LSH-SAMPLING BREAKS THE COMPUTATIONAL CHICKEN-AND-EGG LOOP IN ADAPTIVE STOCHASTIC GRADIENT ESTIMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Stochastic Gradient Descent or SGD is the most popular algorithm for large-scale optimization. In SGD, the gradient is estimated by uniform sampling with sample size one. There have been several results that show better gradient estimates, using weighted non-uniform sampling, which leads to faster convergence. Unfortunately, the per-iteration cost of maintaining this adaptive distribution is costlier than the exact gradient computation itself, which creates a chicken-and-egg loop making the fast convergence useless. In this paper, we break this barrier by providing the first demonstration of a sampling scheme, which leads to superior gradient estimation, while keeping the sampling cost per iteration similar to the uniform sampling. Such a scheme is possible due to recent advances in Locality Sensitive Hashing (LSH) literature. As a consequence, we improve the running time of all existing gradient descent algorithms.

1 MOTIVATION

Stochastic gradient descent or popularly known as SGD is the most popular choice of optimization in large-scale setting for its computational efficiency. A typical interest in machine learning is to minimize the average loss function f over the training data, with respect to the parameters θ, i.e., the objective function of interest is

$$\theta^* = \arg\min_{\theta} F(\theta) = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} f(x_i, \theta) \tag{1}$$

Throughout the paper, our training data $D = \{x_i,\ y_i\}_{i=1}^N$ will have N instances with $x_i \in \mathbb{R}^d$ d dimensional features and y_i will be labels. The labels can be real valued for regression problems. For classification problem, they will take value in a discrete set, i.e., $y_i \in \{1,\ 2, \cdots,\ K\}$. Typically, the function f is a convex function. The popular ones include least squares $f(x_i, \theta) = (\theta \cdot x_i - y_i)^2$, used in regression setting.

During the optimization, SGD (Bottou, 2010) samples an instance x_j, uniformly at random from among the N instances, and performs the gradient descent update

$$\theta_t = \theta_{t-1} - \eta^t \nabla f(x_j, \theta_{t-1}) \tag{2}$$

where the gradient $\nabla f(x_j, \theta_{t-1})$ is only evaluated on x_j, using the current θ_{t-1}. Here, η^t is the step size at the tth iteration.

It should be noted that a full gradient of the objective is given by the average $\frac{1}{N}\sum_{i=1}^N \nabla f(x_i, \theta_{t-1})$. Thus, a uniformly sampled gradient $\nabla f(x_j, \theta_{t-1})$ is an unbiased estimator of the full gradient, i.e.,

$$\mathbb{E}(\nabla f(x_j, \theta_{t-1})) = \frac{1}{N} \sum_{i=1}^{N} \nabla f(x_i, \theta_{t-1}). \tag{3}$$

This is the key reason why despite only using one sample, SGD still converges to the local minima, analogous to full gradient descent, provided η^t is chosen properly (Robbins & Monro, 1951; Bottou, 2010).

However, it is known that the convergence rate of SGD is slower than that of the full gradient descent (Shamir & Zhang, 2013). Nevertheless, the cost of computing the full gradient requires $O(N)$ evaluations of ∇f compared to just constant evaluation in SGD. Thus, with the cost of one epoch of full gradient descent, SGD can perform $O(N)$ epochs, which overcompensates the slow convergence. Therefore, despite fast convergence rates, SGD is almost always the algorithm for choice in large-scale settings as the full calculation of gradient in every epoch is prohibitively slow. Further improving the speed of SGD is still an active area of exploration. Any improvement in the speed of SGD will directly speed up almost all the state-of-the-art in machine learning.

As expected, the slower convergence of SGD is due to the poor estimation of the gradient (the average) by only sampling a single instance uniformly. Clearly, the variance of the one sample estimator is high. As a result, there have been several efforts in finding better sampling strategies for better estimation of the gradients (Zhao & Zhang, 2014; Needell et al., 2014; Zhao & Zhang, 2015; Alain et al., 2015). The key idea behind these methods is to replace the uniform distribution with a weighted distribution with better variance. The weighted distribution has to change (adaptive) in every iteration, when the parameters and the gradients change. Unfortunately, as argued in (Gopal, 2016), all of these methods suffer from what we call the chicken-and-egg loop – maintaining any adaptive non-uniform distribution itself cost $O(N)$ per iteration. It is also the cost of computing the gradient exactly.

To the best of our knowledge, there does not exist any generic sampling scheme for adaptive gradient estimation, where the cost of maintaining and updating the distribution, per update, is constant O(1). It is also comparable in update time with stochastic gradient sampling. Our work provides first such sampling scheme utilizing the recent advances in sampling and unbiased estimation using Locality Sensitive Hashing (Spring & Shrivastava, 2017).

1.1 Adaptive Sampling for SGD

For non-uniform sampling, we can sample each x_i with an associated weight w_i. These w_i's can be tuned to minimize the variance. It was first shown in (Alain et al., 2015), that sampling x_i in proportion to the L_2 norm of the gradient, i.e. $||\nabla f(x_i, \theta_{t-1})||_2$, is the optimal distribution that minimizes the variance. However, sampling x_i in proportion to $w_i = ||\nabla f(x_i, \theta_{t-1})||_2$, requires first computing all the N, w_is, which change with every update because the θ_{t-1} get updated. Thus, with every epoch just to maintain the values of w_i's is costlier than computing the full gradient. (Gopal, 2016) proposed to mitigate this overhead partially by exploiting additional side information such as the cluster structure of the data. Prior to the realization of optimal variance distribution, (Zhao & Zhang, 2014) and (Needell et al., 2014) proposed to sample a training instance with a probability proportional to the Lipschitz constant of the function $f(x_i, \theta_{t-1})$ and $\nabla f f(x_i, \theta_{t-1})$ respectively. Again, as argued, in (Gopal, 2016), the cost of maintaining the distribution is prohibitive.

It is worth mentioning that before these works, a very similar idea was used in designing importance sampling-based low-rank matrix approximation algorithms. The resulting sampling methods, popularly known as leverage score sampling, are again proportional to the squared Euclidean norms of rows and columns of the underlying matrix. See (Drineas et al., 2012).

The Chicken-and-Egg Loop: In summary, to speed up the convergence of stochastic gradient descent, we need non-uniform sampling for better estimates (low variance) of the full gradient. Any interesting non-uniform sampling is dependent on the data and the parameter θ_t which changes in every iteration. Thus, maintaining the non-uniform distribution requires $O(N)$ computations to merely compute the weights w_i, which is the same cost of computing the full gradient. It is not even clear if there is any sweet and adaptive distribution which breaks this computational chicken-and-egg loop. We provide the first affirmative answer by giving an unusual distribution which is derived from probabilistic indexing based on locality sensitive hashing.

Our Contributions: In this work, we propose a novel LSH-based samplers, that breaks the aforementioned chicken-and-egg loop. Our algorithm, which we call LSD (**L**SH Sampled **S**tochastic gradient **D**escent), are generated via hash lookups which cost less than one dot product and the probability of selection x_i is provably adaptive. As a result, the current gradient estimates have lower variance, compared to a single sample SGD, while the computational complexity of sampling is constant and of the order of SGD sampling.

As a direct consequence, we obtain a generic and faster gradient descent algorithm which converges significantly faster than SGD, both concerning epochs as well as running time. It should be noted that simply epoch wise rapid convergence does not imply computational efficiency. Newtons method converges faster, epoch wise, than any first-order gradient descent, but it is prohibitively slow in practice. The wall clock time of convergence or the amount of floating point operations to converge should be the metric of consideration for conclusive comparisons.

Accuracy Vs Running Time: It is rare to see any fair (same computational setting) empirical comparison of SGD with any adaptive SGD scheme, which compares the growth in accuracy with running time on the same computational platform. Almost all methods compare accuracy with the number of epochs, which is unfair to SGD which can complete $O(N)$ epochs at the computational cost (or running time) of 1 epoch for adaptive sampling schemes.

2 BACKGROUND

We first describe a recent advancement in the theory of sampling and estimation using locality sensitive hashing (LSH) (Indyk & Motwani, 1998) which will be heavily used in our proposal. Before we go in sampling, let us revise the two-decade-old theory of LSH.

2.1 LOCALITY SENSITIVE HASHING (LSH)

Locality-Sensitive Hashing (LSH) (Indyk & Motwani, 1998) is a popular, sub-linear time algorithm for approximate nearest-neighbor search. The high-level idea is to place similar items into the same bucket of a hash table with high probability. An LSH hash function maps an input data vector to an integer key

$$h(x) : \mathbb{R}^D \mapsto [0, 1, 2, \ldots, N].$$

A collision occurs when the hash values for two data vectors are equal - $h(x) = h(y)$. The collision probability of most LSH hash functions is generally a monotonic function of the similarity -

$$Pr[h(x) = h(y)] = \mathcal{M}(sim(x, y))$$

where $\mathcal{M}$ is a monotonically increasing function. Essentially, similar items are more likely to collide with each other under the same hash fingerprint.

The algorithm uses two parameters - (K, L). We construct L independent hash tables from the collection $\mathcal{C}$. Each hash table has a meta-hash function H that is formed by concatenating K random independent hash functions from $\mathcal{F}$. Given a query, we collect one bucket from each hash table and return the union of L buckets. Intuitively, the meta-hash function makes the buckets sparse (less crowded) and reduces the number of false positives because only valid nearest-neighbor items are likely to match all K hash values for a given query. The union of the L buckets decreases the number of false negatives by increasing the number of potential buckets that could hold valid nearest-neighbor items.

The candidate generation algorithm works in two phases [See (Spring & Shrivastava, 2017) for details]:

1. **Pre-processing Phase:** We construct L hash tables from the data by storing all elements $x \in \mathcal{C}$. We only store pointers to the vector in the hash tables because storing whole data vectors is very memory inefficient.
2. **Query Phase:** Given a query Q; we will search for its nearest-neighbors. We report the union from all of the buckets collected from the L hash tables. Note, we do not scan all the elements in $\mathcal{C}$, we only probe L different buckets, one bucket for each hash table.

After generating the set of potential candidates, the nearest-neighbor is computed by comparing the distance between each item in the candidate set and the query.

2.2 LSH FOR ESTIMATIONS AND SAMPLING

An item returned as candidate from a (K, L)-parameterized LSH algorithm (section 3.2) is sampled with probability $1 - (1 - p^K)^L$ where p is the collision probability of LSH function. The LSH

family defines the precise form of p used to build the hash tables. This sampling view of LSH was first utilized to perform adaptive sparsification of deep networks in near-constant time, leading to efficient backpropagation algorithm (Spring & Shrivastava, 2016)

A year later, (Spring & Shrivastava, 2017) demonstrated the first theory of using these samples for unbiased estimation of partition functions in log-linear models. More specifically, the authors show that since we know the precise probability of sampled elements $1-(1-p^K)^L$, we can design provably unbiased estimators using importance sampling type idea. This was the first demonstration that random sampling can be beaten with roughly the same computational cost as vanilla sampling. (Luo & Shrivastava, 2017) uses the same approaches for unbiased estimation of anomaly scoring function. (Charikar & Siminelakis) rigorously formalized these notions and show provable improvements in sample complexity of kernel density estimation problems. Recently (Chen et al., 2017) used the sampling in a very different context of connected component estimation for unique entity counts.

2.2.1 MIPS SAMPLING

Recent advances in maximum inner product search (MIPS) using asymmetric locality sensitive hashing has made it possible to sample large inner products.

For this paper, it is sufficient to assume that given a collection $\mathcal{C}$ of vectors and query vector Q, using (K, L)-parameterized LSH algorithm with MIPS hashing (Shrivastava & Li, 2014), we get a candidate set S, such that every element $x_i \in \mathcal{C}$ is sample with probability $p_i \leq 1$ where p_i is a monotonically increasing function of $Q \cdot x_i$. Thus, we can pay a one-time linear cost of preprocessing $\mathcal{C}$ into hash tables, and any further adaptive sampling for query Q only requires few hash lookups and guarantees that returned candidates are likely to have higher $Q \cdot x_i$. We can also compute the exact probability of getting x.

We would like to touch upon some technical details about this sampling. It is not a distribution, i.e., $sum_{x_i \in \mathcal{C}} \neq 1$. Also the probability of getting x_i and x_j is not independent. But the same can be used for unbiased estimation. Reader interested in details of such sampling can refer to (Spring & Shrivastava, 2017). In particular, the form of sampling probability is quite unusual, $p_i = (1-(1-g(q \cdot x_i))^K)^L$, where $g(q \cdot x_i)$ is the collision probability. Overall, it can shown that p_i a monotonic function of $q \cdot x_i$.

3 THE LSD ALGORITHM

3.1 A GENERIC FRAMEWORK FOR EFFICIENT GRADIENT ESTIMATION

Our algorithm leverages the efficient estimations using locality sensitive hashing, which usually beats random sampling estimators while keeping the sampling cost near-constant. We first provide the intuition of our proposal, and the analysis will follow. Consider least square regression with loss function $\frac{1}{N}\sum_{i=1}^{N}(y_i - \theta_t \cdot x_i)^2$, where θ_t is the parameter in the tth iteration. The gradient is just like a partition function, and if we simply follow the procedures in (Spring & Shrivastava, 2017), we can easily show a generic unbiased estimator via adaptive sampling. However, we can be little smarter and get better sampling procedures.

Observe that, the gradient, with respect to θ_t concerning x_i is given by $2(y_i - \theta_t \cdot x_i)x_i$, the L_2 norm of the gradient, which is also the optimal sampling weight w_i^* for x_i according to (Alain et al., 2015), can be written as an absolute value of inner product.

$$||\nabla f(x_i, \theta_t)||_2 = \big|2(\theta_t \cdot x_i - y_i)||x_i||_2\big| = 2\big| < \theta_t, -1 > \cdot < x_i||x_i||, y_i||x_i|| > \big|, \tag{4}$$

where $< \theta_t, -1 >$ is a vector concatenation of θ with -1. If the data is normalized then we should sample x_i in proportion to $w_i* = \big| < \theta_t, -1 > \cdot < x_i, y_i > \big|$, i.e. large magnitude inner products should be sampled with higher probability.

As argued, w_i* changes with θ_t and hence this sampling is costly. It turns out, that we can design a sampling process which does not exactly sample with probability w_i^* but instead samples from a different weighted distribution which is a monotonic function of w_i*. In particular, we sample from $w_i^{lsh} = f(w_i^*)$, where f is some monotonic function. Before we describe the efficient sampling process, we argue that a monotonic sampling is a good choice.

For any monotonic function f, the weighted distribution $w_i^{lsh} = f(w_i*)$ is still adaptive and changes with θ_t. Also, due to monotonicity, if the optimal sampling prefers x_i over x_j i.e. $w_i^* \geq w_j*$, then monotonic sampling will also have same preference, i.e., $w_i^{lsh} \geq w_j^{lsh}$.

The key insight is that there are two quantities in the inner product (equation 4), are $< \theta_t, -1 >$ and $< x_i, y_i >$. With successive iteration $< \theta_t, -1 >$ changes while $< x_i, y_i >$ is fixed. Thus, it is possible to preprocess $< x_i, y_i >$ into hash tables (one time cost) and query with $< \theta_t, -1 >$ for efficient and adaptive sampling. With every iteration, only the query changes to $< \theta_{t+1}, -1 >$ but the hash tables remains the same. Hence, few hash lookups, are sufficient to sample x_i for gradient estimation adaptively. Thus, we only pay one-time preprocessing cost of building hash tables, and further every iteration requires hash lookups, typically just one lookup (Section 2), to get a sample for estimation.

There are few more technical subtleties due to the absolute value of inner product $\left| < \theta_t, -1 > \cdot < x_i, y_i > \right|$, rather than the inner product itself. However, the square of

$$\left| < \theta_t, -1 > \cdot < x_i, y_i > \right|^2 = T(< \theta_t, -1 >) \cdot T(< x_i, y_i >)$$

can also be written as an inner product as it is a quadratic kernel, and T is the corresponding feature expansion transformation. Again square is monotonic function, and therefore, our sampling is still monotonic as composition of monotonic functions is monotonic. Thus, technically we hash $T(< x_i, y_i >)$ to create hash tables and the query at t^{th} step is $T(< \theta_t, -1 >)$.

Once an x_i is sampled via LSH sampling (Algorithm 2), we can precisely compute the probability of its sampling, i.e., p_i (See section 2). Therefore our estimation of full gradient is unbiased.

3.2 Algorithmic Details

We first describe the detailed step of our gradient estimator in Algorithm 1. We also give the sampling algorithm 2 in detail. We assume that we have access to the right LSH function h, and its collision probability expression $cp(x, y) = Pr(h(x) = h(y))$. In case of linear regression, we can use signed random projections or simhash (Charikar, 2002), instead of MIPS hashing, as with normalized data simhash collision probability is $cp(x, y) = 1 - \frac{cos^{-1}(\frac{x \cdot y}{||x||_2 ||y||_2})}{\pi}$ which is monotonic in the inner product.

Algorithm 1 LSH-Sampled Stochastic gradient Descent (LSD) Algorithm

Input: $D = x_i,\ y_i$**,** N**,** θ_0**,** η
Input: LSH Family H**, parameters** K**,** L
Output: θ^*
$HT =$ Preprocess all data vectors $< x_i,\ y_i >$ into LSH Data structure (Section 2).
$t = 0$
while $NotConverged$ **do**
    $x,\ p = Sample(H, \text{HT}, K, < \theta_t, -1 >)$ (Algorithm 2)
    $\theta_{t+1} := \theta_t - \eta_t(\frac{\nabla f(x, \theta_t)}{p \times N})$
end while
return θ^*

3.2.1 Running Time of Sampling

The computational cost of SGD sampling is merely a single random number generator. The cost of gradient update (equation 2) is one inner product, which is d multiplications. If our adaptive sampling beats SGD, the sampling cost cannot be much larger than d multiplications.

The cost of LSD sampling (Algorithm 2) is $K \times l$ hash computations followed by $l + 1$ random number generator, (1 extra for sampling from the bucket). However, the scheme works for any K. Thus, we can always choose K small enough so that empty buckets are rare (see (Spring & Shrivastava, 2017)). In all our experiments, $K = 5$ for which l is almost always 1. Thus, we require K hash computations and only 2 random number generation. If we use very sparse random projections, then K hash computations only require a constant $\ll d$ multiplications. For example,

Algorithm 2 Sample

Input: H **(Hash functions),** $HT[][]$ **(**L **Hash Tables), K,** $Query$
$cp(x, Q)$ **is the collision probability Pr(h(x)= h(Q)), under given LSH (known)**
Output: sampled data x**, probability of sampling** p
$l,\ S = 0$
while true **do**
    $ti = random(1, L)$
    $bucket = H(Query, ti)$ (table specific hash)
    if HT[ti][bucket] = empty **then**
        l++
        continue;
    end if
    $S = |HT[ti][bucket]|$ (size of bucket)
    x = randomly pick one element from $HT[ti][bucket]$
    break;
end while
$p = (1 - (1 - cp(x, Query)^K)^l) \times \frac{1}{S}$
return $x,\ p$

in all our experiments we only need d/6 multiplication, in expectation, to get all the hashes using sparse projections. Thus, our sampling cost is significantly less than d multiplication which is the cost of gradient update. Using fast hash computation is critical for our method to work in practice.

3.2.2 Near-Neighbor is Costlier than LSH-Sampling

It might be tempting to use approximate near-neighbor search with query θ_t to find x_i. However, near-neighbor queries are costly due to candidate generation and filtering. It is still sub-linear in N (and not constant). Moreover, the sampling probability of x cannot be calculated for near-neighbor search which will cause bias in the gradient estimate.

It is important to note that although LSH is heavily used for near-neighbor search, in this case, we just use them to sample. For efficient near neighbor search, K and L grows with N (Indyk & Motwani, 1998). In contrast, the sampling works for any K and L as small as 1. Efficient unbiased estimation is the key difference that makes sampling practical while near-neighbor query prohibitive.

3.3 Variance Analysis

In this section, we first prove that our estimator of gradient is unbiased with lower variance than SGD for most real datasets. Call S the bucket that provided the sample x from in Algorithm 2. For simplicity denote the query with θ_t. Denote $p_i = 1 - (1 - cp(x_i, \theta_t)^K)^l$ is the probability of finding x_i in bucket S.

Theorem 1. *The following expression is an unbiased estimator of the full gradient*

$$Est = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{x_i \in S}\, \mathbb{1}_{(x_i = x_m | x_i \in S)} \frac{\nabla f(x_i, \theta_t) \cdot |S|}{p_i} \tag{5}$$

$$\mathbb{E}[Est] = \frac{1}{N}\sum_{i=1}^{N} \nabla f(x_i, \theta_t) \tag{6}$$

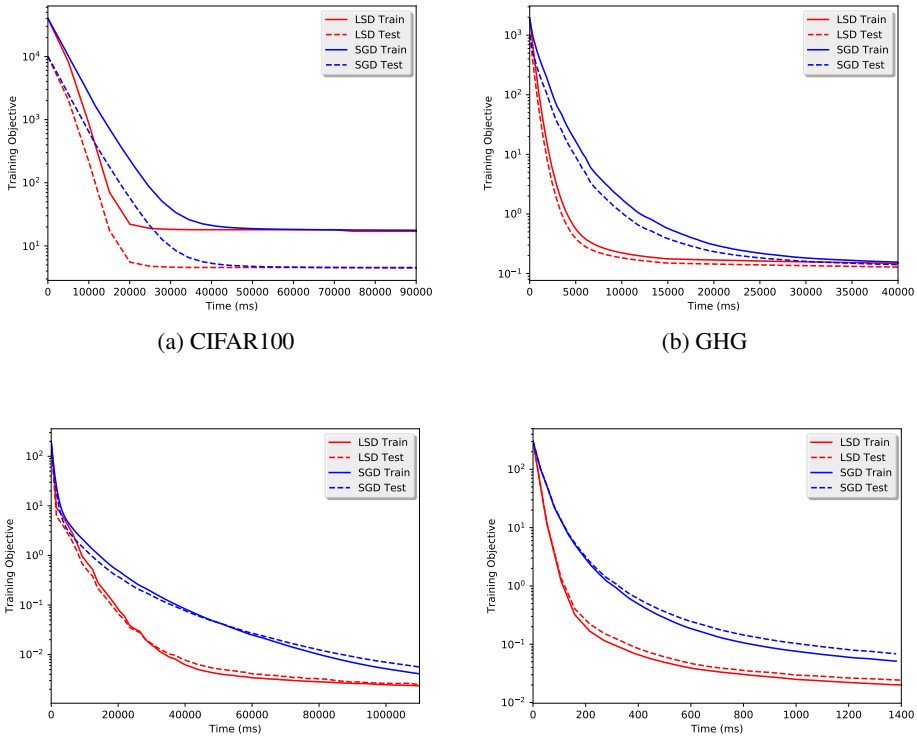
Theorem 2. *The variance of our estimator is:*

$$\mathbb{V}[Est] = \frac{1}{N^2}\sum_{i=1}^{N} \frac{\nabla f(x_i, \theta_t)^2 \cdot |S|}{p_i} - \frac{1}{N^2}(\sum_{i=1}^{N} \nabla f(x_i, \theta_t))^2 \tag{7}$$

Theorem 3. *Variance of LSD's estimator is smaller than variance of SGD's estimator if*

$$\frac{1}{N}\sum_{i=1}^{N} \frac{\nabla f(x_i, \theta_t)^2 \cdot |S|}{p_i} < (\sum_{i=1}^{N} \nabla f(x_i, \theta_t))^2 \tag{8}$$

Recall that the collision probability $p_i = 1 - (1 - p^K)^l$ mentioned in Section 2.2. Noted that l here according to Algorithm 2 is the number of tables that have being looked up. In most practical cases and also in our experiment, K and L are relative small. LSD can achieve a much smaller variance than SGD by setting small values of K and L. It is not difficult to see that if several terms in the summation satisfy $\frac{|S|}{p_i N} \leq 1$, then the variance of our estimator is better than random sampling. If the data is clustered nicely, i.e. a random pair has low similarity, then we will have small $|S|$ and p_i can be controlled by tuning K. See Spring & Shrivastava (2017) for more details on when LSH sampling is better than random sampling.



(a) CIFAR100

(b) GHG

(c) ElectricityLoad

(d) DrivFace

Figure 1: Wall clock convergence comparisons of plain LSD and SGD.

4 EXPERIMENTS

We evaluate the effectiveness of our algorithm on four datasets covering a wide range of applications from image to time series data;

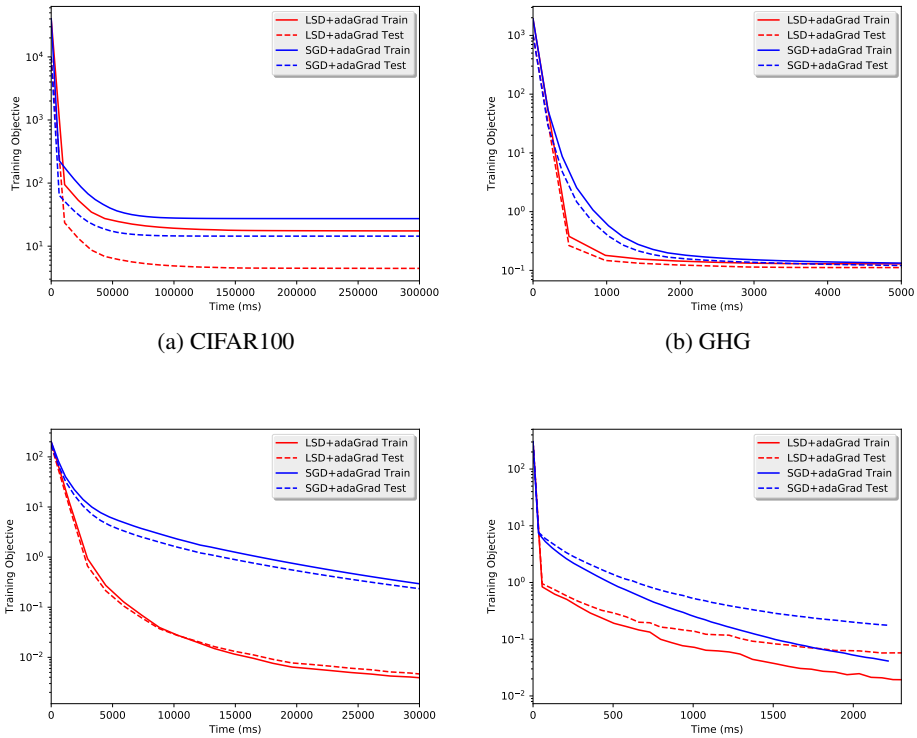
CIFAR100: (Krizhevsky, 2009) An image database with 100 classes containing 600 images each. 50,000 of them were used for training. We used applied standard RGB channel preprocessing to the 32x32 images, leading to 3,072 features for each instance.

ElectricityLoad: (Lichman, 2013) A set of 370 records of electricity load of different clients. The dimension is 140,256. There are 296 training examples and 74 testing examples.

DrivFace: (Diaz-Chito et al., 2016)s A dataset containing 606 instances of images of 4 drivers with several facial features, each with 6,400 attributes. We used 303 instances for training and 303 for testing.

GHG : (Lucas et al., 2015) A collection of 2921 time series of greenhouse gas concentrations in California. 1900 of them were used for training. There are 5232 features in total.

All datasets were preprocessed to have zero mean and unit variance for each feature column. Furthermore, each row was normalized using $L2$ norm. Noted that for all the experiments, the choice of the gradient decent algorithm is the same. For both SGD and LSD, the only difference in the gradient algorithm was the gradient estimator. For SGD a random sampling estimator was used, while for LSD the estimator used the adaptive estimator. We used fixed values $K = 5$ and $L = 3$



(a) CIFAR100

(b) GHG

(c) ElectricityLoad

(d) DrivFace

Figure 2: Wall clock convergence comparisons of adagrad with LSD and SGD gradient estimation.

for all the datasets. Our hash function is simhash (or signed random projections) and we use sparse random projection with sparsity 1/30.

To the best of our knowledge, there is no other adaptive estimation baseline, where the cost of sampling per iteration is less than linear $O(N)$. Since our primary focus will be on wall clock speed up, no $O(N)$ estimation method will be able to outperform O(1) SGD (and LSD) estimates on the same platform.

LSD vs. SGD In the first experiment, we compare vanilla SGD with LSD, i.e., we use simple SGD with fixed learning rate. This basic experiment aims to demonstrate the performance of pure LSD and SGD without involving other factors like L1/L2 regularizations on linear regression task. In such a way, we can quantify the superiority of LSD.

Figure 4 (in Appendix) shows the decrease in the squared loss error with epochs. Blue lines represent SGD and red lines represent LSD. It is obvious that LSD converges much faster than SGD in either training or testing loss comparisons. This is not surprising with the claims in Section 3.2.1 and theoretical proof in Section 3.3. Since LSD uses slightly more computations per epoch than SGD updates, it is hard to defend if LSD gains enough benefits simply from the epoch wise comparisons. We therefore also show the decrease in error with wall clock time. Wall clock time is the actual quantification of speedups. Again, on every single dataset, LSD shows faster convergence both in epochs as well as running time.

LSD+AdaGrad vs. SGD+AdaGrad As argued, our LSD algorithm is not an alternative but a complimentary to another gradient-based optimization algorithm. We repeated the first experiment one but using AdaGrad (Duchi et al., 2011) instead of plain SGD. Again, the only change in the competing algorithm is the gradient estimates per epoch. Figure 5 (in Appendix) shows epoch wise comparison, while Figure 2 shows running time comparisons. The trends as expected are similar. LSD with adagrad outperforms adagrad with SGD estimates of gradients.

LSD, SGD vs. True Gradient: In this section, as a sanity check, we compare the quality of estimates with SGD and LSD. We chose estimation DrivFace dataset. During an intermediate iteration, we compute the true gradient. We then estimate the gradient using m samples of SGD and LSD. We plot the estimation error with m in Figure 3 (in Appendix). It is not surprising to see LSD has better estimates of the true gradient than SGD most of the time confirming our theoretical results empirically.

REFERENCES

Guillaume Alain, Alex Lamb, Chinnadhurai Sankar, Aaron Courville, and Yoshua Bengio. Variance reduction in sgd by distributed importance sampling. *arXiv preprint arXiv:1511.06481*, 2015.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer, 2010.

Moses Charikar and Paris Siminelakis. Hashing-based-estimators for kernel density in high dimensions.

Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 380–388. ACM, 2002.

Beidi Chen, Anshumali Shrivastava, and Rebecca C Steorts. Unique entity estimation with application to the syrian conflict. *arXiv preprint arXiv:1710.02690*, 2017.

Katerine Diaz-Chito, Aura Hernández-Sabaté, and Antonio M. López. A reduced feature set for driver head pose estimation. *Appl. Soft Comput.*, 45(C):98–107, August 2016. ISSN 1568-4946. doi: 10.1016/j.asoc.2016.04.027. URL http://dx.doi.org/10.1016/j.asoc.2016.04.027.

Petros Drineas, Malik Magdon-Ismail, Michael W Mahoney, and David P Woodruff. Fast approximation of matrix coherence and statistical leverage. *Journal of Machine Learning Research*, 13 (Dec):3475–3506, 2012.

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

Siddharth Gopal. Adaptive sampling for sgd by exploiting side information. In *International Conference on Machine Learning*, pp. 364–372, 2016.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613. ACM, 1998.

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

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

D. D. Lucas, C. Yver Kwok, P. Cameron-Smith, H. Graven, D. Bergmann, T. P. Guilderson, R. Weiss, and R. Keeling. Designing optimal greenhouse gas observing networks that consider performance and cost. *Geoscientific Instrumentation, Methods and Data Systems*, 4(1):121–137, 2015. doi: 10.5194/gi-4-121-2015. URL https://www.geosci-instrum-method-data-syst.net/4/121/2015/.

Chen Luo and Anshumali Shrivastava. Arrays of (locality-sensitive) count estimators (ace): High-speed anomaly detection via cache lookups. *arXiv preprint arXiv:1706.06664*, 2017.

Deanna Needell, Rachel Ward, and Nati Srebro. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in Neural Information Processing Systems*, pp. 1017–1025, 2014.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Ohad Shamir and Tong Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *International Conference on Machine Learning*, pp. 71–79, 2013.

Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, pp. 2321–2329, 2014.

Ryan Spring and Anshumali Shrivastava. Scalable and sustainable deep learning via randomized hashing. *arXiv preprint arXiv:1602.08194*, 2016.

Ryan Spring and Anshumali Shrivastava. A new unbiased and efficient class of lsh-based samplers and estimators for partition function computation in log-linear models. *arXiv preprint arXiv:1703.05160*, 2017.

Peilin Zhao and Tong Zhang. Accelerating minibatch stochastic gradient descent using stratified sampling. *arXiv preprint arXiv:1405.3080*, 2014.

Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1–9, 2015.

A EPOCH PLOTS AND PROOFS

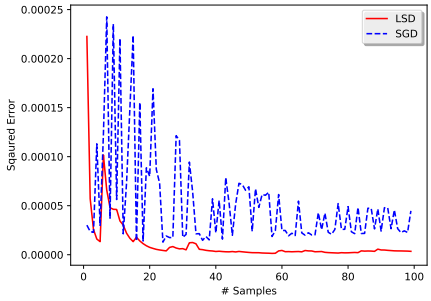
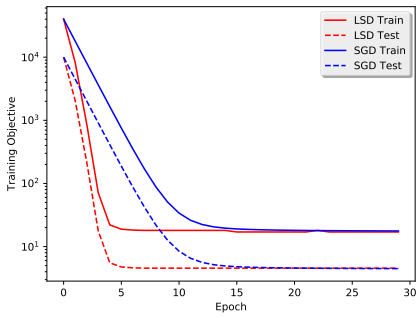
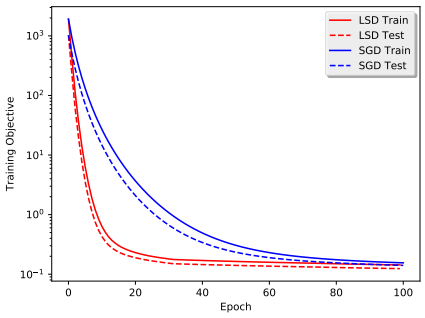


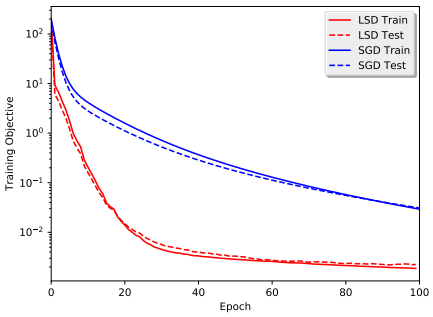
Figure 3: Error in estimating full gradient for SGD and LSD with sample size m



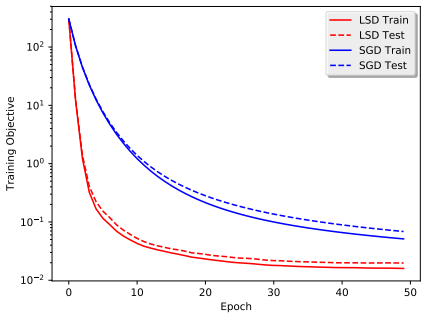
(a) CIFAR100



(b) GHG



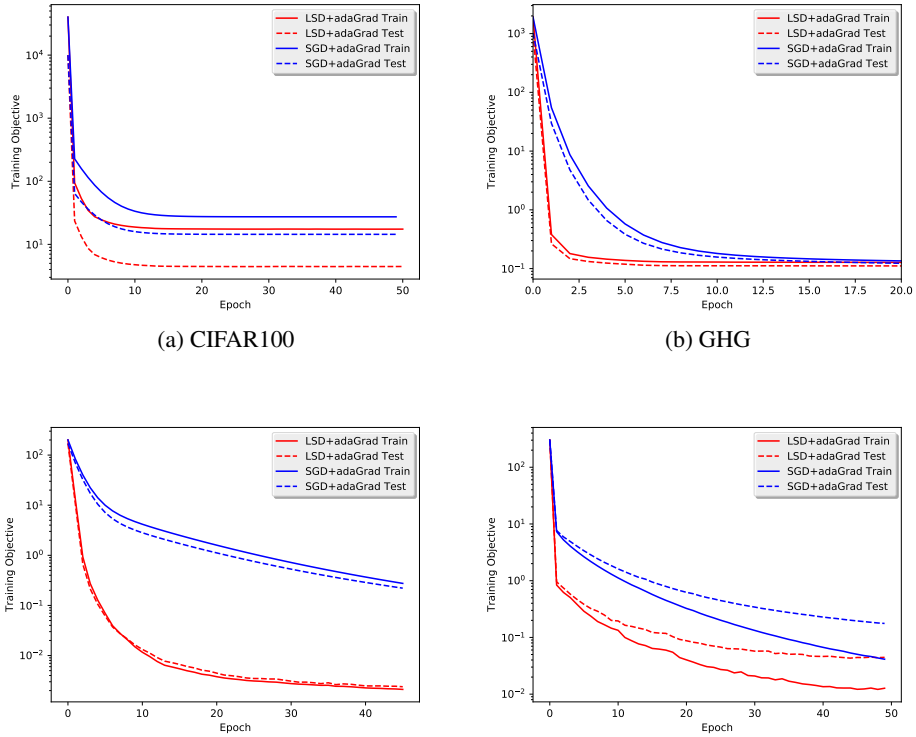
(c) ElectricityLoad



(d) DrivFace

Figure 4: shows epoch wise the comparisons of training and testing loss of LSD and SGD for all four datasets

Theorem 4. *Let S be the bucket that sample x_m is chosen from in Algorithm 2. Let p_m be the sampling probability associated with sample x_m. Suppose we query a sample with θ_t. Then we have*



(a) CIFAR100 (b) GHG

(c) ElectricityLoad (d) DrivFace

Figure 5: Epoch wise convergence comparisons of training and testing loss of LSD+adaGrad and SGD+adaGrad.

an unbiased estimator of the full gradient:

$$\mathit{Est} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{x_i \in S}\, \mathbb{1}_{(x_i = x_m | x_i \in S)} \frac{\nabla f(x_i, \theta_t) \cdot |S|}{p_i}$$

$$\mathbb{E}[\mathit{Est}] = \frac{1}{N}\sum_{i=1}^{N} \nabla f(x_i, \theta_t)$$

Proof.

$$\mathbb{E}[\mathbb{1}_{x_i \in S}] = p_i \quad \textit{and} \quad \mathbb{E}[\mathbb{1}_{x_i = x_m | x_i \in S}] = \frac{1}{|S|}$$

Also note that $\mathbb{E}[\mathbb{1}_{x_i \in S}\, \mathbb{1}_{x_i = x_m | x_i \in S}] = \mathbb{E}[\mathbb{1}_{x_i \in S}]\mathbb{E}[\mathbb{1}_{x_i = x_m | x_i \in S}]$

$$
\begin{aligned}
\mathbb{E}[\text{Est}] &= \frac{1}{N}\mathbb{E}[\sum_{i=1}^{N} \mathbb{1}_{x_i \in S}\mathbb{1}_{x_i = x_m | x_i \in S}\frac{\nabla f(x_i,\theta_t)\cdot |S|}{p_i}] \\
&= \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}[\mathbb{1}_{x_i \in S}\mathbb{1}_{x_i = x_m | x_i \in S}]\cdot \mathbb{E}[\frac{\nabla f(x_i,\theta_t)\cdot |S|}{p_i}] \\
&= \frac{1}{N}\sum_{i=1}^{N} p_i \cdot \frac{1}{|S|}\frac{\nabla f(x_i,\theta_t)\cdot |S|}{p_i} \\
&= \frac{1}{N}\sum_{i=1}^{N}\nabla f(x_i,\theta_t)
\end{aligned}
$$

□

Theorem 5. *The variance of our estimator is:*

$$
\mathbb{V}[\textit{Est}] = \frac{1}{N^2}\sum_{i=1}^{N}\frac{\nabla f(x_i,\theta_t)^2\cdot |S|}{p_i} - \frac{1}{N^2}(\sum_{i=1}^{N}\nabla f(x_i,\theta_t))^2
$$

Proof.

$$
\mathbb{V}[\text{Est}] = \mathbb{E}[Est^2] - \mathbb{E}[\text{Est}]^2
$$

$$
\begin{aligned}
\text{Est}^2 &= \frac{1}{N^2}\sum_{i,j}^{N}\mathbb{1}_{x_i \in S}\mathbb{1}_{x_j \in S}\mathbb{1}_{x_i = x_m | x_i \in S}\mathbb{1}_{x_j = x_m | x_j \in S}\frac{\nabla f(x_i,\theta_t)\cdot \nabla f(x_j,\theta_t)\cdot |S|^2}{p_i \cdot p_j} \\
&= \frac{1}{N^2}\sum_{i}^{N}\mathbb{1}_{x_i \in S}\cdot \mathbb{1}_{x_i = x_m | x_i \in S}\frac{(\nabla f(x_i,\theta_t))^2\cdot |S|^2}{p_i^2} \\
\mathbb{E}[\text{Est}^2] &= \frac{1}{N^2}\sum_{i}^{N}\frac{(\nabla f(x_i,\theta_t))^2\cdot |S|}{p_i} \\
\mathbb{V}[\text{Est}] &= \frac{1}{N^2}\sum_{i=1}^{N}\frac{\nabla f(x_i,\theta_t)^2\cdot |S|}{p_i} - \frac{1}{N^2}(\sum_{i=1}^{N}\nabla f(x_i,\theta_t))^2
\end{aligned}
$$

□

Theorem 6. *Variance of LSD's estimator is smaller than variance of SGD's estimator if*

$$
\frac{1}{N}\sum_{i=1}^{N}\frac{\nabla f(x_i,\theta_t)^2\cdot |S|}{p_i} < \sum_{i=1}^{N}\nabla f(x_i,\theta_t)^2
$$

Proof. The variance of regular SGD is

$$
\mathbb{V}[\text{Est'}] = \frac{1}{N}\sum_{i}^{N}\nabla f(x_i,\theta_t)^2 - \frac{1}{N^2}(\sum_{i=1}^{N}\nabla f(x_i,\theta_t))^2
$$

By (11) and (17), one can easily see that $\mathbb{V}[\text{Est'}] < \mathbb{V}[\text{Est}]$ when (16) satisfies. □