Dynamic Trace Estimation

Prathamesh Dharangutte Christopher Musco
Tandon School of Engineering
New York University
prathamesh.d@nyu.edu,cmusco@nyu.edu

Abstract

We study a *dynamic* version of the implicit trace estimation problem. Given access to an oracle for computing matrix-vector multiplications with a dynamically changing matrix A, our goal is to maintain an accurate approximation to A's trace using as few multiplications as possible. We present a practical algorithm for solving this problem and prove that, in a natural setting, its complexity is quadratically better than the standard solution of repeatedly applying Hutchinson's stochastic trace estimator. We support this theory with empirical results, showing significant computational improvements on two applications in machine learning and network science: tracking moments of the Hessian spectral density during neural network optimization, and counting triangles in a dynamically changing graph.

1. Introduction

1.1. Implicit Trace Estimation

Implicit or "matrix-free" trace estimation is a ubiquitous computational primitive in linear algebra, which has become increasingly important in machine learning and data science. Given access to an oracle for computing matrix-vector products $Ax_1, \ldots, Ax_m$ between an $n \times n$ matrix A and chosen vectors $x_1, \ldots, x_m$, the goal is to compute an approximation to A's trace, $\mathrm{tr}(A) = \sum_{i=1}^{n} A_{ii}$. This problem arises when A's diagonal entries cannot be accessed explicitly, usually because forming A is computationally prohibitive. As an example, consider A which is the Hessian matrix of a loss function involving a neural network. While forming the Hessian is infeasible when the network is large, backpropagation can be used to very efficiently compute Hessian-vector products (Pearlmutter, 1994).

In other important applications, A is a *matrix function* of some other matrix B. For example, if B is an $n \times n$ graph adjacency matrix, $\mathrm{tr}(B^3)$ equals six times the number of triangle in the graph (Avron, 2010). Computing $A = B^3$ explicitly to evaluate the trace would require $O(n^3)$ time, whereas a matrix-vector multiplication $Ax = B \cdot (B \cdot (Bx))$ only requires $O(n^2)$ time. Similarly, in the application of log-determinant approximation, useful in e.g. Bayesian log likelihood computations or determinantal point process methods, we want to approximate the trace of $A = \log(B)$ (Boutsidis et al., 2015; Han et al., 2015; Saibaba et al., 2017). Again, A takes $O(n^3)$ time to form explicitly, but Ax can be computed efficiently using iterative methods like the Lanczos algorithm in roughly $O(n^2)$ time (Higham, 2008).

In data science and machine learning, other applications of implicit trace estimation include matrix norm and spectral sum estimation (Han et al., 2017; Ubaru & Saad, 2018; Musco et al., 2018), as well as methods for eigenvalue counting (Di Napoli et al., 2016) and spectral density estimation (Weiße et al., 2006; Lin et al., 2016). Spectral density estimation methods typically use implicit trace estimation to estimate *moments* of a matrix's eigenvalue distribution – i.e., $\mathrm{tr}(A), \mathrm{tr}(A^2), \mathrm{tr}(A^3)$, etc. – which can then be used to compute an approximation to that entire distribution. In deep learning, spectral density estimation is used to quickly analyze the spectra of weight matrices (Pennington et al., 2018; Mahoney & Martin, 2019) or to probe information about the Hessian matrix during optimization (Ghorbani et al., 2019; Yao et al., 2021). Trace estimation has also been used for neural networks weight quantization (Dong et al., 2020; Qian et al., 2020) and to understand training dynamics (Wei & Schwab, 2019).

1.2. Static Setting

The mostly widely used algorithm for implicit trace estimation is Hutchinson's estimator (Girard, 1987; Hutchinson, 1990), which approximates $\mathrm{tr}(A)$ via the average:

$$h_\ell(A) = \frac{1}{\ell} \sum_{i=1}^{1} g_i^T \left(A g_i\right),$$

where $g_1, \ldots, g_\ell \in \mathbb{R}^n$ are random vectors with i.i.d. mean 0 and unit variance entries (e.g. standard Gaussians or ± 1 Rademachers). Hutchinson's estimator requires ℓ matrix-vector multiplications to compute, and can be shown to have variance $O(\|A\|_F^2/\ell)$ (Avron & Toledo, 2011). With high probability, if $\ell = O(1/\epsilon^2)$, we have the error gaurantee

$$|h_\ell - \mathrm{tr}(A)| < \epsilon \|A\|_F. \tag{1}$$

When A is positive semidefinite, a modified version of Hutchinson's estimator obtains a relative error approximation with $\ell = O(1/\epsilon)$ (Meyer et al., 2021), and improved methods have also been studied for sparse, structured, or nearly low-rank matrices (Tang & Saad, 2011; Stathopoulos et al., 2013; Saibaba et al., 2017; Hanyu Li, 2020). For general matrices, however, no known methods outperform Hutchinon's estimator – i.e., no methods can achieve guarantee (1) with $o(1/\epsilon^2)$ matrix-vector products.

1.3. Dynamic Setting

We explore a natural and widely applicable dynamic version of the implicit trace estimation problem: given access to a matrix-vector multiplication oracle for a *dynamically changing* matrix A, we want to maintain an approximation to A's trace. For example, in the above applications involving optimization in machine learning, we need to estimate the trace of a constantly changing Hessian matrix H (or some function of H) during the course of model training. In other applications, A is dynamic because it is repeatedly modified by some algorithmic process. For example, in the transit planning method of (Wang et al., 2020), edges are added to a network to optimally increase a measure of connectivity known as the "Estrada index" (Estrada & Hatano, 2008). This index depends on the quantity $\mathrm{tr}(\exp(B))$, where B is the network adjacency matrix and $\exp(B)$ is a matrix exponential. Implicit trace estimation is repeatedly used to estimate $\mathrm{tr}(\exp(B))$ over the course of optimization (i.e., for a series of slowly changing graphs).

A naive solution to the above problems is to simply apply Hutchinson's estimator to every snapshot of A as it changes over time. After m changes, we will require $O(m/\epsilon^2)$ matrix-vector multiplies to achieve a guarantee like (1) for each time step. The goal of this paper is to improve on this bound when the changes to A are *bounded*. Formally, we abstract the problem as follows:

Problem 1 (Dynamic trace estimation)**.** *Let $A_1, ..., A_m$ be $n \times n$ matrices satisfying:*

1. $\|A_i\|_F \leq 1$, *for all* $i \in [1, m]$.
2. $\|A_{i+1} - A_i\|_F \leq \alpha$, *for all* $i \in [1, m-1]$.

Given implicit matrix-vector multiplication access to each A_i in sequence, the goal is to compute trace approximations $t_1, \ldots, t_m$ for $\mathrm{tr}(A_1), ..., \mathrm{tr}(A_m)$ such that, for each $i \in 1, \ldots, m$,

$$\mathbb{P}[|t_i - \mathrm{tr}(A_i)| \geq \epsilon] \leq \delta.$$

Here $A_1, \ldots, A_m$ represent different snapshots of a dynamic matrix at m time steps. Without loss of generality, we can assume all matrices have Frobenius norm $\|A_i\|_F$ bounded by 1, since otherwise we can rescale the matrices before estimating their trace. The second condition bounds how much the matrices change over time. For simplicity, we assume a fixed upper bound of α on the difference at each time step, but the algorithms presented in this paper will be adaptive to changing gaps between A_i and A_{i+1}, and will perform better when these gaps are small on average. By triangle inequality, $\alpha \leq 2$, but in applications we typically have $\alpha \ll 1$, meaning that the changes in the dynamic matrix are small relative to its Frobenius norm.

We will measure the complexity of any algorithm for solving Problem 1 in the **matrix-vector multiplication oracle model of computation**, meaning that we consider the cost of matrix-vector products (which are the only way $A_1, \ldots, A_m$ can be accessed) to be significantly larger than other computational costs, and thus seek to minimize the number of such products used (Sun et al., 2019). The matrix-vector oracle model has seen growing interest in recent years as it generalizes both the matrix sketching and Krylov subspace models in linear algebra, naturally captures the true computational cost of algorithms in these classes, and is amenable to proving strong lower-bounds, as in other restricted computational models like the first-order oracle model in convex optimization (Simchowitz et al., 2018; Braverman et al., 2020).

1.4. Main Result

Our main result is an algorithm for solving Problem 1 more efficiently than the naive Hutchinson's estimator solution:

Theorem 1.1. *For any $\epsilon, \delta, \alpha \in (0, 1)$, the DeltaShift algorithm (Algorithm 1) solves Problem 1 with*

$$O\left(m \cdot \frac{\alpha \log(1/\delta)}{\epsilon^2} + \frac{\log(1/\delta)}{\epsilon^2}\right)$$

total matrix-vector multiplications involving $A_1, \ldots, A_m$.

For large m, the first term dominates the asymptotic complexity in Theorem 1.1. For comparison, a tight analysis of Hutchinson's estimator via e.g., the Hanson-Wright inequality (Meyer et al., 2021; Martinsson & Tropp, 2020) establishes that the naive approach requires $O\left(m \cdot \frac{\log(1/\delta)}{\epsilon^2}\right)$, which is worse than Theorem 1.1 by a factor of α. A natural setting is when $\alpha = O(\epsilon)$, in which case Algorithm 1 requires $O(\log(1/\delta)/\epsilon)$ matrix-multiplications on average per time step in comparison to $O(\log(1/\delta)/\epsilon^2)$ for Hutchinson's estimator, a quadratic improvement in ϵ.

To achieve the guarantee of Theorem 1.1, we introduce a dynamic *variance reduction* scheme. By linearity of the trace, we have that $\mathrm{tr}(A_{i+1}) = \mathrm{tr}(A_i) + \mathrm{tr}(\Delta_i)$, where $\Delta_i = A_{i+1} - A_i$. Instead of estimating $\mathrm{tr}(A_{i+1})$ directly, we can combine our previous estimate for $\mathrm{tr}(A_i)$ with an

estimate for $\mathrm{tr}(\Delta_i)$, computed via Hutchinson's estimator. Each sample for Hutchinson's estimator applied to Δ_i requires just two matrix-vector multiplies – one with A_i and one with A_{i+1}. At the same time, when Δ_i has small Frobenius norm (bounded by α), we can estimate its trace more accurately than $\mathrm{tr}(A_{i+1})$ directly.

While intuitive, this approach requires care to make work, as in a naive implementation, error in estimating $\mathrm{tr}(\Delta_1), \mathrm{tr}(\Delta_2), \ldots,$ builds quickly, eliminating any computational savings. To avoid this issue, we introduce a novel damping strategy that actually estimates $\mathrm{tr}(A_{i+1} - (1-\gamma)A_i))$ for a positive damping factor γ.

We compliment our main result, Theorem 1.1, which a nearly matching *conditional lower bound*. Specifically, in Section 4 we argue that our DeltaShift method cannot be improved in the dynamic setting unless Hutchinson's estimator can be improved in the static setting for general matrices.

1.5. Related Work

Prior work on implicit trace estimation and applications in machine learning is discussed in Section 1.1. To the best of our knowledge, no dynamic version of the problem akin to Problem 1 has been studied. However, the idea of variance reduction has found applications in other work on implicit trace estimation (Adams et al., 2018; Gambhir et al., 2017; Lin, 2017; Meyer et al., 2021). In these results, the trace of a matrix A is estimated by decomposing $A = B + \Delta$ where B has an easily computed trace (e.g., it is low-rank) and $\|\Delta\|_F \ll \|A\|_F$, so $\mathrm{tr}(\Delta)$ is more easily approximated with Hutchinson's estimator than $\mathrm{tr}(A)$ directly.

2. Preliminaries

2.1. Notation

We let $B \in \mathbb{R}^{m\times k}$ denote a real-valued matrix with m rows and k columns. $x \in \mathbb{R}^n$ denotes a real-valued vector with n entries. Subscripts like B_i or x_j typically denote a matrix or vector in a sequence, but we use double subscripts with matrices to denote entries: B_{ij} being the entry at the i^{th} row and j^{th} column. $\|B\|_F$ denotes the Frobenius norm of B, $\sqrt{\sum_{i,j} B_{ij}^2}$. We let $\mathbb{E}[v]$ and $\mathrm{Var}[v]$ denote the expectation and variance of a random variable v.

2.2. Hutchinson's Estimator

Our algorithm will use Hutchinson's trace estimator with Rademacher ± 1 random variables as a subroutine. Specifically, let $g_1, \ldots, g_\ell \in \mathbb{R}^n$ be independent random vectors, with each entry $+1$ or -1 with probability $1/2$. Let $A \in \mathbb{R}^{n\times n}$. Hutchinson's can estimator for $\mathrm{tr}(A)$ is:

$$h_\ell(A) = \frac{1}{\ell}\sum_{i=1}^{\ell} g_i^T (Ag_i) \tag{2}$$

Claim 2.1 (Hutchinson's expectation and variance)**.** *For any positive integer ℓ and matrix A we have:*

$$\mathbb{E}[h_\ell(A)] = \mathrm{tr}(A)$$
$$\mathrm{Var}[h_\ell(A)] = \frac{2}{\ell}\left(\|A\|_F^2 - \sum_{i=1}^{n} A_{ii}^2\right) \le \frac{2}{\ell}\|A\|_F^2.$$

The above follow from simple calculations, found e.g. in (Avron & Toledo, 2011). Similar bounds can be obtained when Hutchinson's estimator is implemented with different random variables. For example, random Gaussians also lead to a variance bound of $\frac{2}{\ell}\|A\|_F^2$. We use Rademachers because they tend to work better empirically.

Given the variance bound of Claim 2.1, Chebyshev's inequality immediately implies a concentration bound for Hutchinson's estimator.

Claim 2.2 (Chebyshev's Inequality)**.** *For a random variable X with mean $\mathbb{E}[X] = \mu$ and variance $\mathrm{Var}[X] = \sigma^2$, for any $k \ge 1$,*

$$\mathbb{P}\left(|X - \mu| \ge k\sigma\right) \le \frac{1}{k^2}.$$

Corollary 2.2.1. *For any $\epsilon, \delta \in (0,1)$, if $\ell = \frac{2}{\epsilon^2\delta}$ then*

$$\Pr\left[|h_\ell(A) - \mathrm{tr}(A)| \ge \epsilon\|A\|_F\right] \le \delta.$$

The δ dependence in Corollary 2.2.1 can be improved from $\frac{1}{\delta}$ to $\log(1/\delta)$ via the Hanson-Wright inequality, which shows that $h_\ell(A)$ is a sub-exponential random variable (Rudelson et al., 2013; Meyer et al., 2021). We also require Hanson-Wright to obtain our bound involving $\log(1/\delta)$.

By applying this tighter bound to each matrix in Problem 1, Hutchinson's yields a total matrix-vector multiplication bound of $O\left(m \cdot \frac{\log(1/\delta)}{\epsilon^2}\right)$ for solving the problem.

3. Main Algorithmic Result

As discussed in Section 1.3, a natural idea for solving Problem 1 with fewer than $O(m/\epsilon^2)$ queries is to take advantage of the small differences between A_{i+1} and A_i to compute a running estimate of the trace. In particular, instead of estimating $\mathrm{tr}(A_1), \mathrm{tr}(A_2), \ldots, \mathrm{tr}(A_m)$ individually using Hutchinson's estimator, we use linearity of the trace to write:

$$\mathrm{tr}(A_j) = \mathrm{tr}(A_1) + \sum_{i=2}^{j} \mathrm{tr}(\Delta_i) \tag{3}$$

Algorithm 1 DeltaShift

Input: Implicit matrix-vector multiplication access to $A_1, ..., A_m \in \mathbb{R}^{n\times n}$, positive integer constants ℓ_0, ℓ, damping factor $\gamma \in [0, 1]$.
Output: $t_1, \ldots, t_m$ approximating $\mathrm{tr}(A_1), \ldots, \mathrm{tr}(A_m)$.

Draw ℓ_0 random ± 1 vectors $g_1, \ldots, g_{\ell_0} \in \mathbb{R}^n$
Initialize $t_1 \leftarrow \frac{1}{\ell_0}\sum_{i=1}^{\ell_0} g_i^T A_1 g_i$
for $j \leftarrow 2$ **to** m **do**
  Draw ℓ random ± 1 vectors $g_1, \ldots, g_\ell \in \mathbb{R}^n$
  $z_1 \leftarrow A_{j-1}g_1, \ldots, z_\ell \leftarrow A_{j-1}g_\ell$
  $w_1 \leftarrow A_j g_1, \ldots, w_\ell \leftarrow A_j g_\ell$
  $t_j \leftarrow (1-\gamma)t_{j-1} + \frac{1}{\ell}\sum_{i=1}^{\ell} g_i^T\,(w_i - (1-\gamma)z_i)$
end for

where $\Delta_i = A_i - A_{i-1}$.

Then, we can compute an accurate approximation $h_{\ell_0}(A_1) \approx \mathrm{tr}(A_1)$ for large ℓ_0, and for $i > 1$ and some $\ell < \ell_0$, approximate:

$$\mathrm{tr}(A_j) \approx h_{\ell_0}(A_1) + \sum_{i=2}^{j} h_\ell(\Delta_i) \tag{4}$$

The above expression is still an unbiased estimate for $\mathrm{tr}(A_j)$, and since $\|\Delta_i\|_F \le \alpha \ll \|A_{i+1}\|_F$, we expect to approximate $\mathrm{tr}(\Delta_2), \ldots, \mathrm{tr}(\Delta_m)$ much more accurately than $\mathrm{tr}(A_2), \ldots, \mathrm{tr}(A_m)$ directly. At the same time, the estimator in (4) only incurs a 2 factor overhead in matrix-vector multiplies in comparisons to Hutchinson's: it requires $2\cdot(m-1)\ell$ to compute $h_\ell(\Delta_2), \ldots, h_\ell(\Delta_m)$ versus $(m-1)\ell$ to compute $h_\ell(A_2), \ldots, h_\ell(A_m)$. The cost of the initial estimate $h_{\ell_0}(A_1)$ is necessarily higher, but can be amortized over time.

3.1. Our Approach

While intuitive, the problem with this approach is that error compounds over time due to the sum in (4). Each $h_\ell(\Delta_i)$ is roughly $\alpha/\sqrt{\ell}$ away from $\mathrm{tr}(\Delta_i)$, so after j steps we naively expect to incur error $\sim j\cdot\alpha/\sqrt{\ell}$. We can do slightly better by taking advantage of the randomness in our errors, and argue that they accumulate as $\sim \sqrt{j}\cdot\alpha/\sqrt{\ell}$, but regardless, there is accumulation. One option is to "restart" the estimation process – i.e. after some fixed number of steps q, throw out all previous trace approximations, compute an accurate estimate for $\mathrm{tr}(A_q)$, and for $j \ge q$ construct an estimator based on $\mathrm{tr}(A_j) = \mathrm{tr}(A_q) + \sum_{i=q}^{j-1}\mathrm{tr}(\Delta_i)$. While possible to analyze theoretically, this approach turns out to be difficult to implement in practice due to several competing parameters – see the experiments in Section 5 for details.

Instead, we introduce a cleaner approach based on a *damped variance reduction* strategy, which is detailed in Algorithm 1, which we calll DeltaShift. Instead being based on (3), DeltaShift uses the following recursive identity involving a fixed parameter $0 \le \gamma < 1$ (to be chosen later):

$$\mathrm{tr}(A_j) = (1-\gamma)\mathrm{tr}(A_{j-1}) + \mathrm{tr}(\widehat{\Delta_j}) \tag{5}$$

where

$$\widehat{\Delta_j} = A_j - (1-\gamma)A_{j-1}.$$

Given an estimate t_{j-1} for $\mathrm{tr}(A_{j-1})$, DeltaShift estimates $\mathrm{tr}(A_j)$ by $(1-\gamma)t_{j-1} + h_\ell(\widehat{\Delta_j})$. This approach has several useful properties: 1) if t_{j-1} is an unbiased estimate for $\mathrm{tr}(A_{j-1})$, t_j is an unbiased estimate for $\mathrm{tr}(A_j)$, 2) $\|\widehat{\Delta_j}\|_F$ is not much larger than $\|\Delta_j\|_F$ if γ is small, and 3) by shrinking t_{j-1} by a factor of $(1-\gamma)$ when computing t_j, the estimator reduces the variance of this leading term. The last property suffices to ensure that error does no accumulate over time, leading to our main result:

Theorem 1.1 (Restated). *For any $\epsilon, \delta, \alpha \in (0,1)$, Algorithm 1 run with $\gamma = \alpha$, $\ell_0 = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$, and $\ell = O\left(\frac{\alpha\log(1/\delta)}{\epsilon^2}\right)$ solves Problem 1. In total, it requires*

$$O\left(m\cdot\frac{\alpha\log(1/\delta)}{\epsilon^2} + \frac{\log(1/\delta)}{\epsilon^2}\right)$$

matrix-vector multiplications with $A_1, \ldots, A_m$.

The full proof of Theorem 1.1 relies on the Hanson-Wright inequality, and is given in Appendix A. Here, we give a simple proof of essentially the same statement, but with a slightly weaker dependence on the failure probability δ.

Proof. Let $\gamma = \alpha$, $\ell_0 = \frac{2}{\epsilon^2\delta}$, and $\ell = \frac{8\alpha}{\epsilon^2\delta}$. The proof is based on an inductive analysis of the variance of t_j. Specifically, we claim that that for $j = 1, \ldots, m$:

$$\mathrm{Var}[t_j] \le \delta\epsilon^2. \tag{6}$$

For the base case, consider t_1, which is simply Hutchinson's estimator applied to A_1. The bound of (6) follows directly from Claim 2.1 and our assumption that $\|A_1\|_F \le 1$.

Now consider the inductive case, t_j. This random variable is the sum of two independent estimators, t_{j-1} and $h_\ell(\widehat{\Delta_j})$. So, to bound its variance, we just need to bound the variance of these two terms. To address the second, note that by triangle inequality, $\|\widehat{\Delta_j}\|_F = \|A_j - (1-\gamma)A_{j-1}\|_F \le \|A_j - A_{j-1}\|_F + \gamma\|A_{j-1}\|_F \le 2\alpha$. So, by our inductive assumption that $\mathrm{Var}[t_{j-1}] \le \delta\epsilon^2$, and Claim 2.1, we have:

$$\begin{aligned}\mathrm{Var}[t_j] &= (1-\gamma)^2\mathrm{Var}[t_{j-1}] + \mathrm{Var}[h_\ell(\widehat{\Delta_j})]\\ &\le (1-\alpha)^2\delta\epsilon^2 + \frac{2}{\ell}\|\widehat{\Delta_j}\|_F^2\\ &\le (1-\alpha)\delta\epsilon^2 + \alpha\delta\epsilon^2 = \delta\epsilon^2.\end{aligned}$$

This proves (6) for all $j = 1, \ldots, m$, and by Chebyshev's inequality with $k = \sqrt{1/\delta}$, we thus have that, for each j,

$$\Pr\left[|t_j - \mathrm{tr}(A_j)| \geq \epsilon\right] \leq \delta.$$

□

3.2. Selecting γ in Practice

While DeltaShift is simple to implement, in practice, its performance is sensitive to the choice of γ. For the Theorem 1.1 analysis, we assume $\gamma = \alpha$, but α may not be known apriori, and may change over time. To address this issue, we describe a practical way of selecting a *near optimal* γ at each time step j (the choice may change over time) with very little additional computational overhead. Let $v_{j-1} = \mathrm{Var}[t_{j-1}]$ be the variance of our estimator for $\mathrm{tr}(A_{j-1})$. We have that $v_j = \mathrm{Var}[t_j] = (1-\gamma)^2 v_{j-1} + \mathrm{Var}[h_\ell(A_j - (1-\gamma)A_{j-1})] \leq (1-\gamma)^2 v_{j-1} + \frac{2}{\ell}\|A_j - (1-\gamma)A_{j-1}\|_F^2$. At time step j, a natural goal is to choose damping parameter γ^* that minimizes this upper bound on the variance of t_j:

$$\gamma^* = \arg\min_{\gamma} \left[(1-\gamma)^2 v_{j-1} + \frac{2}{\ell}\|\widehat{\Delta_j}\|_F^2\right], \qquad (7)$$

where $\widehat{\Delta_j} = A_j - (1-\gamma)A_{j-1}$ as before.

While (7) cannot be computed directly, observing that $\|B\|_F^2 = \mathrm{tr}(B^T B)$ for any matrix B, so for any choice of γ, the above quantity can be estimated as:

$$\tilde{v}_j = (1-\gamma)^2 \tilde{v}_{j-1} + \frac{2}{\ell} h_\ell(\widehat{\Delta_j}^T \widehat{\Delta_j}), \qquad (8)$$

where $\tilde{v}_{j-1}$ is an estimate for v_{j-1}. Importantly, the estimate $h_\ell(\widehat{\Delta_j}^T \widehat{\Delta_j})$ can be computed using *exactly* the same ℓ matrix-vector products involving A_j and A_{j-1} that are used to estimate $\mathrm{tr}(\widehat{\Delta_j})$, so there is little computational overhead. Moreover, since $\widehat{\Delta_j}^T \widehat{\Delta_j}$ is positive semidefinite, as long as ℓ exceeds $\log(1/\delta)$ as in Theorem 1.1, we will obtain a constant factor *relative error approximation* to its trace in (8) with probability $1 - \delta$.

An alternative approach to estimating v_j would be to simply compute the empirical variance of the average $h_\ell(\widehat{\Delta_j})$, but this requires fixing γ. An advantage of (8) is that it can be used to analytically optimize γ. Specifically, expanding $\widehat{\Delta_j}^T \widehat{\Delta_j}$, we have that:

$$\tilde{v}_j = (1-\gamma)^2 \tilde{v}_{j-1} + \frac{2}{\ell}\left(h_\ell(A_j^T A_j) + (1-\gamma)^2 h_\ell(A_j^T A_j) - 2(1-\gamma) h_\ell(A_{j-1}^T A_j)\right).$$

Above, each estimate h_ℓ is understood to use the same set of random vectors. Taking the derivative and setting to zero, we have that the minimizer of (8), denoted $\tilde{\gamma}^*$, equals:

$$\tilde{\gamma}^* = 1 - \frac{2h_\ell(A_{j-1}^T A_j)}{\ell \tilde{v}_{j-1} + 2h_\ell(A_{j-1}^T A_{j-1})}. \qquad (9)$$

The above expression motivates an essentially parameter free version of DeltaShift, summarized as Algorithm 2, which is what we use in our experimental evaluation.

Algorithm 2 Parameter Free DeltaShift

Input: Implicit matrix-vector multiplication access to $A_1, ..., A_m \in \mathbb{R}^{n\times n}$, positive integer constant ℓ.
Output: $t_1, \ldots, t_m$ approximating $\mathrm{tr}(A_1), \ldots, \mathrm{tr}(A_m)$.

```
Draw ℓ random ±1 vectors g_1, ..., g_ℓ ∈ R^n
z_1 ← A_1 g_1, ..., z_ℓ ← A_1 g_ℓ
N ← (1/ℓ) Σ_{i=1}^ℓ z_i^T z_i                    (estimate for ||A_1||_F^2)
Initialize t_1 ← (1/ℓ) Σ_{i=1}^ℓ g_i^T z_i and v ← (2/ℓ) N
for j ← 2 to m do
    Draw ℓ random ±1 vectors g_1, ..., g_ℓ ∈ R^n
    z_1 ← A_{j-1} g_1, ..., z_ℓ ← A_{j-1} g_ℓ
    w_1 ← A_j g_1, ..., w_ℓ ← A_j g_ℓ
    N ← (1/ℓ) Σ_{i=1}^ℓ z_i^T z_i                (estimate for tr(A_{j-1}^T A_{j-1}))
    M ← (1/ℓ) Σ_{i=1}^ℓ w_i^T w_i                (estimate for tr(A_j^T A_j))
    C ← (1/ℓ) Σ_{i=1}^ℓ w_i^T z_i                (estimate for tr(A_{j-1}^T A_j))
    γ ← 1 − 2C / (ℓ v_{j-1} + 2N)                (optimal damping factor)
    t_j ← (1−γ) t_{j-1} + (1/ℓ) Σ_{i=1}^ℓ g_i^T (w_i − (1−γ) z_i)
    v ← (1−γ)^2 v + (2/ℓ)(N + (1−γ)^2 M − 2(1−γ) C)
end for
```

The only input to this version of the algorithm is ℓ, which controls the number of matrix-vector products used at each time step. For simplicity, unlike in Algorithm 1, we do not use a larger number of matrix-vector multiplies when estimating A_1. In practice this leads to somewhat higher error for the first matrices in the sequence $A_1, \ldots, A_m$, but error quickly falls off for large j.

4. Lower Bound and Open Problems

In this section, we prove a lower bound to compliment Theorem 1.1, and discuss directions for further research.

4.1. Lower Bound

As noted, for large m, the matrix-vector multiplication complexity of DeltaShift is dominated by the leading term in Theorem 1.1, $O\left(m \cdot \frac{\alpha \log(1/\delta)}{\epsilon^2}\right)$. We note that it is unlikely an improvement on this term can be obtained in general. Specifically, any improvement would immediately imply a better estimator than Hutchinon's estimator:

Lemma 4.1. *If there is an algorithm $\mathcal{S}$ for solving Prob. 1 with $o\left(m \cdot \frac{\alpha \log(1/\delta)}{\epsilon^2}\right)$ total matrix-vector multiplications*

*with $A_2, \ldots, A_m$, and **any number** of matrix-vector multiplications with A_1, then there is an algorithm $\mathcal{T}$ for achieving guarantee (1) for a single matrix A with $o\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ matrix-vector multiplications.*

Proof. The proof is via a direct reduction. Given a matrix A, positive integer $m > 1$, and parameter $\alpha = \frac{1}{m-1}$, construct the sequence of matrices:

$$\begin{aligned} A_1 &= 0 \\ A_2 &= \alpha \cdot A \\ &\vdots \\ A_{1/\alpha} &= (1-\alpha)A \\ A_m &= A \end{aligned}$$

Since $A = 0$, and every $A_2, \ldots, A_m$ is a scaling of A, any algorithm $\mathcal{S}$ satisfying the assumption of Lemma 4.1 can be implemented with $o\left((m-1) \cdot \frac{\alpha \log(1/\delta)}{\epsilon^2}\right) = o\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ matrix-vector multiplications with A. Moreover, if $\mathcal{S}$ is run on this sequence of matrices, on the last step it outputs an approximation t_m to $\mathrm{tr}(A)$ with $\Pr[|t_m - \mathrm{tr}(A)| \geq \epsilon] \leq \delta$. So algorithm $\mathcal{T}$ can simply simulate $\mathcal{S}$ on $A_1, \ldots, A_m$ and return its final estimate to satisfy (1). □

Lemma 4.1 implies a *conditional* lower-bound on matrix-vector query algorithms for solving Problem 1: if Hutchinson's estimator cannot be improved for static trace estimation (and it hasn't been for 30 years) then DeltaShift cannot be improved for dynamic trace estimation.

Open Question. An interesting question is if the lower bound of Lemma 4.1 can be made *unconditional*. A natural approach would be to first prove an unconditional lower bound arguing that any algorithm requires $O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ calls to a matrix-vector multiplication oracle to achieve guarantee (1), and thus Hutchinson's estimator is tight in general. Lower bounds for trace estimation have been studied, but only in the special case of positive semidefinite matrices (Meyer et al., 2021; Wimmer et al., 2014).

4.2. Positive semidefinite matrices

As discussed, a natural setting for Problem 1 has $\alpha \approx \epsilon$. In this case, the matrix-vector multiplication complexity of DeltaShift is roughly $O(m/\epsilon)$, which is a quadratic improvement over the $O(m/\epsilon^2)$ required by Hutchinson's estimator. However, in the special case when $A_1, \ldots, A_m$ are all positive semidefinite (PSD), such an improvement is already possible in the *static setting*. Specifically, recent work shows that just $O\left(\frac{\log(1/\delta)}{\epsilon}\right)$ matrix-vector multiplications are needed to find an estimate t such that $|\mathrm{tr}(A) - t| \leq \epsilon \cdot \mathrm{tr}(A)$ with probability $1 - \delta$ (Meyer et al., 2021).

Algorithm 3 Dynamic trace estimation with restarts

Input: Implicit matrix-vector multiplication access to $A_1, ..., A_m \in \mathbb{R}^{n \times n}$, positive integer constant $\ell_0, \ell, q \leq m$.
Output: $t_1, \ldots, t_m$ approximating $\mathrm{tr}(A_1), \ldots, \mathrm{tr}(A_m)$.

```
Draw ℓ_0 random ±1 vectors g_1, ..., g_{ℓ_0} ∈ R^n
Initialize t_1 ← (1/ℓ_0) Σ_{i=1}^{ℓ_0} g_i^T A_1 g_i
for j ← 2 to m do
  if j ≡ 1 (mod n) then
    Draw ℓ_0 random ±1 vectors g_1, ..., g_{ℓ_0} ∈ R^n
    t_j ← (1/ℓ_0) Σ_{i=1}^{ℓ_0} g_i^T A_j g_i
  else
    Draw ℓ_0 random ±1 vectors g_1, ..., g_ℓ ∈ R^n
    t_j ← t_{j-1} + (1/ℓ) Σ_{i=1}^{ℓ} g_i^T (A_j − A_{j-1}) g_i
  end if
end for
```

Open Question. A natural question is if such a guarantee can be obtained in the dynamic setting with $o\left(m \cdot \frac{\log(1/\delta)}{\epsilon}\right)$ matrix-vector multiplication when each of $A_1, \ldots, A_m$ is PSD. Sequences of PSD matrices arise in many applications – for example, in the Estrada index application discussed in Section 1.3, A_j is PSD for all j because the exponential of a matrix is always PSD. We note that answering this question requires some care: even though $A_1, \ldots, A_m$ are PSD, it is not typical that $A_j - A_{j-1}$ is PSD. As such, the improved method from (Meyer et al., 2021), which is called Hutch++, cannot be immediately applied in our variance reduction framework.

5. Experiments

We show that our proposed algorithm outperforms three alternatives on both synthetic and real-world trace estimation problems. Specifically, we evaluate the following methods:

Hutchinson. The naive method of estimating each $\mathrm{tr}(A_1), \ldots, \mathrm{tr}(A_m)$ using an independent Hutchinson's estimator. Discussed in Section 2.2.

NoRestart. The estimator of (4), which uses the same variance reduction strategy as DeltaShift for all $j \geq 2$, but does not restart or add damping to reduce error accumulation.

Restart The estimator discussed in the beginning of Section 3.1, which periodically restarts the variance reduction strategy, using Hutchinson's to obtain a fresh estimator for $\mathrm{tr}(A_j)$. Pseudocode is included as Algorithm 3.

DeltaShift Our parameter free, damped variance reduction estimator described in Algorithm 2.[1]

We allocated to each method a fixed number of matrix-

[1] We ran our large scale experiments before realizing $\tilde{\gamma}^*$ had a closed form solution – see (3.2). Instead, we optimized γ numerically at each step using a simple grid search.

vector queries, Q, to be used over all time steps $1, \ldots, m$. For Hutchinson and DeltaShift, the same number of vectors Q/m was used for each A_j. For Restart and NoRestart, the distribution was non-uniform, and parameter selections are described in detail below.

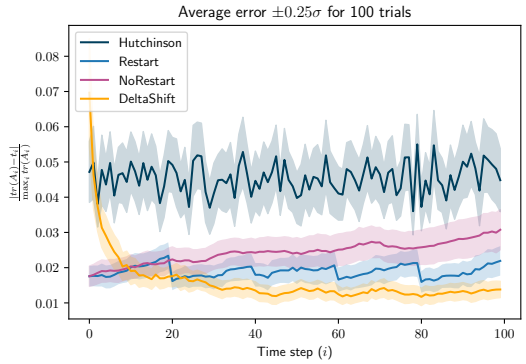


Figure 1. Synthetic data with low perturbation ($Q = 10^4$)

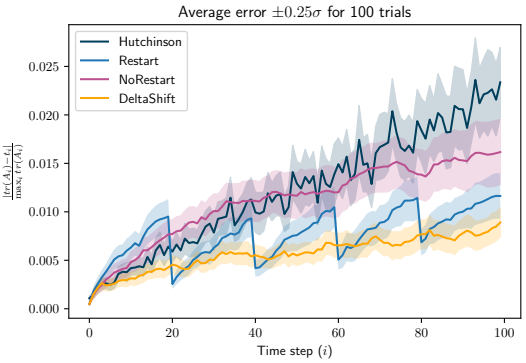


Figure 2. Synthetic data with significant perturbation ($Q = 10^4$)

Synthetic data: To stimulate the dynamic setting, we consider a random symmetric matrix $A \in \mathbb{R}^{n \times n}$ and random perturbation $\Delta \in \mathbb{R}^{n \times n}$ to A for 100 time steps, with $n = 2000$. We consider two cases, one where the perturbations are small (Fig. 1) and one where the perturbations are significant (Fig. 2). For the first case, A is a symmetric matrix with uniformly random eigenvectors and eigenvalues in $[-1, 1]$. Each perturbation is a random rank-1 matrix: $\Delta_j = 5e^{-5} \cdot gg^T$ where r is random ± 1 and $g \in \mathbb{R}^n$ is random Gaussian. For the large perturbation case, each Δ_j is a random rank-25 positive semidefinite matrix. As such, A's trace and Frobenius norm monotonically increase over time, which is reflected in increasing absolute error among all algorithms. For the Restart algorithm, we restart every $q = 20$ time steps, resulting in periodic dips in error. The Q matrix-vector multiplications were evenly distributed to each block of 20 matrices, and then $1/3$ of those used for estimating the trace of the first matrix in the block, and the rest split evenly among the remaining 19. For NoRestart, the same number of vectors were allocated to A_1 as for Restart, and the rest evenly divided among all 99 remaining steps.

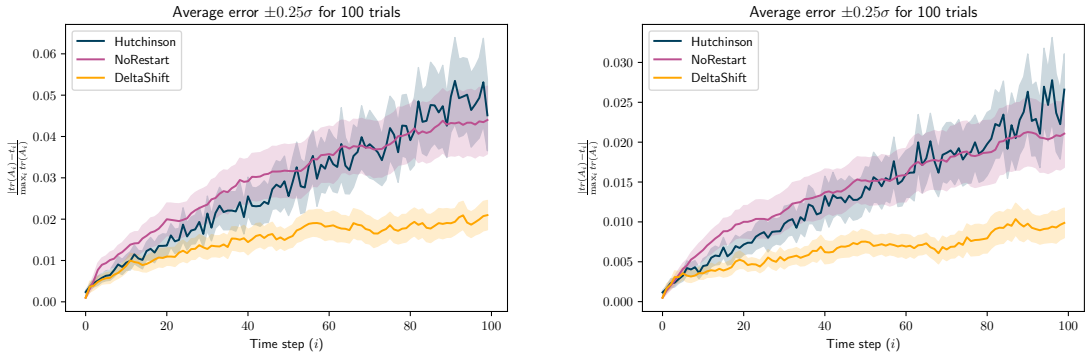
We report scaled absolute error between the estimator at time, t_j, and the true trace $\mathrm{tr}(A_j)$. We average error over 100 trials and report $\pm 0.25\sigma$ of the error. As expected, Hutchinson is outperformed even by NoRestart when each perturbation Δ_j is small. DeltaShift performs best, and its error actually improve slightly over time. DeltaShift also performs best for the large Δ_j experiment. We note that choosing the multiple parameters for the Restart algorithm was a major challenge in comparison to DeltaShift. Tuning the method becomes infeasible for larger experiments, or for multiple Q, so we exclude this method in our other experiments. That includes for the plots in Fig. 3, which show that DeltaShift continues to outperform Hutchinson and NoRestart for lower values of Q.

Estimating triangle counts in Graph: Our first real-data experiment is on estimating the number of triangles in a dynamic graph. For an undirected graph G with adjacency matrix B and entries either 0 or 1, the number of triangles in the graph is $\frac{1}{6}\mathrm{tr}(B^3)$. The number of triangles can be thus approximated by estimating the trace of B^3. The graph dataset we use is the Wikipedia vote network dataset with 7115 nodes (Leskovec et al., 2010a;b). At each timestep we perturb the graph by sampling a number of nodes (k) uniformly between 10 and 150 and adding a complete subgraph of size k. After 75 time steps, we start randomly deleting among the subgraphs added. This setup ensures that the perturbations to the graph in the initial stages are significant. For small perturbations, DeltaShift easily performs better than Hutchinson's, but we also wish to evaluate its performance in more challenging settings. We follow the same setup for number of matrix-vector products used by the estimators and the error reported as in the synthetic experiments. Note that for this particular application, the actual number of matrix-vector multiplications with B is $3Q$, since each oracle call computes $B(B(Bx))$. As seen in Fig. 4, DeltaShift provides the best estimates overall.

Hessian spectral density: Our second real-data experiment involves a dynamically changing Hessian matrix, H, for a neural network training problem. As discussed in Section 1, a common goal is to approximate the *spectral density* of H. One common way to do this is via the Kernel Polynomial Method (Weiße et al., 2006), which requires computing the trace of polynomials of the matrix H. Specifically, we consider the sequence of Chebyshev polynomials, but note that other polynomial basis sets can also be used (e.g., Legendre polynomials). The three term recurrence relation for the Chebyshev polynomials of first kind is:

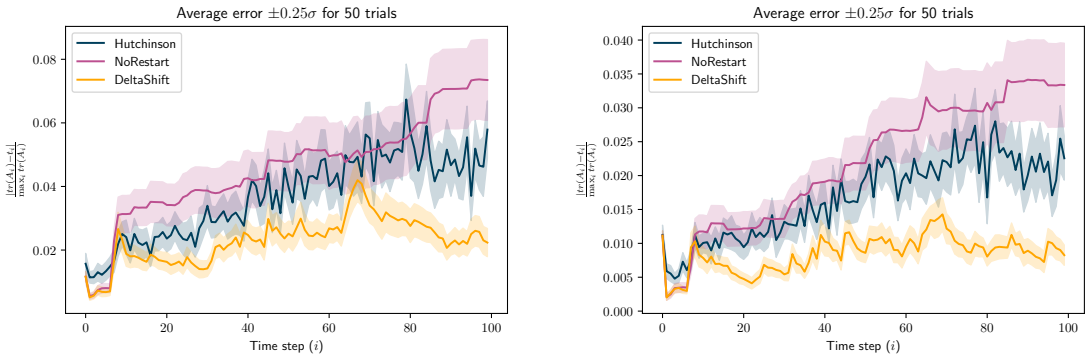
$$T_0(H) = I \qquad T_1(H) = H$$
$$T_{n+1}(H) = 2HT_n(H) - T_{n-1}(H). \tag{10}$$

Here I is the identity matrix. The Chebyshev polynomials form an orthogonal basis for functions in the range $[-1, 1]$. So, as a first step, we estimate the maximum eigenvalue of



(a) Synthetic data with $Q = 2 * 10^3$ (b) Synthetic data with $Q = 8 * 10^3$

Figure 3. Comparison of DeltaShift with Hutchinson and NoRestart for trace estimation on synthetic data.



(a) Graph data with $Q = 2 * 10^3$ (b) Synthetic data with $Q = 10^4$

Figure 4. Comparison of DeltaShift with Hutchinson and NoRestart for triangle counting in dynamic graph.

the Hessian using power iteration and scale $\tilde{H} = H/\lambda_{\max}$. Like trace estimation, power iteration requires computing Hessian-vector products, which we compute approximately using the PyHessian library (Yao et al., 2020). For a given neural network and loss function, PyHessian efficiently approximates Hessian-vector products by applying Pearlmutter's method to a randomly sampled batch of data points (Pearlmutter, 1994). To compute matrix-vector products with $T_0(H), T_1(H), \ldots, T_q(H)$, which are needed to approximate the trace of these matrices, we simply implement the recurrence of (10) using PyHessian's routine for Hessian-vector products. Multiplying by $T_q(H)$ requires q Hessian-vector products in total.

We consider a ResNet model with 269722 parameters and train it on the CIFAR-10 dataset. The model is trained using SGD and a batch size of 128, and we pre-train for 50 epochs before tracking the Hessian for 25 more steps. We report error in estimating the trace of the first 5 Chebyshev polynomials of H, averaged over 5 trials. As it is impossible to compute the true trace of these matrices, we use Hutchinson's estimator with a greater number of queries as placeholder for ground-truth, and compare the performance against the computed values.

As can be seen in Tables 1 and 2, DeltaShift obtains uniformly better approximation to the trace values, although the improvement is only marginal. This makes sense, as more progress on each training epoch implies a greater change in the Hessian over time, meaning α is larger and thus DeltaShift's advantage over Hutchinson's is smaller.

Table 1. Average error for trace of polynomials of Hessian with learning rate 0.001 and $Q = 2000$

	HUTCHINSON	NORESTART	DELTASHIFT
$T_1(H)$	2.5E-02	3.7E-02	1.7E-02
$T_2(H)$	1.2E-06	1.7E-06	8.0E-07
$T_3(H)$	4.0E-02	4.1E-02	3.1E-02
$T_4(H)$	1.5E-06	1.7E-06	1.0E-06
$T_5(H)$	2.1E-02	4.3E-02	1.9E-02

References

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

Avron, H. Counting triangles in large graphs using random-

Table 2. Average error for trace of polynomials of Hessian with learning rate 0.01 and $Q = 2000$

	HUTCHINSON	NOREstart	DELTASHIFT
$T_1(H)$	1.9E-02	5.0E-02	1.5E-02
$T_2(H)$	1.2E-06	2.9E-06	9.9E-07
$T_3(H)$	7.7E-02	9.4E-02	6.1E-02
$T_4(H)$	1.7E-06	2.8E-06	1.5E-06
$T_5(H)$	2.1E-02	4.2E-02	1.8E-02

ized matrix trace estimation. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2010.

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

Boutsidis, C., Drineas, P., Kambadur, P., and Zouzias, A. A randomized algorithm for approximating the log determinant of a symmetric positive definite matrix. *Linear Algebra and its Applications*, 533, 03 2015.

Braverman, M., Hazan, E., Simchowitz, M., and Woodworth, B. The gradient complexity of linear regression. In *Proceedings of the 33rd Annual Conference on Computational Learning Theory (COLT)*, volume 125, pp. 627–647, 2020.

Di Napoli, E., Polizzi, E., and Saad, Y. Efficient estimation of eigenvalue counts in an interval. *Numerical Linear Algebra with Applications*, 2016.

Dong, Z., Yao, Z., Cai, Y., Arfeen, D., Gholami, A., Mahoney, M. W., and Keutzer, K. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020.

Estrada, E. and Hatano, N. Communicability in complex networks. *Phys. Rev. E*, 77:036111, Mar 2008.

Gambhir, A. S., Stathopoulos, A., and Orginos, K. Deflation as a method of variance reduction for estimating the trace of a matrix inverse. *SIAM Journal on Scientific Computing*, 39(2):A532–A558, 2017.

Ghorbani, B., Krishnan, S., and Xiao, Y. An investigation into neural net optimization via hessian eigenvalue density. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97, pp. 2232–2241, 2019.

Girard, D. Un algorithme simple et rapide pour la validation croisee géénéralisée sur des problémes de grande taille. Technical report, 1987.

Han, I., Malioutov, D., and Shin, J. Large-scale log-determinant computation through stochastic Chebyshev expansions. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pp. 908–917, 2015.

Han, I., Malioutov, D., Avron, H., and Shin, J. Approximating the spectral sums of large-scale matrices using stochastic Chebyshev approximations. *SIAM Journal on Scientific Computing*, 2017.

Hanson, D. L. and Wright, F. T. A bound on tail probabilities for quadratic forms in independent random variables. *Ann. Math. Statist.*, 42(3):1079–1083, 06 1971.

Hanyu Li, Y. Z. Randomized block Krylov space methods for trace and log-determinant estimators. *arXiv:2003.00212*, 2020.

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

Higham, N. J. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, 2008.

Hutchinson, M. F. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450, 1990.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Leskovec, J., Huttenlocher, D., and Kleinberg, J. *Signed Networks in Social Media*, pp. 1361–1370. Association for Computing Machinery, New York, NY, USA, 2010a. ISBN 9781605589299. URL https://doi.org/10.1145/1753326.1753532.

Leskovec, J., Huttenlocher, D., and Kleinberg, J. Predicting positive and negative links in online social networks. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pp. 641–650, New York, NY, USA, 2010b. Association for Computing Machinery. ISBN 9781605587998. doi: 10.1145/1772690.1772756. URL https://doi.org/10.1145/1772690.1772756.

Lin, L. Randomized estimation of spectral densities of large matrices made accurate. *Numerische Mathematik*, 136 (1):183–213, 2017.

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

Mahoney, M. and Martin, C. Traditional and heavy tailed self regularization in neural network models. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97, pp. 4284–4293, 2019.

Martinsson, P.-G. and Tropp, J. A. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica*, 29:403–572, 2020.

Meyer, R. A., Musco, C., Musco, C., and Woodruff, D. Hutch++: optimal stochastic trace estimation. *Proceedings of the 4th Symposium on Simplicity in Algorithms (SOSA)*, 2021.

Musco, C., Netrapalli, P., Sidford, A., Ubaru, S., and Woodruff, D. P. Spectrum approximation beyond fast matrix multiplication: Algorithms and hardness. *Proceedings of the 9th Conference on Innovations in Theoretical Computer Science (ITCS)*, 2018.

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

Pennington, J., Schoenholz, S., and Ganguli, S. The emergence of spectral universality in deep networks. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1924–1932, 2018.

Qian, X., Li, V., and Darren, C. Channel-wise hessian aware trace-weighted quantization of neural networks. *arXiv preprint arXiv:2008.08284*, 2020.

Rudelson, M., Vershynin, R., et al. Hanson-Wright inequality and sub-Gaussian concentration. *Electronic Communications in Probability*, 18, 2013.

Saibaba, A. K., Alexanderian, A., and Ipsen, I. C. F. Randomized matrix-free trace and log-determinant estimators. *Numerische Mathematik*, 137(2):353–395, 2017.

Simchowitz, M., El Alaoui, A., and Recht, B. Tight query complexity lower bounds for PCA via finite sample deformed Wigner law. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 1249–1259, 2018.

Stathopoulos, A., Laeuchli, J., and Orginos, K. Hierarchical probing for estimating the trace of the matrix inverse on toroidal lattices. *SIAM Journal on Scientific Computing*, 35(5):S299–S322, 2013.

Sun, X., Woodruff, D. P., Yang, G., and Zhang, J. Querying a matrix through matrix-vector products. In *Proceedings of the 46th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 132, pp. 94:1–94:16, 2019.

Tang, J. M. and Saad, Y. Domain-decomposition-type methods for computing the diagonal of a matrix inverse. *SIAM Journal on Scientific Computing*, 33(5):2823–2847, 2011.

Ubaru, S. and Saad, Y. Applications of trace estimation techniques. In *High Performance Computing in Science and Engineering*, pp. 19–33, 2018.

Vershynin, R. *Introduction to the non-asymptotic analysis of random matrices*, pp. 210–268. Cambridge University Press, 2012. doi: 10.1017/CBO9780511794308.006.

Wainwright, M. J. *Basic tail and concentration bounds*, pp. 21–57. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2019. doi: 10.1017/9781108627771.002.

Wang, S., Sun, Y., Musco, C., and Bao, Z. Public transport planning: When transit network connectivity meets commuting demand. *Preprint*, 2020.

Wei, M. and Schwab, D. J. How noise affects the hessian spectrum in overparameterized neural networks. *arXiv preprint arXiv:1910.00195*, 2019.

Weiße, A., Wellein, G., Alvermann, A., and Fehske, H. The kernel polynomial method. *Reviews of modern physics*, 78(1):275, 2006.

Wimmer, K., Wu, Y., and Zhang, P. Optimal query complexity for estimating the trace of a matrix. In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 1051–1062, 2014.

Yao, Z., Gholami, A., Keutzer, K., and Mahoney, M. W. Pyhessian: Neural networks through the lens of the hessian. In *IEEE BigData*, 2020.

Yao, Z., Gholami, A., Shen, S., Keutzer, K., and Mahoney, M. W. ADAHESSIAN: An adaptive second order optimizer for machine learning. *AAAI Conference on Artificial Intelligence (AAAI 2021)*, 2021.

A. High Probability Proofs

In this section we give a full proof of Theorem 1.1 with the correct logarithmic dependence on $1/\delta$. Before doing so, we collect several definitions and results required for proving the theorem.

Definition 1. *(Wainwright, 2019) A random variable* X *with* $\mathbb{E}[\mathrm{X}] = \mu$ *is* sub-exponential *with parameters* (ν, β) *if its moment generating function satisfies:*

$$\mathbb{E}[e^{\lambda(\mathrm{X}-\mu)}] \leq e^{\frac{\nu^2\lambda^2}{2}} \quad \textit{for all } |\lambda| < \frac{1}{\beta}.$$

Claim A.1. *(Wainwright, 2019) Any sub-exponential random variable with parameters (ν, β) satisfies the tail bound*

$$\Pr\left[|\mathrm{X} - \mu| \geq t\right] \leq \begin{cases} 2e^{\frac{-t^2}{2\nu^2}} & \text{if } 0 \leq t \leq \frac{\nu^2}{\beta} \\ 2e^{\frac{-t}{2\beta}} & \text{for } t > \frac{\nu^2}{\beta}. \end{cases}$$

Claim A.2. *Let $\mathrm{X}_1, \mathrm{X}_2, \ldots, \mathrm{X}_k$ be independent random variables with mean $\mu_1, ..., \mu_k$ and sub-exponential parameters $(\nu_1, \beta_1), ...(\nu_k, \beta_k)$, then $\sum_{i=1}^k a_i \mathrm{X}_i$ is sub-exponential with parameters (ν_*, β_*) where,*

$$\nu_* = \sqrt{\sum_{i=1}^k a_i^2 \nu_i^2} \quad \text{and} \quad \beta_* = \max_{i=1,\ldots,k} a_i \beta_i$$

Proof. The proof is straight-forward by computing the moment generating function of $\sum_{i=1}^k a_i \mathrm{X}_i$ using the independence of $\mathrm{X}_1, \mathrm{X}_2, \ldots, \mathrm{X}_k$. Specifically, for $|\lambda| < 1/(\max_{i=1,\ldots,k} a_i \beta_i)$ we have:

$$\begin{aligned} \mathbb{E}[e^{\lambda \sum_{i=1}^k a_i(\mathrm{X}_i - \mu_i)}] &= \prod_{i=1}^k \mathbb{E}[e^{\lambda a_i(\mathrm{X}_i - \mu_i)}] \\ &\leq \prod_{i=1}^k e^{\frac{\lambda^2 a_i^2 \nu_i^2}{2}} = e^{\frac{\lambda^2 \nu_*^2}{2}}. \end{aligned}$$

□

As discussed, a tight analysis of Hutchinson's estimator, and also our DeltaShift algorithm, relies on the *Hanson-Wright inequality* (Hanson & Wright, 1971), which shows that any quadratic form involving a vector with i.i.d. sub-Gaussian entries is a sub-exponential random variable. Specifically, we use the following version of the inequality:

Claim A.3. *[Corollary of Theorem 1.1, (Rudelson et al., 2013)] For $A \in \mathbb{R}^{n \times n}$, let $h_\ell(A)$ be Hutchinson's estimator as defined in Section 2.2, implemented with Rademacher random vectors. $h_\ell(A)$ is a sub-exponential random variable with parameters*

$$\nu = \frac{c_1 \|A\|_F}{\sqrt{\ell}} \quad \text{and} \quad \beta = \frac{c_2 \|A\|_2}{\ell},$$

where c_1, c_2 are absolute constants.

Proof. Recall that $h_\ell(A)$ is an average of ℓ independent random variables, each of the form $g^T A g$, where $g \in \mathbb{R}^n$ is a vector with independent ± 1 Rademacher random entries. We start by decoupling $g^T A g$ in two sums involving diagonal and off-diagonal terms in A:

$$g^T A g = \sum_{i=1}^n g_i^2 A_{ii} + \sum_{i,j:i \neq j} A_{ij} g_i g_j.$$

Here g_i denotes the i^{th} entry of g. Since each g_i is sampled i.i.d. from a ± 1 Rademacher distribution, the first term is constant with value $\sum_{i=1}^n A_{ii} = \mathrm{tr}(A)$. Rudelson et al. (2013) derive a bound on the moment generating function for the off-diagonal term, which they denote $S = \sum_{i,j:i \neq j}^n A_{ij} g_i g_j$. Specifically, they show that

$$\mathbb{E}\left[e^{\lambda S}\right] \leq e^{c_1^2 \|A\|_F^2 \lambda^2 / 2}, \quad \text{for all } |\lambda| < \frac{1}{c_2 \|A\|_2},$$

where c_1, c_2 are positive constants. As S is mean zero, we conclude that it is sub-exponential with parameters $(c_1\|A\|_F, c_2\|A\|_2)$ (refer to Definition 1), and thus $g^T A g$ (which is just S added to a constant) is sub-exponential with same parameters. Finally, from Claim A.2, we immediately have that $h_\ell(A)$ is sub-exponential with parameters $\left(\frac{c_1\|A\|_F}{\sqrt{\ell}}, \frac{c_2\|A\|_2}{\ell}\right)$. Note that, while we only consider ± 1 Rademacher random vectors, a similar analysis can be performed for any i.i.d. sub-Gaussian random entries by showing that the diagonal term is itself subexponential (it will no longer be constant). The result will involve additional constants depending on the choice of g_i. In the case when g_i are i.i.d. standard normals, the diagonal term is a scaled chi-squared random variable. □

Now, we are ready to move on to the main result.

Theorem 1.1 (Restated)**.** *For any $\epsilon, \delta, \alpha \in (0,1)$, Algorithm 1 run with $\gamma = \alpha$, $\ell_0 = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$, and $\ell = O\left(\frac{\alpha \log(1/\delta)}{\epsilon^2}\right)$ solves Problem 1. In total, it requires*

$$O\left(m \cdot \frac{\alpha \log(1/\delta)}{\epsilon^2} + \frac{\log(1/\delta)}{\epsilon^2}\right)$$

matrix-vector multiplications with $A_1, \ldots, A_m$.

Proof. The proof is by induction. Let $t_1, \ldots, t_m$ be the estimators for $\mathrm{tr}(A_1), \ldots, \mathrm{tr}(A_m)$ returned by Algorithm 1. We claim that, for all $j = 1, \ldots, m$, t_j is sub-exponential with parameters

$$\nu_j \leq \frac{\epsilon}{2\sqrt{\log(2/\delta)}}, \qquad \beta_j \leq \frac{\epsilon^2}{4\log(2/\delta)}. \tag{11}$$

If we can prove (11), the theorem immediately follows by applying Claim A.1 with $t = \epsilon$ to the random variable t_j. Recall that $\mathbb{E}[t_j] = \mathrm{tr}(A_j)$

First consider the base case, $t_1 = h_{l_0}(A_1)$. By Claim A.3, $h_{l_0}(A_1)$ is sub-exponential with parameters $\left(\frac{c_1}{\sqrt{\ell_0}}, \frac{c_2\|A_1\|_2}{\ell_0}\right)$. Noting that $\|A_1\|_2 \leq \|A_1\|_F \leq 1$ and setting constants appropriately on ℓ_0 gives the bound.

Next consider the inductive case. Recall that $t_j = (1-\gamma)t_{j-1} + h_\ell(\widehat{\Delta_j})$, where $\widehat{\Delta_j} = A_j - (1-\gamma)A_{j-1}$. As

shown in Section 3.1, $\|\widehat{\Delta_j}\|_F \leq 2\alpha$. So by Claim A.3, $h_\ell(\widehat{\Delta_1})$ is sub-exponential with parameters $\left(\frac{2c_1\alpha}{\sqrt{\ell}}, \frac{2c_2\alpha}{\ell}\right)$. As long as $\ell = c \cdot \frac{\alpha \log(2/\delta)}{\epsilon^2}$ for sufficiently large constant c, we therefore have by Claim A.2 that

$$\beta_j = \max\left[(1-\gamma)\beta_{j-1}, \frac{2c_2\alpha}{\ell}\right] \leq \frac{\epsilon^2}{4\log(2/\delta)}.$$

Note that above we used the $\|\widehat{\Delta_j}\|_2 \leq \|\widehat{\Delta_j}\|_F \leq 2\alpha$. Setting $\gamma = \alpha$, we also have

$$\begin{aligned}\nu_j^2 &= (1-\alpha)^2\nu_{j-1}^2 + \left(\frac{2c_1\alpha}{\sqrt{\ell}}\right)^2 \\ &\leq (1-\alpha)\nu_{j-1}^2 + \alpha\nu_{j-1}^2 = \nu_{j-1}^2.\end{aligned}$$

The inequality $\left(\frac{2c_1\alpha}{\sqrt{\ell}}\right)^2 \leq \alpha\nu_{j-1}^2$ follows as long as $\ell = c \cdot \frac{\alpha \log(2/\delta)}{\epsilon^2}$ for sufficiently large constant c. We have thus proven (11) and the theorem follows. □

B. Experimental setup

All experiments were run on server with 2vCPU @2.2GHz and 27 GB main memory and P100 GPU with 16GB memory.