian and we leave more fine-grained analysis as future works.

Convergence analysis of general adaptive gradient methods There is extensive convergence analysis for adaptive gradient methods. For instance, researchers study the convergence of AMSGrad (Reddi et al., 2018; Zhou et al., 2018), RMSprop (Zaheer et al., 2018), AdaFom (Chen et al., 2019), AdaBound (Luo et al., 2018), and Adam with iterate-dependent hyperparameters (Zou et al., 2019; Chen et al., 2022; Gadat & Gavra, 2022). The convergence of Adam is also explored in (Défossez et al., 2022; Wang et al., 2023a). There is also an active line of theoretical research on the convergence of AdaGrad (Duchi et al., 2011), we recommend (Wang et al., 2023b) for more detailed introduction. In this work, we do not focus on the convergence analysis. Rather, we explore the quantitative difference between the loss landscape of CNNs and Transformers and how it impact the behaviors of SGD and Adam.

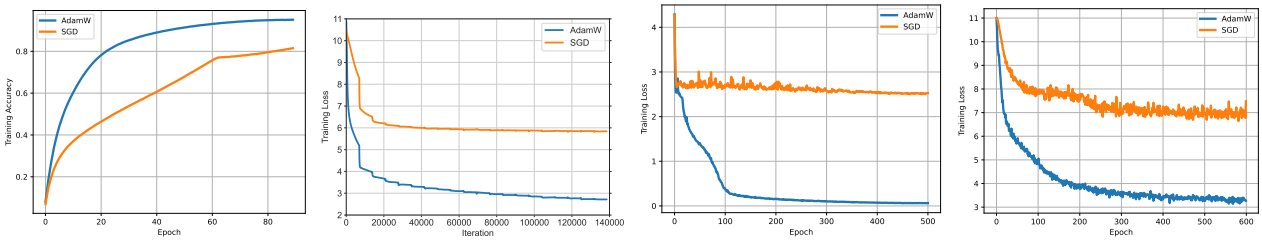
D. More Results and Discussions

Performance comparison of AdamW and SGD on different Architectures. Here, we show the performance comparison of AdamW and SGD on different models. All the vision models are trained on ImageNet. Language models are trained on different English corpus. We grid-search the learning rates for SGD and Adam under the same budget and report the best result for each optimizer. See Appendix F.1 for more implementation details.



(a) ResNet18 (b) VGG16

Figure 11: Performance of AdamW and SGD on CNNs including ResNet18 and VGG16. SGD and Adam perform similarly on these tasks.



(a) ViT (b) BERT (c) GPT2-nano (d) GPT2

Figure 12: Performance of AdamW and SGD on Transformers including ViT, BERT, GPT2-nano, and GPT2. SGD performs significantly worse than Adam on these tasks.

Block-diagonal structure in the existing literature. We remark that Collobert (2004); Roux et al. (2007); Martens & Grosse (2015) also observed the block-diagonal structure in (approximated) Hessian of small-scaled MLPs. We restate their findings in Figure 13. (Collobert, 2004, Section 7) further theoretically proved that the block diagonal structure stems from (i) the layer-by-layer structure and (ii) the Cross-Entropy loss. These results suggest that block-diagonal structure might be common in NNs.

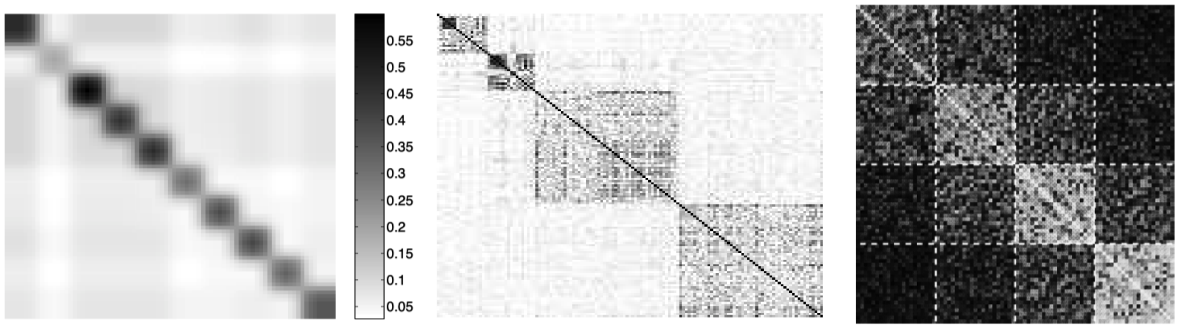
E. More Preliminaries

E.1. Preliminaries on Optimizers

Here we provide a detailed description of the optimizers mentioned in the full script. We consider the minimizing $\mathcal{L}(w) \equiv \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_i(w)$, where n is the number of minibatches, $\mathcal{L}_i(w)$ is the loss of i-th minibatch and $w \in \mathbb{R}^d$ is the neural network parameters. We denote the gradient of the training loss w.r.t. neural network parameters as $\nabla\mathcal{L}(w) \in \mathbb{R}^d$. We use $\nabla\mathcal{L}_i(w) \in \mathbb{R}^d$ to denote the i-th minibatch counterparts. We use w^t to denote the variable at the t-th step. In Algorithm 2 and 3, $\circ$, division and square-root are elementwise operations. In the line 7 and 8 of Algorithm 2, $(\beta_1)^t$ and $(\beta_2)^t$ indicates the t-th power of β_1, β_2. In the PyTorch default setting, $(\beta_1, \beta_2, \epsilon) = (0.9, 0.999, \text{1e-8})$ for Adam and $\beta_1 = 0.9$ for SGD.

E.2. Preliminaries on the Stochastic Lanczos Quadrature Method

Additional notations. Given a real symmetric matrix $H \in \mathbb{R}^{d\times d}$, we denote $tr(H)$ as its trace and $Q^T\Lambda Q$ as its spectral decomposition, where $Q = [q_1, \ldots, q_d]$, $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_d)$ and $\lambda_1 \geq \lambda_2 \cdots \geq \lambda_d$. We denote the condition number of



(a) Exact Hessian of a MLP in (Collobert, 2004), Figure 7.4

(b) Approximated Hessian of a MLP in (Roux et al., 2007), Figure 1

(c) Approximated Hessian of a MLP in (Martens & Grosse, 2015), Figure 6

Figure 13: The block-diagonal structure in the (approximated) Hessian of MLPs reported in the literature.

Algorithm 1 Stochastic Gradient Descent with Momentum (SGD)

Initialize w^0 and choose $0 \leq \beta_1 < 1$ and $\eta_0 > 0$
for $t = 1 \to \infty$ **do**
  Uniformly sample τ^t from the index set $\{1, 2, \cdots, n\}$
  $m^t = \beta_1 m^t + \nabla \mathcal{L}_{\tau^t}(x^t)$
  $x^{t+1} = x^t - \eta_t m^t$
end for

Algorithm 2 AdamW

Initialize x^0, $m^0 = v^0 = 0$, $0 \leq \beta_1 < 1$, $0 \leq \beta_2 < 1$, $\epsilon > 0$, $\eta^0 > 0$, and weight decay coefficient λ
for $t = 1 \to \infty$ **do**
  Uniformly sample τ^t from the index set $\{1, 2, \cdots, n\}$
  $w^{t+1} = w^t - \eta^t \lambda w^t$
  $m^t = \beta_1 m^t + (1 - \beta_1) \nabla \mathcal{L}_{\tau^t}(w^t)$
  $v^t = \beta_2 v^t + (1 - \beta_2) \nabla \mathcal{L}_{\tau^t}(w^t) \circ \nabla \mathcal{L}_{\tau^t}(w^t)$
  $\hat{m}^t = \frac{m^t}{1-(\beta_1)^t}$
  $\hat{v}^t = \frac{v^t}{1-(\beta_2)^t}$
  $w^{t+1} = w^{t+1} - \eta_t \frac{\hat{m}^t}{\sqrt{\hat{v}^t} + \epsilon}$
end for

Algorithm 3 Adam with no bias correction

Initialize x^0, $m^0 = \nabla \mathcal{L}_{\tau^t}(w^0)$, $v^0 = \nabla \mathcal{L}_{\tau^t}(w^0) \circ \nabla \mathcal{L}_{\tau^t}(w^0)$, $0 \leq \beta_1 < 1$, $0 \leq \beta_2 < 1$, $\epsilon > 0$, $\eta^0 > 0$
for $t = 1 \to \infty$ **do**
  Uniformly sample τ^t from the index set $\{1, 2, \cdots, n\}$
  $m^t = \beta_1 m^t + (1 - \beta_1) \nabla \mathcal{L}_{\tau^t}(w^t)$
  $v^t = \beta_2 v^t + (1 - \beta_2) \nabla \mathcal{L}_{\tau^t}(w^t) \circ \nabla \mathcal{L}_{\tau^t}(w^t)$
  $w^{t+1} = w^{t+1} - \eta_t \frac{m^t}{\sqrt{v^t} + \epsilon}$
end for

H as $\kappa = \lambda_1/\lambda_d$. We define matrix function as $f(H) := Q^T f(\Lambda) Q$, where $f(\Lambda) = \operatorname{diag}\left(f\left(\lambda_1\right), \ldots f\left(\lambda_d\right)\right) \in \mathbb{R}^{d\times d}$. We use $\mathbb{N}$ to denote the set of positive integers. We use $\|\cdot\|_2$ to denote the Euclidean norm.

Approximation of the Hessian spectrum can be formulated as a trace estimation problem, as introduced in (Lin et al., 2016; Ubaru et al., 2017). First, the spectrum (eigenvalue density) of Hessian H can written as: $\phi(t) = \frac{1}{d}\sum_{i=1}^{d} \delta\left(t - \lambda_i\right)$, where λ_i are the eigenvalues of H and δ is the Dirac δ-function. Then, we replace the delta functions by a Gaussian blurring function: $\phi(t) \approx g(t) := \frac{1}{d}\sum_{i=1}^{d} f\left(\lambda_i\right)$, where $f(\lambda) := \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(t-\lambda)^2}{2\sigma^2}\right)$. By definition of matrix function, it is easy to see that $g(t) = \frac{1}{d} tr(f(H))$. As such, spectrum approximation could be formulated as a trace estimation problem, i.e., estimating $\frac{1}{d}\operatorname{tr}(f(H))$, where $H \in \mathbb{R}^{d\times d}$ is a real symmetric matrix.

Trace estimation problems could be solved efficiently by the Stochastic Lanczos Quadrature Method (SLQ) (Golub & Strakoš, 1994). For the ease of readers, we re-organize and summarize the existing literature ((Golub & Strakoš, 1994; Ubaru et al., 2017; Ghorbani et al., 2019)) and provide a detailed description of SLQ in our context. SLQ consists of the following steps.

Step 1. We Approximate the trace of matrix function as $\frac{1}{d} tr(f(H)) = \mathbb{E}(v^T f(H) v) \approx \frac{1}{n_v}\sum_{i}^{n_v} v_i^T f(H) v_i$, where $v = u/\|u\|_2$ and u is a Rademacher random vector (each entry of u independently takes ± 1 with probability $1/2$). This step is called Huchinson's estimation (Hutchinson, 1989).

Note that we can also replace the Rademacher random vector u by a unit Gaussian vector (i.e., $u \sim N(0, I_{d\times d})$) and the unbiasedness still holds (Avron & Toledo, 2011). In our implementation, we sample $u \sim N(0, I_{d\times d})$ because there is an efficient built-in PyTorch function for generating Gaussian vectors.

SLQ estimates $v_i^T f(H) v_i$ for $i \in [n_v]$ and then take the average. To understand SLQ, we only need to understand how it estimates each individual quadratic form. To simplify the notation regarding i, from now on, we will discuss how to estimate $v^T f(H) v$, where $v = u/\|u\|_2$ and u is a unit Gaussian vector.

Step 2-1. We rewrite $v^T f(H) v$ as a Riemann-Stieltjes integral (Golub & Meurant, 2009):

$$v^T f(A) v = \sum_{i=1}^{d} \left(v^T q_i\right)^2 f\left(\lambda_i\right) = \int_{\lambda_d}^{\lambda_1} f(\lambda) d\mu(\lambda), \tag{7}$$

where μ is a measure on $(\mathbb{R}, \mathbb{B})$ defined as follows ($\mu(\lambda)$ denotes the measure of set $\{x; x \le \lambda\}$):

$$\mu(\lambda) = \begin{cases} 0 & \lambda < \lambda_d \\ \sum_{i=1}^{k} \left(v^T q_i\right)^2 & \lambda_k \le \lambda < \lambda_{k+1} \\ \sum_{i=1}^{d} \left(v^T q_i\right)^2 & \lambda \ge \lambda_1 \end{cases}. \tag{8}$$

Step 2-2. Unfortunately, this integral is difficult to compute. This is because the measure μ are related to the eigen-pairs of H, which are unknown. It seems unclear how to directly integrate over an unknown measure. As such, we further approximate this integral by a computationally friendly quantity, such as:

$$\int_{\lambda_d}^{\lambda_1} f(\lambda) d\mu(\lambda) \approx \sum_{j=1}^{m} c_j f(x_j). \tag{9}$$

We hope to design $\{(c_j, x_j)\}_{j=1}^m$ with a reasonable number of m such that the estimation error is small. Fortunately, the Gaussian Quadrature method provides a generic design principle of $\{(c_j, x_j)\}_{j=1}^m$ (Golub & Meurant, 2009; Epperson, 2013). It is proved that: when $f(\lambda)$ is not "too complicated" (e.g. $f(\lambda)$ is a polynomial), then there exists $\{(c_j, x_j)\}_{j=1}^m$ which gives a high quality estimation of integral (7). The required number of m is related to "how complicated the $f(\lambda)$ is". Such $\{(c_j, x_j)\}_{j=1}^m$ are called the Gaussian Quadrature rules. c_j and x_j are called the "weights" and the "nodes" of the Gaussian Quadrature rules. A representative theorem is as follows: when $f(\lambda)$ is a polynomial with degree $< 2m$, then the Gaussian Quadrature rules give the exact approximation of integral (7).

Theorem 2. [Rewrited based on (Wikipedia, 2023)] Suppose we have a sequence of orthogonal polynomials $\{p_k(x)\}_{k=1}^m$ w.r.t. measure μ, that is: $\int_{\lambda_d}^{\lambda_1} p_n(x)p_m(x)d\mu(x) = \delta_{m,n}$, where $\delta_{m,n} = 1$ if $m = n$ and $\delta_{m,n} = 0$, otherwise. Assume $f(x)$ is a polynomial with degree $< 2m$, then there exists $\{(c_j, x_j)\}_{j=1}^m$ s.t. $\int_{\lambda_d}^{\lambda_1} f(\lambda)d\mu(\lambda) = \sum_{i=j}^m c_j f(x_j)$. The equality holds when x_j are the roots of $p_m(x)$ and $c_j = \int_{\lambda_d}^{\lambda_1} \prod_{j\neq i} \frac{x-x_i}{x_j-x_i} d\mu$. Such choice of $\{(c_j, x_j)\}_{j=1}^m$ are called the Gaussian Quadrature rules.

Theorem 2 shows the existence of good $\{(c_j, x_j)\}_{j=1}^m$ and their general form. In fact, it is also shown that Gaussian Quadrature is optimal: no other$\{(c_j, x_j)\}_{j=1}^m$ can achieve zero approximation error for higher degree polynomials $f(\lambda)$ (Golub & Meurant, 2009). However, it is often difficult to find these quadrature rules (Golub & Welsch, 1969). There are at least three questions in sequel:

- **1)** how to find the orthogonal polynomials $\{p_k(x)\}_{k=1}^m$ w.r.t. an unknown measure μ?
- **2)** how to efficiently find the roots of $p_m(x)$, which gives the nodes x_j?
- **3)** how to efficiently calculate the weights $c_j = \int_{\lambda_d}^{\lambda_1} \prod_{j\neq i} \frac{x-x_i}{x_j-x_i} d\mu$?

We first answer question **2)** and **3)** and leave question **1)** for later discussion.

Now suppose that we have found the orthogonal polynomials $\{p_k(x)\}_{k=1}^m$ w.r.t. μ. Recall that any orthogonal polynomial has the following "three-term" recursion (Golub & Meurant, 2009):

$$p_{k+1}(x) = (x - \alpha_{k+1})\, p_k(x) - \beta_k p_{k-1}(x), k = 0, 1, \ldots,$$

where $p_{-1}(x) \equiv 0, p_0(x) \equiv 1, \alpha_{k+1} = \frac{\langle xp_k, p_k\rangle}{\langle p_k, p_k\rangle}$ and $\beta_k = \frac{\langle p_k, p_k\rangle}{\langle p_{k-1}, p_{k-1}\rangle}$. Define $P_m(x) = (p_0(x), p_1(x), \ldots p_{m-1}(x))^T \in \mathbb{R}^m$, we can rewrite the recursion formula in matrix form (given x): $xP_m = J_m P_m + \beta_m p_m(x) e^m$, where e^m is the last column of identity matrix $I_{m,m}$ and J_m is called Jacobi matrix of order m:

$$J_m = \begin{pmatrix} \alpha_1 & \sqrt{\beta_1} & & & \\ \sqrt{\beta_1} & \alpha_2 & \sqrt{\beta_2} & & \\ & \sqrt{\beta_2} & \alpha_3 & \sqrt{\beta_3} & \\ & & \ddots & \ddots & \ddots \end{pmatrix} \in \mathbb{R}^{m\times m}$$

It turns out that J_m can help us find the Gaussian Quadrature rules $\{(c_j, x_j)\}_{j=1}^m$ and thus provide answers for question **2)** and **3)**. This is shown in the following theorem.

Theorem 3. (Golub & Meurant, 2009) For the Gaussian Quadrature, $\{x_j\}_{j=1}^m$ are the eigenvalues of J_m and $\{c_j\}_{j=1}^m$ are the squares of the first elements of the normalized eigenvectors of J_m.

The proof of Theorem 3 is based on Christoffel-Darboux relation (Brezinski, 1990). Now, the remaining question is: how to find the Jacobian matrix J_m of a sequence of orthogonal polynomials w.r.t. an unknown measure μ? Note that we no longer need to answer question **1)** if J_m is found, since J_m is sufficient for us to find the Gaussian quadrature rules. However, it seems impossible to find J_m if no information of μ is provided. The good news is: when the μ is specified as in (8), there exists an efficient way to find J_m.

Step 3. When μ is specified as in (8), J_m can be exactly found in m steps using the Lanczos algorithm (Lanczos, 1950), as shown in Algorithm 4. This method takes a real symmetric matrix as input and returns a tridiagonal matrix. It was originally proposed to solve eigenvalue problems. Later, researchers found a deep connection between the Lanczos algorithm and orthogonal polynomials, which further connects this method to the Gaussian quadrature. The method (of finding the Gaussian quadrature by the Lanczos algorithm) is called the Lanczos quadrature (Golub & Strakoš, 1994; Bai & Golub, 1996; Golub & Meurant, 2009). An extremely elegant but highly nontrivial result is as follows:

Theorem 4. (Golub & Meurant, 2009) Given a real symmetric matrix $H \in \mathbb{R}^{d\times d}$ and an arbitrary vector $v \in \mathbb{R}^d$ with unit Euclidean norm, we define the measure μ as in (8) based on this H and v. Then m steps of the Lanzcos algorithm return the Jacobian matrix J_m of orthogonal polynomials w.r.t. to μ.

After J_m is found by the Lanczos algorithm, we perform spectral decomposition of $J_m \in \mathbb{R}^{m\times m}$ to get its eigen-pairs. Using Theorem 3, we successfully get the Gaussian quadrature rules and thus we can approximate the quadratic form $v^T f(H)v$. By averaging over different random vectors v we can then approximate $\frac{1}{d}tr(f(H))$. This concludes the derivation of SLQ for the trace estimation problem.

The full procedure of SLQ is shown in Algorithm 5. We note that SLQ is efficient in theory. Ubaru et al. (2017) show that SLQ converges faster than any other polynomial expansion method for spectrum estimation (e.g., Chebyshev methods used in (Adams et al., 2018)). See (Ubaru et al., 2017, Theorem 4.1) for a formal statement.

We remark that there are at least four versions of the Lanczos algorithm in **Step 3**. Here, we adopt the version in Algorithm 4 since it is known to be the most numerically stable version (Cullum & Willoughby, 2002; Saad, 2011; Wikipedia, 2023). Throughout this work, we choose $f(\cdot)$ as the Gaussian blurring function $f(\lambda) := \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(t-\lambda)^2}{2\sigma^2}\right)$ for spectrum approximation. We plot the spectrum by sweeping t from the minimal node to the maximal node in Gaussian Quadrature rules.

Algorithm 4 The Lanczos Algorithm

1: Input a matrix-vector product $Hv_1 \in \mathbb{R}^d$, where H is a real symmetric matrix and v_1 is an arbitrary vector with Euclidean norm 1. Choose $m \in \mathbb{N}$
2: **Initialization:** Let $w_1' = Hv_1$, $\alpha_1 = (w_1')^T v_1$, $w_1 = w_1' - \alpha_1 v_1$
3: **for** $j = 2 \to m$ **do**
4: Let $\beta_j = \|w_{j-1}\|_2$ (also Euclidean norm)
5: If $\beta_j \neq 0$, then let $v_j = w_{j-1}/\beta_j$,
else pick as v_j an arbitrary vector with Euclidean norm 1 that is orthogonal to all of $v_1, \ldots, v_{j-1}$
6: Let $w_j' = Av_j$
7: Let $\alpha_j = (w_j')^T v_j$
8: Let $w_j = w_j' - \alpha_j v_j - \beta_j v_{j-1}$
9: **end for**
10: Let V be the matrix with columns $v_1, \ldots, v_m$
11: Let $T = \begin{pmatrix} \alpha_1 & \beta_2 & & & & 0 \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \ddots & & \\ & & \ddots & \ddots & \beta_{m-1} & \\ & & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ 0 & & & & \beta_m & \alpha_m \end{pmatrix}$
12: Return T

Algorithm 5 The Stochastic Lanczos Quadrature Method

1: Choose $\mathtt{num}_v, m \in \mathbb{N}$. Sample $\mathtt{num}_v$ i.i.d. v_i from normalized Rademacher distribution, $i \in [\mathtt{num}_v]$
2: **for** $i = 1 \to \mathtt{num}_v$ **do**
3: Run m steps of the Lanczos Algorithm 4 with input Hv_i, returns $T \in \mathbb{R}^{m\times m}$
4: Compute eigenvalue decomposition $T = Q\Lambda Q^T$
5: Compute the nodes $x_i = (\Lambda_{ii})_{i=1}^m$ and weights $c_i = \left(Q_{1,i}^2\right)_{i=1}^m$
6: Return $q_i(t) = \sum_{i=1}^m c_i f\left(x_i; t, \sigma^2\right)$
7: **end for**
8: Return $\frac{1}{\mathtt{num}_v}\sum_{i=1}^{\mathtt{num}_v} f\left(\ell_i; t, \sigma^2\right)$

F. More Eperimental Details

F.1. Implementation Details on SLQ and Training Configurations

Implementation and Running Time Analysis. We provide a simple PyTorch implementation of SLQ. The only query SLQ makes to the neural network is the Hessian vector product, which is attained using the auto-differentiation framework

(Pearlmutter, 1994). To assure the accuracy of the Lanczos algorithm, we remove all the randomness in the forward and backward passes, including: data shuffling order, data augmentation, and dropout, etc.. Since Flash Attention (Dao et al., 2022) does not support the calculation of Hessian-vector product, we implement all attention blocks in the naive way. For the calculation of the blockwise Hessian spectrum $\nabla^2\mathcal{L}(w_l)$, we sample $u_l \sim N(0, I_{d_l \times d_l})$ and set $v_l = u_l/\|u_l\|_2 \in \mathbb{R}^{d_l}$. Then we run Algorithm 5 by taking $\nabla^2\mathcal{L}(w_l)$ and v_l as inputs. We choose the hyperparameters as $m = 100$ and $n_v = 10$ in all experiments. σ is tuned based on visual effects. These hyperparameters are reported to reach highly accurate estimation with error $< 10^{-14}$ (Ghorbani et al., 2019).

We now briefly discuss the computational cost of SLQ. The major computational expense of SLQ is the repeated Hessian-vector product operations in Lanczos algorithm in **Step 3**. Recall $\nabla^2\mathcal{L}(w)d = \frac{1}{n}\sum_{i=1}^{n}\nabla^2\mathcal{L}_i(w)d$, so each Hessian-vector product operation requires **(i)** calculating $\nabla^2\mathcal{L}_i(w)d$; **(ii)** repeating **(i)** on all data. We point out that **(i)** can be computed efficiently and precisely with just two backpropagation passes (Pearlmutter, 1994). The major computational bottleneck lies in **(ii)** due to the large n. Our largest-scale experiment for Hesian spectrum is GPT2 (125M) on Openwebtext, where the number of tokens $n = 9$ Billon. To calculate $\nabla^2\mathcal{L}(w)d$ on all these 9B tokens, it requires about 9 GPU days on eight A100-80GB GPUs. Since SLQ requires at least 1,000 times query of $\nabla^2\mathcal{L}(w)d$, a complete run of SLQ would take at least 9,000 days on eight A100-80GB GPUs, which is unaffordable. In this work, we use the largest possible batch size (with gradient accumulation tricks) to approximate $\nabla^2\mathcal{L}(w)$ under the constraints of GPU bandwidth and time limit. More detailed setup of SLQ are shown as follows.

- **ResNet18 (18M) and VGG16 (138M) on ImageNet.** We use the code base of PyTorch Examples [3]. We use batch size $= 1024$. For the calculation of the blockwise Hessian spectra, we apply SLQ to all parameter blocks except for the BatchNorm layers. In total, it takes about 3 days on one V100 GPU to estimate all the blockwise Hessian spectra and the full Hessian spectrum.
- **ViT-base (86M) on ImageNet.** We use the code base of PyTorch Image Models [4]. We use batch size $= 1024$. Due to the large number of parameters, we are not able to calculate the blockwise Hessian spectra for all parameter blocks. Instead, we apply SLQ to: the embedding layer; the output layer; the 1-st, 6-th, 12-th attention blocks; and the 1-st, 6-th, 12-th MLP blocks (note that the 12-th attention and MLP blocks are the final ones). In total, it takes about 3 days on one V100 GPU to estimate all the blockwise Hessian spectra and the full Hessian spectrum.
- **BERT(40M) on Cornell Movie-Dialogs Corpus.** We use the code base from the blog [5]. We use batch size $= 327,680$ tokens. For the calculation of the blockwise Hessian spectra, we apply SLQ to all parameter blocks except for the LayerNorm layers. In total, it takes about 12 hours on one V100 GPU to estimate all the blockwise Hessian spectra and the full Hessian spectrum.
- **GPT2-nano (11M) on Shakespeare.** We use the code base of NanoGPT [6]. We use batch size $= 163,840$ tokens. For the calculation of the blockwise Hessian spectra, we apply SLQ to all parameter blocks with even indices, except for the LayerNorm layers. In total, it takes about 12 hours on one V100 GPU to estimate all the blockwise Hessian spectra and the full Hessian spectrum.
- **GPT2 (125M) on Openwebtext [7].** We use the code base of NanoGPT. We use batch size $= 245,760$ tokens. Due to the large number of parameters, we are not able to calculate the blockwise Hessian spectra for all parameter blocks. Instead, we apply SLQ to: the embedding layer; the output layer; the 1-st, 4-th, 8-th, 12-th attention blocks; and the 1-st, 4-th, 8-th, 12-th MLP blocks (note that the 12-th attention and MLP blocks are the final ones). In total, it takes about 7 days on one A100 GPU to estimate all the blockwise Hessian spectra and the full Hessian spectrum.

Training configuration. In all cases, we train all the models under the default configurations in the above codebase. We grid-search the learning rates for SGD and Adam under the same budget and report the best result for each optimizer. We use the cosine-decay learning rate schedule for vision tasks. For SFT task, we use nanoGPT codebase. We first pre-train GPT2 on OpenwebText for 25B tokens and then fine-tune it on a subset of Alpaca Eval [8].

[3] https://github.com/pytorch/examples/blob/main/imagenet/main.py
[4] https://github.com/huggingface/pytorch-image-models
[5] https://medium.com/data-and-beyond/complete-guide-to-building-bert-model-from-sratch-3e6562228891
[6] https://github.com/karpathy/nanoGPT/
[7] https://huggingface.co/datasets/Skylion007/openwebtext
[8] https://huggingface.co/datasets/tatsu-lab/alpaca_eval

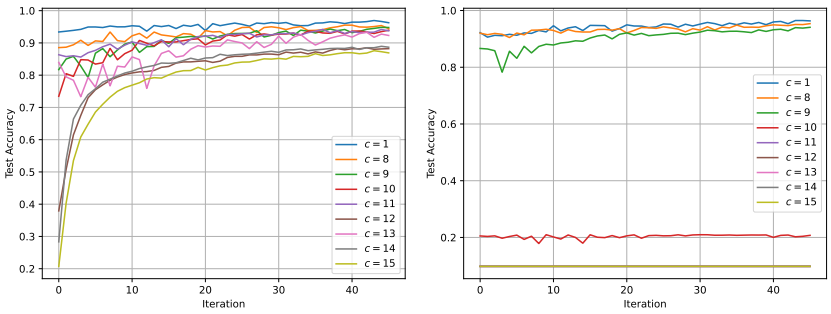
F.2. Implementation Details on Figure 1

We use the code base of NanoGPT to train decoder-only Transformers on 4 consecutive tokens randomly selected from Openwebtext. We set the model configuration as: context window = 2, number of heads = 2 and the embedding dimension = 4. In MLP layers, all widths equal to 4. We choose the number of layers to be 2, 4, and 8. We remove all the LayerNorms in the model. The rest of the model configurations are set to their default values in the code base. We compute the Hessian on all the parameters blocks in attention and MLP layers. In Figure 1 (a), the variables each block corresponds to the parameters in Query and Key; Value and Projection; and MLP, respectively. For better visualization, We take the absolute value of each entry to distinguish non-zero values (including negatives) from those near 0 and then report the average value in each block. Similarly for the rest of the figures.

Due to the intensive overhead of computing and storing the whole Hessian, we have yet to check the block-diagonal structure on larger models. Rigorously speaking, so far we have not gotten sufficient evidence to claim this structure commonly holds in larger Transformers. It requires new numerical methods to efficiently check the block-diagonal Hessian structure without explicitly calculating them. We leave it as an interesting future direction.

F.3. More Details on the MLP experiments in Figure 4

We train a 4-layer MLP on MNIST. We use batch size = 128 and width = 300, 128, and 64 for the hidden layers. We use ReLU activation. We change the degree of heterogeneity by scaling the output of each layer with constant $c \in \mathbb{N}$. We scale c from 1 to 15. For each c, we train SGD and Adam with default hyperparameters by grid-searching the learning rate from 1e-4 to 1e-1 and report the best test accuracy after 1 epoch.



(a) Training curves of Adam (b) Training curves of SGD

Figure 14: The training curves of SGD and Adam on MNIST with 4-layer MLPs under different degrees of block heterogeneity c. We observe that SGD performs worse as heterogeneity grows, while Adam remains unaffected.

G. Proofs

G.1. Proof of Proof of Proposition 1

Let $H = \begin{bmatrix} L & 0 \\ 0 & \mu \end{bmatrix}$, where $L > \mu > 0$. We choose the initial point as $w^0 = (w_1^0, w_2^0) = (\sqrt{\mu/L}, \sqrt{L/\mu})$. By the update rule of GD, we have

$$\begin{aligned} \mathcal{L}(w^{t+1}) &= \mathcal{L}\left(w^t - \eta \nabla \mathcal{L}(w^t)\right) \\ &= \frac{1}{2}(w^t - \eta H w^t)^T H (w^t - \eta H w^t) \\ &= (w_1^t)^2 |1 - \eta L| L + (w_2^t)^2 |1 - \eta\mu| \mu \\ &= |1 - \eta L|^t L \frac{\mu}{L} + |1 - \eta\mu|^t \mu \frac{L}{\mu} \\ &= \mu |1 - \eta L|^t + L |1 - \eta\mu|^t \end{aligned} \tag{10}$$

To proceed, we discuss the following cases:

When $\eta \leq 1/L$, since $|1-\eta L|^t$ and $|1-\eta\mu|^t$ are monotonically decreasing , the optimal solution is $\eta = 1/L$.

When $\eta \geq 1/\mu$, since $|1-\eta L|^t$ and $|1-\eta\mu|^t$ are monotonically increasing , the optimal solution is $\eta = 1/\mu$.

When $1/L \leq \eta \leq 1/\mu$, (10) can be written as $g_t(\eta) = \mu(\eta L - 1)^t + L(1-\eta\mu)^t$. Take the first-order and the second-order derivative of the g, we can obtain $g_t'(\eta) = tL\mu(\eta_L - 1)^{t-1} - t\mu L(1-\eta\mu)^{t-1}$ and $g_t''(\eta) = t(t-1)L^2\mu(\eta L-1)^{t-2} + t(t-1)\mu^2(1-\eta\mu)$. Since $g_t''(\eta) \geq 0$ for all $\eta \in [1/L, 1\mu]$, the function g is convex. By solving the equation that $g_t'(\eta) = 0$, we can obtain $\eta = \frac{2}{L+\mu}$ is a solution for all t. Plugging this result into (10) and rearranging the terms, we conclude the proof of Proposition 1.

G.2. Proof of Theorem 1

We first show that $C_{l,2}$ and $C_{l,1}$ are non-zero w.p.1. under the random initialization in Assumption 1. We define set $S_i = \{w; h_i^T w = 0\}$ where $h_i \in \mathbb{R}^d$ is the i-th row of H. Since H is positive definite, there is at least one non-zero entry in h_i, $i \in [d]$. As such, S_i is a $(d-1)$-dimensional subspace of $\mathbb{R}^d$ and thus has zero Lebesgue measure in $\mathbb{R}^d$. Since w^0 follows continuous distribution, we have $\Pr\left(\{w^0; h_i^T w^0 = 0\}\right) = 0$, for $i = [d]$. Then we have

$$\begin{aligned}
\Pr\left(\nabla\mathcal{L}(w^0) \text{ has at least one zero entry}\right) &= \Pr\left(Hw^0 \text{ has at least one zero entry}\right) && (11)\\
&= \Pr\left(\cup_{i=1}^d \{w^0; h_i^T w^0 = 0\}\right) && (12)\\
&\leq \sum_{i=1}^d \Pr\left(\{w^0; h_i^T w^0 = 0\}\right) && (13)\\
&= 0. && (14)
\end{aligned}$$

Therefore, $\nabla\mathcal{L}(w^0)$ is elementwise non-zero w.p.1.., so $C_{l,1}$ and $C_{l,2}$ are non-zero for all $l \in [L]$, w.p.1.. In the following analysis, We will assume $C_{l,1}$ and $C_{l,2}$ are non-zero.

Without loss of generality, we assume $h = 0$. This is because minimizing $\mathcal{L}(w) = \frac{1}{2}w^T H w - h^T w$ is equivalent to minimizing $\mathcal{L}(w) = \frac{1}{2}(w-w^*)^T H (w-w^*)$ where $w^* = H^{-1}h$. By a linear transformation $z = w - w^*$, Adam for minimizing $\frac{1}{2}(w-w^*)^T H (w-w^*)$ starting from w^0 is equivalent to Adam for minimizing $\frac{1}{2}z^T H z$ starting from $z^0 = w^0 - w^*$. Thus we can assume $w^* = 0$, or equivalently, $h = 0$. The update rule of Adam becomes

$$w^{t+1} = w^t - \eta(D_{Adam}^0)^{-1} H w^t,$$

where $D_{Adam}^0 = \operatorname{diag}(\nabla\mathcal{L}(w^0) \circ \nabla\mathcal{L}(w^0))^{\frac{1}{2}} = \operatorname{diag}(|Hw^0|)$. We denote $d_t = \eta(D_{Adam}^0)^{-1} H w^t$ and thus we have $w^t = \frac{1}{\eta}H^{-1}D_{Adam}^0 d^t$ and $w^{t+1} = w^t - d_t$. These relations also hold for each block by changing the notation to H_l w_l^t, D_{Adam}^0, and d_l^t, etc.. Following the framework in (Sun & Ye, 2021), we try to bound the error yet to be optimized (a.k.a., cost-to-go) and the per-step improvement, respectively. The ratio of these two terms characterizes the rate of convergence. We now express both terms using d_l^t. For the cost-to-go term for the l-th block, we have

$$[\mathcal{L}(w^t)]_l - [\mathcal{L}^*]_l = \frac{1}{2}(w_l^t)^T H_l w_l^t = \frac{1}{2\eta^2}(d_l^t)^T D_{Adam,l}^0 H_l^{-1} D_{Adam,l}^0 d_l^t. \tag{15}$$

For the per-step improvement, we have

$$\begin{aligned}
[\mathcal{L}(w^t)]_l - [\mathcal{L}(w^{t+1})]_l &= \frac{1}{2}(w_l^t)^T H_l w_l^t - \frac{1}{2}(w_l^{t+1})^T H_l w_l^{t+1}\\
&= \frac{1}{2}(w_l^t)^T H_l w_l^{t+1} - \frac{1}{2}(w_l^t - d^t)^T H_l (w_l^t - d_l^t)\\
&= (d_l^t)^T H_l w_l^t - \frac{1}{2}(d_l^t)^T H_l d_l^t
\end{aligned}$$

$$= \frac{1}{2}(d_l^t)^T \left(\frac{2}{\eta} D_{Adam,l}^0 - H_l\right) d_l^t. \tag{16}$$

To proceed, we denote $\hat{H} = (D_{Adam}^0)^{-1} H$ and we denote its eigenvalues as $\hat{\lambda}_1 \geq \hat{\lambda}_2 \geq \cdots \hat{\lambda}_d$. Similarly, we denote $\hat{H}_l = (D_{Adam,l}^0)^{-1} H_l$ and its eigenvalues $\hat{\lambda}_{l,1} \geq \hat{\lambda}_{l,2} \geq \cdots \hat{\lambda}_{l,d_l}$. Let $\eta = \min_{l\in[L]} C_{l,1}$, we have

$$\begin{aligned}
\frac{[\mathcal{L}(w^t)]_l - [\mathcal{L}^*]_l}{[\mathcal{L}(w^t)]_l - [\mathcal{L}(w^{t+1})]_l} &= \frac{\frac{1}{\eta^2}(d_l^t)^T D_{Adam,l}^0 H_l^{-1} D_{Adam,l}^0 d_l^t}{(d_l^t)^T \left(\frac{2}{\eta} D_{Adam,l}^0 - H_l\right) d_l^t} \\
&\leq \left\| \frac{1}{\eta^2} \left(\frac{2}{\eta} D_{Adam,l}^0 - H_l\right)^{-1} D_{Adam,l}^0 H_l^{-1} D_{Adam,l}^0 \right\|_2 \qquad (17) \\
&\overset{(*)}{\leq} \frac{C_{l,2}^2 \lambda_{l,1}^2}{(\min_{l\in[L]} C_{l,1}^2)\lambda_{l,1}\lambda_{l,d_l}} \qquad (18) \\
&\leq \frac{\max_{l\in[L]} C_{l,2}^2}{\min_{l\in[L]} C_{l,1}^2} \kappa_l, \qquad (19)
\end{aligned}$$

where $(*)$ is due to: by Assumption 1, $D_{Adam,l}^0 \preccurlyeq C_{l,2}\lambda_{l,1} I$, $\frac{2}{\eta} D_{Adam,l}^0 - H_l \succcurlyeq \left(\frac{2}{C_{l,1}} C_{1l,}\lambda_{l,1} - \lambda_{l,1}\right) I \succcurlyeq \lambda_{l,1} I$, where $\preccurlyeq$ and $\succcurlyeq$ are matrix inequalities. By rearranging both sides of (19), we have $[\mathcal{L}(w^{t+1})]_l - [\mathcal{L}^*]_l \leq \left(1 - \frac{1}{\left(\frac{\max_{l\in[L]} C_{l,2}^2}{\min_{l\in[L]} C_{l,1}^2}\right)\kappa_l}\right)([\mathcal{L}(w^t)]_l - [\mathcal{L}^*]_l)$. Summing up both sides over $l \in [L]$ and we conclude the proof.

$$\begin{aligned}
\mathcal{L}(w^{t+1}) - \mathcal{L}^* &= \sum_{l=1}^{L} \left([\mathcal{L}(w^{t+1})]_l - [\mathcal{L}^*]_l\right) \\
&\leq \sum_{l=1}^{L} \left(1 - \frac{1}{\left(\frac{\max_{l\in[L]} C_{l,2}^2}{\min_{l\in[L]} C_{l,1}^2}\right)\kappa_l}\right) \left([\mathcal{L}(w^t)]_l - [\mathcal{L}^*]_l\right) \\
&\leq \max_{l\in[L]} \left(1 - \frac{1}{\left(\frac{\max_{l\in[L]} C_{l,2}^2}{\min_{l\in[L]} C_{l,1}^2}\right)\kappa_l}\right) \sum_{l=1}^{L} \left([\mathcal{L}(w^t)]_l - [\mathcal{L}^*]_l\right) \\
&= \max_{l\in[L]} \left(1 - \frac{1}{\left(\frac{\max_{l\in[L]} C_{l,2}^2}{\min_{l\in[L]} C_{l,1}^2}\right)\kappa_l}\right) \left(\mathcal{L}(w^t) - \mathcal{L}^*\right).
\end{aligned}$$