KrADagrad: Kronecker Approximation-Domination Gradient Preconditioned Stochastic Optimization

Jonathan Mei[1†] **Alexander Moreno**[1†] **Luke Walters**[1†]

[1]Independent Researcher

Abstract

Second order stochastic optimizers allow parameter update step size and direction to adapt to loss curvature, but have traditionally required too much memory and compute for deep learning. Recently, Shampoo [Gupta et al., 2018] introduced a Kronecker factored preconditioner to reduce these requirements: it is used for large deep models [Anil et al., 2020] and in production [Anil et al., 2022]. However, it takes inverse matrix roots of ill-conditioned matrices. This requires 64-bit precision, imposing strong hardware constraints. In this paper, we propose a novel factorization, Kronecker Approximation-Domination (KrAD). Using KrAD, we update a matrix that directly approximates the inverse empirical Fisher matrix (like full matrix AdaGrad), avoiding inversion and hence 64-bit precision. We then propose KrADagrad*, with similar computational costs to Shampoo and the same regret. Synthetic ill-conditioned experiments show improved performance over Shampoo for 32-bit precision, while for several real datasets we have comparable or better generalization.

1 INTRODUCTION

Second order stochastic optimization methods adapt to loss curvature, allowing for smaller parameter update steps in regions where the gradient changes quickly, avoiding bouncing behavior, and larger ones in flat regions. Traditionally, they required storing and inverting the Hessian to update parameters: this requires quadratic memory and cubic computation in the number of parameters. Thus, methods using only a diagonal Hessian/Fisher approximation [Duchi et al., 2011, Kingma and Ba, 2015] have dominated the field. However, diagonal preconditioners only scale gradients in the canonical basis, while full preconditioners can potentially perform scaling in a rotated basis aligning more closely with loss curavture.

Recently, Shampoo [Gupta et al., 2018, Anil et al., 2020] proposed approximating the (empirical) Fisher matrices using Kronecker factorized matrices. The matrix version of Shampoo factorizes the full preconditioner matrix into left and right Kronecker factors, which allows storing and inverting the smaller factors instead of the full matrix. For parameters $\mathbf{W} \in \mathbb{R}^{m \times n}$, this reduces computation costs from $O(m^3n^3)$ to $O(m^3 + n^3)$ and storage costs from $O(m^2n^2)$ to $O(m^2 + n^2)$. AdaGrad [Duchi et al., 2011] uses regret bound techniques based on Online Mirror Descent (OMD) Srebro et al. [2011] designed for vector updates. To use these techniques for matrix/tensor updates, [Gupta et al., 2018] exploit domination results that relate vector update preconditioners to their matrix/tensor counterparts. However, Shampoo still requires inverse matrix roots, which are numerically unstable or inaccurate for ill-conditioned matrices in 32-bit precision. For preconditioned gradient descent, it is important for Shampoo to maintain the accuracy of the smallest eigenvalues (largest when inverted). It thus needs 64-bit precision, which requires some combination of slow TPU-CPU data transfers, stale preconditioner matrices, or even new machine learning accelerator hardware [Anil et al., 2020] supporting fast 64-bit matrix multiplication or fast accurate eigendecomposition.

A primary motivator to use any optimizer is to reach the same quality solution in less time or reach a better solution that other optimizers fail to reach. Second order optimizers are currently not as popular as first order methods due to: a) inertia to adoption, with a lack of highly optimized implementations in all major ML frameworks; b) the added compute and memory requirements; c) numerical stability and consequently the additional considerations required to get them to work or to debug them (e.g. numerical linear algebra, computer number formats); and d) even though they sometimes reach a solution unreachable by 1st order methods (or the same solution with lower wall clock time

[2]Funded by and conducted at Luminous Computing.

(WCT) if properly optimized), they don't consistently for every task or architecture.

The key tradeoff is (b, c) vs (d). Shukla [2022] of Weights and Biases noted that some of their customers using Shampoo do find solutions that generalize better than those found with ADAM for their real world tasks (d), but that 2nd order optimizers are still more expensive (b) and have additional considerations (c).

In this paper, we address these limitations (c) by introducing a novel factorization, Kronecker Approximation-Domination (KrAD): it has a simple form that updates the preconditioning matrix without explicitly inverting it. Shampoo constructs Kronecker factors of intermediate statistics such that their Kronecker product dominates the gradient outer product matrix. Our key idea is to construct factors for those statistics such that the *inverse* of their Kronecker product dominates the gradient outer product matrix. This leads to preconditioners that require fractional powers rather than inverse fractional powers of factors. While this does not decrease the computational complexity compared to Shampoo, it avoids needing 64-bit precision.

This paper has three primary contributions: 1) we introduce two new algorithms, both with $O(m^3+n^3)$ and $O(m^2+n^2)$ computational and memory complexity, respectively and only requiring positive matrix roots, in contrast to previous work requiring inverse matrix roots; 2) we show domination properties and use them to prove that our algorithm, which has similar computation cost to Shampoo, achieves optimal regret; 3) we show empirically that in 32-bit precision, we outperform Shampoo in synthetic experiments and perform similarly on some real experiments.We first describe some mathematical tools, set up the problem, and describe second order optimization and established results related to our method in Section 2. Then we present our method and its theoretical properties in Section 3. Next, we consider the practical implementation of our method in Section 4. Then, we show empirical results in Section 5. Finally, we discuss implications in Section 6.

2 BACKGROUND AND RELATED WORK

Here, we set up the notation and the problem and then describe relevant related works. We briefly describe optimization for vector-valued parameters then extend the discussion to matrix-valued parameters. We can further generalize to tensors, but continue with matrices in the main text for clarity and leave the tensor formulation for Appendix C.5.

2.1 NOTATION AND PRELIMINARIES

We use bold lower case letters to denote column vectors (e.g. $\mathbf{g} \in \mathbb{R}^n$), bold upper case letters to denote matrices (e.g. $\mathbf{G} \in \mathbb{R}^{m\times n}$), and calligraphic letters to denote matrices composed of stacking vectors of interest (e.g. $\mathcal{G}_k = \begin{pmatrix}\mathbf{g}_k & \mathbf{g}_{k-1} & \dots & \mathbf{g}_1\end{pmatrix} \in \mathbb{R}^{n\times k}$). For square matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, let the trace be $\operatorname{tr}(\mathbf{A}) = \sum_{i=1}^n a_{i,i}$, where $a_{i,j}$ denotes the element in row i and column j of $\mathbf{A}$. For matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m\times n}$, let $\mathbf{A}\cdot\mathbf{B} = \operatorname{tr}(\mathbf{A}^\top\mathbf{B})$ be the matrix (Frobenius) inner product, and the induced Frobenius norm $\|\mathbf{A}\|_F = (\mathbf{A}\cdot\mathbf{A})^{1/2}$. We use $\|\mathbf{A}\|_2$ to denote the spectral norm, the largest singular value of $\mathbf{A}$. We write $\mathbf{A} \succeq 0$ to mean $\mathbf{A}$ is symmetric positive semi-definite (PSD), while $\mathbf{A} \succ 0$ means $\mathbf{A}$ is symmetric positive definite (PD). For two PSD matrices $\mathbf{B} \succeq \mathbf{A}$ means that $\mathbf{B} - \mathbf{A} \succeq 0$ (similarly for $\succ$). For a PSD matrix, take $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top$ to be the eigenvalue decomposition, which results in orthonormal $\mathbf{V}$ (i.e. $\mathbf{V}^{-1} = \mathbf{V}^\top$). Define $f(\mathbf{\Lambda})$ such that the diagonal elements are $(f(\mathbf{\Lambda}))_{i,i} = f(\lambda_i)$, the principal values of the function applied to the scalar eigenvalues. Then we take $f(\mathbf{A}) = \mathbf{V}f(\mathbf{\Lambda})\mathbf{V}^\top$. In this way, we define a unique value for functions applied to PSD matrices with eigenvalues within the domain of the function. In particular, we have a definition for real powers of PD matrices.

Let $\otimes$ denote the Kronecker product, for matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{q\times r}$ defined as

$$\mathbf{B}\otimes\mathbf{A} = \begin{pmatrix} b_{1,1}\mathbf{A} & \dots & b_{1,r}\mathbf{A} \\ \vdots & \ddots & \vdots \\ b_{q,1}\mathbf{A} & \dots & b_{q,r}\mathbf{A}\end{pmatrix} \in \mathbb{R}^{mq\times nr}.$$

Let the vectorization operation for a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ be

$$\operatorname{vec}(\mathbf{A}) = \begin{pmatrix}\mathbf{a}_1^\top & \dots & \mathbf{a}_n^\top\end{pmatrix}^\top \in \mathbb{R}^{mn}:$$

$\mathbf{a}_i$ is the i-th column of $\mathbf{A}$, and the corresponding inverse vectorization for $\mathbf{a} \in \mathbb{R}^{mn}$ given a target matrix in $\mathbb{R}^{m\times n}$ is

$$\operatorname{vec}^{-1}_{m,n}(\mathbf{a}) = \begin{pmatrix}\mathbf{a}_{1:m} & \dots & \mathbf{a}_{m(n-1)+1:mn}\end{pmatrix} \in \mathbb{R}^{m\times n},$$

where $\mathbf{a}_{i:j} = \begin{pmatrix}a_i & \dots & a_j\end{pmatrix}^\top$.

Several properties [Bellman, 1980, Van Loan, 2000, Boyd and Vandenberghe, 2004, Baumgartner, 2011] of trace and Kronecker products are important for our results. We list them in Appendix D due to space constraints.

2.2 OPTIMIZATION IN MACHINE LEARNING

We are interested in iterative empirical risk minimization under loss f, with $\mathbf{x} \sim p_n(x)$ an empirical density and parameters $\mathbf{w} \in \mathbb{R}^N$

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \mathbb{E}_{p_n}[f(\mathbf{w}, \mathbf{x})].$$

We assume access to gradients $\nabla_{\mathbf{w}} f$. Let $\mathbf{g}_k = \sum_{i\in B_k} \nabla_{\mathbf{w}} f(\mathbf{w}_k, \mathbf{x}_i) \in \mathbb{R}^N$ be the estimated gradient of f w.r.t. $\mathbf{w}$ evaluated at $\mathbf{w}_k$ with data from batch B_k at iteration k. From here, we omit but imply "stochastic" gradients.

For step size $\eta_k \in \mathbb{R}^+$, gradient-based methods update

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta_k \mathbf{P}_k \mathbf{g}_k \tag{1}$$

where $\mathbf{P}_k \in \mathbb{R}^{N \times N}$ is a *preconditioner* matrix (some sources instead refer to $\mathbf{P}_k^{-1}$ as the preconditioner). In some algorithms, additional intermediate *statistics* are stored and updated to aid preconditioner computation. Gradient descent uses $\mathbf{P}_k = \mathbf{I}_N$ (no preconditioning); Newton's method takes $\mathbf{P}_k = \mathbf{H}_k^{-1}$ to be the (pseudo)inverse of the Hessian.

While vanilla gradient descent updates are trivial to compute, convergence can require many iterations. Newton updates are more expensive, but may require far fewer iterations. In practice, the chosen form of the preconditioner matrix appears to exist along a trade-off between computational tractability and improved convergence properties.

2.3 ADAPTIVE GRADIENT PRECONDITIONERS

We can collect gradients through iteration k,

$$\mathcal{G}'_k = \begin{pmatrix} \mathbf{g}_k & \mathbf{g}_{k-1} & \dots & \mathbf{g}_i & \dots & \mathbf{g}_1 \end{pmatrix} \in \mathbb{R}^{N \times k}$$

and augment this with a scaled identity,

$$\mathcal{G}_k = \begin{pmatrix} \mathcal{G}'_k & \epsilon \mathbf{I}_N \end{pmatrix} \in \mathbb{R}^{N \times (k+N)}.$$

One form of adaptive gradient update is

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta_k \left(\mathcal{G}_k \mathcal{G}_k^\top \right)^{-1/2} \mathbf{g}_k.$$

Expressing this in terms of the non-augmented $\mathcal{G}'_k$ and taking $\delta = \epsilon^2$ gives the full version of AdaGrad [Duchi et al., 2011]: $\mathcal{G}_k \mathcal{G}_k^\top$ can be seen as the statistic that is stored and updated in each iteration, and $\left(\delta \mathbf{I}_N + \mathcal{G}'_k \mathcal{G}_k'^\top \right)^{-1/2}$ is the preconditioner computed from the statistic. Unfortunately, storing the full matrix $\mathcal{G}_k \mathcal{G}_k^\top$ is memory intensive at $O(N^2)$, and taking the inverse square root is computationally expensive at $O(N^3)$ to compute the SVD.

Diagonal AdaGrad reduces computational complexity

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta_k \left(\delta \mathbf{I}_N + \operatorname{diag}\left(\mathcal{G}'_k \mathcal{G}_k'^\top \right) \right)^{-1/2} \mathbf{g}_k.$$

This is $O(N)$ complexity in both memory and computation.

2.4 MATRIX VARIABLES AND SHAMPOO

Now, we consider $N = mn$ and optimize w.r.t. a matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$. We consider a single matrix for clarity, but note that the derivations and analyses can be extended to tensors and applied individually to each tensor-valued parameter in a given model (e.g. to compute the total costs). Then we can use the same optimization framework, now taking $\mathbf{w} = \operatorname{vec}(\mathbf{W}) \in \mathbb{R}^{mn}$. However, in forming a preconditioner, we utilize the fact that our parameter now has the additional structure of being a matrix. Consider

$$\mathbf{G}_k = \operatorname{vec}_{m,n}^{-1}(\mathbf{g}_k).$$

One convenient factorized form of a preconditioner is

$$\mathbf{P}_k = \mathbf{R}_k \otimes \mathbf{L}_k,$$

where $\mathbf{L}_k \in \mathbb{R}^{m \times m}$ and $\mathbf{R}_k \in \mathbb{R}^{n \times n}$ are symmetric. This reduces storage and computation while not necessarily being low-rank. To see this, we return to Equation (1) and simplify

$$\begin{aligned} \mathbf{w}_{k+1} &= \mathbf{w}_k - \eta_k \mathbf{P}_k \mathbf{g}_k \\ &= \mathbf{w}_k - \eta_k \operatorname{vec}(\mathbf{L}_k \mathbf{G}_k \mathbf{R}_k) \qquad \because \text{(P18)} \\ \Rightarrow \mathbf{W}_{k+1} &= \mathbf{W}_k - \eta_k \mathbf{L}_k \mathbf{G}_k \mathbf{R}_k. \qquad \because (\operatorname{vec}_{m,n}^{-1}) \end{aligned}$$

This requires $O(m^2 + n^2) = O\left(N\left(\frac{m}{n} + \frac{n}{m}\right)\right)$ storage and $O(N(m+n))$ compute. Unless otherwise specified, we assume w.l.o.g. that $m \leq n$. Shampoo [Gupta et al., 2018] tracks statistics

$$\mathbf{B}_k = \epsilon \mathbf{I}_m + \sum_{i=1}^{k} \mathbf{G}_i \mathbf{G}_i^\top, \quad \mathbf{C}_k = \epsilon \mathbf{I}_n + \sum_{i=1}^{k} \mathbf{G}_i^\top \mathbf{G}_i, \quad (2)$$

and forms the preconditioner from the Kronecker factors

$$(\mathbf{L}_k, \mathbf{R}_k) = (\mathbf{B}_k^{-1/4}, \mathbf{C}_k^{-1/4}). \quad (3)$$

Gupta et al. [2018] relies on 3 key conditions to prove Shampoo achieves optimal regret. First,

$$\mathbf{R}_k^{-2} \otimes \mathbf{L}_k^{-2} \succeq \epsilon \mathbf{I}_{mn} + \sum_{i=1}^{k} \mathbf{g}_i \mathbf{g}_i^\top. \quad (4)$$

It secondly requires that

$$\mathbf{R}_k^{-2} \otimes \mathbf{L}_k^{-2} \succeq \mathbf{R}_{k-1}^{-2} \otimes \mathbf{L}_{k-1}^{-2}. \quad (5)$$

Finally it requires that under mild conditions,

$$\operatorname{tr}(\mathbf{L}_k),\ \operatorname{tr}(\mathbf{R}_k) = O(k^{1/4}). \quad (6)$$

An additional $O(n^3 + m^3)$ cost comes from taking the inverse fractional powers via a high-precision (64-bit) Newton iteration (which involves repeated matrix multiplications; see Equation (25) in Appendix C.3 for further details) or SVD, which dominates the previous $O(N(m+n))$.

The motivation for these properties stems from OMD analysis. The regret vector parameter $\mathbf{w}_k$ updates and general preconditioners $\mathbf{P}_k \succ 0$ is initially bounded by sums of quadratic forms $\mathbf{w}_k^\top \mathbf{P}_k \mathbf{w}_k$. If the domination property holds, these can be bounded in terms of $\operatorname{tr}(\mathbf{P}_k)$. Using Property (P13), this can be further expressed in terms of $\operatorname{tr}(\mathbf{L}_k)$ and $\operatorname{tr}(\mathbf{R}_k)$. The trace growth rates give the final bound.

2.5 RELATED WORK

Recently, there has been a surge in interest in tractable preconditioned gradient methods. We briefly contrast some of the most similar or otherwise notable methods to ours.

Kronecker Factored: KFAC [Martens and Grosse, 2015] and TNT [Ren and Goldfarb, 2021] use Kronecker factors to approximate Fisher matrices while reducing storage and computation costs. KFAC requires knowledge of network architecture and thus modifications or even re-implementations corresponding to each parametric layer type within the network. KFAC and TNT reqiore an additional backward pass and matrix inversion. KBFGS [Goldfarb et al., 2020] does not require matrix inversion but requires an additional forward and backward pass. We note that the empirical performance achieved by KBFGS is partially due to initialization using curvature estimated from the entire training set [Goldfarb et al., 2020], which is not available in a truly online setting. Shampoo is the most closely related work, relying on the empirical Fisher matrix rather than estimating the Fisher matrix, thus not requiring additional sampling or forward/backward passes. In addition, only tensor shape knowledge is required. While estimating the Fisher matrix may have intuitively desirable properties over the empirical Fisher, the empirical Fisher is more practical to compute [Martens, 2020], since in distributed data or model parallel settings, additional forward/backward passes become prohibitive (both in terms of computation and engineering cost). To our knowledge, Shampoo is the only second order optimizer that has been successfully implemented in a large-scale, production, deep learning setting [Anil et al., 2022], which makes it of primary interest.

Limited Memory: GGT [Agarwal et al., 2019] uses a limited history of h past gradients to form a low-rank approximation to the full AdaGrad matrix, reducing storage costs to $O(Nh)$ and compute costs to $O(Nh^2)$; however, this requires many copies (200, for the problems they consider) of the full gradient to be stored as statistics (h still scales as a function of N), which can become prohibitive without modifications to reduce N.

Sketching: AdaHessian [Yao et al., 2021], SENG [Yang et al., 2022], and SketchySGD [Frangella et al., 2022] estimate the Hessian (either the diagonal or a low-rank approximation) via automatic differentiation to compute Hessian-vector products (HVP), which require additional backpropagation steps and two batches of data per step, one for gradient computation and one for Hessian sketching. This is less expensive than a full forward/backward pass, but can still be expensive in distributed settings. In addition, the low-rank factorization still requires many times the storage of the full gradient of the model ($100-200\times$ in the problems they consider). While KrADagrad does not use sketching or HVP, these methods could potentially be combined with KrAD factorization as future work.

3 KRADAGRAD

Here, we derive a pair of new optimization algorithms, KrADagrad and KrADagrad*, presenting along the way intermediate results that allow us to attain domination properties analogous to those in Equations (4)-(5) and trace growth rates in Equation (6) required for good regret. Simultaneously, this maintains low computational complexity achieved by Kronecker factorized methods. To derive the algorithms, we present

1. KrAD, a method for producing a Kronecker factorization approximating a matrix that yields the property in Equation (4)
2. Derivation of the basic form of KrADagrad, applying to AdaGrad style updates KrAD and the Woodbury matrix identity Woodbury [1950] as the key tricks
3. Statements confirming the property in Equation (5)
4. Extension to KrADagrad*.

KrADagrad alternates updates of Kronecker factors of the statistics and has within ε tolerance of optimal regret. With additional insights from KrADagrad regret analysis, we formulate KrADagrad*, which can be seen as an "average" of two KrADagrad estimators. KrADagrad* updates both Kronecker factors of the statistics simultaneously and achieves optimal regret. We present theoretical results along the way in this section as they are needed but defer the proofs to Appendix B. We derive the algorithm for matrix-valued parameters for clarity and again leave the extension to tensor-valued parameters for Appendix C.5.

3.1 KRONECKER APPROXIMATION-DOMINATION

First, we state a Lemma as the goal of KrAD, which is needed to achieve the condition in Equation (4) that allows us to prove optimal regret.

Lemma 3.1. *Let PD matrix $\mathbf{C} \in \mathbb{R}^{n\times n}$, $\mathcal{U} \in \mathbb{R}^{mn\times r}$, $\mathbf{u}_i$ be the i-th column of $\mathcal{U}$, $\mathbf{U}_i = vec^{-1}_{m,n}(\mathbf{u}_i)$. Then*

$$\mathbf{B} = \sum_{i=1}^{r} \mathbf{U}_i \mathbf{C}^{-1} \mathbf{U}_i^\top \succeq 0$$
$$\Rightarrow \mathbf{C} \otimes \mathbf{B} \succeq \frac{1}{n} \mathcal{U}\mathcal{U}^\top .$$

These matrices $\mathbf{U}$ and $\mathcal{U}$ are fairly general. In our setting, we will use KrAD on gradient matrices. In context, this result states that given a PD right matrix, we can express a left matrix in a quadratic form, and the Kronecker product of the right and left matrices will dominate the scaled gradient outer product matrix. We present the proof in Appendix B.1.

3.2 KRADAGRAD UPDATES: DERIVATION

We start with deriving the general statistic update. We outline it here and fill in details in Appendix D. Suppose we have the previous Kronecker factorization of a statistic $\mathbf{Q}_{k-1}$ that dominates the gradient outer product matrix by a factor t (which we will clarify later) and its inverse $\mathbf{P}_{k-1}$

$$\mathbf{Q}_{k-1} = \mathbf{C}_{k-1} \otimes \mathbf{B}_{k-1} \succeq \frac{1}{nt} \mathcal{G}_{k-1} \mathcal{G}_{k-1}^\top \quad (7)$$

$$\mathbf{P}_{k-1} = \mathbf{R}_{k-1} \otimes \mathbf{L}_{k-1} = \mathbf{C}_{k-1}^{-1} \otimes \mathbf{B}_{k-1}^{-1} = \mathbf{Q}_{k-1}^{-1}, \quad (8)$$

where $\mathbf{B}_{k-1}$, $\mathbf{C}_{k-1}$, $\mathbf{L}_{k-1}$, and $\mathbf{R}_{k-1}$ are PD. Our initial intermediate update, which we will apply our KrAD factorization to, is

$$\widetilde{\mathbf{Q}}_k = \tfrac{1}{nt_k}\mathbf{g}_k\mathbf{g}_k^\top + \mathbf{Q}_{k-1},$$

for some $t_k \leq t$. We will then compute an intermediate version of the update on the inverse of the left statistic $\widetilde{\mathbf{B}}_k = \widetilde{\mathbf{L}}_k^{-1}$. Letting $\widetilde{\mathbf{B}}_k = \mathbf{B}_{k-1} + \Delta\mathbf{B}_k$, we can apply KrAD to $\widetilde{\mathbf{Q}}_k$ with a fixed $\mathbf{C}_k = \mathbf{C}_{k-1}$ (i.e., we use the old right statistic to update the current left one),

$$\Delta\mathbf{B}_k = \frac{1}{t_k}\mathbf{G}_k\mathbf{C}_k^{-1}\mathbf{G}_k^\top \tag{9}$$

$$\Rightarrow \mathbf{C}_k \otimes \Delta\mathbf{B}_k \succeq \frac{1}{nt_k}\mathbf{g}_k\mathbf{g}_k^\top \qquad \because \text{Lemma 3.1} \tag{10}$$

$$\Rightarrow \mathbf{C}_k \otimes (\mathbf{B}_{k-1} + \Delta\mathbf{B}_k) \succeq \frac{1}{nt}\mathcal{G}_k\mathcal{G}_k^\top \qquad \because \text{(P15)}. \tag{11}$$

Then, letting $\widehat{\mathbf{C}}_k = \mathbf{C}_k + \frac{1}{t_k}\mathbf{G}_k^\top\mathbf{L}_{k\text{-}1}\mathbf{G}_k$ and applying the Woodbury matrix identity,

$$\widetilde{\mathbf{L}}_k = \widetilde{\mathbf{B}}_k^{-1} = (\mathbf{B}_{k-1} + \Delta\mathbf{B}_k)^{-1} \tag{12}$$

$$\succeq \underbrace{\mathbf{L}_{k\text{-}1} - \frac{1}{t_k}\mathbf{L}_{k\text{-}1}\mathbf{G}_k\mathbf{R}_k\mathbf{G}_k^\top\mathbf{L}_{k\text{-}1}}_{\mathbf{L}_k} \tag{13}$$

(we provide more detail in Appendix D). Note that our update for $\mathbf{L}_k$ neither depends on $\mathbf{B}$, $\mathbf{C}$ nor requires any other expensive matrix inverses to compute. This suggests that we do not need to actually store $\mathbf{B}$ or $\mathbf{C}$ to obtain a computationally tractable implementation.

Here, we state intermediate results that suggest that our proposed updates are reasonable (proof in Appendix B.2).

Proposition 3.2. *Taking* $t_k = 1 + \|\mathbf{L}_{k-1}\mathbf{G}_k\mathbf{R}_{k-1}\mathbf{G}_k^\top\|_2$ *(or the looser but more computationally friendly* $t_k = 1 + \|\mathbf{L}_{k-1}\mathbf{G}_k\mathbf{R}_{k-1}\mathbf{G}_k^\top\|_F$*), the PSD matrix*

$$\mathbf{M}_k \triangleq \tfrac{1}{t_k}\mathbf{L}_{k-1}^{1/2}\mathbf{G}_k\mathbf{R}_{k-1}\mathbf{G}_k^\top\mathbf{L}_{k-1}^{1/2} \prec \mathbf{I}.$$

Corollary 3.3. *If* $\mathbf{L}_{k-1} \succ 0$*, the updated* $0 \prec \mathbf{L}_k \preceq \mathbf{L}_{k-1}$.

In Corollary 3.3, the second inequality $\mathbf{L}_k \preceq \mathbf{L}_{k-1}$ has the effect of not increasing the step size, while the first inequality $\mathbf{L}_k \succ 0$ guarantees that we do not reverse the direction of the gradient. These are both valued theoretical properties of useful preconditioners for avoiding divergent behavior. In practice, they may be lightly violated to great effect; for example, in Adam, it is technically possible to have increasing step size as shown by [Reddi et al., 2018]. The result from Corollary 3.3 also allows us to leverage existing techniques to bound the regret of our algorithm.

3.2.1 Update schemes

Thus far, we have glossed over the fact that we have actually only updated $\mathbf{L}_k$. We may update $\mathbf{R}_k$ in the same way; since we have already achieved domination by just updating $\mathbf{L}_k$, we are interested in the process for jointly updating $(\mathbf{L}_k, \mathbf{R}_k)$. Further, we have only hinted at a method for updating statistics; we must still compute the preconditioner.

In the next few subsections, we discuss these issues more concretely, proposing KrADagrad*, an algorithm that combines two sets of KrAD preconditioners to obtain optimal regret but requires updating two matrix statistics and involves higher order matrix roots. This algorithm is reminiscent of Shampoo, but avoids the numerical difficulty of inverting ill-conditioned matrices. We also propose, KrADagrad, a separate scheme that has suboptimal regret but is more intuitive, showing why we arrive at this form of update. Due to space constraints, we leave this to Appendix C.1.

3.3 KRADAGRAD*: COMBINING PRECONDITIONERS

Suppose we have two distinct sets of KrADagrad preconditioners. Here we overload notation a bit, holding the iteration k fixed and dropping it from the subscript, instead using the subscript to denote the index for the set of preconditioners to which each matrix belongs,

$$\mathbf{R}_1 \otimes \mathbf{L}_1 \succeq \mathcal{G}\mathcal{G}^\top, \qquad \mathbf{R}_2 \otimes \mathbf{L}_2 \succeq \mathcal{G}\mathcal{G}^\top,$$

recalling that $\mathcal{G} = \begin{pmatrix}\mathcal{G}' & \epsilon\mathbf{I}_N\end{pmatrix}$, the augmented matrix collecting the history of observed gradients. Using the matrix geometric mean for two matrices [Ando et al., 2004],

$$\begin{aligned} M_g(\mathbf{A},\mathbf{B}) &\triangleq \mathbf{A}^{1/2}(\mathbf{A}^{-1/2}\mathbf{B}\mathbf{A}^{-1/2})^{1/2}\mathbf{A}^{1/2} \\ &= \mathbf{A}(\mathbf{A}^{-1}\mathbf{B})^{1/2} \end{aligned}$$

and due to the geometry of the manifold of PD matrices [Bhatia, 2009],

$$\mathbf{R}_c \otimes \mathbf{L}_c \succeq \mathcal{G}\mathcal{G}^\top.$$

where $\mathbf{L}_c = M_g(\mathbf{L}_1, \mathbf{L}_2)$ and $\mathbf{R}_c = M_g(\mathbf{R}_1, \mathbf{R}_2)$ form another pair of KrAD estimates[1].

In this case, we may need to take additional square roots of the quantities $(\mathbf{L}_1^{-1}\mathbf{L}_2, \mathbf{R}_1^{-1}\mathbf{R}_2)$ and of $(\mathbf{L}_c, \mathbf{R}_c)$ themselves. We will first show how these combined estimators are relevant to Shampoo, and use this insight to arrive at a second version of preconditioning, which we call KrADagrad*.

3.3.1 Shampoo combines inverse KrAD estimates

If we keep a pair of KrAD estimates for $\mathbf{Q}$ (an integral power of the preconditioner inverse) instead of directly for $\mathbf{P}$, one in which we only update $\mathbf{B}_1$ and keep $\mathbf{C}_1 = \mathbf{I}$ fixed, while in the other we only update $\mathbf{C}_2$ and keep $\mathbf{B}_2 = \mathbf{I}$ fixed, we end up with exactly the Shampoo statistics updates

$$\Delta\mathbf{B}_1 = \mathbf{G}\mathbf{C}_1^{-1}\mathbf{G}^\top = \mathbf{G}\mathbf{G}^\top$$
$$\Delta\mathbf{C}_2 = \mathbf{G}^\top\mathbf{B}_2^{-1}\mathbf{G} = \mathbf{G}^\top\mathbf{G}.$$

[1]the subscript c stands for "combined"

Algorithm 1 KrADagrad*: Precondition *without* inversion

```
Input: Parameters $\mathbf{W}_0 \in \mathbb{R}^{m\times n}$, iterations $K$, step size $\eta$, exponent $\alpha = 1/2$.
Initialize: $(\mathbf{L}_0, \mathbf{R}_0) = (\mathbf{I}_m, \mathbf{I}_n)$
for $k = 1, \dots, K$ do
    Obtain gradient $\mathbf{G}_k$
    Compute $\Delta\mathbf{L}_k = \frac{1}{t_{k,L}}\mathbf{L}_{k-1}\mathbf{G}_k\mathbf{G}_k^\top\mathbf{L}_{k-1}$
    Compute $\Delta\mathbf{R}_k = \frac{1}{t_{k,R}}\mathbf{R}_{k-1}\mathbf{G}_k^\top\mathbf{G}_k\mathbf{R}_{k-1}$
    Update $(\mathbf{L}_k, \mathbf{R}_k) \leftarrow (\mathbf{R}_{k-1} - \Delta\mathbf{R}_k, \mathbf{L}_{k-1} - \Delta\mathbf{L}_k)$
    Compute $(\mathbf{L}_k^{\alpha/2}, \mathbf{R}_k^{\alpha/2})$ from $(\mathbf{L}_k, \mathbf{L}_{k-1}^{\alpha/2}, \mathbf{R}_k, \mathbf{R}_{k-1}^{\alpha/2})$
    Apply preconditioned gradient step
        $\mathbf{W}_k = \mathbf{W}_{k-1} - \eta\mathbf{L}_k^{\alpha/2}\mathbf{G}_k\mathbf{R}_k^{\alpha/2}$
end for
```

Then, the full Shampoo statistics matrices are related to a combination of these two statistics,

$$\mathbf{B}_c = M_g(\mathbf{B}_1, \mathbf{I}) = \mathbf{B}_1^{1/2}$$
$$\mathbf{C}_c = M_g(\mathbf{I}, \mathbf{C}_2) = \mathbf{C}_2^{1/2},$$

These still need to be inverse square rooted to be applied as the preconditioner, since here we have constructed

$$\mathbf{C}_c \otimes \mathbf{B}_c \succeq \mathcal{G}\mathcal{G}^\top.$$

The inverse square root ultimately gets grouped together with the square roots in the expressions above to yield the inverse $1/4$-th power in the preconditioning,

$$\begin{aligned}\mathbf{W}_k &= \mathbf{W}_{k-1} - \eta\mathbf{B}_c^{-1/2}\mathbf{G}_k\mathbf{C}_c^{-1/2}\\ &= \mathbf{W}_{k-1} - \eta\mathbf{B}_1^{-1/4}\mathbf{G}_k\mathbf{C}_2^{-1/4}\end{aligned}$$

(compare with Equations (2)-(3)). Each individual estimator keeps one Kronecker factor as identity.

3.3.2 KrADagrad*

Inspired by Shampoo's optimality, we can maintain a pair of KrADagrad preconditioners $(\mathbf{L}_{k,1}, \mathbf{R}_{k,1}) = (\mathbf{L}_k, \mathbf{I}_m)$ and $(\mathbf{L}_{k,2}, \mathbf{R}_{k,2}) = (\mathbf{I}_n, \mathbf{R}_k)$, where the second index in the LHS subscripts denotes the estimator. Since we will hold the identity matrices constant, we do not need to store or perform multiplication with them explicitly. In addition, this means we can unambiguously drop the second index in the subscripts. On iteration k, we update both $\mathbf{L}_k$ and $\mathbf{R}_k$, just as in shampoo. With

$$t_{k,L} \leftarrow 1 + \|\mathbf{G}_k\mathbf{G}_k^\top\mathbf{L}_{k-1}\|_F$$
$$t_{k,R} \leftarrow 1 + \|\mathbf{G}_k^\top\mathbf{G}_k\mathbf{R}_{k-1}\|_F,$$

we summarize in Algorithm 1. In order to bound the regret, we need a few intermediate results.

Lemma 3.4. *Assume a 1-Lipschitz loss, implying $\|\mathbf{G}_k\|_2 \leq 1\forall k$. Suppose by iteration k, $\mathbf{L}_k$ is updated in $k - k'$ of those steps for $0 \leq k' \leq k$ (and thus $\mathbf{R}_k$ is updated in the remaining k' steps). Letting $\mathbf{B}_k = \mathbf{L}_k^{-1}$, $\mathbf{C}_k = \mathbf{R}_k^{-1}$, and $s = 1 + \|\mathbf{L}_0\|_2\|\mathbf{R}_0\|_2$,*

$$tr(\mathbf{B}_k) \leq tr(\mathbf{B}_0) + k'ms\|\mathbf{R}_0\|_2 \tag{14}$$
$$tr(\mathbf{C}_k) \leq tr(\mathbf{C}_0) + (k-k')ns\|\mathbf{L}_0\|_2. \tag{15}$$

Next, we restate Theorem 7 from Gupta et al. [2018] in our notational setting. We additionally make one minor but straightforward substitution of the smaller matrix dimension instead of the rank, as the dimensions upper bound the rank.

Lemma 3.5 (From Gupta et al. [2018]). *Let $\mathbf{w}_* \in \mathbb{R}^{mn}$ with $m \geq n$, $t > 0$, and $D \triangleq \max_{1\leq k\leq K}\|\mathbf{w}_k - \mathbf{w}_*\|_2$. The regret from using a Kronecker factorized preconditioner*

$$\mathbf{P}_k = \mathbf{C}_k^{-1/2} \otimes \mathbf{B}_k^{-1/2}$$

that dominates the empirical Fisher matrix

$$\mathbf{C}_k \otimes \mathbf{B}_k \succeq \tfrac{1}{nt}\mathcal{G}_k\mathcal{G}_k^\top$$

is bounded

$$\textstyle\sum_{k=1}^K (f_k(\mathbf{w}_k) - f_k(\mathbf{w}_*)) \leq D\sqrt{2tn}\, tr(\mathbf{B}_K^{1/2})\, tr(\mathbf{C}_K^{1/2})$$

Now, we have our regret result: a proof is in Appendix B.4.

Theorem 3.6. *Assuming a 1-Lipschitz loss, the regret from using KrADagrad* scales as $O(\sqrt{K})$.*

4 IMPLEMENTATION

Now we discuss the algorithmic considerations for actually implementing KrADagrad*[2]. The main difficulty lies in efficiently computing the matrix roots. Differentiable matrix square roots for machine learning have been the subject of a substantial amount of research [Song et al., 2021]. For preconditioning, we do not require differentiability for our roots, so they can be computed using numerical linear algebra methods (e.g. SVD) without concern for the backward pass. While such algorithms are gaining hardware support on current GeMM accelerators and software frameworks, they are still not universal (Pytorch supports SVD on CUDA in double precision via cuSOLVER and MAGMA [PyTorch Team, 2021, NVidia Team, 2023, Dongarra et al., 2014], but TPUs to our knowledge do not [Jouppi et al., 2021]).

However, as [Anil et al., 2020] discovered, accurate inverse powers require high precision to retain the important contributions of the eigenvectors corresponding to the smallest eigenvalues, and current general matrix multiplication (GeMM) accelerators do not prioritize 64-bit computation (i.e. GPUs see significant speed reductions, while TPUs do not support it at all). Plus, they assume preconditioners are a slowly-varying sequence of matrices. Thus, they propose computing matrix roots iteratively on CPU due to these algorithmic and hardware architectural constraints.

[2]http://github.com/jonathanmei/kradagrad

In contrast, Kradagrad⋆ deals with positive roots, and both these iterative matrix methods and the numerical linear algebra techniques are otherwise actually amenable to computation using GeMM accelerators. Ultimately, as a straightforward solution for the roots that appear in KrADagrad⋆, the SVD on GPU suffices. We note that avoiding inversion is thus not about reducing the computational complexity, rather we aim to avoid needing 64-bit computation. Further details of matrix roots are discussed in Appendix C.3.

Additionally, diagonal damping is a common feature of preconditioned methods due to their contribution to numerical stability. While we do not need this for numerical stability, we find empirically that diagonal damping still helps reduce the effect of gradient noise and smooth out the loss curve. We describe how this can be applied to KrADagrad⋆ in further detail in Appendix C.4.

So far, our proposed algorithms and analyses apply to matrix parameters. As noted earlier, we can extend this to tensor parameters in a fairly straightforward manner as mainly a matter of additional notation and bookkeeping. Hence, we relegate the extension to Appendix C.5.

The baseline Pytorch Shampoo implementation we use is not optimized, so is not totally fair to the true capability of the baseline. Similarly, our KrADagrad variants, being derived from the mentioned implementation of Shampoo, did not have optimized implementations. Nonetheless, we have conducted some preliminary wall clock time comparisons between our implementation of KrADagrad and Shampoo, which we summarize in Table 1. For each data set, we report the time in seconds to run a single epoch for (KrADagrad, KrADagrad⋆, Shampoo) along with a 2 standard deviation interval, computed from epochs 5 through 10 on an NVidia A40 GPU.

Table 1: Comparison of wall clock times (in seconds) between our implementations of KrADagrad variants and Shampoo on a single epoch and a 2 standard deviation interval computed from 5 epochs for CIFAR-10/100 data sets on an NVidia A40 GPU.

Data set	KrADagrad	KrADagrad⋆	Shampoo
CIFAR-10	29.50± 0.88	45.76± 0.20	35.62 ± 0.17
CIFAR-100	54.71± 0.36	76.12± 1.72	66.45 ± 0.33

4.1 COMPUTE AND MEMORY COSTS

For Kradagrad, computing $\Delta \mathbf{L}_k$ has a computational cost of $(2N(2m+n)+2m^2+2m^3)$ While we still require $O(m^3+n^3)$ matrix square roots for the KrADagrad update, this is less difficult numerically and thus computationally than the -4th root required by Shampoo.

KrADagrad⋆ requires 4th root computation, which is comparable in cost to the inverse 4th root, but without the need for high precision.

Storage involves tracking the two matrix factors, and so is $O(m^2+n^2)$ for all methods.

We summarize these costs in Table 2.

Table 2: Comparison of complexity between our implementations of KrADagrad variants and Shampoo.

	KrADagrad	KrADagrad⋆	Shampoo
Compute	$O(m^3+n^3)$	$O(m^3+n^3)$	$O(m^3+n^3)$
Memory	$O(m^2+n^2)$	$O(m^2+n^2)$	$O(m^2+n^2)$

5 EXPERIMENTS

We address the following: 1) How sensitive is our method to matrix conditioning compared to Shampoo? 2) How does convergence speed compare in the number of training steps? (as measured by task-specific validation metrics) 3) How do our methods compare in the achieved model quality at or near convergence? The goal of each is to compare optimizers in various challenging loss landscapes, rather than to achieve state of the art performance.

To answer 1, in a synthetic experiment we minimize a multi-dimensional quadratic function with a non-diagonal, poorly-conditioned Hessian that neatly factorizes into a Kronecker product of two individually poorly-conditioned PD matrices.

To answer 2 and 3, we compare KrADagrad and KrADagrad⋆ to alternatives across a variety of tasks: image classification (IC), autoencoder problems (AE), recommendation (RecSys), continual learning (CL). For IC experiments we train ResNet-32/56 [He et al., 2016] without BatchNorm (BN) on CIFAR-10/100 [Krizhevsky, 2009]. For AE, we train simple autoencoders consistent with [Goldfarb et al., 2020] on MNIST [Lecun et al., 1998], as well as CURVE and FACES [Hinton and Salakhutdinov, 2006]. For recommendation, we train H+Vamp Gated [Kim and Suh, 2019] on MovieLens20M [Harper and Konstan, 2015]. In the continual learning setting we train on two benchmarks from [Lomonaco et al., 2021]: GEM [Lopez-Paz and Ranzato, 2017] on Permuted MNIST [Goodfellow et al., 2013] and LaMAML [Gupta et al., 2020] on Split CIFAR100 [Zenke et al., 2017].

We choose Shampoo as a baseline 2nd order optimizer as: a.) KrADagrad variants are most similar to Shampoo b.) we expect at best similar performance unless KrADagrad's approximations and lower precision are in practice detrimental. To ground all comparisons, we always include SGD, Adam, plus any unique optimizer from an existing benchmark.

In all our experiments, we initialized from common seeds across optimizers. However, the first points on the training curves follows the number of training steps per evaluation interval, so they do not visually appear to start from the

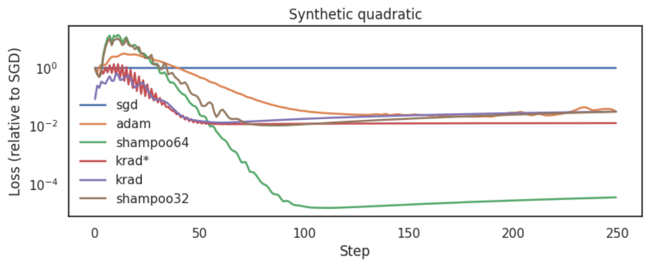

Figure 1: Loss on synthetic quadratic in log scale, relative to SGD. Each curve is *divided* by that of SGD.

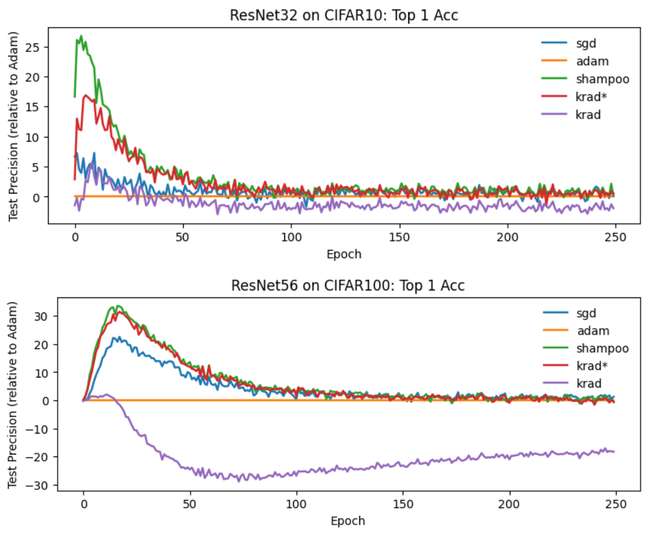

Figure 2: Vision experiments. Each curve is initialized from the same single seed and *subtracted* from that of Adam. Top: Top 1 Accuracy (in %) of ResNet-32 (without BN) on CIFAR-10, relative to Adam; Bottom: Top 1 Accuracy (in %) of ResNet-56 (without BN) on CIFAR-100, relative to Adam.

same point. Doing so would have required modifying the codebases separately for each experiment. In addition, ideally we would have had the resources to run multiple shared seeds for the selected HPs of every task we explored. For smaller tasks where convergence is achieved quickly, such as the autoencoder experiments, it was feasible for us to do this and we share this in Figures 3 and 8. In the tradeoff of how to use our compute and time resources, we opted to present fair evaluations of the optimizers (i.e. ensuring shared seeds and optimized HPs per optimizer per task) across a diversity of tasks, and repeatability statistics from different initial guesses as much as possible.

5.1 CONDITIONING EXPERIMENT

To see the effects of Hessian conditioning on optimizers of interest, we create a synthetic deterministic convex problem. For $\mathbf{X} \in \mathbb{R}^{128\times 128}$, we minimize a quadratic loss function

$$\min_{\mathbf{X}} \ \mathrm{tr}(\mathbf{X}^\top \mathbf{A}\mathbf{X}\mathbf{B})$$

where $\mathbf{A}, \mathbf{B} \succ 0$ are non-diagonal and have condition numbers $\kappa(\mathbf{A}) = \kappa(\mathbf{B}) = 10^{10}$. We seed each optimizer with the same starting point and sweep learning rates and pick the one with the lowest loss for each optimizer. We provide further details in Appendix G. While this stationary problem without a validation or test dataset is different from the online setting assumed in Theorem 3.6, and even from a typical offline machine learning problem, it helps isolate differences in behavior between optimizers in a badly conditioned convex loss landscape.

In Figure 1, while Shampoo in double precision outperforms the others in terms of final loss achieved, single precision KrADagrad* outperforms single precision Shampoo. Adam does not converge quite as quickly or to as good a loss, as the adaptivity it provides is aligned to the canonical axis, while $\mathbf{A}, \mathbf{B}$ being non-diagonal act in a rotated basis.

5.2 REAL DATASETS

For each task and optimizer we sweep hyperparameters (HP), select those yielding the best validation metric at some epoch or iteration, and display the corresponding learning curves in Figures 2 through 5. We set the preconditioner update rate for Shampoo and KrADagrad* to every 20 training steps. For all experiments, Shampoo uses double precision for negative matrix roots, and SGD includes momentum, unless stated otherwise. We summarize our observations here and provide additional detail in Appendix G.

For the three autoencoder experiments, after HP tuning we retrain the best HPs with 5 unique seeds shared across optimizers, with results in Figures 3 and 8. We also find the auto encoder tasks to be relevant baselines for evaluating Shampoo alternatives because Shampoo consistently outperforms SGD and Adam. While KrAdagrad* also outperforms SGD and Adam, Shampoo reaches better solutions or similar solutions in fewer steps. Following the synthetic experiments, we test the hypothesis that KrAdagrad* might perform better than 32-bit Shampoo by running our best HPs for Shampoo with single precision. The 32- and 64-bit versions perform similarly on these tasks, falsifying the hypothesis in the general sense. KrAdagrad, slower in the number of steps and performing worse in general, eventually reaches similar performance to KrAdagrad* on CURVES, with some runs reaching that of 32 bit Shampoo (see Figure 8b). One hypothesis is that on these particular tasks, effectively traversing the loss landscape requires eigenvectors that the KrADagrad approximation has more difficulty capturing than Shampoo.

While Gupta et al. [2018] performs experiments on “standard” machine learning tasks, their regret results pertain to an online setting. We include continual learning problems to test the limits of the theoretical setting. Figure 4a displays learning curves for GEM on Permuted MNIST relative to SGD to illuminate small differences (we include a plot of the actual accuracies in Appendix G). On average, KrADagrad

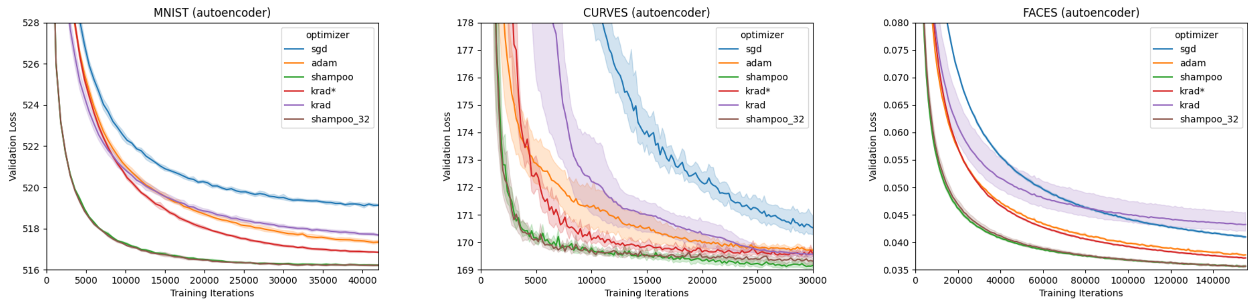

Figure 3: Reconstruction validation error for fully connected auto encoder a) mean cross entropy on MNIST b) mean cross entropy on CURVES c) mean squared error on FACES. The learning curves shown are averaged across 5 unique seeds with CI95 error bars. We do this analysis for the autoencoder experiments since they're relatively inexpensive to run.

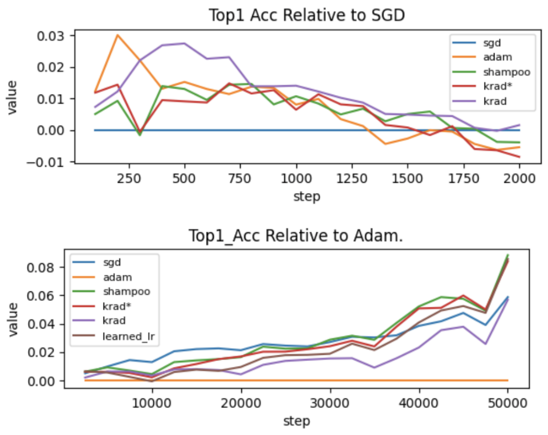

Figure 4: a) Top 1 Accuracy of GEM on Permuted MNIST, relative to SGD. Each curve averages multiple seeds, subtracted from the corresponding SGD curve to illuminate otherwise unobservable differences. See Appendix G for absolute curves. b) Top 1 Accuracy of LaMAML on Split CIFAR-100, relative to Adam.

stays slightly ahead of others throughout training, unlike KrADagrad⋆.

Figure 4b shows similar accuracy curves for LaMAML on Split CIFAR100 relative to Adam (again, the absolute accuracies are in Appendix G). KraADagrad⋆ closely follows Shampoo and the adaptive learning rate optimizer from [Gupta et al., 2020], while KrADagrad performs only better than Adam.

We also evaluate Shampoo and KrAdagrad variants against the baseline for H+Vamp Gated[3]. Shampoo reaches a lower training loss compared to KrAdagrad⋆, while both Adam variants converge faster and to lower loss values (Figure 5).

6 CONCLUSION

We introduced KrADagrad⋆, a preconditioned gradient optimizer that avoids matrix inversion, and proved that it

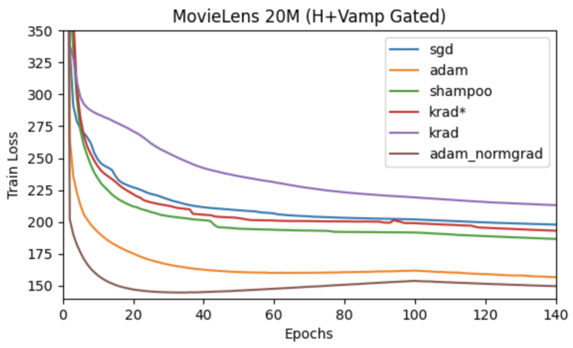

Figure 5: Training loss of H+Vamp Gated on ML-20M. We choose to compare the training loss for this experiment because it comprises a weighted combination of components, RE + $\beta\cdot$ KL, where β changes according to a predefined schedule (causing loss growth in the 1st 100 epochs), which is not consistent in the validation loss.

has optimal regret properties. We showed that its performance exceeds that of Shampoo in single precision on an ill-conditioned synthetic convex optimization problem. Our experiments on real datasets show that KrADagrad⋆ often performs similarly to Shampoo.

Future work includes tractable methods to compute low-rank updates to matrix roots, for instance with rational Krylov subspace methods, to improve the performance of KrADagrad⋆ and other Kronecker-factored preconditioners. It is also worth considering the application of the optimizer to other application areas, such as decision making and controls, or scientific discovery.

Acknowledgements

We thank Luminous Computing for funding this work. In particular, we thank David Baker for encouraging this line of investigation and David Scott for fruitful discussions about numerical linear algebra.

[3]https://github.com/psywaves/EVCF

References

Naman Agarwal, Brian Bullins, Xinyi Chen, Elad Hazan, Karan Singh, Cyril Zhang, and Yi Zhang. Efficient full-matrix adaptive regularization. In *International Conference on Machine Learning*, pages 102–110. PMLR, 2019.

T. Ando, Chi Kwong Li, and Roy Mathias. Geometric means. *Linear Algebra and Its Applications*, 385:305–334, 7 2004. ISSN 00243795. doi: 10.1016/J.LAA.2003.11.019.

Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Scalable second order optimization for deep learning. *arXiv preprint arXiv:2002.09018*, 2020.

Rohan Anil, Sandra Gadanho, Da Huang, Nijith Jacob, Zhuoshu Li, Dong Lin, Todd Phillips, Cristina Pop, Kevin Regan, Gil I. Shamir, Rakesh Shivanna, and Qiqi Yan. On the factory floor: ML engineering for industrial-scale ads recommendation models. ACM RecSys, 9 2022. doi: 10.48550/arxiv.2209.05310. URL `https://orsum.inesctec.pt/orsum2022/assets/files/paper6.pdf`.

Bernhard Baumgartner. An inequality for the trace of matrix products, using absolute values. 6 2011. doi: 10.48550/arxiv.1106.6189.

Richard Bellman. Some inequalities for positive definite matrices. In *General Inequalities 2*, pages 89–90. Springer, 1980.

Rajendra Bhatia. *Positive Definite Matrices*. Princeton University Press, 12 2009. ISBN 9781400827787. doi: 10.1515/9781400827787.

Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Jack Dongarra, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, and Ichitaro Yamazaki. Accelerating numerical dense linear algebra calculations with gpus. *Numerical Computations with GPUs*, pages 1–26, 2014.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7), 2011.

Zachary Frangella, Pratik Rathore, Shipu Zhao, and Madeleine Udell. SketchySGD: Reliable stochastic optimization via robust curvature estimates. 11 2022. doi: 10.48550/arxiv.2211.08597. URL `https://arxiv.org/abs/2211.08597v2`.

Donald Goldfarb, Yi Ren, and Achraf Bahamou. Practical quasi-newton methods for training deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2386–2396, 2020.

Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks, 2013. URL `https://arxiv.org/abs/1312.6211`.

Gunshi Gupta, Karmesh Yadav, and Liam Paull. Look-ahead meta learning for continual learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11588–11598. Curran Associates, Inc., 2020.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pages 1842–1850. PMLR, 2018.

F. Maxwell Harper and Joseph A. Konstan. The MovieLens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), dec 2015. ISSN 2160-6455. doi: 10.1145/2827872.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. URL `http://image-net.org/challenges/LSVRC/2015/`.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. doi: 10.1126/science.1127647. URL `https://www.science.org/doi/abs/10.1126/science.1127647`.

Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David Patterson. Ten Lessons From Three Generations Shaped Google's TPUv4i Industrial Product. 2021. doi: 10.1109/ISCA52012.2021.00010.

Daeryong Kim and Bongwon Suh. Enhancing VAEs for collaborative filtering. In *Proceedings of the 13th ACM Conference on Recommender Systems*. ACM, sep 2019. doi: 10.1145/3298689.3347015.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Vincenzo Lomonaco, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L. Hayes, Matthias De Lange, Marc Masana, Jary Pomponi, Gido van de Ven, Martin Mundt, Qi She, Keiland Cooper, Jeremy Forest, Eden Belouadah, Simone Calderara, German I. Parisi, Fabio Cuzzolin, Andreas Tolias, Simone Scardapane, Luca Antiga, Subutai Amhad, Adrian Popescu, Christopher Kanan, Joost van de Weijer, Tinne Tuytelaars, Davide Bacciu, and Davide Maltoni. Avalanche: an end-to-end library for continual learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2nd Continual Learning in Computer Vision Workshop, 2021. URL `https://github.com/ContinualAI/continual-learning-baselines`.

David Lopez-Paz and Marc' Aurelio Ranzato. Gradient episodic memory for continual learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21:1–76, 2020. ISSN 1533-7928.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning*, pages 2408–2417, 2015.

NVidia Team. cuSOLVER API Reference, January 2023. URL `https://docs.nvidia.com/cuda/cusolver`.

PyTorch Team. The torch.linalg module: Accelerated linear algebra with autograd in PyTorch | PyTorch, June 2021. URL `https://pytorch.org/blog/torch-linalg-autograd/`.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of Adam and beyond. *International Conference on Learning Representations*, 2018.

Yi Ren and Donald Goldfarb. Tensor normal training for deep learning models. *Advances in Neural Information Processing Systems*, 34:26040–26052, 12 2021.

Lavanya Shukla. Understanding the Landscape of the latest Large Models. *Advances in neural information processing systems*, 35, 2022.

Yue Song, Nicu Sebe, and Wei Wang. Why approximate matrix square root outperforms accurate SVD in global covariance pooling? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1115–1123, 2021.

Nati Srebro, Karthik Sridharan, and Ambuj Tewari. On the universality of online mirror descent. *Advances in neural information processing systems*, 24, 2011.

Charles F. Van Loan. The ubiquitous kronecker product. *Journal of Computational and Applied Mathematics*, 123:85–100, 11 2000. ISSN 03770427. doi: 10.1016/S0377-0427(00)00393-9.

Max A Woodbury. *Inverting modified matrices*. Statistical Research Group, 1950.

Minghan Yang, Dong Xu, Zaiwen Wen, Mengyun Chen, and Pengxiang Xu. Sketch-based empirical natural gradient methods for deep learning. *Journal of Scientific Computing*, 92:1–29, 9 2022. ISSN 15737691. doi: 10.1007/S10915-022-01911-X/METRICS.

Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael W Mahoney. AdaHessian: An adaptive second order optimizer for machine learning. 2021.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Improved multitask learning through synaptic intelligence. *CoRR*, abs/1703.04200, 2017. URL `http://arxiv.org/abs/1703.04200`.