Shampoo: Preconditioned Stochastic Tensor Optimization

Vineet Gupta [1] **Tomer Koren** [1] **Yoram Singer** [2 1]

Abstract

Preconditioned gradient methods are among the most general and powerful tools in optimization. However, preconditioning requires storing and manipulating prohibitively large matrices. We describe and analyze a new structure-aware preconditioning algorithm, called Shampoo, for stochastic optimization over tensor spaces. Shampoo maintains a set of preconditioning matrices, each of which operates on a single dimension, contracting over the remaining dimensions. We establish convergence guarantees in the stochastic convex setting, the proof of which builds upon matrix trace inequalities. Our experiments with state-of-the-art deep learning models show that Shampoo is capable of converging considerably faster than commonly used optimizers. Surprisingly, although it involves a more complex update rule, Shampoo's runtime per step is comparable in practice to that of simple gradient methods such as SGD, AdaGrad, and Adam.

1. Introduction

Over the last decade, stochastic first-order optimization methods have emerged as the canonical tools for training large-scale machine learning models. These methods are particularly appealing due to their wide applicability and their low runtime and memory costs.

A potentially more powerful family of algorithms consists of *preconditioned* gradient methods. Preconditioning methods maintain a matrix, termed a preconditioner, which is used to transform (i.e., premultiply) the gradient vector before it is used to take a step. Classic algorithms in this family include Newton's method, which employs the local Hessian as a preconditioner, as well as a plethora of quasi-Newton methods (e.g., Fletcher, 2013; Lewis and Overton, 2013; Nocedal, 1980) that can be used whenever second-order information is unavailable or too expensive to compute. Newer additions to this family are preconditioned online algorithms, most notably AdaGrad (Duchi et al., 2011), that use the covariance matrix of the accumulated gradients to form a preconditioner.

While preconditioned methods often lead to improved convergence properties, the dimensionality of typical problems in machine learning prohibits out-of-the-box use of full-matrix preconditioning. To mitigate this issue, specialized variants have been devised in which the full preconditioner is replaced with a diagonal approximation (Duchi et al., 2011; Kingma and Ba, 2014), a sketched version (Gonen and Shalev-Shwartz, 2015; Pilanci and Wainwright, 2017), or various estimations thereof (Erdogdu and Montanari, 2015; Agarwal et al., 2016; Xu et al., 2016). While the diagonal methods are heavily used in practice thanks to their favorable scaling with the dimension, the other approaches are seldom practical at large scale as one typically requires a fine approximation (or estimate) of the preconditioner that often demands super-linear memory and computation.

In this paper, we take an alternative approach to preconditioning and describe an efficient and practical apparatus that exploits the structure of the parameter space. Our approach is motivated by the observation that in numerous machine learning applications, the parameter space entertains a more complex structure than a monolithic vector in Euclidean space. In multiclass problems the parameters form a matrix of size $m \times n$ where m is the number of features and n is the number of classes. In neural networks, the parameters of each fully-connected layer form an $m \times n$ matrix with n being the number of input nodes and m is the number of outputs. The space of parameters of convolutional neural networks for images is a collection of 4 dimensional tensors of the form input-depth $\times$ width $\times$ height $\times$ output-depth. As a matter of fact, machine learning software tools such as Torch and TensorFlow are designed with tensor structure in mind.

Our algorithm, which we call Shampoo,[1] retains the tensor structure of the gradient and maintains a separate preconditioner matrix for each of its dimensions. An illustration of Shampoo is provided in Figure 1. The set of preconditioners

[1]Google Brain, Mountain View, CA, USA [2]Princeton University, Princeton, NJ, USA. Correspondence to: Tomer Koren <tkoren@google.com>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

[1]We call it Shampoo because it has to do with pre-conditioning.

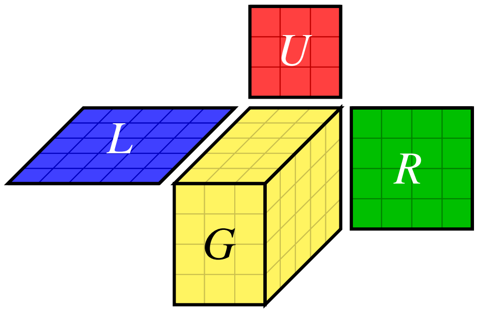

Figure 1. Illustration of Shampoo for a 3-dim tensor $G \in \mathbb{R}^{3\times4\times5}$.

Algorithm 1 Shampoo, matrix case.

```
Initialize $W_1 = \mathbf{0}_{m\times n}$ ; $L_0 = \epsilon I_m$ ; $R_0 = \epsilon I_n$
for $t = 1, \ldots, T$ do:
    Receive loss function $f_t : \mathbb{R}^{m\times n} \mapsto \mathbb{R}$
    Compute gradient $G_t = \nabla f_t(W_t)$        // $G_t \in \mathbb{R}^{m\times n}$
    Update preconditioners:
        $L_t = L_{t-1} + G_t G_t^{\mathsf{T}}$
        $R_t = R_{t-1} + G_t^{\mathsf{T}} G_t$
    Update parameters:
        $W_{t+1} = W_t - \eta L_t^{-1/4} G_t R_t^{-1/4}$
```

is updated by the algorithm in an online fashion with the second-order statistics of the accumulated gradients, similarly to AdaGrad. Importantly, however, each individual preconditioner is a full, yet moderately-sized, matrix that can be effectively manipulated in large scale learning problems.

While our algorithm is motivated by modern machine learning practices, in particular training of deep neural networks, its derivation stems from our analysis in a stochastic convex optimization setting. In fact, we analyze Shampoo in the broader framework of online convex optimization (Shalev-Shwartz, 2012; Hazan, 2016), thus its convergence applies more generally. Our analysis combines well-studied tools in online optimization along with off-the-beaten-path inequalities concerning geometric means of matrices. Moreover, the adaptation to the high-order tensor case is non-trivial and relies on extensions of matrix analysis to the tensor world.

We implemented Shampoo (in its general tensor form) in Python as a new optimizer in the TensorFlow framework (Abadi et al., 2016). Shampoo is extremely simple to implement, as most of the computations it performs boil down to standard tensor operations supported out-of-the-box in TensorFlow and similar libraries. Using the Shampoo optimizer is also a straightforward process. Whereas recent optimization methods, such as (Martens and Grosse, 2015; Neyshabur et al., 2015), need to be aware of the structure of the underlying model, Shampoo only needs to be informed of the tensors involved and their sizes. In our experiments with state-of-the-art deep learning models Shampoo is capable of converging considerably faster than commonly used optimizers. Surprisingly, albeit using more complex update rule, Shampoo's runtime per step is comparable to that of simple methods such as vanilla SGD.

1.1. Shampoo for matrices

In order to further motivate our approach we start with a special case of Shampoo and defer a formal exposition of the general algorithm to later sections. In the two dimensional case, the parameters form a matrix $W \in \mathbb{R}^{m\times n}$. First-order methods update iterates W_t based on the gradient $G_t = \nabla f_t(W_t)$, which is also an $m \times n$ matrix. Here, f_t is the loss function encountered on iteration t that typically represents the loss incurred over a single data point (or more generally, over a batch of data).

A structure-oblivious full-matrix preconditioning scheme would flatten the parameter space into an mn-dimensional vector and employ preconditioning matrices H_t of size $mn \times mn$. In contrast, Shampoo maintains smaller left $L_t \in \mathbb{R}^{m\times m}$ and right $R_t \in \mathbb{R}^{n\times n}$ matrices containing second-moment information of the accumulated gradients. On each iteration, two preconditioning matrices are formed from L_t and R_t and multiply the gradient matrix from the left and right respectively. The amount of space Shampoo uses in the matrix case is $m^2 + n^2$ instead of m^2n^2. Moreover, as the preconditioning involves matrix inversion (and often spectral decomposition), the amount of computation required to construct the left and right preconditioners is $O(m^3 + n^3)$, substantially lower than full-matrix methods which require $O(m^3n^3)$.

The pseudocode of Shampoo for the matrix case is given in Algorithm 1. To recap more formally, Shampoo maintains two different matrices: an $m \times m$ matrix $L_t^{1/4}$ to precondition the rows of G_t and $R_t^{1/4}$ for its columns. The $1/4$ exponent arises from our analysis; intuitively, it is a sensible choice as it induces an overall step-size decay rate of $O(1/\sqrt{t})$, which is common in stochastic optimization methods. The motivation for the algorithm comes from the observation that its update rule is equivalent, after flattening W_t and G_t, to a gradient step preconditioned using the Kronecker product of $L_t^{1/4}$ and $R_t^{1/4}$. The latter is shown to be tightly connected to a full unstructured preconditioner matrix used by algorithms such as AdaGrad. Thus, the algorithm can be thought of as maintaining a "structured" matrix which is implicitly used to precondition the flattened gradient, without either forming a full matrix or explicitly performing a product with the flattened gradient vector.

1.2. Related work

As noted above, Shampoo is closely related to AdaGrad (Duchi et al., 2011). The diagonal (i.e., element-wise) version of AdaGrad is extremely popular in practice and frequently applied to tasks ranging from learning large linear models to training of deep neural networks models. In contrast, the full-matrix version of AdaGrad analyzed in (Duchi et al., 2011) is rarely used in practice due to the prohibitive memory and runtime requirements associated with maintaining a full preconditioner. Shampoo can be viewed as an efficient, practical and provable apparatus for approximately and implicitly using the full AdaGrad preconditioner, without falling back to diagonal matrices.

Another recent optimization method that uses factored preconditioning is K-FAC (Martens and Grosse, 2015), which was specifically designed to optimize the parameters of neural networks. K-FAC employs a preconditioning scheme that approximates the Fisher-information matrix of a generative model represented by a neural network. The Fisher matrix of each layer in the network is approximated by a Kronecker product of two smaller matrices, relying on certain independence assumptions regarding the statistics of the gradients. K-FAC differs from Shampoo in several important ways. While K-FAC is used for training generative models and needs to sample from the model's predictive distribution, Shampoo applies in a general stochastic (more generally, online) optimization setting and comes with convergence guarantees in the convex case. K-FAC relies heavily on the structure of the backpropagated gradients in a feed-forward neural network. In contrast, Shampoo is virtually oblivious to the particular model structures and only depends on standard gradient information. As a result, Shampoo is also much easier to implement and use in practice as it need not be tailored to the particular model or architecture.

2. Background and technical tools

We use lowercase letters to denote scalars and vectors and uppercase letters to denote matrices and tensors. Throughout, the notation $A \succeq 0$ (resp. $A \succ 0$) for a matrix A means that A is *symmetric* and positive semidefinite (resp. definite), or PSD (resp. PD) in short. Similarly, the notations $A \succeq B$ and $A \succ B$ mean that $A - B \succeq 0$ and $A - B \succ 0$ respectively, and both tacitly assume that A and B are symmetric. Given $A \succeq 0$ and $\alpha \in \mathbb{R}$, the matrix A^α is defined as the PSD matrix obtained by applying $x \mapsto x^\alpha$ to the eigenvalues of A; formally, if we rewrite A using its spectral decomposition $\sum_i \lambda_i u_i u_i^\mathsf{T}$ in which (λ_i, u_i) is A's i'th eigenpair, then $A^\alpha = \sum_i \lambda_i^\alpha u_i u_i^\mathsf{T}$. We denote by $\|x\|_A = \sqrt{x^\mathsf{T} A x}$ the Mahalanobis norm of $x \in \mathbb{R}^d$ as induced by a positive definite matrix $A \succ 0$. The dual norm of $\|\cdot\|_A$ is denoted $\|\cdot\|_A^*$ and equals $\sqrt{x^\mathsf{T} A^{-1} x}$. The inner product of two matrices A and B is denoted as $A \bullet B = \mathrm{Tr}(A^\mathsf{T} B)$. The spectral norm of a matrix A is denoted $\|A\|_2 = \max_{x \neq 0} \|Ax\|/\|x\|$ and the Frobenius norm is $\|A\|_\mathsf{F} = \sqrt{A \bullet A}$. We denote by e_i the unit vector with 1 in its i'th position and 0 elsewhere.

2.1. Online convex optimization

We use Online Convex Optimization (OCO) (Shalev-Shwartz, 2012; Hazan, 2016) as our analysis framework. OCO can be seen as a generalization of stochastic (convex) optimization. In OCO a learner makes predictions in the form of a vector belonging to a convex domain $\mathcal{W} \subseteq \mathbb{R}^d$ for T rounds. After predicting $w_t \in \mathcal{W}$ on round t, a convex function $f_t : \mathcal{W} \mapsto \mathbb{R}$ is chosen, potentially in an adversarial or adaptive way based on the learner's past predictions. The learner then suffers a loss $f_t(w_t)$ and observes the function f_t as feedback. The goal of the learner is to achieve low cumulative loss compared to any fixed vector in the $\mathcal{W}$. Formally, the learner attempts to minimize its *regret*, defined as the quantity

$$\mathcal{R}_T = \sum_{t=1}^T f_t(w_t) - \min_{w \in \mathcal{W}} \sum_{t=1}^T f_t(w) ,$$

Online convex optimization includes stochastic convex optimization as a special case. Any regret minimizing algorithm can be converted to a stochastic optimization algorithm with convergence rate $O(\mathcal{R}_T/T)$ using an online-to-batch conversion technique (Cesa-Bianchi et al., 2004).

2.2. Adaptive regularization in online optimization

We next introduce tools from online optimization that our algorithms rely upon. First, we describe an adaptive version of Online Mirror Descent (OMD) in the OCO setting which employs time-dependent regularization. The algorithm proceeds as follows: on each round $t = 1, 2, \ldots, T$, it receives the loss function f_t and computes the gradient $g_t = \nabla f_t(w_t)$. Then, given a positive definite matrix $H_t \succ 0$ it performs an update according to

$$w_{t+1} = \operatorname*{argmin}_{w \in \mathcal{W}} \left\{ \eta g_t^\mathsf{T} w + \tfrac{1}{2} \|w - w_t\|_{H_t}^2 \right\} . \tag{1}$$

When $\mathcal{W} = \mathbb{R}^d$, Eq. (1) is equivalent to a preconditioned gradient step, $w_{t+1} = w_t - \eta H_t^{-1} g_t$. More generally, the update rule can be rewritten as a projected gradient step,

$$w_{t+1} = \Pi_{\mathcal{W}} \left[w_t - \eta H_t^{-1} g_t ; H_t \right],$$

where $\Pi_{\mathcal{W}}[z; H] = \mathrm{argmin}_{w \in \mathcal{W}} \|w - z\|_H$ is the projection onto the convex set $\mathcal{W}$ with respect to the norm $\|\cdot\|_H$. The following lemma provides a regret bound for Online Mirror Descent (see, e.g., Duchi et al., 2011).

Lemma 1. For any sequence of matrices $H_1, \ldots, H_T \succ 0$, the regret of online mirror descent is bounded above by,

$$\frac{1}{2\eta}\sum_{t=1}^{T}\left(\|w_t - w^\star\|_{H_t}^2 - \|w_{t+1} - w^\star\|_{H_t}^2\right) + \frac{\eta}{2}\sum_{t=1}^{T}\left(\|g_t\|_{H_t}^*\right)^2 .$$

In order to analyze particular regularization schemes, namely specific strategies for choosing the matrices $H_1, \ldots, H_T$, we need the following lemma, adopted from Gupta et al. (2017); for completeness, we provide a short proof in the full version of the paper (Gupta et al., 2018).

Lemma 2 (Gupta et al. (2017))**.** Let $g_1, \ldots, g_T$ be a sequence of vectors, and let $M_t = \sum_{s=1}^{t} g_s g_s^\mathsf{T}$ for $t \geq 1$. Given a function Φ over PSD matrices, define

$$H_t = \operatorname*{argmin}_{H \succ 0} \left\{ M_t \bullet H^{-1} + \Phi(H) \right\}$$

(and assume that a minimum is attained for all t). Then

$$\sum_{t=1}^{T}\left(\|g_t\|_{H_t}^*\right)^2 \leq \sum_{t=1}^{T}\left(\|g_t\|_{H_T}^*\right)^2 + \Phi(H_T) - \Phi(H_0) .$$

2.3. Kronecker products

We recall the definition of the Kronecker product, the vectorization operation and their calculus. Let A be an $m \times n$ matrix and B be an $m' \times n'$ matrix. The Kronecker product, denoted $A \otimes B$, is an $mm' \times nn'$ block matrix defined as,

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \ldots & a_{1n}B \\ a_{21}B & a_{22}B & \ldots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \ldots & a_{mn}B \end{pmatrix} .$$

For an $m \times n$ matrix A with rows $a_1, \ldots, a_m$, the *vectorization* (or flattening) of A is the $mn \times 1$ column vector[2]

$$\overline{\mathrm{vec}}(A) = (a_1 \quad a_2 \quad \cdots \quad a_m)^\mathsf{T} .$$

The next lemma collects several properties of the Kronecker product and the $\overline{\mathrm{vec}}(\cdot)$ operator, that will be used throughout the paper. For proofs and further details, we refer to (Horn and Johnson, 1991).

Lemma 3. Let A, A', B, B' be matrices of appropriate dimensions. The following properties hold:

(i) $(A \otimes B)(A' \otimes B') = (AA') \otimes (BB')$;
(ii) $(A \otimes B)^\mathsf{T} = A^\mathsf{T} \otimes B^\mathsf{T}$;
(iii) If $A, B \succeq 0$, then for any $s \in \mathbb{R}$ it holds that $(A \otimes B)^s = A^s \otimes B^s$, and in particular, if $A, B \succ 0$ then $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$;
(iv) If $A \succeq A'$ and $B \succeq B'$ then $A \otimes B \succeq A' \otimes B'$, and in particular, if $A, B \succeq 0$ then $A \otimes B \succeq 0$;
(v) $\mathrm{Tr}(A \otimes B) = \mathrm{Tr}(A)\,\mathrm{Tr}(B)$;
(vi) $\overline{\mathrm{vec}}(uv^\mathsf{T}) = u \otimes v$ for any two column vectors u, v.

[2]This definition is slightly non-standard as it uses a row-major order rather than a column-major order typically used to define vec(); the notation $\overline{\mathrm{vec}}()$ is used to distinguish it from the latter.

The following identity connects the Kronecker product and the $\overline{\mathrm{vec}}$ operator; it facilitates an efficient computation of a matrix-vector product where the matrix is a Kronecker product of two smaller matrices. See the full version of the paper (Gupta et al., 2018) for a proof.

Lemma 4. Let $G \in \mathbb{R}^{m \times n}$, $L \in \mathbb{R}^{m \times m}$ and $R \in \mathbb{R}^{n \times n}$. Then, one has $(L \otimes R^\mathsf{T})\overline{\mathrm{vec}}(G) = \overline{\mathrm{vec}}(LGR)$.

2.4. Matrix inequalities

Our analysis requires the following result concerning the geometric means of matrices. Recall that by writing $X \succeq 0$ we mean, in particular, that X is a symmetric matrix.

Lemma 5 (Ando et al. (2004))**.** Assume that $0 \preceq X_i \preceq Y_i$ for all $i = 1, \ldots, n$. Assume further that all X_i commute with each other and all Y_i commute with each other. Let $\alpha_1, \ldots, \alpha_n \geq 0$ such that $\sum_{i=1}^{n} \alpha_i = 1$, then

$$X_1^{\alpha_1} \cdots X_n^{\alpha_n} \preceq Y_1^{\alpha_1} \cdots Y_n^{\alpha_n} .$$

In words, the (weighted) geometric mean of commuting PSD matrices is operator monotone.

Ando et al. (2004) proved a stronger result which does not require the PSD matrices to commute with each other, relying on a generalized notion of geometric mean, but for our purposes the simpler commuting case suffices. We also use the following classic result from matrix theory, attributed to Löwner (1934), which is an immediate consequence of Lemma 5.

Lemma 6. The function $x \mapsto x^\alpha$ is operator-monotone for $\alpha \in [0, 1]$, that is, if $0 \preceq X \preceq Y$ then $X^\alpha \preceq Y^\alpha$.

3. Analysis of Shampoo for matrices

In this section we analyze Shampoo in the matrix case. The analysis conveys the core ideas while avoiding numerous the technical details imposed by the general tensor case. The main result of this section is stated in the following theorem.

Theorem 7. Assume that the gradients $G_1, \ldots, G_T$ are matrices of rank at most r. Then the regret of Algorithm 1 compared to any $W^\star \in \mathbb{R}^{m \times n}$ is bounded as follows,

$$\sum_{t=1}^{T} f_t(W_t) - \sum_{t=1}^{T} f_t(W^\star) \leq \sqrt{2r} D\, \mathrm{Tr}(L_T^{1/4})\, \mathrm{Tr}(R_T^{1/4}) ,$$

where $L_T = \epsilon I_m + \sum_{t=1}^{T} G_t G_t^\mathsf{T}$, $R_T = \epsilon I_n + \sum_{t=0}^{T} G_t^\mathsf{T} G_t$ and $D = \max_{t \in [T]} \|W_t - W^\star\|_\mathsf{F}$.

Let us make a few comments regarding the bound. First, under mild conditions, each of the trace terms on the right-hand side of the bound scales as $O(T^{1/4})$. Thus, the overall scaling of the bound with respect to the number of iterations T is $O(\sqrt{T})$, which is the best possible in the context of online (or stochastic) optimization. For example, assume that the functions f_t are 1-Lipschitz with respect to the spectral norm, that is, $\|G_t\|_2 \le 1$ for all t. Let us also fix $\epsilon = 0$ for simplicity. Then, $G_t G_t^\mathsf{T} \preceq I_m$ and $G_t^\mathsf{T} G_t \preceq I_n$ for all t, and so we have $\mathrm{Tr}(L_T^{1/4}) \le mT^{1/4}$ and $\mathrm{Tr}(R_T^{1/4}) \le nT^{1/4}$. That is, in the worst case, while only assuming convex and Lipschitz losses, the regret of the algorithm is $O(\sqrt{T})$.

Second, we note that D in the above bound could in principle grow with the number of iterations T and is not necessarily bounded by a constant. This issue can be easily addressed, for instance, by adding an additional step to the algorithm in which W_t is projected W_t onto the convex set of matrices whose Frobenius norm is bounded by $D/2$. Concretely, the projection at step t needs to be computed with respect to the norm induced by the pair of matrices (L_t, R_t), defined as $\|A\|_t^2 = \mathrm{Tr}(A^\mathsf{T} L_t^{1/4} A R_t^{1/4})$; it is not hard to verify that the latter indeed defines a norm over $\mathbb{R}^{m\times n}$, for any $L_t, R_t \succ 0$. Alas, the projection becomes computationally expensive in large scale problems and is rarely performed in practice. We therefore omitted the projection step from Algorithm 1 in favor of a slightly looser bound.

The main step in the proof of the theorem is established in the following lemma. The lemma implies that the Kronecker product of the two preconditioners used by the algorithm is lower bounded by a full $mn \times mn$ matrix often employed in full-matrix preconditioning methods.

Lemma 8. Assume that $G_1, \ldots, G_T \in \mathbb{R}^{m\times n}$ are matrices of rank at most r. Let $g_t = \overline{\mathrm{vec}}(G_t)$ denote the vectorization of G_t for all t. Then, for any $\epsilon \ge 0$,

$$\begin{aligned}\epsilon I_{mn} &+ \frac{1}{r}\sum_{t=1}^{T} g_t g_t^\mathsf{T} \\ &\preceq \left(\epsilon I_m + \sum_{t=1}^{T} G_t G_t^\mathsf{T}\right)^{1/2} \otimes \left(\epsilon I_n + \sum_{t=1}^{T} G_t^\mathsf{T} G_t\right)^{1/2} .\end{aligned}$$

In particular, the lemma shows that the small eigenvalues of the full-matrix preconditioner on the left, which are the most important for effective preconditioning, do not vanish as a result of the implicit approximation. In order to prove Lemma 8 we need the following technical result, the proof of which is relegated to the full version of the paper (Gupta et al., 2018).

Lemma 9. Let G be an $m \times n$ matrix of rank at most r and denote $g = \overline{\mathrm{vec}}(G)$. Then,

$$\frac{1}{r} g g^\mathsf{T} \preceq I_m \otimes (G^\mathsf{T} G) \quad \text{and} \quad \frac{1}{r} g g^\mathsf{T} \preceq (G G^\mathsf{T}) \otimes I_n .$$

Proof of Lemma 8. Let us introduce the following notations to simplify our derivation,

$$A_m \overset{\mathrm{def}}{=} \epsilon I_m + \sum_{t=1}^{T} G_t G_t^\mathsf{T} , \qquad B_n \overset{\mathrm{def}}{=} \epsilon I_n + \sum_{t=1}^{T} G_t^\mathsf{T} G_t .$$

From Lemma 9 we get,

$$\epsilon I_{mn} + \frac{1}{r}\sum_{t=1}^{T} g_t g_t^\mathsf{T} \preceq I_m \otimes B_n, \quad \epsilon I_{mn} + \frac{1}{r}\sum_{t=1}^{T} g_t g_t^\mathsf{T} \preceq A_m \otimes I_n .$$

Now, observe that $I_m \otimes B_n$ and $A_m \otimes I_n$ commute with each other. Using Lemma 5 followed by Lemma 3(iii) and Lemma 3(i) yields

$$\begin{aligned}\epsilon I_{mn} + \frac{1}{r}\sum_{t=1}^{T} g_t g_t^\mathsf{T} &\preceq \big(I_m \otimes B_n\big)^{1/2}\big(A_m \otimes I_n\big)^{1/2} \\ &= \big(I_m \otimes B_n^{1/2}\big)\big(A_m^{1/2} \otimes I_n\big) \;=\; A_m^{1/2} \otimes B_n^{1/2} ,\end{aligned}$$

which completes the proof. □

We can now prove the main result of the section.

Proof of Theorem 7. Recall the update performed in Algorithm 1,

$$W_{t+1} \;=\; W_t - \eta L_t^{-1/4} G_t R_t^{-1/4} .$$

Note that the pair of left and right preconditioning matrices, $L_t^{1/4}$ and $R_t^{1/4}$, is equivalent due to Lemma 4 to a single preconditioning matrix $H_t = L_t^{1/4} \otimes R_t^{1/4} \in \mathbb{R}^{mn\times mn}$. This matrix is applied to flattened version of the gradient $g_t = \overline{\mathrm{vec}}(G_t)$. More formally, letting $w_t = \overline{\mathrm{vec}}(W_t)$ we have that the update rule of the algorithm is equivalent to,

$$w_{t+1} = w_t - \eta H_t^{-1} g_t . \tag{2}$$

Hence, we can invoke Lemma 1 in conjuction the fact that $0 \prec H_1 \preceq \ldots \preceq H_T$. The latter follows from Lemma 3(iv), as $0 \prec L_1 \preceq \ldots \preceq L_T$ and $0 \prec R_1 \preceq \ldots \preceq R_T$. We thus further bound the first term of Lemma 1 by,

$$\begin{aligned}&\sum_{t=1}^{T}(w_t - w^\star)^\mathsf{T}(H_t - H_{t-1})(w_t - w^\star) \\ &\qquad \le D^2 \sum_{t=1}^{T} \mathrm{Tr}(H_t - H_{t-1}) = D^2\, \mathrm{Tr}(H_T) ,\end{aligned} \tag{3}$$

for $D = \max_{t\in[T]} \|w_t - w^\star\| = \max_{t\in[T]} \|W_t - W^\star\|_\mathsf{F}$ where $w^\star = \overline{\mathrm{vec}}(W^\star)$ and $H_0 = 0$. We obtain the regret bound

$$\sum_{t=1}^{T} f_t(W_t) - f_t(W^\star) \le \frac{D^2}{2\eta}\mathrm{Tr}(H_T) + \frac{\eta}{2}\sum_{t=1}^{T}\big(\|g_t\|_{H_t}^*\big)^2 . \tag{4}$$

Let us next bound the sum on the right-hand side of Eq. (4). First, according to Lemma 8 and the monotonicity (in the operator sense) of the square root function $x \mapsto x^{1/2}$ (recall Lemma 6), for the preconditioner H_t we have that

$$\widehat{H}_t \overset{\text{def}}{=} \left(r\epsilon I + \sum_{s=1}^{t} g_s g_s^\top \right)^{1/2} \preceq \sqrt{r} H_t \ . \qquad (5)$$

On the other hand, invoking Lemma 2 with the choice of potential $\Phi(H) = \mathrm{Tr}(H) + r\epsilon\,\mathrm{Tr}(H^{-1})$ and $M_t = \sum_{s=1}^t g_t g_t^\top$, we get,

$$\operatorname*{argmin}_{H \succ 0} \left\{ M_t \bullet H^{-1} + \Phi(H) \right\} = \operatorname*{argmin}_{H \succ 0} \mathrm{Tr}\left(\widehat{H}_t^2 H^{-1} + H \right) = \widehat{H}_t \ .$$

To see the last equality, observe that for any symmetric $A \succeq 0$, the function $\mathrm{Tr}(AX + X^{-1})$ is minimized at $X = A^{-1/2}$, since $\nabla_X \mathrm{Tr}(AX + X^{-1}) = A - X^{-2}$. Hence, Lemma 2 implies

$$\begin{aligned} \sum_{t=1}^{T} (\|g_t\|_{\widehat{H}_t}^*)^2 &\le \sum_{t=1}^{T} (\|g_t\|_{\widehat{H}_T}^*)^2 + \Phi(\widehat{H}_T) - \Phi(\widehat{H}_0) \\ &\le \left(r\epsilon I + \sum_{t=1}^{T} g_t g_t^\top \right) \bullet \widehat{H}_T^{-1} + \mathrm{Tr}(\widehat{H}_T) \qquad (6) \\ &= 2\,\mathrm{Tr}(\widehat{H}_T) \ . \end{aligned}$$

Using Eq. (5) twice along with Eq. (6), we obtain

$$\sum_{t=1}^{T} (\|g_t\|_{H_t}^*)^2 \le \sqrt{r} \sum_{t=1}^{T} (\|g_t\|_{\widehat{H}_t}^*)^2 \le 2\sqrt{r}\,\mathrm{Tr}(\widehat{H}_T) \le 2r\,\mathrm{Tr}(H_T) \ .$$

Finally, using the above upper bound in Eq. (4) and choosing $\eta = D/\sqrt{2r}$ gives the desired regret bound:

$$\begin{aligned} &\sum_{t=1}^{T} f_t(W_t) - \sum_{t=1}^{T} f_t(W^\star) \\ &\quad \le \left(\frac{D^2}{2\eta} + \eta r \right) \mathrm{Tr}(H_T) = \sqrt{2r} D\,\mathrm{Tr}(L_T^{1/4})\,\mathrm{Tr}(R_T^{1/4}) \ . \qquad \square \end{aligned}$$

4. Shampoo for tensors

In this section we introduce the Shampoo algorithm in its general form, which is applicable to tensors of arbitrary dimension. Before we can present the algorithm, we review further definitions and operations involving tensors.

4.1. Tensors: notation and definitions

A tensor is a multidimensional array. The *order* of a tensor is the number of dimensions (also called modes). For an order-k tensor A of dimension $n_1 \times \cdots \times n_k$, we use the notation $A_{j_1,\ldots,j_k}$ to refer to the single element at position j_i on the i'th dimension for all i where $1 \le j_i \le n_i$. We also denote

$$n = \prod_{i=1}^{k} n_i \quad \text{and} \quad \forall i : \ n_{-i} = \prod_{j \ne i} n_j \ .$$

The following definitions are used throughout the section.

- A *slice* of an order-k tensor along its i'th dimension is a tensor of order $k-1$ which consists of entries with the same index on the i'th dimension. A slice generalizes the notion of rows and columns of a matrix.
- An $n_1 \times \cdots \times n_k$ tensor A is of *rank one* if it can be written as an outer product of k vectors of appropriate dimensions. Formally, let $\circ$ denote the vector outer product and and set $A = u^1 \circ u^2 \circ \cdots \circ u^k$ where $u^i \in \mathbb{R}^{n_i}$ for all i. Then A is an order-k tensor defined through

$$\begin{aligned} A_{j_1,\ldots,j_k} &= (u^1 \circ u^2 \circ \cdots \circ u^k)_{j_1,\ldots,j_k} \\ &= u^1_{j_1} u^2_{j_2} \cdots u^k_{j_k}, \qquad \forall\, 1 \le j_i \le n_i \ (i \in [k]) \ . \end{aligned}$$

- The *vectorization* operator flattens a tensor to a column vector in $\mathbb{R}^n$, generalizing the matrix $\overline{\mathrm{vec}}$ operator. For an $n_1 \times \cdots \times n_k$ tensor A with slices $A^1_1, \ldots, A^1_{n_1}$ along its first dimension, this operation can be defined recursively as follows:

$$\overline{\mathrm{vec}}(A) = \left(\overline{\mathrm{vec}}(A^1_1)^\top \ \cdots \ \overline{\mathrm{vec}}(A^1_{n_1})^\top \right)^\top,$$

where for the base case ($k = 1$), we define $\overline{\mathrm{vec}}(u) = u$ for any column vector u.
- The *matricization* operator $\mathrm{mat}_i(A)$ reshapes a tensor A to a matrix by vectorizing the slices of A along the i'th dimension and stacking them as rows of a matrix. More formally, for an $n_1 \times \cdots \times n_k$ tensor A with slices $A^i_1, \ldots, A^i_{n_i}$ along the i'th dimension, matricization is defined as the $n_i \times n_{-i}$ matrix,

$$\mathrm{mat}_i(A) = \left(\overline{\mathrm{vec}}(A^i_1) \ \cdots \ \overline{\mathrm{vec}}(A^i_{n_i}) \right)^\top \ .$$

- The matrix product of an $n_1 \times \cdots \times n_k$ tensor A with an $m \times n_i$ matrix M is defined as the $n_1 \times \cdots \times n_{i-1} \times m \times n_{i+1} \times \cdots \times n_k$ tensor, denoted $A \times_i M$, for which the identity $\mathrm{mat}_i(A \times_i M) = M \mathrm{mat}_i(A)$ holds. Explicitly, we define $A \times_i M$ element-wise as

$$(A \times_i M)_{j_1,\ldots,j_k} = \sum_{s=1}^{n_i} M_{j_i s} A_{j_1,\ldots j_{i-1},s,j_{i+1},\ldots,j_k} \, .$$

A useful fact, that follows directly from this definition, is that the tensor-matrix product is commutative, in the sense that $A \times_i M \times_{i'} M' = A \times_{i'} M' \times_i M$ for any $i \ne i'$ and matrices $M \in \mathbb{R}^{n_i \times n_i}$, $M' \in \mathbb{R}^{n_{i'} \times n_{i'}}$.
- The *contraction* of an $n_1 \times \cdots \times n_k$ tensor A with itself along all but the i'th dimension is an $n_i \times n_i$ matrix defined as $A^{(i)} = \mathrm{mat}_i(A)\mathrm{mat}_i(A)^\top$, or more explicitly as

$$A^{(i)}_{j,j'} = \sum_{\alpha_{-i}} A_{j,\alpha_{-i}} A_{j',\alpha_{-i}} \qquad \forall\, 1 \le j, j' \le n_i,$$

where the sum ranges over all possible indexings α_{-i} of all dimensions $\ne i$.

4.2. The algorithm

We can now describe the Shampoo algorithm in the general, order-k tensor case, using the definitions established above. Here we assume that the optimization domain is $\mathcal{W} = \mathbb{R}^{n_1 \times \cdots \times n_k}$, that is, the vector space of order-k tensors, and the functions $f_1, \ldots, f_T$ are convex over this domain. In particular, the gradient ∇f_t is also an $n_1 \times \cdots \times n_k$ tensor.

The Shampoo algorithm in its general form, presented in Algorithm 2, is analogous to Algorithm 1. It maintains a separate preconditioning matrix H_t^i (of size $n_i \times n_i$) corresponding to for each dimension $i \in [k]$ of the gradient. On step t, the i'th mode of the gradient G_t is then multiplied by the matrix $(H_t^i)^{-1/2k}$ through the tensor-matrix product operator $\times_i$. (Recall that the order in which the multiplications are carried out does not affect the end result.) After all dimensions have been processed and the preconditioned gradient $\widetilde{G}_t$ has been obtained, a gradient step is taken.

Algorithm 2 Shampoo, general tensor case.

```
Initialize: $W_1 = \mathbf{0}_{n_1 \times \cdots \times n_k}$ ; $\forall i \in [k] : H_0^i = \epsilon I_{n_i}$
for $t = 1, \ldots, T$ do:
    Receive loss function $f_t : \mathbb{R}^{n_1 \times \cdots \times n_k} \mapsto \mathbb{R}$
    Compute $G_t = \nabla f_t(W_t)$          // $G_t \in \mathbb{R}^{n_1 \times \cdots \times n_k}$
    $\widetilde{G}_t \leftarrow G_t$          // $\widetilde{G}_t$ is preconditioned gradient
    for $i = 1, \ldots, k$ do:
        $H_t^i = H_{t-1}^i + G_t^{(i)}$
        $\widetilde{G}_t \leftarrow \widetilde{G}_t \times_i (H_t^i)^{-1/2k}$
    Update: $W_{t+1} = W_t - \eta \widetilde{G}_t$
```

The tensor operations $A^{(i)}$ and $M \times_i A$ can be implemented using tensor contraction, which is a standard library function in scientific computing libraries and is fully supported by modern machine learning frameworks such as TensorFlow (Abadi et al., 2016). See Section 5 for further details on our implementation of the algorithm in TensorFlow.

We now state the main result of this section.

Theorem 10. Assume that for all $i \in [k]$ and $t = 1, \ldots, T$ it holds that $\text{rank}(\text{mat}_i(G_t)) \le r_i$, and let $r = (\prod_{i=1}^k r_i)^{1/k}$. Then the regret of Algorithm 2 compared to any $W^\star \in \mathbb{R}^{n_1 \times \cdots \times n_k}$ is

$$\sum_{t=1}^{T} f_t(W_t) - \sum_{t=1}^{T} f_t(W^\star) \le \sqrt{2r} D \prod_{i=1}^{k} \text{Tr}\big((H_T^i)^{1/2k}\big),$$

where $H_T^i = \epsilon I_{n_i} + \sum_{t=1}^T G_t^{(i)}$ for all $i \in [k]$ and $D = \max_{t \in [T]} \|W_t - W^\star\|_\mathsf{F}$.

The comments following Theorem 7 regarding the parameter D in the above bound and the lack of projections in the algorithm apply also in the general tensor case. Furthermore, as in the matrix case, under standard assumptions each of the trace terms on the right-hand side of the above bound is bounded by $O(T^{1/2k})$. Therefore, their product, and thereby the overall regret bound, is $O(\sqrt{T})$.

The proof of Theorem 10 is based on a technical generalization of the arguments in the matrix case, and can be found in the full version of the paper (Gupta et al., 2018).

5. Implementation details

We implemented Shampoo in its general tensor form in Python as a new TensorFlow (Abadi et al., 2016) optimizer. Our implementation follows almost verbatim the pseudocode shown in Algorithm 2. We used the built-in `tensordot` operation to implement tensor contractions and tensor-matrix products. Matrix powers were computed simply by constructing a singular value decomposition (SVD) and then taking the powers of the singular values. These operations are fully supported in TensorFlow on GPUs. We plan to implement Shampoo in PyTorch in the near future.

Our optimizer treats each tensor in the input model as a separate optimization variable and applies the Shampoo update to each of these tensors independently. This has the advantage of making the optimizer entirely oblivious to the specifics of the architecture, and it only has to be aware of the tensors involved and their dimensions. In terms of preconditioning, this approach amounts to employing a block-diagonal preconditioner, with blocks corresponding to the different tensors in the model. In particular, only intra-tensor correlations are captured and correlations between parameters in different tensors are ignored entirely.

Our optimizer also implements a diagonal variant of Shampoo which is automatically activated for a dimension of a tensor whenever it is considered too large for the associated preconditioner to be stored in memory or to compute its SVD. Other dimensions of the same tensor are not affected and can still use non-diagonal preconditioning (unless they are too large themselves). See the full version of the paper (Gupta et al., 2018) for a detailed description of this variant and its analysis. In our experiments, we used a threshold of around 1200 for each dimension to trigger the diagonal version with no apparent sacrifice in performance. This option gives the benefit of working with full preconditioners whenever possible, while still being able to train models where some of the tensors are prohibitively large, and without having to modify either the architecture or the code used for training.

6. Experimental results

We performed experiments with Shampoo on several datasets, using standard deep neural-network models. We focused on two domains: image classification on CIFAR-10/100, and statistical language modeling on LM1B. In each experiment, we relied on existing code for training the mod-

Dataset	SGD	Adam	AdaGrad	Shampoo
CIFAR10 (ResNet-32)	2.184	2.184	2.197	2.151
CIFAR10 (Inception)	3.638	3.667	3.682	3.506
CIFAR100 (ResNet-55)	1.210	1.203	1.210	1.249
LM1B (Transformer)	4.919	4.871	4.908	3.509

Table 1. Average number of steps per second (with batch size of 128) in each experiment, for each algorithm we tested.

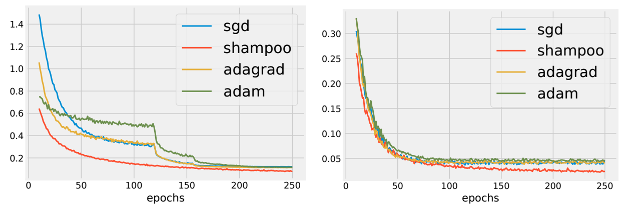

Figure 2. Convergence of training (cross-entropy) loss for a 32-layer ResNet (left) and an Inception network (right) on CIFAR-10.

els, and merely replaced the TensorFlow optimizer without making any other changes to the code.

In all of our experiments, we worked with a mini-batch of size 128. In Shampoo, this simply means that the gradient G_t used in each iteration of the algorithm is the average of the gradient over 128 examples, but otherwise has no effect on the algorithm. Notice that, in particular, the preconditioners are also updated once per batch using the averaged gradient rather than with gradients over individual examples.

We made two minor heuristic adjustments to Shampoo to improve performance. First, we employed a delayed update for the preconditioners, and recomputed the roots of the matrices H_t^i once in every 20–100 steps. This had almost no impact on accuracy, but helped to improve the amortized runtime per step. Second, we incorporated momentum into the gradient step, essentially computing the running average of the gradients $\overline{G}_t = \alpha \overline{G}_{t-1} + (1-\alpha) G_t$ with a fixed setting of $\alpha = 0.9$. This slightly improved the convergence of the algorithm, as is the case with many first-order methods.

Quite surprisingly, while the Shampoo algorithm performs significantly more computation per step than algorithms like SGD (with momentum), AdaGrad, and Adam, its actual runtime in practice is not much worse. Table 1 shows the average number of steps (i.e., batches of size 128) per second on a Tesla K40 GPU, for each of the algorithms we tested. As can be seen from the results, each step of Shampoo is typically slower than that of the other algorithms by a small margin, and in some cases (ResNet-55) it is actually faster.

Image classification. We experimented with Shampoo over the CIFAR-10 benchmark with several different architectures. For each optimization algorithm, we explored 10 different learning rates between 0.01 and 10.0 (scaling the entire range for Adam by a factor of 10^{-4}), and chose the one with the best loss. (See the full version of the paper (Gupta et al., 2018) for the precise learning rates and schedules used in all of our experiments.) Fig. 2 shows the training loss convergence for a 32-layer ResNet with 2.4M parameters. This network is capable of reaching an error rate of 5% on the test set. We also ran on the 20-layer small inception network described in Zhang et al. (2017), with 1.65M trainable parameters, capable of reaching an error rate of 7.5% on test data. On CIFAR-100 (Fig. 3), we used a standard 55-layer residual network with 13.5M trainable parameters. In this model, the trainable variables are all tensors of order 4 (all layers are convolutional), where the largest layer is of dimension (256, 3, 3, 256); it does not employ batch-norm, dropout, etc., and reaches 24% top-1 error on the test set.

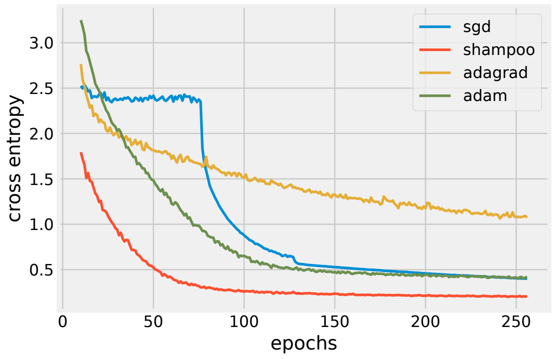

Figure 3. Convergence of training loss for a 55-layer ResNet on CIFAR-100.

Language models. Our next experiment was on the LM1B benchmark for statistical language modeling (Chelba et al., 2013). We used the Transformer model with 9.8M trainable parameters from (Vaswani et al., 2017). This model has a succession of fully connected-layers, with corresponding tensors of order at most 2, the largest of which is of dimension (2000, 256). For Shampoo, we simply used the default learning rate of $\eta = 1.0$. For the other algorithms we explored various different settings of the learning rate. The convergence of the test perplexity is shown in Fig. 4.

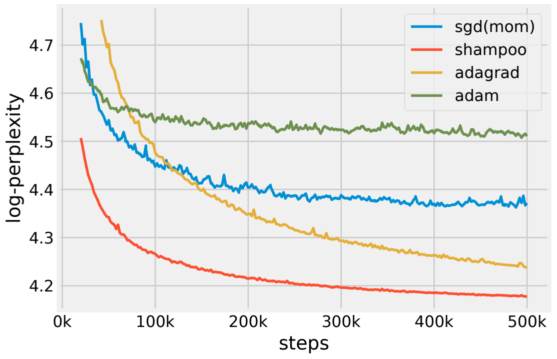

Figure 4. Convergence of test loss for the Transformer model for machine translation (Vaswani et al., 2017) on LM1B.

Acknowledgements

We are grateful to Samy Bengio, Roy Frostig, Phil Long, Aleksander Mądry and Kunal Talwar for numerous discussions and helpful suggestions. Special thanks go to Roy for coming up with the name “Shampoo.”

References

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

N. Agarwal, B. Bullins, and E. Hazan. Second order stochastic optimization in linear time. *arXiv preprint arXiv:1602.03943*, 2016.

T. Ando, C.-K. Li, and R. Mathias. Geometric means. *Linear algebra and its applications*, 385:305–334, 2004.

N. Cesa-Bianchi, A. Conconi, and C. Gentile. On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057, 2004.

C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson. One billion word benchmark for measuring progress in statistical language modeling. Technical report, Google, 2013.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

M. A. Erdogdu and A. Montanari. Convergence rates of sub-sampled newton methods. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pages 3052–3060. MIT Press, 2015.

R. Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

A. Gonen and S. Shalev-Shwartz. Faster sgd using sketched conditioning. *arXiv preprint arXiv:1506.02649*, 2015.

V. Gupta, T. Koren, and Y. Singer. A unified approach to adaptive regularization in online and stochastic optimization. *arXiv preprint arXiv:1706.06569*, 2017.

V. Gupta, T. Koren, and Y. Singer. Shampoo: Preconditioned stochastic tensor optimization. *arXiv preprint arXiv:1802.09568*, 2018.

E. Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3-4):157–325, 2016.

R. A. Horn and C. R. Johnson. Topics in matrix analysis, 1991. *Cambridge University Presss, Cambridge*, 37:39, 1991.

A. Kalai and S. Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

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

A. S. Lewis and M. L. Overton. Nonsmooth optimization via quasi-newton methods. *Mathematical Programming*, 141(1-2):135–163, 2013.

K. Löwner. Über monotone matrixfunktionen. *Mathematische Zeitschrift*, 38(1):177–216, 1934.

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

B. Neyshabur, R. R. Salakhutdinov, and N. Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2422–2430, 2015.

J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

M. Pilanci and M. J. Wainwright. Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence. *SIAM Journal on Optimization*, 27(1):205–245, 2017.

S. Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2012.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.

P. Xu, J. Yang, F. Roosta-Khorasani, C. Ré, and M. W. Mahoney. Sub-sampled newton methods with non-uniform sampling. In *Advances in Neural Information Processing Systems*, pages 3000–3008, 2016.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.