Proceedings of Machine Learning Research vol 107:75–92, 2021 2nd Annual Conference on Mathematical and Scientific Machine Learning

BEAR: Sketching BFGS Algorithm for Ultra-High Dimensional Feature Selection in Sublinear Memory

Amirali Aghazadeh AMIRALIA@BERKELEY.EDU
Vipul Gupta VIPUL_GUPTA@BERKELEY.EDU
Alex DeWeese MDEWEESE@BERKELEY.EDU
O. Ozan Koyluoglu OZAN.KOYLUOGLU@BERKELEY.EDU
Kannan Ramchandran KANNANR@EECS.BERKELEY.EDU
Department of Electrical Engineering and Computer Science, University of California, Berkeley

Editors: Joan Bruna, Jan S Hesthaven, Lenka Zdeborova

Abstract

We consider feature selection for applications in machine learning where the dimensionality of the data is so large that it exceeds the working memory of the (local) computing machine. Unfortunately, current large-scale sketching algorithms show poor memory-accuracy trade-off in selecting features in high dimensions due to the irreversible collision and accumulation of the stochastic gradient noise in the sketched domain. Here, we develop a second-order feature selection algorithm, called BEAR, which avoids the extra collisions by efficiently storing the *second-order stochastic gradients* of the celebrated Broyden–Fletcher–Goldfarb–Shannon (BFGS) algorithm in Count Sketch, using a memory cost that grows sublinearly with the size of the feature vector. BEAR reveals an unexplored advantage of second-order optimization for memory-constrained high-dimensional gradient sketching. Our extensive experiments on several real-world data sets from genomics to language processing demonstrate that BEAR requires *up to three orders of magnitude* less memory space to achieve the same classification accuracy compared to the first-order sketching algorithms with a comparable run time. Our theoretical analysis further proves the global convergence of BEAR with $\mathcal{O}(1/t)$ rate in t iterations of the sketched algorithm.

Keywords: Feature selection, sketching, second-order optimization, sublinear memory.

1. Introduction

Consider a data set comprising n data points $(\boldsymbol{\theta}_i)_{i=1}^n = (\mathbf{x}_i, y_i)_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^p$ denotes the data vectors representing p features and $(y_i)_{i=1}^n$ denote the corresponding labels. Feature selection seeks to select a small subset of the features of size $k \ll p$ that best models the relationship between $\mathbf{x}_i$ and y_i. In this paper, we consider the feature selection problem in ultra-high dimensional settings where dense feature vectors in $\mathbb{R}^p$ cannot fit in the working memory of the computer because of the sheer dimensionality of the problem (p). Such problems have become increasingly important in biology, chemistry, networking, and streaming applications. In biology, it is common to represent a DNA sequence comprised of four nucleotides A, T, C, G, as well as 11 wild-card characters in the FASTQ format Deorowicz and Grabowski (2011), using the frequency of sub-sequences of length k, called k-mers, with $k \geq 12$ Vervier et al. (2016); Aghazadeh et al. (2016). A feature vector of size $15^{k=12}$ with floating-point numbers requires more than a petabyte of memory to store. This is simply

larger than the memory capacity of the computers today. In streaming, the memory budget of the local edge computing devices is extremely small compared to the dimension of the data streams Yu et al. (2013). In both scenarios, it is critical to select a subset of the features that are most predictive of the outputs with *lowest memory* cost in the dimensionality of the data.

Recently, first-order stochastic gradient descent (SGD) algorithms Aghazadeh et al. (2018); Tai et al. (2018) have been developed which extend the ideas in feature hashing (FH) Weinberger et al. (2009) to feature selection. Instead of explicitly storing the feature vectors, these algorithms store a low-dimensional sketch of the features in a data structure called Count Sketch Charikar et al. (2002), originated from the streaming literature. The key idea behind Count Sketch is in linearly binning (colliding) a random subset of features into the same bucket (i.e., an entry of Count Sketch). As long as the total number of features with high weight (i.e., the heavy hitters) is small, such collisions won't affect the weights of the heavy hitters. In particular, Count Sketch preserves the weights of the top-k features with high probability using a memory cost that grows sublinearly with the size of the feature vector (p). This high probability guarantee, however, depends on the energy of the non-top-k coordinates in the SGD algorithm. In particular, the noise components of the gradients, which normally average out in the regular stochastic optimization, accumulate in the non-top-k coordinates of Count Sketch. This unwanted sketched noise increases the probability of deleterious collision of heavy hitters in Count Sketch, deteriorates the quality of the recovered features, and results in poor memory-accuracy trade-offs. This is a critical problem since the only class of optimization algorithms that operates in such ultra-high dimensions does not select high-quality features when the memory budget is small.

We propose a novel optimization scheme to solve this critical problem in sketching. We improve the quality of the sketched gradients and correct for the unwanted collisions in the sketched domain using the information from the second-derivative of the loss function. Second-order methods have recently gained increasing attention in machine learning for their faster convergence Agarwal et al. (2017), less reliance on the step size parameter Xu et al. (2020), and their superior communication-computation trade-off in distributed processing Yao et al. (2019). Here, we uncover another key advantage of second-order optimization in improving memory-accuracy trade-off in sketching models trained on ultra-high dimensional data sets. We develop a second-order optimization algorithm with a memory cost that grows sublinearly with the size of the feature vector (p). Our algorithm finds high-quality features by *limiting the probability of extra collisions* due to the stochastic noise in Count Sketch. The contributions of the paper are as follows:

Algorithm. We develop BEAR which, to the best of our knowledge, is the first quasi-Newton-type algorithm that achieves a memory cost that grows sublinearly with the size of the feature vector (p). We demonstrate that applying sublinear memory data structures such as Count Sketch in the second-order optimization is challenging particularly because Hessian, unlike the gradient, cannot be directly sketched. In response, BEAR stores the product of the inverse Hessian and the gradient in the Broyden–Fletcher Goldfarb–Shannon (BFGS) algorithm using Count Sketch. BEAR updates Count Sketch in time quadratic in the sparsity of the data by operating only on the features that are active in each minibatch.

Theory. We theoretically demonstrate that BEAR maintains the $\mathcal{O}(1/t)$ global convergence rate of the online version of limited-memory BFGS algorithm (oLBFGS) Mokhtari and Ribeiro (2015) in t iterations. We show that the convergence rate is retained as we go from the ambient domain to the sketched domain. The analysis employs the matrix Bernstein inequality to bound the non-zero eigenvalues of the projection operator in Count Sketch. In practice, we demonstrate that BEAR

converges faster than the first-order feature selection algorithms—although improving convergence time is not the main focus of this work.

Experiments. In real-world, ultra-high dimensional data sets from genomics, natural language processing, and networking, we demonstrate that BEAR requires $10-1000\times$ less memory space to achieve the same classification accuracy as the first-order methods. Moreover, BEAR achieves $10-20\%$ higher classification accuracy given the same memory budget to store Count Sketch and selects more interpretable features in ultra-high dimensions using a personal laptop-size machine. Importantly, our results show an increase in the performance gap between the first- and second-order methods as the memory budget to store the model parameters decrease, which highlights the important advantages of second-order optimization in storing the sketched stochastic gradient vectors with a lower collision rate.[1]

Simulations. We performed extensive controlled sparse recovery simulations with data points drawn from the normal distribution. We demonstrate that, given a fixed memory budget to store the weights, BEAR recovers the ground truth features with a large phase transition gap — an important statistical performance metric from the compressive sensing literature. We show that BEAR's performance is highly consistent across a large range of values for the step size parameter because of the second-order nature of the algorithm.

2. Review: Count Sketch

Count Sketch is a data structure which is originated from the streaming literature Charikar et al. (2002). Its primary application is to approximately count the number of occurrences of a very large number (p) of objects in sublinear memory when only the frequency of the most recurring elements (i.e., the heavy hitters) are of interest. Instead of storing a counter for all the p objects, Count Sketch *linearly* projects the count values using d independent random hash functions into a $m \ll p$ dimensional subspace. Count Sketch keeps a matrix of counters (or bins) $\mathcal{S}$ of size $c \times d = m \sim \mathcal{O}(\log p)$ and uses d random hash functions $h_j : \forall j \in \{1,\ 2, ...,\ d\}$ to map p-dimensional vectors to $c = m/d$ bins, that is, $h_j : \{1,\ 2, ...,\ p\} \rightarrow \{1,\ 2, ...,\ c\}$. For any row j of sketch $\mathcal{S}$, component i of the vector is hashed into bin $\mathcal{S}(j, h_j(i))$. In addition to h_j, Count Sketch uses d random sign functions to map the components of the vectors randomly to $\{+1,\ -1\}$, that is, $s_j : \{1,\ 2, ...,\ p\} \rightarrow \{+1, -1\}$.

Count Sketch supports two operations: ADD(item i, increment Δ) and QUERY(item i). The ADD operation updates the sketch with any observed increment. More formally, for an increment Δ to an item i, the sketch is updated by adding $s_j(i)\Delta$ to the cell $\mathcal{S}(j, h_j(i))\ \forall j \in \{1,\ 2, ...,\ d\}$. The QUERY operation returns an estimate for component i, the median of all the d different associated counters. In this paper, the objects that we aim to "count" are the weights of the coordinates of the gradient vector in the feature selection algorithm. Count Sketch provides the following bound in recovering the top-k coordinates of the feature vector $\mathbf{z} \in \mathbb{R}^p$:

Theorem 1 *Charikar et al. (2002) Count Sketch finds approximate top-k coordinates z_i with $\pm\varepsilon\|\mathbf{z}\|_2$ error, with probability at least $1-\delta$, in space $\mathcal{O}(log(\frac{p}{\delta})(k+\frac{\|\mathbf{z}^{tail}\|_2^2}{(\varepsilon\zeta)^2}))$, where $\|\mathbf{z}^{tail}\|_2^2 = \sum_{i \notin top-k} z_i^2$ is the energy of the non-top-k coordinates and ζ is the k^{th} largest value in $\mathbf{z}$.*

1. Codes are available at https://github.com/BEAR-algorithm/BEAR

Count Sketch recovers the top-k coordinates with a memory cost that grows only logarithmic with the dimension of the data p; and naturally, it requires the energy of the non-top-k coordinates to be sufficiently small. This is the property that we leverage in this paper in order to improve feature selection accuracy in ultra-high dimensions.

3. Stochastic Sketching for Feature Selection

We first elaborate on how can we perform feature selection using Count Sketch. Recall that feature selection seeks to select a small subset of the features that best models the relationship between $\mathbf{x}_i$ and y_i. This relationship is captured using a sparse feature vector $\boldsymbol{\beta}^* \in \mathbb{R}^p$ that minimizes a given loss function $f(\boldsymbol{\beta}, \boldsymbol{\theta}) : \mathbb{R}^p \to \mathbb{R}$ using the optimization problem $\min_{\boldsymbol{\beta}} \mathbb{E}_{\boldsymbol{\theta}}[f(\boldsymbol{\beta}, \boldsymbol{\theta})]$, where $\boldsymbol{\theta} \in \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_n\}$ denotes a data point in a data set of size n. This problem is solved using the empirical risk minimization

$$\boldsymbol{\beta}^* \coloneqq \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} f(\boldsymbol{\beta}, \boldsymbol{\theta}_i), \tag{1}$$

using the SGD algorithm which produces the updates $\boldsymbol{\beta}_{t+1} \coloneqq \boldsymbol{\beta}_t - \eta_t \mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$ at iteration t, where η_t is the step size, the minibatch $\boldsymbol{\Theta}_t = \{\boldsymbol{\theta}_{t1}, \boldsymbol{\theta}_{t2}, \ldots, \boldsymbol{\theta}_{tb}\}$ contains b independent samples from the data, and $\mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t) = \sum_{i=1}^{b} \nabla_{\boldsymbol{\beta}_t} f(\boldsymbol{\beta}_t, \boldsymbol{\theta}_{ti})$ is the stochastic gradient of the instantaneous loss function $f(\boldsymbol{\beta}, \boldsymbol{\Theta}_t)$. In this paper, we are interested in the setting where the dense feature vector $\boldsymbol{\beta}_t$ of size p cannot be stored in the memory of a computer. The most common approach in machine learning when dealing with such high dimensional problem is to project the data points (i.e., the features) into a lower dimensional space. Feature hashing (FH) is one of the most popular algorithms Weinberger et al. (2009) which uses a universal hash function to project the features. While FH is ideal for prediction, it is not suited for feature selection; that is, the original important features cannot be recovered from the hashed ones.

The reason to stay hopeful in recovering the important features using sublinear memory is that the feature vector $\boldsymbol{\beta}^*$ is typically sparse in ultra-high dimensions. However, while the final feature vector is sparse, $\boldsymbol{\beta}_t$ becomes dense in the intermediate iterations of the algorithm. The workaround is to store a sketch of the intermediate non-sparse $\boldsymbol{\beta}_t$ using a low dimensional sketched vector $\boldsymbol{\beta}_t^s \in \mathbb{R}^m$ (with $m \ll p$) such that the important features are still recoverable. This results in the following sketched optimization steps $\boldsymbol{\beta}_{t+1}^s \coloneqq \boldsymbol{\beta}_t^s - \eta_t \mathbf{g}^s(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$, where $\mathbf{g}^s(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$ is the sketched gradient vector. To enable the recovery of the important features from the sketched features the weights $\boldsymbol{\beta}_t^s \in \mathbb{R}^m$ can be stored in Count Sketch Charikar et al. (2002); Aghazadeh et al. (2018). Count Sketch preserves the information of the top-k elements (i.e., the heavy hitters) with high probability as long as the energy of the non-top-k coefficients is sufficiently small (see Theorem 1) . The noise term $\mathbf{g}^s(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$ in the SGD algorithm, however, contributes to the energy of the non-top-k coefficients. This is a critical problem since, unlike SGD in the ambient dimension, this spurious sketched noise does not cancel out until it becomes so large that it shows up in the top-k coordinates in Count Sketch. As a result, a large fraction of the memory in Count Sketch will be wasted to store the sketched noise term in the non-top-k coordinates, which results in poor memory-accuracy trade off in selecting features in first-order methods.

4. Challenges of Second-order Sketching in Ultra-High Dimension

Algorithm 1 Limited-memory BFGS

Input: $\mathbf{g}(\hat{\boldsymbol{\beta}}_t, \boldsymbol{\Theta}_t)$ and $\{\mathbf{s}_i, \mathbf{r}_i\}_{i=t-\tau+1}^{t}$

1. $\rho_t = \frac{1}{\mathbf{r}_t^T \mathbf{s}_t}$.
2. $\mathbf{q}_t = \mathbf{g}(\hat{\boldsymbol{\beta}}_t, \boldsymbol{\Theta}_t)$,
   for $i = t$ to $t - \tau + 1$:
   $\alpha_i = \rho_i \mathbf{s}_i^T \mathbf{q}_i$,
   $\mathbf{q}_{i-1} = \mathbf{q}_i - \alpha_i \mathbf{r}_i$.
3. $\mathbf{z}_{t-\tau} = \frac{\mathbf{r}_t^T \mathbf{s}_t}{\mathbf{r}_t^T \mathbf{r}_t} \mathbf{q}_{t-\tau}$,
   for $i = t - \tau + 1$ to t:
   $\gamma_i = \rho_i \mathbf{r}_i^T \mathbf{z}_i$.
   $\mathbf{z}_i = \mathbf{z}_{i-1} + \mathbf{s}_i(\alpha_i - \gamma_i)$.

Return: $\mathbf{z}_t$

In this paper, we propose a second-order optimization algorithm for feature selection to reduce the effect of collisions while sketching SGD into lower dimensions. Recall that the stochastic second-order Newton's method produces the updates $\boldsymbol{\beta}_{t+1} \coloneqq \boldsymbol{\beta}_t - \eta_t \mathbf{B}_t^{-1} \mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$, where $\mathbf{B}_t = \nabla^2_{\boldsymbol{\beta}_t} f(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t) \in \mathbb{R}^{p \times p}$ is the instantaneous Hessian at iteration t computed over the minibatch $\boldsymbol{\Theta}_t$. However, there are critical challenges in sketching these updates for ultra-high dimensional feature selection: First, computing the Hessian and finding the matrix inverse is computationally hard as the matrix inverse operation is going to have cubic computational complexity in the problem dimension (i.e., $\mathcal{O}(p^3)$) in the worst case. Recent works have allowed for efficient inversion of the Hessian matrix with a computational complexity which grows linearly with p (i.e., $\mathcal{O}(pr^2)$) using a rank-r approximation of Hessian or grouping the eigenvalues of Hessian into r clusters (see e.g., Bollapragada et al. (2019); O'Leary-Roseberry et al. (2019)). However, the computational complexity of all these efficient algorithms grow in the best case linearly with p. Second, even if we assume that the Hessian is approximately diagonal (e.g., in AdaHessian Yao et al. (2020)), storing the diagonal elements will have linear memory cost in p, which we can not afford in ultra-high dimension. Third, sketching the Hessian directly is not possible using Count Sketch since the linear increments are happening over the gradients and not the Hessian in SGD.

Unfortunately, all the recent literature in fast second-order optimization also operates with at least linear memory and time complexity. Namely, Quasi-Newton's methods such as the BFGS algorithm reduce the time complexity of Newton's method using an iterative update of the Hessian as a function of the variations in the gradients $\mathbf{r}_t = \mathbf{g}(\boldsymbol{\beta}_{t+1}, \boldsymbol{\Theta}) - \mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta})$ and the feature vectors $\mathbf{s}_t = \boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}_t$ which ensures that the Hessian satisfies the so-called secant equation $\mathbf{B}_{t+1}\mathbf{s}_t = \mathbf{r}_t$. While BFGS avoids the heavy computational cost involved in the matrix inversion, it still has a quadratic memory requirement. The limited-memory BFGS (LBFGS) algorithm reduces the memory requirement of the BFGS algorithm from quadratic to linear by estimating the product of the inverse Hessian and the gradient vector $\mathbf{z}_t = \mathbf{B}_t^{-1} \mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$ without explicitly storing the Hessian Nocedal (1980). This makes use of the difference vectors $\mathbf{r}_t$ and $\mathbf{s}_t$ from the last τ iteration of the algorithm (Alg. 1). Recently, an online LBFGS algorithm (oLBFGS) Mokhtari and Ribeiro (2015) with global linear convergence guarantee and an incremental greedy BFGS algorithm (IGS) Gao et al. (2020) with non-asymptotic local superlinear convergence guarantee have been developed. However, both the online limited-memory and incremental BFGS algorithms fail to run on ultra-high dimensional data sets due to their linear memory requirement. This motivates the central question that we aim to address: how can we enjoy the benefits of second-order optimization for feature selection in sublinear memory?

Algorithm 2 BEAR

Initialize: $t = 0$, Count Sketch $\boldsymbol{\beta}^s_{t=0} = 0$, top-$k$ heap.

while stopping criteria not satisfied **do**

1. Sample b independent data points in a minibatch $\boldsymbol{\Theta}_t = \{\boldsymbol{\theta}_{t1}, \ldots, \boldsymbol{\theta}_{tb}\}$.
2. Find the active set $\mathcal{A}_t$.
3. QUERY the feature weights in $\mathcal{A}_t \cap$ top-k from Count Sketch $\boldsymbol{\beta}_t = query(\boldsymbol{\beta}^s_t)$.
4. Compute stochastic gradient $\mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$.
5. Compute the descent direction with Alg. 1 $\mathbf{z}_t$ = LBFGS($\mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$, $\{\mathbf{s}_i, \mathbf{r}_i\}^t_{i=t-\tau+1}$).
6. ADD the sketch of $\mathbf{z}_t$ at the active set $\hat{\mathbf{z}}_t = {\mathbf{z}_t}^{\mathcal{A}_t}$ to Count Sketch $\boldsymbol{\beta}^s_{t+1} \coloneqq \boldsymbol{\beta}^s_t - \eta_t \hat{\mathbf{z}}^s_t$.
7. QUERY the features weights in $\mathcal{A}_t \cap$ top-k from Count Sketch $\boldsymbol{\beta}_{t+1} = query(\boldsymbol{\beta}^s_{t+1})$.
8. Compute stochastic gradient $\mathbf{g}(\boldsymbol{\beta}_{t+1}, \boldsymbol{\Theta}_t)$.
9. Set $\mathbf{s}_{t+1} = \boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}_t$, and $\mathbf{r}_{t+1} = \mathbf{g}(\boldsymbol{\beta}_{t+1}, \boldsymbol{\Theta}_t) - \mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$.
10. Update the top-k heap.
11. $t = t + 1$.

end while

Return: The top-k heavy-hitters in Count Sketch.

5. The BEAR Algorithm

Our feature selection algorithm, BEAR, estimates the second-derivative of the loss function from sketched features. Instead of explicitly storing the product of the inverse Hessian and the gradient (done in oLBFGS), BEAR maintains a Count Sketch $\mathcal{S}$ to store the feature weights in sublinear memory and time quadratic in the sparsity of the input data. The key insight is to update the Count Sketch by the sketch of the gradient corrected by the sketch of the difference vectors $\mathbf{r}_t$ and $\mathbf{s}_t$. As detailed in Alg. 2, BEAR first initializes Count Sketch with zero weights and a top-k heap to store the top-k features. In every iteration, it samples b independent data points $\boldsymbol{\Theta}_t$ and identifies the active set $\mathcal{A}_t$, that is, the features that are present in $\boldsymbol{\Theta}_t$. It then queries Count Sketch to retrieve the feature weights that are in the intersection of the active set $\mathcal{A}_t$ and the top-k heap and set the weights for the rest of the features to zero. Next, it computes the stochastic gradient $\mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$ and uses it along with the difference vectors $\mathbf{r}_i$ and $\mathbf{s}_i$ from the last τ iterations to find the descent direction $\mathbf{z}_t$ using the LBFGS algorithm detailed in Alg. 1. Then, it adds the sketch of the descent direction $\mathbf{z}_t$ *only at the features in the active set* $\hat{\mathbf{z}}_t = {\mathbf{z}_t}^{\mathcal{A}_t}$ to Count Sketch. BEAR queries Count Sketch for the second time in order to update the difference vector $\mathbf{r}_{t+1} = \mathbf{g}(\boldsymbol{\beta}_{t+1}, \boldsymbol{\Theta}_t) - \mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$ and uses the sketch vector $\hat{\mathbf{z}}_t$ to set $\mathbf{s}_{t+1}$. The difference vector $\mathbf{r}_{t+1}$ captures the changes in the gradient vector as the content of Count Sketch change *over a fixed minibatch* $\boldsymbol{\Theta}_t$ Mokhtari and Ribeiro (2015). Finally, BEAR updates the top-k heap with the active set $\mathcal{A}_t$ and moves on to the next iteration until the convergence criteria is met. To update the top-k heap, BEAR scans the features that have been changed in Count Sketch over the past iteration. If those features are already in the heap, it updates the values of those elements, and if the features are new, it inserts the new elements into the heap with a worst-case time complexity that grows logarithmic with the number of features k. In the rare scenario, where the intersection of the active set $\mathcal{A}_t$ and the top-k heap is empty the gradient can still be non-zero and can change the feature weightings.

The most time-costly step of BEAR, which computes the descent direction (step 5), is quadratic time in the sparsity of the data $|\mathcal{A}_t|$. Table 1 summarizes the worst-case memory complexity of the vectors involved in BEAR. The dominant term is Count Sketch $\boldsymbol{\beta}^s_t$ for which the memory complexity

$\boldsymbol{\beta}_t$	$\mathbf{s}_t$	$\mathbf{r}_t$	$\mathbf{z}_t$	$\boldsymbol{\beta}_t^s$	$\mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$
k	$2\lvert\mathcal{A}_t\rvert$	$2\lvert\mathcal{A}_t\rvert$	$2\tau\lvert\mathcal{A}_t\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{A}_t\rvert$

Table 1: Memory cost of the vectors in BEAR.

in terms of the top-k value are established in Theorem 1. The memory requirement to store the auxiliary vector $\mathbf{z}_t$ is only a constant τ times larger than the size of the active set which is negligible in streaming data-sparse features compared to the size of Count Sketch (see section 7).

Convergence Analysis. We now analyze the convergence of the algorithm. Consider the following standard assumptions Mokhtari and Ribeiro (2015) on the instantaneous functions $f(\boldsymbol{\beta}, \boldsymbol{\Theta})$ to prove the convergence of the BEAR algorithm: 1) The instantaneous objective functions are twice differentiable with the instantaneous Hessian being positive definite, that is, the eigenvalues of the instantaneous Hessian satisfy $M_1\mathbf{I} \preceq \nabla^2_{\boldsymbol{\beta}} f(\boldsymbol{\beta}, \boldsymbol{\Theta}) \preceq M_2\mathbf{I}$, for some $0 < M_1 < M_2$. 2) The norm of the gradient of the instantaneous functions is bounded for all $\boldsymbol{\beta}$, that is, $\mathbb{E}_{\boldsymbol{\Theta}}[\|\mathbf{g}(\boldsymbol{\beta}, \boldsymbol{\Theta})\|^2 \mid \boldsymbol{\beta}] \leq S^2$. 3) The step sizes η_t are square-summable. More specifically, $\sum_{t=0}^{\infty} \eta_t = \infty$ and $\sum_{t=0}^{\infty} \eta_t^2 < \infty$. We prove the following theorem.

Theorem 2 *Let $f(\cdot)$ and the step sizes η_t satisfy the assumptions above. Let the size of Count Sketch be $m = \theta(\varepsilon^{-2} \log 1/\delta)$ with number of hashes $d = \theta(\varepsilon^{-1} \log 1/\delta)$ for $\varepsilon, \delta > 0$. Then, the Euclidean distance between updates $\boldsymbol{\beta}_t^s$ in the* BEAR *algorithm and the sketch of the solution of problem (1) converges to zero with probability $1 - \delta$, that is,*

$$\mathbb{P}(\lim_{t \to \infty} \|\boldsymbol{\beta}_t^s - \boldsymbol{\beta}^{s*}\|^2 = 0) = 1 - \delta, \tag{2}$$

where the probability is over the random realizations of random samples $\{\boldsymbol{\Theta}_t\}_{t=0}^{\infty}$. Furthermore, for the specific step size $\eta_t = \eta_0/(t + T_0)$ for some constants η_0 and T_0, the model parameters at iteration t satisfy

$$\mathbb{E}_{\boldsymbol{\Theta}}[f(\boldsymbol{\beta}_t^s, \boldsymbol{\Theta}) - \mathbb{E}[f(\boldsymbol{\beta}^{s*}, \boldsymbol{\Theta})] \leq \frac{C_0}{T_0 + t}, \tag{3}$$

with probability $1 - \delta$. Here, C_0 is a constant depending on the parameters of the sketching scheme, the above assumptions, and the objective function.

The proof makes use of the matrix Bernstein inequality in projecting the second-order gradients in Count Sketch Kane and Nelson (2014) which we defer to the Proofs section. For sufficiently sparse solutions the convergence in the ambient domain follows from convergence in the sketched domain (i.e., Theorem (2)) and the Count Sketch guarantee (i.e., Theorem (1)):

Corollary 3 *Let $\pi(\cdot)$ be a permutation on $\{1, 2, \cdots, p\}$ such that $\beta^*_{\pi(1)} \geq \beta^*_{\pi(2)} \geq \cdots \beta^*_{\pi(p)}$, where $\boldsymbol{\beta}^* = [\beta_1^*, \beta_2^*, \cdots, \beta_p^*]$ is the optimal solution to (1). Also, let*

$$m = \max\left[\mathcal{O}\left(log(\frac{2p}{\delta})\left(k + \frac{\|\boldsymbol{\beta}^{*tail}\|_2^2}{(\varepsilon\zeta)^2}\right)\right), \theta\left(\frac{1}{\varepsilon^2}\log\frac{2}{\delta}\right)\right] \tag{4}$$

and number of hashes $d = \theta(\varepsilon^{-1} \log 2/\delta)$ *where* $\varepsilon, \delta > 0, \|\boldsymbol{\beta}^{*tail}\|_2^2 = \sum_{i=k+1}^{p} (\beta^*_{\pi(i)})^2$ *and* $\zeta = \beta^*_{\pi(k)}$. *Then,*

$$|\beta^*_{\pi(i)} - \beta_{t\pi(i)}| \leq \epsilon \|\boldsymbol{\beta}^*\|_2 \quad \textit{for all } i \in \{1, 2, \cdots, k\} \textit{ with probability } 1 - \delta, \tag{5}$$

where $\boldsymbol{\beta}_t = [\beta_{t1}, \beta_{t2}, \cdots, \beta_{tp}]$ *is the output of the* BEAR *algorithm.*

This completes the convergence proof of BEAR in sublinear memory in the ambient space.

6. Simulations

We have conducted sparse recovery simulations to evaluate the performance BEAR compared to the first-order feature selection algorithm in ultra-high dimension MISSION Aghazadeh et al. (2018). MISSION is one of the only first-order optimization algorithms that enables feature selection in a memory cost that grows sublinearly with the size of the feature vector and as a result runs in the scale of problems that we are considering in this paper. In addition, comparing the accuracy of BEAR with MISSION as a baseline shows the power of sketching the second-order gradients (done in BEAR) against sketching first-order gradients (done in MISSION). The synthetic simulations described in this section have ground truth features, so we can assess the algorithms in a more controlled environment and compare the results using a variant of the phase transition plot from the compressive sensing literature Maleki and Donoho (2010). We also show the results of the full Newton's method version of our BEAR algorithm where we compute the Hessian rather than its oLBFGS approximation (this algorithm cannot operate in large-scale settings). The same hash table (hash functions and random seeds) and step sizes are used for BEAR and MISSION. Hyperparameter search is performed to select the value of the step sizes in both algorithms. The entries of the data vectors $\mathbf{x}_i$ are sampled from an i.i.d. Gaussian distribution with zero mean and unit variance. The output labels y_i are set using a linear forward model $y_i = \mathbf{x}_i \boldsymbol{\beta}^*$, where $\boldsymbol{\beta}^*$ is a k-sparse ground truth feature vector. The indices of the support (i.e., the non-zero entries) and the weights of the non-zero entries in $\boldsymbol{\beta}^*$ are drawn uniformly at random respectively from the sets $[1, p]$ and $[0.8, 1.2]$ and MSE is used as the loss function. The same experiment is repeated 200 times with different realization of the data vectors $\mathbf{x}$. Convergence at iteration t is reached when the norm of the gradient drops below 10^{-7} consistently in all the algorithms. The algorithms are compared in terms of the accuracy in selecting the ground truth features as well as the sensitivity of the algorithms to the choice of the value of the step size.

Feature Selection Accuracy. The task is to select $k = 8$ features in a data set with $n = 900$ rows (data points) and $p = 1000$ columns (features). The size of Count Sketch is varied from 10% to 60% of the total memory required to store a $p = 1000$ dimensional feature vector. This ratio, that is the ratio of data dimension p to Count Sketch size, is called the compression factor. For each value of the compression factor, the experiment is repeated 200 times. Fig. 1A shows the fraction of iterations in which the algorithms find *all* the ground truth features correctly, that is, the probability of success. Fig. 1B illustrates the same results in terms of the average ℓ_2-norm of the error of the recovered feature vectors $\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|_2$. BEAR significantly outperforms MISSION in terms of the probability of success and the average ℓ_2-norm error. The gap is more pronounced in higher compression factors; given a compression factor of 3, MISSION has almost no power in predicting the correct features while the BEAR and Newton's methods achieve a 0.5 probability of success. Fig. 1A and B further suggest that the performance gap between BEAR and it's exact Hessian counterpart is small showing that the oLBFGS makes a good approximation to the Hessian in terms of the selected features.

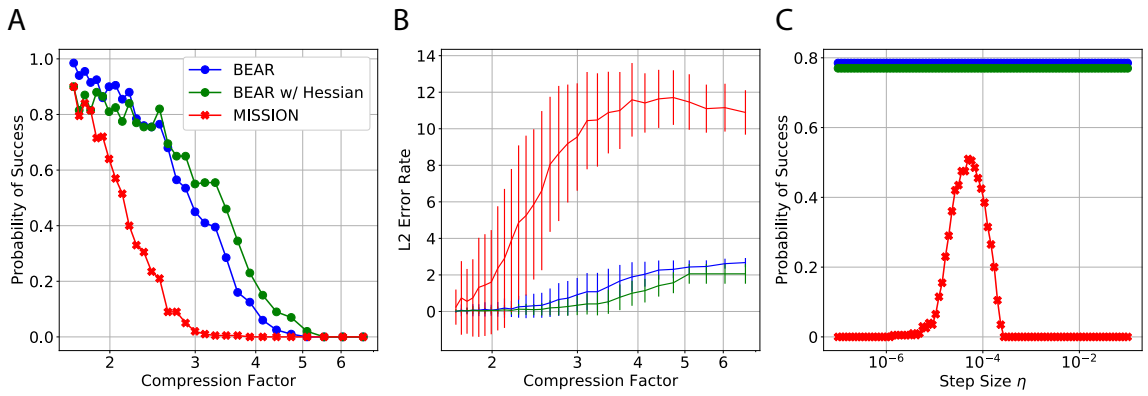

Figure 1: Feature selection experiments on $p = 1000$-dimensional synthetic data sets with entries drawn from normal distribution. A) Probability of success in recovering all features correctly as a function of compression factor. B) Recovery error rate in terms of the ℓ_2-norm. C) Probability of success as a function of the value of the step size (compression factor $= 2.22$).

Sensitivity to Step Size. The experimental setup is similar to the previous section except the Sketch size is fixed and step size varies. The experiment is repeated 200 times while varying the values of the step size η ranging from 10^{-7} to 10^{-1} and the probability of success is reported. Count Sketch of size 150×3 is used for both MISSION and BEAR. Fig. 1C illustrates the probability of success for BEAR and MISSION as a function of the step size. The plot shows that BEAR is fairly agnostic and MISSION is dependent on the choice of the step size η. MISSION's accuracy peaks around $\eta = 10^{-4}$ and sharply drops as η deviates from this value. BEAR's lower-dependence on step size is ideal for streaming settings where the statistics of the data might change over time and there is not enough time and memory budget to do step size selection.

7. Experiments

We designed the experiment in a way to answer the following questions:

- Does BEAR outperform MISSION in terms of classification accuracy? In particular, how does the performance gap between the algorithms change as a function of the memory allocated for Count Sketch?
- How does BEAR perform on real-world large-scale data sets ($p >$ 50 million)?
- How does BEAR perform in terms of classification accuracy compared to FH?
- How does changing the number of top-k features affect the accuracy of the feature selection algorithms?
- What is the convergence behaviour of BEAR when the memory budget is small?
- What is the run time of BEAR compared to MISSION?

We compare the performance of these baseline algorithms with BEAR: 1) **Stochastic Gradient Descent (SGD)**: For data sets with sufficiently small dimension and size to be able to train a classifier on our laptop machine, we perform the vanilla SGD algorithm (with $\mathcal{O}(p)$ memory). 2) **oLBFGS**: Similar to SGD, for the data sets that the dimension and size allows to train a classifier on our laptop

Data set	Dim (p)	#Train (n)	#Test	Size	#Act.
RCV1	47,236	20,242	677,399	1.2GB	73
Webspam	16,609,143	280,000	70,000	25GB	3730
DNA	16,777,216	600,000	600,000	1.5GB	89
KDD 2012	54,686,452	119,705,032	29,934,073	22GB	12

Table 2: Summary of the real-world data sets.

machine, we perform the vanilla oLBFGS algorithm (neither SGD nor the oLBFGS techniques do feature selection or model compression). 3) **Feature Hashing (FH)**: FH Weinberger et al. (2009) is a standard algorithm to do prediction (classification) in large-scale machine learning problems. FH hashes the data features into a lower dimensional space *before* the training process and is not a feature selection algorithm. 4) **MISSION**: As mentioned earlier, MISSION is a first-order optimization algorithm for feature selection which sketches the noisy stochastic gradients into Count Sketch.

Performance Metrics. The algorithms are assessed in terms of the following performance metrics: 1) **Classification accuracy**: Once the algorithms converge, the performance of the algorithms in terms of classification accuracy are compared, that is, the fraction of test samples that are classified to correct classes. 2) **Area under the ROC curve (AUC)**: For the data sets that the class distribution are highly skewed the area under the ROC curve (AUC) is reported instead of the classification accuracy. In these data sets, the class probabilities are taken as the output of the classifiers. 3) **Compression factor (CF)**: The compression factor is defined as the dimension of the data set p divided by the size of Count Sketch m. For multi-class classification problems, m is the total memory of all the Count Sketches used for all the classes. A higher compression factor means a smaller memory budget is allocated to store the model parameters. SGD and oLBFGS have a compression factor of one. 4) **Run time**: The run time of the algorithms to converge in minutes.

Real-World Data sets. The key statistics of the data sets used in the paper are tabulated in Table 2 including the dimension of the data set (p), number of training data (n), number of test data, total size of the data set, and the average number of active (non-zero) features per data point. All the data is analyzed in the Vowpal Wabbit format.

1) **RCV1: Reuters Corpus Volume I.** RCV1 is an archive of manually categorized news wire stories made available by Reuters, Ltd. for research purposes. The negative class label includes Corporate/Industrial/Economics topics and positive class labels includes Government/Social/Markets topics (see Lewis et al. (2004)). The data set is fairly balanced between the two classes.

2) **Webspam: Web Spam Classification.** Web spam refers to Web pages that are created to manipulate search engines and Web users. The data set is a large collection of annotated spam/nonspam hosts labeled by a group of volunteers (see Webb et al. (2006)). It is slightly class-imbalanced with 60% samples from class 1.

3) **DNA: Metagenomics.** A data set that we dub "DNA" from metagenomics. Metagenomics studies the composition of microbial samples collected from various environments (for example human gut) by sequencing the DNA of the living organisms in the sample. The data set comprises of short DNA sequences which are sampled from a set of 15 DNA sequences of bacterial genomes. The task is to train a classifier to label the DNA sequences with their corresponding bacteria. DNA sequences are encoded using their constituent sub-sequences called K-mers (see Vervier et al. (2016)). The training and test data have an equal number of samples for each class. A naive guessing strategy achieves a classification accuracy of 0.06.

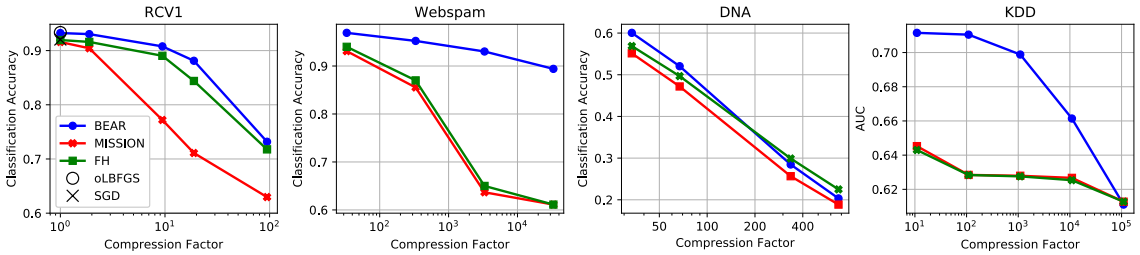

Figure 2: Classification performance as a function of the compression factor in real-world data sets.

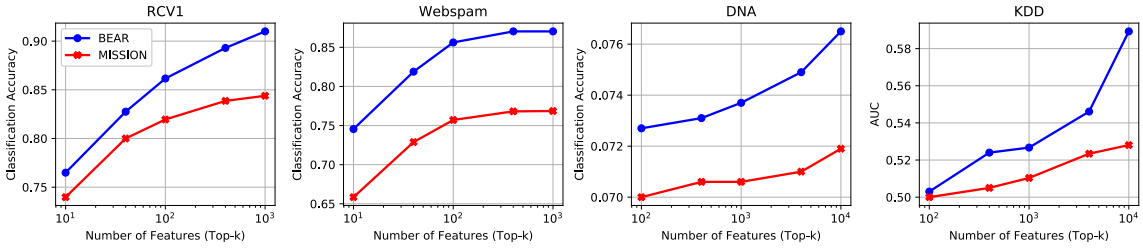

Figure 3: Classification performance as a function of the number of the top-k features.

4) **KDD Cup 2012: Click-Through Rate Prediction.** A key idea in search advertising is to predict the click-through rate (pCTR) of ads, as the economic model behind search advertising requires pCTR values to rank ads and to price clicks. The KDD Cup 2012 data set comprises training instances derived from session logs of the Tencent proprietary search engine (see Juan et al. (2016)). The data set is highly class-imbalanced with 96% samples from class 1 (click).

Multi-class Extension. For the multi-class classification problems stated above, we developed a multi-class version of the BEAR algorithm. In the multi-class problem one natural assumption is that there are separate subsets of features that are most predictive for each class. Our multi-class BEAR algorithm accommodates for this by maintaining a separate Count Sketch and heap to store the the top-k features associated with each class. The total memory complexity of the algorithm grows linearly with the number of classes. For a fair comparison, we use the exact same multi-class Count Sketch extension for MISSION. We have also implemented the single Count Sketch version of BEAR, however, since the multi-class Count Sketch extension performs better for feature selection we report the results of the former in our experiments.

Experimental Setup. MurmurHash3 with 32-bit hash values is used to implement the hash functions in MISSION, BEAR, and FH. The algorithms are trained in a streaming fashion using the cross entropy loss. The algorithms are run for a single epoch so that each algorithm sees a data point once on average. The size of the minibatches and the step size are kept consistent across the algorithms. The constant $\tau = 5$ in BEAR however the results are consistent across a large range of values for τ. Both in BEAR and MISSION a Count Sketch with 5 rows (hash functions) is used. The lower dimensional embedding size of FH is set equal to the total size of Count Sketch in BEAR. The experiments are performed on a single laptop machine - 2.4 GHz Quad-Core Intel Core i5 with 16 GB of RAM. We chose an edge device as opposed to a computing server for our real-world experiments to showcase the applicability of BEAR in a resource constrained environment.

BEAR	manage	entrepreneur	colombian	decade	oppress
MISSION	peach	week	nora	demand	incomplete

Table 3: Examples of the features selected in RCV1.

data set (CF)	RCV1 (95)	Webs (332)	DNA (22)	KDD (10^3)
BEAR	0.1	5	26	25
MISSION	0.3	19	55	33

Table 4: Overall run time comparison (minutes).

Result I) Classification Performance vs. Compression Factor. We assess the classification performance of BEAR compared to the baseline algorithms for different compression factors in Fig. 2. All the active features in the test data are used at the inference step for a fair comparison with FH. BEAR's classification performance is consistently better than MISSION and FH across all the data sets over a wide range of compression factors while showing a hysteresis behaviour: the performance gap increases as the compression factor grows until Count Sketch is too small to yield any prediction power. The classification performance of all algorithms degrades with larger compression factors, which is expected since lower Count Sketch sizes increase the probability of collisions in both BEAR and MISSION. The degradation, however, impacts MISSION significantly more that BEAR. In particular, BEAR's performance stays relatively robust for compression rates in the range of $1-10$ in RCV1, $1-1000$ in Webspam, and $10-100$ in KDD, while the classification performance of MISSION drops rapidly. The increasing performance gap between BEAR and MISSON with compression factor highlights the unique advantage of BEAR in storing the second-order steps in Count Sketch and lowering the probability of collisions. Note that this performance gap is less pronounced in the DNA data set while the general trend still follows the other data sets. This is because the DNA data set has 15 balanced classes and its K-mer features have relatively more distributed information content compared to the features in the other data sets, which poses a harder feature selection task for the algorithms.

Result II) Classification Performance vs. Top-k Features. We assess the performance of BEAR in terms of the classification accuracy against the number of selected top-k features. The compression factors are fixed to 10, 330, 330, and 1100 respectively for the data sets in Fig. 3. SGD, oLBFGS, and FH cannot select features, therefore, they are not included in this analysis. The plots shows that BEAR selects features that are better in terms of prediction accuracy for a wide range of values of k. The gap grows for larger k. We analyzed the selected features in RCV1 for which a proper documentation of the features is publicly available (unlike the other data sets). Some of the selected features are shared among the algorithms, for example, "shareholder", "nigh", and "company", which can be attributed to the Markets, Social, and Industrial subjects, respectively. Other terms, however, are uniquely chosen by one of the algorithms as tabulated in Table 3. Compared to BEAR, the terms selected by MISSION are less frequent (e.g., "peach") and do not discriminate between the subject classes (e.g., "incomplete").

Result III) Run Time. We compare the overall run time of BEAR with MISSION in Table 4. BEAR is significantly faster than MISSION consistently in all the data sets; BEAR makes a better use of the data by estimating the curvature of the loss function and converges faster.

8. Discussion and Conclusion

We have developed BEAR, which to the best of our knowledge is the first second-order optimization algorithm for ultra-high dimensional feature selection in sublinear memory. Our results demonstrate that BEAR has up to three orders of magnitude smaller memory footprint in feature selection compared to the first-order sketching algorithms with comparable (and sometimes superior) run time. We showed that the benefits of BEAR is far more pronounced while sketching into lower-dimensional subspaces, which is due to the more accurate decent directions of second-order gradients resulting in less collision-causing noise in Count Sketch. The implications of memory-accuracy advantage of second-order methods goes beyond hashing and streaming and can be applied to improve the communication-computation trade-off in distributed learning in communicating the sketch of the stochastic gradients between nodes Ivkin et al. (2019); Gupta et al. (2019). Moreover, while we laid out the algorithmic principles in sketching second-order gradient for training ultra-high dimensional generalized linear classifiers with theoretical guarantees, similar algorithmic principles can be used in sketching nonlinear models such as deep neural networks on lower-dimensional data sets Yao et al. (2020). We believe that our work will open up new research directions towards understanding the benefits of second-order optimization in training massive-scale machine learning models in memory-constrained environments.

Proofs

Theorem 2. Before stating the proof, for more clarity, we will reiterate the problem setup and our assumptions from the main paper here. We are interested in solving the following problem using BEAR

$$\boldsymbol{\beta}^* \coloneqq \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^p} f(\boldsymbol{\beta}, \boldsymbol{\Theta}) = \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{T}\sum_{t=1}^{T} f(\boldsymbol{\beta}, \boldsymbol{\theta}_t) = \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{T}\sum_{t=1}^{T} f(\mathbf{X}_t\boldsymbol{\beta}, \mathbf{y}_t), \tag{6}$$

where $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_T\}$ and $\boldsymbol{\theta}_t = (\mathbf{X}_t, \mathbf{y}_t)\ \forall\ t \in [1, T]$. We make the following standard assumptions Mokhtari and Ribeiro (2015):

1. The instantaneous objective functions, $f(\cdot)$, in Eq. (6) are twice differentiable with the instantaneous Hessian being positive definite. That is, the eigenvalues of the instantaneous Hessian satisfy

$$M_1\mathbf{I} \preceq \nabla^2_{\boldsymbol{\beta}} f(\boldsymbol{\beta}, \boldsymbol{\Theta}) \preceq M_2\mathbf{I}, \tag{7}$$

   for some $0 < M_1 \leq M_2$.

2. The norm of the gradient of the instantaneous functions $f(\cdot)$ in Eq. 6 is bounded for all $\boldsymbol{\beta}$, that is

$$\mathbb{E}_{\boldsymbol{\Theta}}[\|\mathbf{g}(\boldsymbol{\beta}, \boldsymbol{\Theta})\|^2 \mid \boldsymbol{\beta}] \leq S^2. \tag{8}$$

3. The step-sizes η_t are square-summable. More specifically,

$$\sum_{t=0}^{\infty} \eta_t = \infty \text{ and } \sum_{t=0}^{\infty} \eta_t^2 < \infty. \tag{9}$$

Lemma 4 *The solution of problem in* (6) *using* BEAR *(or its first-order variant MISSION) is equivalent to the solution of the following problem in the sketched domain,*

$$\boldsymbol{\beta}^{s*} \coloneqq \underset{\boldsymbol{\beta}^s \in \mathbb{R}^m}{\arg\min} \frac{1}{T} \sum_{t=1}^{T} f(\mathbf{X}_t \mathbf{S} \boldsymbol{\beta}^s, \mathbf{y}_t), \tag{10}$$

where multiplication by $\mathbf{S} \in \mathbb{R}^{p \times m}$ *is the linear projection operator in Count Sketch and* $\boldsymbol{\beta}^s \in \mathbb{R}^m$ *is the projected model parameters.*

Proof Let the update for online gradient descent for the original problem in Eq. (6) be given by

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \eta_t \nabla f(\mathbf{X} \boldsymbol{\beta}_t, \mathbf{y}_t) \tag{11}$$

For BEAR/MISSION type algorithms, the model parameters are stored in a Count Sketch based hash table. The compressed vector can be represented by an affine transformation as $\boldsymbol{\beta}_t^s = \mathbf{S}^T \boldsymbol{\beta}_t$, where $\mathbf{S} \in \mathbb{R}^{p \times m}$ is the Count Sketch matrix Kane and Nelson (2014). While updating the model, the indices corresponding to the non-zero values in the gradient (the oLBFGS update in case of BEAR) are updated by querying Count Sketch. For Count Sketch with mean query operator, the update for MISSION can be written as

$$\boldsymbol{\beta}_{t+1}^s = \boldsymbol{\beta}_t^s - \eta_t \mathbf{S}^T \nabla f(\mathbf{X} \mathcal{Q}(\boldsymbol{\beta}_t^s), \mathbf{y}_t) \tag{12}$$

where $\mathcal{Q}(\cdot)$ is the query function and $\boldsymbol{\beta}_t^s = \mathbf{S}^T \boldsymbol{\beta}_t$ is the sketched model parameter vector. When the query is the mean operator, the $\mathcal{Q}(\cdot)$ is the affine transformation $\mathcal{Q}(x) = \mathbf{S}\mathbf{x}$ for any $\mathbf{x} \in \mathbb{R}^m$ Kane and Nelson (2014); Woodruff et al. (2014). Thus, the MISSION update equation is given by

$$\boldsymbol{\beta}_{t+1}^s = \boldsymbol{\beta}_t^s - \eta_t \mathbf{S}^T \nabla f(\mathbf{X} \mathbf{S} \boldsymbol{\beta}_t^s, \mathbf{y}_t). \tag{13}$$

The gradient for the problem in Eq. (10) is given by $\nabla f_{\boldsymbol{\beta}^s}(\cdot) = \mathbf{S}^T \nabla f(\cdot)$. Hence, its online gradient descent update is the same as MISSION's update in Eq. (13). Since BEAR is a second-order variant of MISSION, it attempts to solve the same problem as MISSION. Next, we show that it indeed solves the problem with high probability at a linear convergence rate. ■

Now, to show that BEAR converges to $\boldsymbol{\beta}^{s*}$, we first need to show that the problem in Eq. (10) also satisfies the assumptions in Eq. (7) and (8) (albeit with different constants). Then, we can invoke the convergence guarantees for oLBFGS from Mokhtari and Ribeiro (2015) to show that BEAR converges at a linear rate.

Lemma 5 *Assume Count Sketch has a size* $m = \Theta(\varepsilon^{-2} \log 1/\delta)$ *with the number of hashes* $d = \Theta(\varepsilon^{-1} \log 1/\delta)$. *If the instantaneous function* $f(\boldsymbol{\beta}, \boldsymbol{\Theta})$ *satisfy Assumptions 1 and 2 (Eq.* (7) *and* (8)*, respectively), then, the corresponding instantaneous function for the sketched problem* $f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})$ *also satisfy*

$$\frac{p}{m}(1-\varepsilon) M_1 \mathbf{I} \preceq \nabla^2_{\boldsymbol{\beta}^s} f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta}) \preceq \frac{p}{m} M_2 (1+\varepsilon) \mathbf{I}, \tag{14}$$

$$\mathbb{E}_{\boldsymbol{\theta}}[\|\nabla_{\boldsymbol{\beta}^s} f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})\|^2 \mid \boldsymbol{\beta}] \leq \frac{p}{m} M_2 (1+\varepsilon) S^2, \tag{15}$$

with probability $1 - \delta$ *each.*

Proof The instantaneous Hessian for the sketched problem (say $\mathbf{H}^s$) is given by

$$\mathbf{H}^s = \nabla^2_{\boldsymbol{\beta}^s} f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta}) = \mathbf{S}^T \nabla^2 f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta}) \mathbf{S} = \mathbf{S}^T \mathbf{H} \mathbf{S}, \tag{16}$$

where $\mathbf{H} = \nabla^2 f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})$ is the instantaneous Hessian for the original problem. Since commuting matrices have the same set of non-zero eigenvalues, the eigenvalues of $\mathbf{S}^T\mathbf{H}\mathbf{S}$ are equal to the eigenvalues of $\mathbf{H}\mathbf{S}\mathbf{S}^T$. Hence,

$$\begin{aligned}
\lambda_{\max}(\mathbf{H}^s) &= \lambda_{\max}(\mathbf{S}^T\mathbf{H}\mathbf{S}) = \lambda_{\max}(\mathbf{H}\mathbf{S}\mathbf{S}^T) && (17)\\
&\leq \lambda_{\max}(\mathbf{H})\lambda_{\max}(\mathbf{S}\mathbf{S}^T) && (18)\\
&\leq M_2 \, \lambda_{\max}(\mathbf{S}\mathbf{S}^T) && (19)\\
&= M_2 \, \lambda_{\max}(\mathbf{S}^T\mathbf{S}), && (20)
\end{aligned}$$

where $\lambda_{\max}(\cdot)$ denotes the maximum eigenvalue and $\lambda_{\max}(\mathbf{H}) \leq M_2$ by assumption. Also, Eq. (18) uses the fact that the maximum eigenvalue of the product of two symmetric matrices is upper bounded by the product of maximum eigenvalues of individual matrices.

For the count sketch matrix $\mathbf{S} \in \mathbb{R}^{p \times m}$, we have $\mathbb{E}[\mathbf{S}^T\mathbf{S}] = \frac{p}{m}\mathbf{I}$. Moreover, by applying the matrix Bernstein inequality (Tropp, 2015) on the matrix $\mathbf{Z} = \mathbf{S}^T\mathbf{S} - \frac{p}{m}\mathbf{I} = \sum_{i=1}^{p} \left(\mathbf{S}_i^T\mathbf{S}_i - \frac{1}{m}\mathbf{I}\right) = \sum_{i=1}^{p} \mathbf{Z}_i$, where $\mathbf{S}_i$ is the i-th row in $\mathbf{S}$ and $\mathbf{Z}_i = \left(\mathbf{S}_i^T\mathbf{S}_i - \frac{1}{m}\mathbf{I}\right) \; \forall \; i \in [1, p]$, we get the following bound $\mathbb{P}\left(\|\mathbf{Z}\| \geq \epsilon\frac{p}{m}\right) \leq 2m \exp \frac{-\epsilon^2 p^2/(2m^2)}{v(\mathbf{Z}) + L\epsilon p/(3m)}$, where $\|\mathbf{Z}_i\| \leq L \; \forall \; i$ and $v(\mathbf{Z}) = \|\sum_{i=1}^{p} \mathbf{Z}_i^T\mathbf{Z}_i\|$. For the count sketch matrix, we have $L = 1$ and $v(\mathbf{Z}) = \frac{d(k-1)}{k^2}$. Thus, we get $\mathbb{P}\left(\|\mathbf{Z}\| \geq \epsilon\frac{p}{m}\right) \leq 2m \exp \frac{-\epsilon^2 p^2/(2m^2)}{p(m-1)/m^2 + \epsilon p/(3m)}$. Further, for any $m = O(\sqrt{p})$, the R.H.S. in the above inequality is upper bounded by δ. Note that the sketch-size m is generally independent and (order-wise) much less than the bigger dimension p, and hence $m = O(\sqrt{p})$ can be easily satisfied by choosing appropriate constants $\epsilon > 0$ and $\delta < 1$. Thus, we get the following bound on the eigenvalues of $\mathbf{S}^T\mathbf{S}$ (also the non-zero eigenvalues in $\mathbf{S}\mathbf{S}^T$)

$$\frac{p}{m}(1-\varepsilon) \leq \lambda_i(\mathbf{S}^T\mathbf{S}) \leq \frac{p}{m}(1+\varepsilon) \text{ for all } i \in [1, m] \tag{21}$$

with probability $1 - \delta$. Using this in (18), we get

$$\lambda_{\max}(\mathbf{H}^s) \leq \frac{p}{m} M_2 (1+\varepsilon) \tag{22}$$

with probability $1 - \delta$.

Similarly, we can write the smallest eigenvalue of $\mathbf{H}^s$ as

$$\lambda_{\min}(\mathbf{H}^s) = \frac{1}{\lambda_{\max}((\mathbf{H}^s)^{-1})} = \frac{1}{\lambda_{\max}((\mathbf{S}^T\mathbf{H}\mathbf{S})^{-1})} = \frac{1}{\lambda_{\max}((\mathbf{H}\mathbf{S}\mathbf{S}^T)^{\dagger})}, \tag{23}$$

where $(\cdot)^{\dagger}$ is the Moore-Penrose inverse and the last inequality again uses the fact that commuting matrices have the same set of non-zero eigenvalues. Thus, $\mathbf{S}^T\mathbf{H}\mathbf{S}$ and $\mathbf{H}\mathbf{S}\mathbf{S}^T$, and their corresponding inverses, have the same set of non-zero eigenvalues. Let's define the truncated eigenvalue decomposition of $\mathbf{S}\mathbf{S}^T$ as $\mathbf{S}\mathbf{S}^T = \mathbf{U}\Lambda\mathbf{U}^T$ and note that $(\mathbf{H}\mathbf{S}\mathbf{S}^T)^{\dagger} = (\mathbf{H}\mathbf{U}\Lambda\mathbf{U}^T)^{\dagger} = (\mathbf{U}^T)^{\dagger}\Lambda^{-1}(\mathbf{U})^{\dagger}\mathbf{H}^{-1}$. Hence, we get

$$\begin{aligned}
\lambda_{\max}((\mathbf{H}\mathbf{S}\mathbf{S}^T)^{\dagger}) &= \lambda_{\max}((\mathbf{U}^T)^{\dagger}\Lambda^{-1}(\mathbf{U})^{\dagger}\mathbf{H}^{-1})\\
&\leq \lambda_{\max}(\Lambda^{-1})\lambda_{\max}((\mathbf{H})^{-1})\\
&= \lambda_{\max}((\mathbf{S}\mathbf{S}^T)^{\dagger})\lambda_{\max}((\mathbf{H})^{-1}) && (24)
\end{aligned}$$

since Λ^{-1} contains the non-zero eigenvalues of $(\mathbf{S}\mathbf{S}^T)^\dagger$. Thus,

$$\begin{aligned}\lambda_{\min}(\mathbf{H}^s) &\geq \frac{1}{\lambda_{\max}((\mathbf{S}\mathbf{S}^T)^\dagger)\lambda_{\max}((\mathbf{H})^{-1})} \\ &\geq \lambda_{\min}(\mathbf{S}\mathbf{S}^T)\lambda_{\min}(\mathbf{H}) \\ &\geq \lambda_{\min}(\mathbf{S}\mathbf{S}^T)M_1 \\ &= \lambda_{\min}(\mathbf{S}^T\mathbf{S})M_1 \\ &\geq \frac{p}{m}M_1(1-\varepsilon), \end{aligned} \tag{25}$$

with probability $1-\delta$. where the last inequality follows from (21). This proves the desired result.

Similarly, to prove that the gradient of the sketched problem is bounded, observe that $\|\nabla_{\boldsymbol{\beta}^s} f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})\|^2 = \|\mathbf{S}^T\nabla f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})\|^2 \leq \frac{p}{m}M_2(1+\varepsilon)\|\nabla f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})\|^2$ with probability $1-\delta$, where the last inequality follows from (21). Hence,

$$\mathbb{E}_{\boldsymbol{\theta}}[\|\nabla_{\boldsymbol{\beta}^s} f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})\|^2 \leq \frac{p}{m}M_2(1+\varepsilon)\mathbb{E}_{\boldsymbol{\theta}}[\|\nabla f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})\|^2 \leq \frac{p}{m}M_2(1+\varepsilon)S^2 \tag{26}$$

with probability $1-\delta$, where the second inequality follows from assumption in (8). ■

Finally, to prove Theorem 2, we invoke the results from Mokhtari and Ribeiro (2015). According to Theorem 6 in Mokhtari and Ribeiro (2015), oLBGS with instantaneous functions satisfying assumptions in Eqs. (14) and (26) converges with probability one. Hence, for BEAR, we get

$$\mathbb{P}(\lim_{t\to\infty} \|\boldsymbol{\beta}_t^s - \boldsymbol{\beta}^{s*}\|^2 = 0) = 1-\delta. \tag{27}$$

Moreover, for the specific step-size $\eta_t = \eta_0/(t+T_0)$, where η_0 and T_0 satisfy the inequality $2m_1\eta_0 T_0 > C$ for some constant C, BEAR satisfies the following rate of convergence (Theorem 7 in Mokhtari and Ribeiro (2015))

$$\mathbb{E}[f(\boldsymbol{\beta}_t^s, \boldsymbol{\Theta}) - \mathbb{E}[f(\boldsymbol{\beta}^{s*}, \boldsymbol{\Theta})] \leq \frac{C_0}{T_0+t}, \tag{28}$$

with probability $1-\delta$, where the constant C_0 is given by

$$C_0 = \max\left\{\frac{\eta_0^2 T_0^2 C M_2 p^2 S^2(1+\varepsilon)^2}{2c^2 m[M_1 p\eta_0 T_0(1-\varepsilon) - Cm]}, T_0(\mathbb{E}[f(\boldsymbol{\beta}^s, \boldsymbol{\Theta}) - \mathbb{E}[f(\boldsymbol{\beta}^{s*}, \boldsymbol{\Theta})])\right\}.$$

Acknowledgements

This work is supported by the following grants: NSF CIF-1703678, NSF CNS-1748692, NSF CCF-1748585, and MLWiNS 2002821.

References

N. Agarwal, B. Bullins, and E. Hazan. Second-order stochastic optimization for machine learning in linear time. *The Journal of Machine Learning Research*, 18(1):4148–4187, 2017.

A. Aghazadeh, A. Y. Lin, M. A. Sheikh, A. L. Chen, L. M. Atkins, C. L. Johnson, J. F. Petrosino, R. A. Drezek, and R. G. Baraniuk. Universal microbial diagnostics using random DNA probes. *Sci. Adv.*, 2(9):e1600025, 2016.

A. Aghazadeh, R. Spring, D. Lejeune, G. Dasarathy, A. Shrivastava, et al. MISSION: Ultra Large-Scale Feature Selection using Count-Sketches. In *Intl. Conf. Machine Learning (ICML)*, pages 80–88, 2018.

Raghu Bollapragada, Richard H Byrd, and Jorge Nocedal. Exact and inexact subsampled newton methods for optimization. *IMA Journal of Numerical Analysis*, 39(2):545–578, 2019.

M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Intl. Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

S. Deorowicz and S. Grabowski. Compression of DNA sequence reads in FASTQ format. *Bioinformatics*, 27(6):860–862, 2011.

Zhan Gao, Alec Koppel, and Alejandro Ribeiro. Incremental greedy bfgs: An incremental quasi-newton method with explicit superlinear rate. 2020.

V. Gupta, S. Kadhe, T. Courtade, M. W. Mahoney, and K. Ramchandran. Oversketched Newton: Fast convex optimization for serverless systems. *arXiv preprint arXiv:1903.08857*, 2019.

N. Ivkin, D. Rothchild, E. Ullah, V. Braverman, I. Stoica, and R. Arora. Communication-efficient distributed SGD with Sketching. *arXiv preprint arXiv:1903.04488*, 2019.

Y. Juan, Y. Zhuang, W. Chin, and C. Lin. Field-aware factorization machines for CTR prediction. In *Proc.* 10^{th} *ACM Conf. Recommender Syst.*, pages 43–50. ACM, 2016.

D. Kane and J. Nelson. Sparser Johnson-Lindenstrauss transforms. *J. ACM*, 61(1):4, 2014.

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5(Apr):361–397, 2004.

A. Maleki and D. L. Donoho. Optimally tuned iterative reconstruction algorithms for compressed sensing. *IEEE J Selected Topics in Sig. Proces.*, 4(2):330–341, 2010.

A. Mokhtari and A. Ribeiro. Global convergence of online limited memory BFGS. *J. Mach. Learn. Res.*, 16(1):3151–3181, 2015.

J. Nocedal. Updating quasi-Newton matrices with limited storage. *Math. Comput.*, 35(151):773–782, 1980.

Thomas O'Leary-Roseberry, Nick Alger, and Omar Ghattas. Inexact newton methods for stochastic nonconvex optimization with applications to neural network training. *arXiv preprint arXiv:1905.06738*, 2019.

K. S. Tai, V. Sharan, P. Bailis, and G. Valiant. Sketching Linear Classifiers over Data Streams. In *Proc. of the 2018 Intl. Conf. on Management of Data*, pages 757–772. ACM, 2018.

Joel A Tropp. An introduction to matrix concentration inequalities. *arXiv preprint arXiv:1501.01571*, 2015.

K. Vervier, P. Mahé, M. Tournoud, J. Veyrieras, and J. Vert. Large-scale machine learning for metagenomics sequence classification. *Bioinformatics*, 32(7):1023–1032, 2016.

S. Webb, J. Caverlee, and C. Pu. Introducing the Webb Spam Corpus: Using Email Spam to Identify Web Spam Automatically. In *CEAS*, 2006.

K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proc. of the* 26^{th} *Annual Intl. Conf. on Machine Learning*, pages 1113–1120. ACM, 2009.

D. Woodruff et al. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

P. Xu, F. Roosta, and M. Mahoney. Second-order optimization for non-convex machine learning: An empirical study. In *Proceedings of the 2020 SIAM International Conference on Data Mining*, pages 199–207. SIAM, 2020.

Z. Yao, A. Gholami, K. Keutzer, and M. Mahoney. Pyhessian: Neural networks through the lens of the Hessian. *arXiv preprint arXiv:1912.07145*, 2019.

Zhewei Yao, Amir Gholami, Sheng Shen, Kurt Keutzer, and Michael W Mahoney. ADAHESSIAN: An adaptive second order optimizer for machine learning. *arXiv preprint arXiv:2006.00719*, 2020.

M. Yu, L. Jose, and R. Miao. Software Defined Traffic Measurement with OpenSketch. In *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, pages 29–42, 2013.