Differentially Private Stochastic Gradient Descent for in-RDBMS Analytics

Xi Wu[†], Arun Kumar[†], Kamalika Chaudhuri[‡], Somesh Jha[†], Jeffrey F. Naughton[§]
[†]UW-Madison, [‡]UCSD, [§]Google
{xiwu, arun, jha}@cs.wisc.edu, kamalika@cs.ucsd.edu, naughton@google.com

June 16, 2016

Abstract

In-RDBMS data analysis has received considerable attention in the past decade and has been widely used in sensitive domains to extract patterns in data using machine learning. For these domains, there has also been growing concern about privacy, and differential privacy has emerged as the gold standard for private data analysis. However, while differentially private machine learning and in-RDBMS data analytics have been studied separately, little previous work has explored private learning in an in-RDBMS system.

This work considers a specific algorithm – stochastic gradient descent (SGD) for differentially private machine learning – and explores how to integrate it into an RDBMS system. We find that previous solutions on differentially private SGD have characteristics that render them unattractive for an RDBMS implementation. To address this, we propose an algorithm that runs SGD for a constant number of passes and then adds noise to the resulting output. We provide a novel analysis of the privacy properties of this algorithm. We then integrate it, along with two existing versions of differentially private SGD, in the Postgres RDBMS. Experimental results demonstrate that our proposed approach can be easily integrated into an RDBMS, incurs virtually no overhead, and yields substantially better test accuracy than the other private SGD algorithm implementations.

1 Introduction

The past decade has seen significant interest in integrating complex data analytics into RDBMS systems (for example, MADlib [12], Bismarck [9]) for large-scale enterprise applications. A core type of data analytics supported by these systems is *machine learning*, which is widely used to extract valuable patterns from data in domains such as personalized medicine, finance, web search histories and social networks. For these domains, there has also been an ever growing concern about individuals' privacy. To this end, *differential privacy*, a cryptographically motivated privacy notion, has emerged as the gold standard for protecting data privacy. *Differentially private learning* has been intensively studied in recent years by researchers from the database, machine learning and theoretical computer science communities [2, 4, 6, 14, 16, 26, 27].

Interestingly, while in-RDBMS machine learning and differentially private learning have been separately studied, to the best of our knowledge little previous work has examined private learning in an RDBMS system. We observe that an ideal solution would satisfy three properties: (1) *High accuracy*. The algorithm must learn private models of accuracy competitive with the non-private models. (2) *Low overhead*. Ideally, the private learning algorithm should only incur a small runtime overhead compared to non-private algorithms. (3) *Ease of integration*. The private algorithm must fit naturally into an RDBMS code base and should not require a significant modification to the RDBMS.

This work considers a *specific* algorithm – *stochastic gradient descent (SGD)* for differentially private machine learning in an in-RDBMS system. There are two main reasons to specifically consider SGD. First, it is a fundamental machine learning algorithm that lies at the heart of many in-RDBMS data analytics

systems, as it is simple to implement, easily parallelizable [29] and is known to have strong robustness properties. For example, a key feature of Bismarck [9] is a single framework to implement all convex analysis techniques available in RDBMSes based on a highly efficient implementation of SGD using User Defined Aggregates (UDA), a widely-available RDBMS abstraction. As a result, integrating a private version of SGD is relatively simple, and it automatically provides private in-RDBMS versions of all these convex analysis techniques.

The second reason to consider SGD has to do with a core problem for both in-RDBMS learning and private learning: *empirical risk minimization*. A long line of work, that includes the algorithms in [4, 16, 22], and a subset of algorithms in [2, 13, 14], has studied how to privately solve this problem assuming that the *exact* empirical risk minimizer can be computed. However, as also observed by Jain, Kothari and Thakurta [13], this assumption often does not hold in reality, and it is not clear if the privacy guarantees based on it still hold if only approximate solutions can be computed. By directly using private SGD, we do not have to rely on the "exact minimizer" assumption any more, and can provide a more general and practical solution.

While previous work has considered differentially private versions of SGD, each of the solutions has characteristics that render them unattractive for implementation in an RDBMS. Song, Chaudhuri and Sarwate [25] adds a large amount of noise at each iteration of SGD to make it differentially private; this results in a noisy solution, which is partially mitigated in practice by mini-batching. Bassily, Smith and Thakurta [2] provides a second solution following the same paradigm, but with less noise per iteration. This is achieved by first, using a novel subsampling technique and second, relaxing the privacy guarantee to (ε, δ)-differential privacy for $\delta > 0$. This relaxation is necessary as they need to use advanced composition results for (ε, δ)-differential privacy. Therefore, both of these solutions require modifying gradient update steps of a non-private SGD algorithm and inject noise for each step. This makes it more difficult to integrate them with an in-RDBMS analytics system. In fact, when we integrated them with Bismarck, it required a somewhat deep modification of the existing code base.

Main Contributions. In this paper we make the following contributions in improving private SGD for better private in-RDBMS machine learning.

- We show, perhaps somewhat surprisingly, that the *output perturbation method*, one of the most basic paradigms for achieving differential privacy, can be applied to achieve private SGD with small noise. Specifically, we propose running SGD for a constant number of passes and then add noise to the resulting output. We provide guarantees on the privacy properties of this approach, which are based on a careful and improved analysis of the L_2-sensitivity of SGD. Our analysis leverages the well-known *expansion properties* of gradient operators [18, 20].

- Because output perturbation assumes black-box access to non-private algorithms, our method thus directly inherits many of the nice properties of SGD, while allowing easier integration. We integrate our private SGD algorithms, as well as SCS13 [25] and BST14 [2] into Bismarck [9], an in-RDBMS data analytics platform. We demonstrate practically that our algorithms can be integrated more easily with little effort and also runs faster than the others.

- We provide a comprehensive empirical study comparing our method with SCS13 and BST14 . Using standard benchmark datasets of different scales, we analyze the effects of different parameters for running private SGD, and demonstrate that our method yields substantially better test accuracy than SCS13 or BST14 for the same number of passes over the data.

The rest of this paper is organized as follows: In Section 2 we present preliminaries. Then in Section 3, we present our private SGD algorithms and analyze their privacy and convergence guarantees. Along the way, we extend our main algorithms in various ways to incorporate common practices of SGD. We then provide a comprehensive empirical study in Section 4 to evaluate private SGD with respect to the three aforementioned properties for in-RDBMS analytics: High accuracy, lower overhead, and ease of integration. We provide more remarks on related theory work in Section 5, and conclude with future directions in Section 6.

2 Preliminaries

This section reviews important definitions and existing results.

Machine Learning and Convex ERM. We begin with a description of our problem in the supervised learning setting. We have a sample space $Z = X \times Y$, where X is a space of feature vectors and Y is a label space. We also have an ordered training set $((x_i, y_i))_{i=1}^m$. Let $\mathcal{W} \subseteq \mathbb{R}^d$ be a hypothesis space equipped with the standard inner product and 2-norm $\|\cdot\|$.[1] We are given a loss function $\ell : \mathcal{W} \times Z \mapsto \mathbb{R}$ which measures the how well a w classifies an example $(x, y) \in Z$, so that given a hypothesis $w \in \mathcal{W}$ and a sample $(x, y) \in Z$, we have a loss $\ell(w, (x, y))$. Our goal is to minimize the *empirical risk* over the training set S (i.e., the empirical risk minimization, or ERM), defined as

$$L_S(w) = \frac{1}{m} \sum_{i=1}^{m} \ell(w, (x_i, y_i)).$$

For fixed S, we think of $\ell_i(w) = \ell(w, (x_i, y_i))$ as a function of w.

A common theme in in-RDBMS and private learning is the *convex* ERM problem, where every ℓ_i is *convex*. We list the following core definitions:

Definition 1. *Let $f : \mathcal{W} \mapsto \mathbb{R}$ be a function:*

- *f is L-*Lipschitz *if for any $u, v \in \mathcal{W}$,*
$$\|f(u) - f(v)\| \leq L\|u - v\|.$$
- *f is* convex *if for any $u, v \in \mathcal{W}$,*
$$f(u) \geq f(v) + \langle \nabla f(v), u - v \rangle.$$
- *f is γ-*strongly convex *if*
$$f(u) \geq f(v) + \langle \nabla f(v), u - v \rangle + \frac{\gamma}{2}\|u - v\|^2.$$
- *f is β-*smooth *if*
$$\|\nabla f(u) - \nabla f(v)\| \leq \beta\|u - v\|.$$

Stochastic Gradient Descent. In a nutshell, SGD performs a series of gradient updates: given time t, iterate w_t, and an individual data point (x_t, y_t), the update rule is as follows:

$$w_{t+1} = G_{\ell_t, \eta_t}(w_t) = w_t - \eta_t \ell_t'(w_t) \tag{1}$$

where $\ell_t(\cdot) = \ell(\cdot; (x_t, y_t))$ is the loss function and $\eta_t \in \mathbb{R}$ is a parameter called the *learning rate*, or *step size*. We will denote G_{ℓ_t, η_t} as G_t. A form of SGD that is commonly used in practice is permutation-based SGD (PSGD): first sample a random permutation τ of $[m]$ (m is the size of the training set S), and then repeatedly apply (1) by cycling through S according to τ. In particular, if we cycle through the dataset k times, we say that it is a k-pass PSGD. We will need two important properties of SGD, *expansiveness* and *boundedness*, which are described in [18, 20].

Definition 2 (Expansiveness)**.** *Let $G : \mathcal{W} \mapsto \mathcal{W}$ be an operator that maps a hypothesis to another hypothesis. G is said to be ρ-expansive if* $\sup_{w,w'} \frac{\|G(w) - G(w')\|}{\|w - w'\|} \leq \rho$.

Definition 3 (Boundedness)**.** *Let $G : \mathcal{W} \mapsto \mathcal{W}$ be an operator that maps a hypothesis to another hypothesis. G is said to be σ-bounded if* $\sup_{w \in \mathcal{W}} \|G(w) - w\| \leq \sigma$.

Lemma 1 (Expansiveness ([18, 20]))**.** *Assume that ℓ is β-smooth. Then, the following hold.*

[1] Using standard results in machine learning, our results easily extend to the case when w lies in a Hilbert Space.

1. *If ℓ is convex, then for any $\eta \le 2/\beta$, $G_{\ell,\eta}$ is 1-expansive.*

2. *If ℓ is γ-strongly convex, then for $\eta \le \frac{2}{\beta+\gamma}$, $G_{\ell,\eta}$ is $(1 - \frac{2\eta\beta\gamma}{\beta+\gamma})$-expansive.*

In particular we use the following simplification due to [11].

Lemma 2 ([11]). *Suppose that ℓ is β-smooth and γ-strongly convex. If $\eta \le \frac{1}{\beta}$, then $G_{\ell,\eta}$ is $(1 - \eta\gamma)$-expansive.*

Lemma 3 (Boundedness). *Assume that ℓ is L-Lipschitz. Then the gradient update $G_{\ell,\eta}$ is (ηL)-bounded.*

We are ready to describe a key quantity studied in this paper.

Definition 4 (δ_t). *Let $w_0, w_1, \ldots, w_T$, and $w'_0, w'_1, \ldots, w'_T$ be two sequences in $\mathcal{W}$. We define δ_t as $\|w_t - w'_t\|$.*

The following lemma bounds δ_t using expansiveness and boundedness properties (Lemma 1 and 3).

Lemma 4 (Growth Recursion [11]). *Fix any two sequences of updates $G_1, \ldots, G_T$ and $G'_1, \ldots, G'_T$. Let $w_0 = w'_0$ and $w_t = G_t(w_{t-1})$ and $w'_t = G'_t(w'_{t-1})$ for $t = 1, 2, \ldots, T$. Then*

$$\delta_0 = 0, \text{ and for } 0 < t \le T$$

$$\delta_t \le \begin{cases} \rho\delta_{t-1} & G_t = G'_t \text{ is } \rho\text{-expansive.} \\ \min(\rho, 1)\delta_{t-1} + 2\sigma_t & G_t \text{ and } G'_t \text{ are } \sigma_t\text{-bounded,} \\ & G_t \text{ is } \rho\text{-expansive.} \end{cases}$$

Differential Privacy. We say that two datasets S, S' are *neighboring*, denoted as $S \sim S'$, if they differ on a single individual's private value. Recall the following definition:

Definition 5 ((ε, δ)-differential privacy). *A (randomized) algorithm A is said to be* ε-differentially private *if for any neighboring datasets S, S', and any event $E \subseteq \mathrm{Range}(A)$, $\Pr[A(S) \in E] \le e^{\varepsilon} \Pr[A(S') \in E] + \delta$.*

In particular, if $\delta = 0$, we will use ε-differential privacy instead of $(\varepsilon, 0)$-differential privacy. A basic paradigm to achieve ε-differential privacy is to examine its L_2-sensitivity,

Definition 6 (L_2-sensitivity). *Let f be a deterministic query that maps a dataset to a vector in $\mathbb{R}^d$. The L_2-sensitivity of f is defined to be $\Delta_2(f) = \max_{S \sim S'} \|f(S) - f(S')\|$.*

The following theorem relates ε-differential privacy and L_2-sensitivity.

Theorem 1 ([7]). *Let f be a deterministic query that maps a database to a vector in $\mathbb{R}^d$. Then publishing $f(D) + \kappa$ where κ is sampled from the distribution with density*

$$p(\kappa) \propto \exp\left(-\frac{\varepsilon\|\kappa\|}{\Delta_2(f)}\right) \tag{2}$$

ensures ε-differential privacy.

Importantly, the L_2-norm of the noise vector, $\|\kappa\|$, is distributed according to the Gamma distribution $\Gamma\left(d, \frac{\Delta_2(q)}{\varepsilon}\right)$. We have the following fact about Gamma distributions:

Theorem 2 ([4]). *For the noise vector κ, we have that with probability at least $1 - \gamma$, $\|\kappa\| \le \frac{d \ln(d/\gamma)\Delta_2(q)}{\varepsilon}$.*

Finally, by changing the noise to Gaussian noise, one obtains (ε, δ)-differential privacy.

Theorem 3 ([8]). *Let f be a deterministic query that maps a database to a vector in $\mathbb{R}^d$. Let $\varepsilon \in (0, 1)$ be arbitrary. For $c^2 > 2\ln(1.25/\delta)$, adding Gaussian noise sampled according to*

$$\mathcal{N}(0, \sigma^2); \quad \sigma \ge \frac{c\Delta}{\varepsilon}, \quad c^2 > 2\ln\left(\frac{1.25}{\delta}\right) \tag{3}$$

ensures (ε, δ)-differentially privacy.

3 Private SGD

In this section we present differentially private PSGD algorithms and analyze their privacy and convergence guarantees. Specifically, we present a new analysis of the output perturbation method for PSGD. Our new analysis shows that very little noise is needed to achieve differential privacy. In fact, the resulting private algorithms have good convergence rates with even *one pass* through the data. Since output perturbation also uses standard PSGD algorithm as a black-box, this makes our algorithms attractive for in-RDBMS scenarios.

This section is structured accordingly in two parts. In Section 3.1 we give two main differentially private algorithms for convex and strongly convex optimization. In Section 3.2 we first prove that these two algorithms are differentially private (Section 3.2.1 and 3.2.2), then extend them in various ways (Section 3.2.3), and finally prove their convergence (Section 3.2.4).

3.1 Algorithms

As we mentioned before, our differentially private PSGD algorithms uses one of the most basic paradigms for achieving differential privacy – the *output perturbation* method [7] based on L_2-*sensitivity* (Definition 6). Specifically, our algorithms are "instantiations" of the output perturbation method where the L_2-sensitivity parameter Δ_2 comes out of our new analysis. To describe the algorithms, we assume a standard permutation-based SGD procedure (denoted as PSGD) which can be invoked as a black-box. To facilitate the presentation, Table 1 summarizes the parameters and their meaning.

Parameter	Meaning
L	Lipschitz constant.
γ	Strong convexity.
β	Smoothness.
ε, δ	Privacy parameters.
η_t	Learning rate or step size at iteration t.
$\mathcal{W}$	Hypothesis space.
$\ell(w, z)$	$w \in \mathcal{W}$, z is a single individual's data.
S	Training set.
m	$\lvert S \rvert$, the size of S.
R	Radius of the convex set.

Table 1: Parameters.

Algorithm 1 Private Convex Permutation-based SGD

Require: $\ell(\cdot, z)$ is convex for every z, $\eta \leq 2/\beta$.
Input: Data S, parameters k, η, ε
1: **function** PrivateConvexPSGD(S, k, ε, η)
2: $w \leftarrow \mathsf{PSGD}(S)$ with k passes and $\eta_t = \eta$
3: $\Delta_2 \leftarrow 2kL\eta$
4: Sample noise vector κ according to (2).
5: **return** $w + \kappa$

Algorithms 1 and 2 give our private SGD algorithms for convex and strongly convex cases, respectively. A key difference between these two algorithms is at line 3 where different L_2-sensitivities are used to sample the noise κ. Note that different learning rates are used: In the convex case, a constant rate is used, while a decreasing rate $\frac{1}{\gamma t}$ is used in the strongly convex case. Finally, note that the standard PSGD is invoked as a black box at line 2.

Algorithm 2 Private Strongly Convex Permutation-based SGD

Require: $\ell(\cdot, z)$ is γ-strongly convex for every z

Input: Data S, parameters k, ε

1: **function** PrivateStronglyConvexPSGD(S, k, ε)
2: $w \leftarrow \text{PSGD}(S)$ with k passes and $\eta_t = \min(\frac{1}{\beta}, \frac{1}{\gamma t})$
3: $\Delta_2 \leftarrow \frac{2L}{\gamma m}$
4: Sample noise vector κ according to (2).
5: **return** $w + \kappa$

3.2 Analysis

In this section we establish privacy and convergence guarantees of Algorithms 1 and 2. Along the way, we also describe extensions to accommodate common practices in running stochastic gradient descent. Most proofs in this section are deferred to the appendix.

We start with an overview of the privacy analysis. Let $A(r; S)$ denote a randomized non-private algorithm where r denotes the randomness (e.g., random permutations sampled by SGD) and S denotes the input training set. On a pair of neighboring datasets S, S' (i.e., S, S' differ on a single data point), we want to bound $\max_{r,r'} \|A(r; S) - A(r'; S')\|$, where r, r' can be *different randomness sequences of* A in general. This can be complicated in general since $A(r; \cdot)$ and $A(r'; \cdot)$ may access the data in vastly different patterns.

Fortunately, we observe that for non-adaptive randomized algorithms, one can consider randomness sequences one at a time, and it suffices to bound $\max_r \|A(r; S) - A(r; S')\|$. We have the following definition,

Definition 7 (Non-Adaptive Algorithms)**.** *A randomized algorithm A is* non-adaptive *if its random choices do not depend on the input data values.*

PSGD is clearly non-adaptive as a single random permutation is sampled at the very beginning of the algorithm. Another common SGD variant, where one independently and uniformly samples $i_t \sim [m]$ at iteration t and picks the i_t-th data point, is also non-adaptive. In fact, more modern SGD variants, such as Stochastic Variance Reduced Gradient (SVRG [15]) and Stochastic Average Gradient (SAG [21]), are non-adaptive as well. Now we have the following lemma for non-adaptive algorithms and differential privacy.

Lemma 5. *Let $A(r; S)$ be a non-adaptive randomized algorithm where r denotes the randomness of the algorithm and S denotes the dataset A works on. Suppose that*

$$\sup_{S \sim S'} \sup_r \|A(r; S) - A(r; S')\| \le \Delta.$$

Then publishing $A(r; S) + \kappa$ where κ is sampled with density $p(\kappa) \propto \exp\left(-\frac{\varepsilon \|\kappa\|_2}{\Delta}\right)$ ensures ε-differential privacy.

Proof. Let $\widetilde{A}$ denote the private version of A. $\widetilde{A}$ has two parts of randomness: One part is r, which is used to compute $A(r; S)$; the second part is κ, which is used for perturbation (i.e. $A(r; S) + \kappa$). Let R be the random variable corresponding to the randomness of A. Note that R does not depend on the input training set. Thus for any event E,

$$\begin{aligned} &\Pr[\widetilde{A}((r, \kappa); S) \in E] \\ &= \sum_r \Pr[R = r] \cdot \Pr_\kappa[A((r, \kappa); S) \in E \mid R = r]. \end{aligned} \tag{4}$$

Denote $\Pr_\kappa[A((r, \kappa); S) \in E \mid R = r]$ by $p_\kappa(A_r(S) \in E)$. Then similarly for S' we have that

$$\begin{aligned} &\Pr[\widetilde{A}((r, \kappa); S') \in E] \\ &= \sum_r \Pr[R = r] \cdot p_\kappa(A_r(S') \in E). \end{aligned} \tag{5}$$

Compare (4) and (5) term by term (for every r): the lemma then follows as we calibrate the noise κ so that $p_\kappa(A_r(S) \in E) \leq e^\varepsilon p_\kappa(A_r(S') \in E)$. □

From now on we denote PSGD by A. With Definition 4, our goal is thus to bound $\sup_{S\sim S'} \sup_r \delta_T$. Fortunately, one can now bound this using the expansion and boundedness properties. A final issue is the contributions of the differing data point. Because for PSGD in each pass the differing data point is only encountered once, one can easily accumulate their contributions and obtains the desired L_2-sensitivity bounds.

3.2.1 Convex Optimization

In this section we prove privacy guarantee when $\ell(\cdot, z)$ is convex. Recall that for general convex optimization, we have 1-expansiveness by Lemma 1.1. We thus have the following lemma that bounds δ_T.

Lemma 6. *Consider k-passes PSGD for L-Lipschitz, convex and β-smooth optimization where $\eta_t \leq \frac{2}{\beta}$ for $t = 1, \ldots, T$. Let S, S^i be any neighboring datasets. Let r be a random permutation of $[m]$. Suppose that $r(i) = i^*$. Let $T = km$, then $\delta_T \leq 2L \sum_{j=0}^{k-1} \eta_{i^*+jm}$.*

We immediately have the following corollary on L_2-sensitivity with constant step size,

Corollary 1 (Constant Step Size)**.** *Consider k-passes PSGD for L-Lipschitz, convex and β-smooth optimization. Suppose further that we have constant learning rate $\eta_1 = \eta_2 = \cdots = \eta_T = \eta \leq \frac{2}{\beta}$. Then $\sup_{S\sim S'} \sup_r \delta_T \leq 2kL\eta$.*

This directly yields the following theorem,

Theorem 4. *Algorithm 1 is ε-differentially private.*

We now give L_2-sensitivity results for two different choices of step sizes, which are also common for convex optimization.

Corollary 2 (Decreasing Step Size)**.** *Let $c \in [0, 1)$ be some constant. Consider k-passes PSGD for L-Lipschitz, convex and β-smooth optimization. Suppose further that we take decreasing step size $\eta_t = \frac{2}{\beta(t+m^c)}$ where m is the training set size. Then $\sup_{S\sim S'} \sup_r \delta_T = \frac{4L}{\beta}\left(\frac{1}{m^c} + \frac{\ln k}{m}\right)$.*

Corollary 3 (Square-Root Step Size)**.** *Let $c \in [0, 1)$ be some constant. Consider k-passes PSGD for L-Lipschitz, convex and β-smooth optimization. Suppose further that we take square-root step size $\eta_t = \frac{2}{\beta(\sqrt{t}+m^c)}$. Then*

$$\begin{aligned}\sup_{S\sim S'} \sup_r \delta_T \leq & \frac{4L}{\beta}\left(\sum_{j=0}^{k-1} \frac{1}{\sqrt{jm+1}+m^c}\right) \\ = & O\left(\frac{L}{\beta}\left(\frac{1}{m^c} + \min\left(\frac{k}{m^c}, \sqrt{\frac{k}{m}}\right)\right)\right).\end{aligned}$$

Two remarks are in order: First, in Lemma 1 we use "constant" step size for the SGD. However, one should note that "constant step size" does not mean "constant noise" in a typical differential privacy sense. Constant step size for SGD can depend on the size of the training set, and in particular can vanish to zero as training set size increases. This, in particular, implies that our private PSGD algorithm is *consistent* (i.e., L_2-sensitivity vanishes to 0 as training set size increases). In fact, in typical convergence results of SGD (see, for example in [3, 17]) the constant step size η is set to $1/T^{O(1)}$ where T is the total number of iterations. In such cases, T asymptotically depends on m, the size of the training set. For example, if the step size is $1/\sqrt{T}$, and we run a single epoch through the data, then the L_2-sensitivity is indeed $\frac{2L}{\sqrt{m}}$, and so the noise vanishes to 0 as m increases.

Second, in SGD we may want to approximately measure the "maximal progress" one can make, which is how far away one can walk from the starting point. Formally, this is the sum of all step sizes. We note that, for this to be large in the case of decreasing and square-root step sizes, c should not be set to be too large, while making it large is beneficial for reducing noise. Indeed, since our initial step size is $O(1/m^c)$, and the final step size is $O(1/km)$, the sum is roughly $\int_{m^c}^{km} \frac{1}{x} dx = O((1-c)\ln m)$.

3.2.2 Strongly Convex Optimization

Now we consider the case where $\ell(\cdot, z)$ is γ-strongly convex. In this case the sensitivity is smaller because the gradient operators are ρ-expansive for $\rho < 1$ so in particular they become contractions. We have the following lemmas.

Lemma 7 (Constant Step Size)**.** *Consider PSGD for L-Lipschitz, γ-strongly convex and β-smooth optimization with constant step sizes $\eta \le \frac{1}{\beta}$. Let k be the number of passes. Let S, S' be two neighboring datasets differing at the i-th data point. Let r be a random permutation of $[m]$. Suppose that $r(i) = i^*$. Let $T = km$, then $\delta_T \le 2L\eta \sum_{j=0}^{k-1} (1-\eta\gamma)^{(k-j)m-i^*}$. In particular,*

$$\sup_{S \sim S'} \sup_r \delta_T \le \frac{2\eta L}{1-(1-\eta\gamma)^m}.$$

Lemma 8 (Decreasing Step Size)**.** *Consider k-passes PSGD for L-Lipschitz, γ-strongly convex and β-smooth optimization. Suppose further that we use decreasing step length: $\eta_t = \min(\frac{1}{\gamma t}, \frac{1}{\beta})$. Let S, S' be two neighboring datasets differing at the i-th data point. Let r be a random permutation of $[m]$. Suppose that $r(i) = i^*$. Let $T = km$, then $\sup_{S \sim S'} \sup_r \delta_T \le \frac{2L}{\gamma m}$.*

In particular, Lemma 8 yields the following theorem,

Theorem 5. *Algorithm 2 is ε-differentially private.*

One should contrast this theorem with Theorem 4: In the convex case we bound L_2-sensitivity by $2kL\eta$, while in the strongly convex case we bound it by $2L/\gamma m$.

3.2.3 Extensions

In this section we extend our main argument in several ways: (ε, δ)-differential privacy, mini-batching, model averaging, fresh permutation at each pass, and finally constrained optimization. These extensions can be easily incorporated to standard PSGD algorithm, as well as our private algorithms 1 and 2, and are used in our empirical study later.

(ε, δ)-Differential Privacy. We can also obtain (ε, δ)-differential privacy easily using Gaussian noise (see Theorem 3). We have that

Lemma 9. *Let $A(r; S)$ be a non-adaptive randomized algorithm where r denotes the randomness of the algorithm and S denote the dataset. Suppose that*

$$\sup_{S \sim S'} \sup_r \|A(r; S) - A(r; S')\| \le \Delta.$$

Then for any $\varepsilon \in (0, 1)$, publishing $A(r; S) + \kappa$ where each component of κ is sampled using (3) ensures (ε, δ)-differential privacy.

In particular, combining this with our L_2-sensitivity results, we get the following two theorems,

Theorem 6 (Convex and Constant Step)**.** *Algorithm 1 is (ε, δ)-differentially private if each component of κ at line 3 is sampled according to equation (3).*

Theorem 7 (Strongly Convex and Decreasing Step)**.** *Algorithm 2 is (ε, δ)-differentially private if each component of κ at line 3 is sampled according to equation (3).*

Mini-batching. A popular way to do SGD is that at each step, instead of sampling a single data point z_t and do gradient update with respect to it, we randomly sample a batch $B \subseteq [m]$ of size b, and do

$$w_t = w_{t-1} - \eta_t \frac{1}{b}\left(\sum_{i\in B} \ell_i'(w_{t-1})\right) = \frac{1}{b}\sum_{i\in B} G_i(w_{t-1}).$$

For permutation SGD, a natural way to employ mini-batch is to partition the m data points into mini-batches of size b (for simplicity let us assume that b divides m), and do gradient updates with respect to each chunk. In this case, we notice that mini-batch indeed improves the sensitivity by a factor of b. In fact, let us consider neighboring datasets S, S', and at step t, we have batches B, B' that differ in at most one data point. Without loss of generality, let us consider the case where B, B' differ at one data point, then on S we have $w_t = \frac{1}{b}\sum_{i\in B} G_i(w_{t-1})$, and on S' we have $w_t' = \frac{1}{b}\sum_{i\in B} G_i'(w_{t-1}')$, and so

$$\begin{aligned}\delta_t &= \left\| \frac{1}{b}\sum_{i\in B} G_i(w_{t-1}) - G_i'(w_{t-1}') \right\| \\ &\le \frac{1}{b}\sum_{i=1}^{B} \|G_i(w_{t-1}) - G_i'(w_{t-1}')\|.\end{aligned}$$

We note that for all i except one in B, $G_i = G_i'$, and so by the Growth Recursion Lemma 4, $\|G_i(w_{t-1}) - G_i'(w_{t-1}')\| \le \rho\delta_{t-1}$ if G_i is ρ-expansive, and for the differing index i^*, $\|G_{i^*}(w_{t-1}) - G_{i^*}'(w_{t-1}')\| \le \min(\rho, 1)\delta_{t-1} + 2\sigma_t$. Therefore, for a uniform bound ρ_t on expansiveness and σ_t on boundedness (for all $i \in B$, which is the case in our analysis), we have that $\delta_t \le \rho_t\delta_{t-1} + \frac{2\sigma_t}{b}$. This implies a factor b improvement for all our sensitivity bounds.

Model Averaging. Model averaging is a popular technique for SGD. For example, given iterates $w_1, \ldots, w_T$, a common way to do model averaging is either to output $\frac{1}{T}\sum_{t=1}^T w_t$ or output the average of the last $\log T$ iterates. We show that model averaging will not affect our sensitivity result, and in fact it will give a constant-factor improvement when earlier iterates have smaller sensitivities. We have the following lemma.

Lemma 10 (Model Averaging)**.** *Suppose that instead of returning w_T at the end of the optimization, we return an averaged model $\bar{w} = \sum_{t=1}^T \alpha_t w_t$, where α_t is a sequence of coefficients that only depend on t, T. Then,*

$$\sup_{S\sim S'}\sup_r \|\bar{w} - \bar{w}'\| \le \sum_{t=1}^T \alpha_t \|w_t - w_t'\| = \sum_{t=1}^T \alpha_t\delta_t.$$

In particular, we notice that the δ_t's we derived before are non-decreasing, so the sensitivity is bounded by $(\sum_{t=1}^T \alpha_t)\delta_T$.

Fresh Permutation at Each Pass. We note that our analysis extends verbatim to the case where in each pass a new permutation is sampled. This is because our analysis applies to *any* fixed permutation.

Constrained Optimization. Until now, our SGD algorithm is for unconstrained optimization. That is, the hypothesis space $\mathcal{W}$ is the entire $\mathbb{R}^d$. Our results easily extend to constrained optimization where the hypothesis space $\mathcal{W}$ is a convex set $\mathcal{C}$. That is, our goal is to compute $\min_{w\in\mathcal{C}} L_S(w)$. In this case, we change the original gradient update rule 1 to the *projected gradient update rule*:

$$w_t = \prod_{\mathcal{C}}\left(w_{t-1} - \eta_t \ell_t'(w_{t-1})\right), \tag{6}$$

where $\prod_{\mathcal{C}}(w) = \arg\min_v \|v - w\|$ is the projection of w to $\mathcal{C}$. It is easy to see that our analysis carries over verbatim to the projected gradient descent. In fact, our analysis works as long as the optimization is carried over a Hilbert space (i.e., the $\|\cdot\|$ is induced by some inner product). The essential reason is that projection will not increase the distance ($\|\prod u - \prod v\| \le \|u - v\|$), and thus will not affect our sensitivity argument.

3.2.4 Convergence of Optimization

We now bound the optimization error of our private PSGD algorithms. More specifically, we bound the *excess empirical risk* $L_S(w) - L_S^*$ where $L_S(w)$ is the loss of the output w of our private SGD algorithm and L_S^* is the minimum obtained by any w in the feasible set $\mathcal{W}$. Note that in PSGD we sample data points *without replacement*. While sampling without replacement benefits our L_2-sensitivity argument, its convergence behavior is poorly understood in theory. Our results are based on very recent advances by Shamir [24] on the sampling-without-replacement SGD.

As in Shamir [24], we assume that the loss function ℓ_i takes the form of $\ell_i(\langle w, x_i\rangle) + r(w)$ where r is some fixed function. Further we assume that the optimization is carried over a convex set $\mathcal{C}$ of radius R (i.e., $\|w\| \le R$ for $w \in \mathcal{C}$). We use projected PSGD algorithm (i.e., we use the projected gradient update rule 6).

Finally, $R(T)$ is a regret bound if for any $w \in \mathcal{W}$ and convex-Lipschitz $\ell_1, \ldots, \ell_T$, $\sum_{t=1}^T \ell_t(w_t) - \sum_{t=1}^T \ell_t(w) \le R(T)$ and $R(T)$ is sublinear in T. We use the following regret bound,

Theorem 8 (Zinkevich [28]). *For SGD with constant step size $\eta_1 = \eta_2 = \cdots = \eta_T = \eta$, the regret bound $R(T)$ is bounded by $\frac{R^2}{2\eta} + \frac{L^2 T \eta}{2}$.*

We are now ready to bound the excess empirical risk. We start with the following simple lemma.

Lemma 11 (Error Introduced by Privacy). *Consider L-Lipschitz and β-smooth optimization. Let w be the output of the non-private SGD algorithm, κ be the noise of the output perturbation, and $\widetilde{w} = w + \kappa$. Then $L_S(w) - L_S(\widetilde{w}) \le L\|\kappa\|$.*

Convex Optimization. If $\ell(\cdot, z)$ is convex, we use the following theorem from Shamir [24],

Theorem 9 (Corollary 1 of Shamir [24]). *Let $T \le m$ (that is we take at most 1-pass over the data). Suppose that each iterate w_t is chosen from $\mathcal{W}$, and the SGD algorithm has regret bound $R(T)$, and that $\sup_{t, w\in\mathcal{W}} |\ell_t(w)| \le R$, and $\|w\| \le R$ for all $w \in \mathcal{W}$. Finally, suppose that each loss function ℓ_t takes the form $\tilde{\ell}(\langle w, x_t\rangle) + r(w)$ for some L-Lipschitz $\tilde{\ell}(\cdot, x_t)$ and $\|x_t\| \le 1$, and a fixed r, then*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^T L_S(w_t) - L_S(w^*)\right] \le \frac{R(T)}{T} + \frac{2(12 + \sqrt{2}L)R}{\sqrt{m}}.$$

Together with Theorem 8, we thus have the following lemma,

Lemma 12. *Consider the same setting as in Theorem 9, and 1-pass PSGD optimization defined according to rule (6). Suppose further that we have constant learning rate $\eta = \frac{R}{L\sqrt{m}}$. Finally, let $\bar{w}_m$ be the model averaging $\frac{1}{m}\sum_{t=1}^T w_t$. Then,*

$$\mathbb{E}[L_S(\bar{w}_T) - L_S^*] \le \frac{(L + 2(12 + \sqrt{L}))R}{\sqrt{m}}.$$

Now we can bound the excess empirical risk as follows,

Theorem 10 (Convex and Constant Step Size). *Consider the same setting as in Lemma 12 where the step size is constant $\eta = \frac{R}{L\sqrt{m}}$. Let $\tilde{w} = \bar{w}_T + \kappa$ be the result of output perturbation. Then*

$$\mathbb{E}[L_S(\tilde{w}) - L_S^*] \le \frac{(L + (2(12 + \sqrt{L}))R}{\sqrt{m}} + \frac{2dLR}{\varepsilon\sqrt{m}}.$$

Note that the term $\frac{2dLR}{\varepsilon\sqrt{m}}$ corresponds to the expectation of $L\|\kappa\|$.

Strongly Convex Optimization. If $\ell(\cdot, z)$ is γ-strongly convex, we instead use the following theorem,

Theorem 11 (Theorem 3 of Shamir [24]). *Suppose $\mathcal{W}$ has diameter R, and $L_S(\cdot)$ is γ-strongly convex on $\mathcal{W}$. Assume that each loss function ℓ_t takes the for $\bar{\ell}(\langle w_t, x_t\rangle) + r(w)$ where $\|x_i\| \leq 1$, $r(\cdot)$ is possibly some regularization term, and each $\bar{\ell}(\cdot, x_t)$ is L-Lipschitz and β-smooth. Furthermore, suppose $\sup_{w\in\mathcal{W}} \|\ell_t'(w)\| \leq G$. Then for any $1 < T \leq m$, if we run SGD for T iterations with step size $\eta_t = 1/\gamma t$, we have*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} L_S(w_t) - L_S(w^*)\right] \leq c \cdot \frac{((L+\beta R)^2 + G^2)\log T}{\gamma T},$$

where c is some universal positive constant.

Using the same argument as in the convex case, we immediately have the following theorem,

Theorem 12 (Strongly Convex and Decreasing Step Size). *Consider the same setting as in Theorem 11 where the step size is $\eta_t = \frac{1}{\gamma t}$. Consider 1-pass PSGD. Let $\bar{w}_T$ be the result of model averaging and $\tilde{w} = \bar{w}_T + \kappa$ be the result of output perturbation. Then $\mathbb{E}[L_S(\tilde{w}) - L_S(w^*)] \leq c \cdot \frac{((L+\beta R)^2+G^2)\log m}{\gamma m} + \frac{2dG^2}{\varepsilon\gamma m}$.*

Several remarks are in order. (i) Our excess empirical risk bounds are weaker than those obtained in Bassily et al. [2], where in the convex case their bound is $O\left(\frac{LR\log^{3/2}(m/\delta)\sqrt{d\log(1/\delta)}}{\varepsilon m}\right)$, and in the strongly convex case their bound is $O\left(\frac{L^2\log^2(m/\delta)d\log(1/\delta)}{\gamma\varepsilon^2 m^2}\right)$. (ii) However, our bounds are for a *single pass* through the data, while theirs need m passes. (iii) Our bounds are for ε-differential privacy, while theirs are for (ε, δ)-differential privacy ($\delta > 0$). (iv) Finally, our bounds need more assumptions: Specifically, we assume that ℓ_t takes the special form of $\bar{\ell}(\langle w, x_t\rangle) + r(w)$.

4 Empirical Evaluation

In this section we conduct a comprehensive empirical study investigating three alternatives for private SGD in RDBMSes: Two previously proposed state-of-the-art private SGD algorithms, SCS13 [25] and BST14 [2], and our algorithms which are instantiations of the output perturbation method with our new analysis.

In our study we try to answer the following four main questions regarding the three issues for a good private in-RDBMS SGD: High accuracy, low overhead and ease of integration.

1. *What is the effort to integrate each algorithm into an in-RDBMS system?*
2. *How does the test accuracy of our algorithms compare to SCS13 and BST14?*
3. *How do various parameters, such as mini-batch sizes, number of passes (epochs), and privacy parameters, affect the test accuracy?*
4. *What is the runtime overhead when we deploy these algorithms in real data analytics systems?*

As a summary, our main findings are the following: **(i)** Our SGD algorithm requires almost no changes to Bismarck, while both SCS13 and BST14 require deeper changes. **(ii)** Under the same differential privacy guarantees, our private SGD algorithms yield substantially better accuracy than SCS13 and BST14, for all datasets and settings of parameters we test. **(iii)** As for the effect of parameters, our empirical results align well with the theory. For example, as one might expect, mini-batch sizes are important for reducing privacy noise. The number of passes is more subtle. For our algorithm, if the learning task is only convex, more passes result in larger noise (e.g., see Lemma 1), and so give rise to potentially worse test accuracy. On the other hand, if the learning task is strongly convex, the number of passes will not affect the noise magnitude

(e.g., see Lemma 8). As a result, doing more passes may lead to better convergence and thus potentially better test accuracy. Interestingly, we note that slightly enlarging mini-batch size can reduce noise very effectively so it is affordable to run our private algorithms for more passes to get better convergence in the convex case. This corroborates the results of [25] that mini-batches are helpful in private SGD settings. **(iv)** Our algorithms incur virtually no overhead, while SCS13 and BST14 run much slower. The reason is that our algorithms only need to inject noise once at the end while SCS13 and BST14 need to inject noise at every gradient update step.

In the rest of this section we give more details of our empirical study. Our discussion is structured as follows: In Section 4.1 we first discuss the implemented algorithms. In particular, we discuss how we modify SCS13 and BST14 to make them better fit into our experiments. We also give some remarks on other relevant previous algorithms, and on parameter tuning. Then in Section 4.2 we discuss the effort of integrating different algorithms into Bismarck. Next in Section 4.3 we discuss the experimental design and datasets for testing accuracy and runtime. Then in Section 4.4, we report the results on test accuracy for various datasets and parameter settings, and discuss the effects of parameters. Finally in Section 4.5, we report runtime overhead.

4.1 Implemented Algorithms

We first discuss implementations of our algorithms, SCS13 and BST14. Importantly, we extend both SCS13 and BST14 to make them better fit into our experiments. Among these extensions, probably most importantly, we extend BST14 to support a smaller number of iterations through the data and *reduce the amount of noise* needed for each iteration. Our extension makes BST14 more competitive in our experiments.

Our Algorithms. We implement Algorithms 1 and 2 with the extensions of mini-batching and constrained optimization (see Section 3.2.3). Note that our algorithms invoke a standard PSGD algorithm as a black-box and Bismarck already supports mini-batching and constrained optimization. Therefore the only change we need to make for Algorithms 1 and 2 is the setting of L_2-sensitivity parameter Δ_2 at line 3 of respective algorithms, which we divide by b if the mini-batch size is b.

BST14 [2]. The original BST14 algorithm needs $O(m^2)$ iterations to finish, which is prohibitive for even moderate sized datasets. We extend it to support cm iterations for some constant c. Reducing the number of iterations means that potentially we can *reduce the amount of noise for privacy* because data is "less examined." This is indeed the case: One can go through the same proof in [2] with a smaller number of iterations, and show that each iteration only needs a smaller amount of noise than before (unfortunately this does not give convergence results). Our extension makes BST14 *more competitive*. In fact it yields significantly better test accuracy compared to the case where one naïvely stops BST14 after c passes, but the noise magnitude in each iteration is the same as in the original paper [2] (which is for m passes). The extended BST14 algorithms are given in Algorithm 3 and 4. Finally, we also make straightforward extensions so that BST14 supports mini-batching.

SCS13 [25]. We also modify [25], which originally only supports one pass through the data, to support multi-passes over the data.

Other Related Work. We also note the work of Jain, Kothari and Thakurta [13] which is related to our setting. In particular their Algorithm 6 is similar to our private SGD algorithm in the setting of strong convexity and (ε, δ)-differential privacy. However, we note that their algorithm uses Implicit Gradient Descent (IGD), which belongs to *proximal algorithms* (see for example Parikh and Boyd [19]) and is known to be more difficult to implement than stochastic gradient methods. Due to this consideration, in this study we will not compare empirically with this algorithm. Finally, we also note that [13] also has an SGD-style algorithm (Algorithm 3) for strongly convex optimization and (ε, δ)-differential privacy. This algorithm adds noise comparable to our algorithm *at each step* of the optimization, and as a result, we do not compare with this algorithm either.

Parameter Tuning. We observe that for all the SGD algorithms we considered, multiple parameters need

Algorithm 3 Convex BST14 Constant with Constant Epochs

Require: $\ell(\cdot, z)$ is convex for every z, $\eta \leq 2/\beta$.

Input: Data S, parameters $k, \varepsilon, \delta, d, L, R$

function ConvexBST14ConstNpass($S, k, \varepsilon, \delta, d, L, R$)
  $m \leftarrow |S|$
  $T \leftarrow km$
  $\delta_1 \leftarrow \delta/km$
  $\varepsilon_1 \leftarrow$ Solution of $\varepsilon = T\varepsilon_1(e^{\varepsilon}_1 - 1) + \sqrt{2T\ln(1/\delta_1)}\varepsilon_1$
  $\varepsilon_2 \leftarrow \min(1, m\varepsilon_1/2)$
  $\sigma^2 \leftarrow 2\ln(1.25/\delta_1)/\varepsilon_2^2$
  $w \leftarrow 0$
  for $t = 1, 2, \ldots, T$ **do**
    $i_t \sim [m]$ and let (x_{i_t}, y_{i_t}) be the data point.
    $z \sim \mathcal{N}(0, \sigma^2 \iota I_d)$ ▷ $\iota = 1$ for logistic regression, and in general is the L_2-sensitivity localized to an iteration; I_d is d-dimensional identity matrix.
    $w \leftarrow \prod_{\mathcal{W}} \left(w - \eta_t(\nabla\ell(w; (x_{i_t}, y_{i_t}) + z)\right)$ where $\eta_t = \frac{2R}{G\sqrt{t}}$ and $G = \sqrt{d\sigma^2 + b^2L^2}$.
  return w_T

Algorithm 4 Strongly Convex BST14 with Constant Epochs

Input: Data S, parameters $k, \varepsilon, \delta, d, L, R$

function StronglyConvexBST14ConstNpass($S, k, \varepsilon, \delta, d, L, R$)
  $m \leftarrow |S|$
  $T \leftarrow km$
  $\delta_1 \leftarrow \delta/km$
  $\varepsilon_1 \leftarrow$ Solution of $\varepsilon = T\varepsilon_1(e^{\varepsilon}_1 - 1) + \sqrt{2T\ln(1/\delta_1)}\varepsilon_1$
  $\varepsilon_2 \leftarrow \min(1, m\varepsilon_1/2)$
  $\sigma^2 \leftarrow 2\ln(1.25/\delta_1)/\varepsilon_2^2$
  $w \leftarrow 0$
  for $t = 1, 2, \ldots, T$ **do**
    $i_t \sim [m]$ and let (x_{i_t}, y_{i_t}) be the data point.
    $z \sim \mathcal{N}(0, \sigma^2 \iota I_d)$
    $w \leftarrow \prod_{\mathcal{W}} \left(w - \eta_t(\nabla\ell(w; (x_{i_t}, y_{i_t}) + z)\right)$, $\eta_t = \frac{1}{\gamma t}$.
  return w

to be fine-tuned to achieve the best performance. In particular, the mini-batch sizes, number of passes, L_2-regularization parameter, and privacy parameters all have significant impact on the final performance. Therefore, a natural attempt is to show how one can *tune* these parameters in order to achieve best performance. However, one should note that this is an issue orthogonal to our main questions: We are interested in obtaining *relative* performance of the algorithms for some same (sensible) setting of parameters, rather than seeking the best parameters for one particular algorithm. Moreover, we note that under differential privacy, tuning parameters must also be done *privately*, and there has been an independent line of research [4, 5] investigating it. As a result, we choose to compare performance of these algorithms under *various settings of parameters*, and leave tuning for the best performing parameters as separate future work.

4.2 Integration with Bismarck

We now explain how we integrate private SGD algorithms in RDBMS. To begin with, we note that the state-of-the-art way to do in-RDBMS is via the User Defined Aggregates (UDA) offered by almost all RDBM-

Ses [10]. Using UDAs enables scaling to larger-than-memory datasets seamlessly while still being fast.[2] A well-known open source implementation of the UDAs required is Bismarck [9]. Bismarck achieves high performance and scalability through a unified architecture of in-RDBMS data analytics systems using the permutation-based stochastic gradient descent.

Therefore, we use Bismarck to experiment with private SGD inside RDBMS. Specifically, we use Bismarck on top of PostgreSQL, which implements the UDA for SGD in C to provide high runtime efficiency. Our results carry over naturally to any other UDA-based implementation of analytics in an RDBMS. The rest of this section is organized as follows. We first describe Bismarck's system architecture. We then compare the system extensions and the implementation effort needed for integrating our private PSGD algorithm as well as SCS13 and BST14.

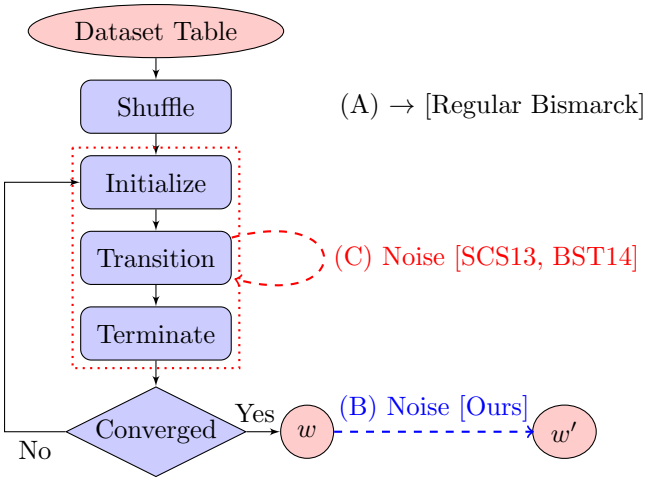


Figure 1: (A) System architecture of regular Bismarck. (B) Extension to implement our algorithms. (C) Extension to implement any of SCS13 and BST14.

Figure 1 (A) gives an overview of Bismarck's architecture. The dataset is stored as a table in PostgreSQL. Bismarck permutes the table using an SQL query with a shuffling clause, viz., `ORDER BY RANDOM()`. A pass (or epoch, which is used more often in practice) of SGD is implemented as a C UDA and this UDA is invoked with an SQL query for each epoch. A front-end controller in Python issues the SQL queries and also applies the convergence test for SGD after each epoch. The developer has to provide implementations of three functions in the UDA's C API: initialize, transition, and terminate, all of which operate on the `aggregation state`, which is the quantity being computed over the table.

To explain how this works, we compare SGD with a standard SQL aggregate: `AVG`. The state for `AVG` is the 2-tuple $(\texttt{sum}, \texttt{count})$, while that for SGD is the model vector w. The function initialize sets $(\texttt{sum}, \texttt{count}) = (0, 0)$ for `AVG`, while for SGD, it sets w to the value given by the Python controller (the previous epoch's output model). The function `transition` updates the state based on a single tuple (one example). For example, given a tuple with value `x`, the state update for `AVG` is as follows: $(\texttt{sum}, \texttt{count}) \mathrel{+}= (\texttt{x}, 1)$. For SGD, `x` is the feature vector and the update is the update rule for SGD with the gradient on `x`. If mini-batch SGD is used, the updates are made to a temporary accumulated gradient that is part of the aggregation state along with counters to track the number of examples and mini-batches seen so far. When a mini-batch is over, the transition function updates w using the accumulated gradient for that mini-batch using an appropriate step size. The function terminate computes `sum/count` and outputs it for `AVG`, while for SGD, it simply returns w at the end of that epoch.

It is easy to see that our private SGD algorithm requires almost no change to Bismarck – simply add noise to the final w output after all epochs, as illustrated in Figure 1 (B). Thus, our algorithm does not modify any

[2] The MapReduce abstraction is similar to an RDBMS UDA [1]. Thus our implementation ideas apply to MapReduce-based systems as well.

of the RDBMS-related C UDA code. In fact, we were able to implement our algorithm in about 10 lines of code (LOC) in Python within the front-end Python controller. In contrast, both SCS13 and BST14 require deeper changes to the UDA's transition function because they need to add noise at the end of each mini-batch update. Thus, implementing them required adding dozens of LOC in C to implement their noise addition procedure within the transition function, as illustrated in Figure 1 (C). Furthermore, Python's scipy library already provides the sophisticated distributions needed for sampling the noise (gamma and multivariate normal), which our algorithm's implementation exploits. But for both SCS13 and BST14, we need to implement some of these distributions in C so that it can be used in the UDA.[3]

4.3 Experimental Method and Datasets

We now describe our experimental method and datasets used for testing accuracy and runtime overhead.

Test Scenarios. We consider four main scenarios to evaluate the algorithms: (1) Convex, ε-differential privacy, (2) Convex, (ε, δ)-differential privacy, (3) Strongly Convex, ε-differential privacy, and finally (4) Strongly Convex, (ε, δ)-differential privacy. Note that BST14 only supports (ε, δ)-differential privacy. Thus for tests (1) and (3) we compare non-private algorithm, our algorithms, and SCS13. For tests (2) and (4), we compare non-private algorithm, our algorithms, SCS13 and BST14. For each of these scenarios, we train models on standard datasets used for evaluating SGD and measure the test accuracy of the resulting models on the test datasets. As is common in the literature for evaluating SGD in the context of convex optimization, we choose *logistic regression* models. We use the standard logistic regression for the convex case (Tests (1) and (2)), and logistic regression regularized by L_2-regularizer for the strongly convex case (Tests (1) and (3)). We now describe the datasets and the parameter spaces considered for each of them.

Dataset	Task	Train Size	Test Size	#Dimensions
MNIST	10 classes	60000	10000	784 (50) [*]
Protein	Binary	72876	72875	74
Forest	Binary	498010	83002	54

Table 2: Datasets. Each row gives the name of the dataset, number of classes in the classification task, sizes of training and test sets, and finally the number of dimensions. [*]: For MNIST, it originally has 784 dimensions, which is difficult for differential privacy as the noise scales linearly with the number of dimensions. Therefore we randomly project it to 50 dimensions. All data points are normalized to the unit sphere.

Datasets. We consider three standard benchmark datasets: MNIST[4], Protein[5], and Forest Covertype[6]. MNIST is a popular dataset used for image classification. MNIST poses a challenge to differential privacy for three reasons: First, it "high-dimensional" with 784 dimensions. To get meaningful test accuracy we thus use Gaussian Random Projection, which is known to preserve differential privacy, to randomly project to 50 dimensions. This random projection only incurs very small loss in test accuracy, and thus the performance of non-private SGD on 50 dimensions will serve the baseline. Second, MNIST is of medium size and differential privacy is known to be more difficult for medium or small sized datasets. Finally, MNIST is a multiclass classification (there are 10 digits), we built "one-vs.-all" multiclass logitstic regression models. Using the "one-vs.-all" method means that we need to construct 10 binary models (one for each digit). Thus for privacy, one needs to split the privacy budget across sub-models. We used the simplest composition theorem [8], and divide the privacy budget evenly.

For Protein dataset, because its test dataset does not have labels, we randomly partition the training set into halves to form train and test datasets. Logistic regression models have very good test accuracy on it.

[3] One could use the Python-based UDAs in PostgreSQL but that incurs a significant runtime performance penalty compared to C UDAs.

[4] http://yann.lecun.com/exdb/mnist/.

[5] http://osmot.cs.cornell.edu/kddcup/datasets.html.

[6] https://archive.ics.uci.edu/ml/datasets/Covertype.

Finally, Forest Covertype is a large dataset with 581012 data points, almost 6 times larger than previous ones. We split it to have 498010 training points and 83002 test points. We use this large dataset for two purposes: First, in this case, one may expect that privacy will follow more easily. We test to what degree this holds for different private algorithms. Second, since training on such large datasets is time consuming, it is desirable to use it to measure runtime overheads of various private algorithms.

Parameters. For MNIST we vary ε in the set $\{0.1, 0.2, 0.5, 1, 2, 4\}$. For Protein and Covertype, we vary ε in the set $\{0.01, 0.02, 0.05, 0.1, 0.2, 0.4\}$ (as they are binary classification and we do not need to divide by 10). δ is set to be $1/m$ where m is the size of the training set size. We vary mini-batch sizes in $\{1, 10, 50\}$, and number of passes in $\{1, 10, 20\}$. The L_2-regularization parameter is 0.0001. Finally, Table 3 summarizes step sizes for different algorithms and tests.

	Non-private	Ours	SCS13	BST14
C + ε-DP	Constant	Constant	$\frac{1}{\sqrt{t}}$	×
C + (ε, δ)-DP	Constant	Constant	$\frac{1}{\sqrt{t}}$	Alg. 3
SC + ε-DP	$\frac{1}{\gamma t}$	$\min(\frac{1}{\beta}, \frac{1}{\gamma t})$	$\frac{1}{\sqrt{t}}$	×
SC + (ε, δ)-DP	$\frac{1}{\gamma t}$	$\min(\frac{1}{\beta}, \frac{1}{\gamma t})$	$\frac{1}{\sqrt{t}}$	Alg. 4

Table 3: Step Sizes for different test scenarios and algorithms. C: Convex, SC: Strongly Convex, DP: differential privacy. For SCS13 and strongly convex case we choose step size $1/\sqrt{t}$ because results in[25] suggest that this step size yields smaller variance with small $\lambda = 0.0001$. In our experiments, these two step sizes yield very similar accuracy.

Experimental Environment. All the experiments were run on a machine with Intel Xeon E5-2680 2.50GHz CPUs (48-core) and 64GB RAM running Ubuntu 14.04.4.

4.4 Accuracy and Effects of Parameters

We now present results on test accuracy and analyze the effects of parameters. Due to lack of space, we only report partial results for representative parameter settings.

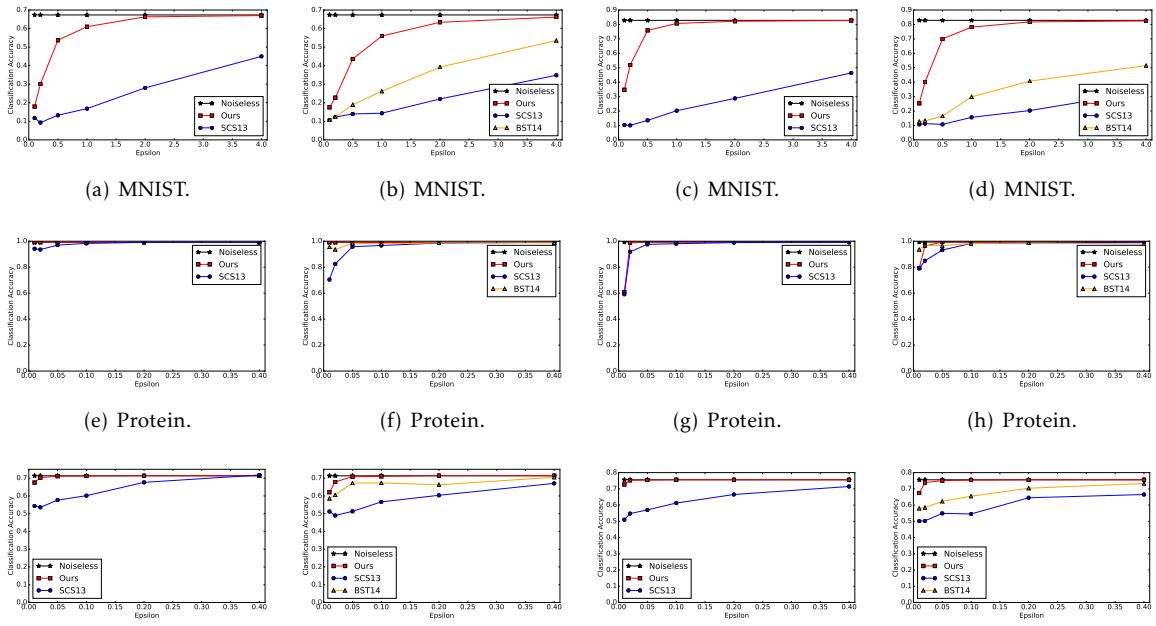
Test Accuracy. Figure 2 gives the test accuracy results of MNIST, Protein and Covertype for all 4 test scenarios with mini-batch size 50 and 10 passes over the data. *For all the four tests we see that our algorithms give significantly better accuracy, up to 4x better than SCS13 and up to 3.5x better than BST14.*

SCS13 and BST14 exhibit much better accuracy on Protein than on MNIST, since logistic regression fits well to the problem. Specifically, BST14 has very close accuracy as our algorithms, though our algorithms still consistently outperform BST14. The accuracy of SCS13 decreases significantly with smaller ε.

For Covertype, even on this large dataset, SCS13 and BST14 give much worse accuracy compared to ours. The accuracy of our algorithms is close to the baseline at around $\varepsilon = 0.05$. The accuracy of SCS13 and BST14 slowly improves with more passes over the data. Specifically, the accuracy of BST14 approaches the baseline only after $\varepsilon = 0.4$.

Finally, we observe that in some cases our private algorithms actually yield better test accuracy than the baseline. This is not surprising because test accuracy amounts to *generalization risk*, and it is known that deliberately introducing small noise can better stabilize the output hypothesis and helps generalization [23].

Number of Passes (Epochs). In the case of convex optimization, more passes through the data indicates larger noise magnitude according to our theory results. This translates empirically to worse test accuracy as we perform more passes. Figure 3 (a) reports test accuracy in the convex case as we run our algorithm 1 pass, 10 passes and 20 passes through the MNIST data. The accuracy drops from 0.71 to 0.45 for $\varepsilon = 4.0$. One should contrast this with results reported in Figure 3 (b) where doing more passes actually *improves*



(a) MNIST. (b) MNIST. (c) MNIST. (d) MNIST.

(e) Protein. (f) Protein. (g) Protein. (h) Protein.

(i) Forest Covertype. (j) Forest Covertype. (k) Forest Covertype. (l) Forest Covertype.

Figure 2: **Test accuracy** for all 4 tests with mini-batch size 50 and 10 passes. ε is varied in $\{0.1, 0.2, 0.5, 1.0, 2.0, 4.0\}$, and $\delta = 1/m$. For Test 3 and Test 4, the L_2-regularization parameter $\lambda = 0.0001$. For constrained optimization (SCS13 in the regularized case), the radius is set to $1/\lambda$, otherwise we report results of unconstrained optimization. Each row gives the results of 4 tests, where Test 1 is Convex, $(\varepsilon, 0)$-DP, Test 2 is Convex, (ε, δ)-DP, Test 3 is Strongly Convex, $(\varepsilon, 0)$-DP, and Test 4 is Strongly Convex, (ε, δ)-DP. For Test 1 and 3, we compare Noiseless, our algorithm and SCS13. For Test 2 and 4, we compare all four algorithms.

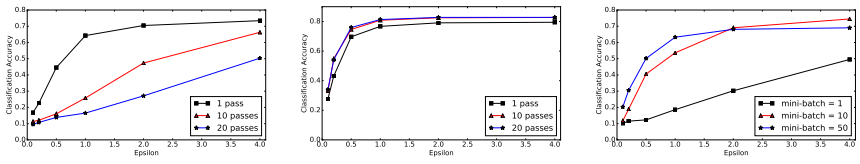
the test accuracy. This is because in the strongly convex case more passes will not introduce more noise for privacy while it can potentially improve the convergence.

Mini-batch Sizes. We find that slightly enlarging the mini-batch size can effectively reduce the noise and thus allow the private algorithm to run more passes in the convex setting. This is useful since it is very common in practice to adopt a mini-batch size at around 10 to 50. To illustrate the effect of mini-batch size we consider the same test as we did above for measuring the effect of number of passes: We run Test 1 with 20 passes through the data, but vary mini-batch sizes in $\{1, 10, 50\}$. Figure 3 (c) reports the test accuracy for this experiment: As soon as we increase mini-batch size to 10 the test accuracy already improves drastically from 0.45 to 0.71.

4.5 Runtime Overhead

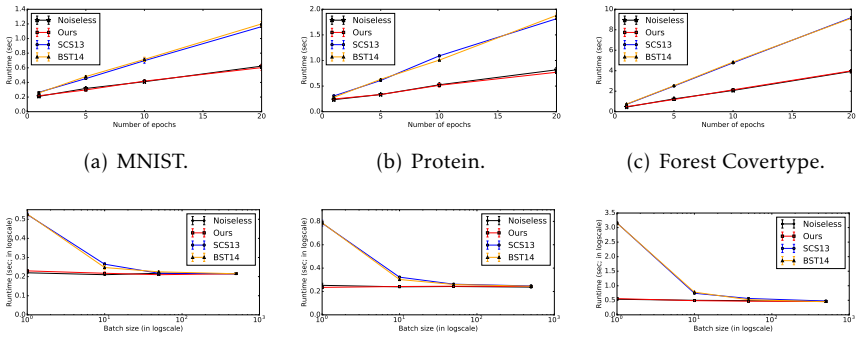
We now compare the runtime overheads of our private SGD algorithm integrated into Bismarck against the regular, noiseless version, as well as the other algorithms. Unlike the accuracy experiments, the key parameters that affect runtimes are the number of epochs and the batch sizes. Thus, we vary each of these parameters, while fixing the others. The runtimes are the average of 4 warm-cache runs and all datasets fit in the buffer cache of PostgreSQL. The error bars represent 90% confidence intervals. The results are plotted in Figure 4 (a)–(c) and Figure 4 (d)–(f) (only the results of strongly convex, (ε, δ)-differential privacy are reported; the other results are similar and thus, we skip them here for brevity).

The first observation is that our algorithm incurs virtually no runtime overhead over noiseless Bismarck,



(a) Convex, Test 1 (b) Strongly Convex, Test 3 (c) Convex, Test 1

Figure 3: **(a), (b) The effect of number of passes**: We report the results on MNIST dataset. We contrast Test 1 (Convex ε-DP) using mini-batch size 1, with Test 3 (Strongly Convex ε-DP) using mini-batch size 50. In the former case, more passes through the data introduces more noise due to privacy and thus results in worse test accuracy. In the latter case, more passes improves the test accuracy as it helps convergence while no more noise is needed for privacy. **(c) The effect of mini-batch size**. We run again Test 1 (Convex, ε-DP) with 20 passes through the data, and vary mini-batch size in $\{1, 10, 50\}$. We note that as soon as we increase mini-batch size to 10 the test accuracy improves drastically from 0.45 to 0.71.



(a) MNIST. (b) Protein. (c) Forest Covertype.

(d) MNIST. (e) Protein. (f) Forest Covertype.

Figure 4: **Runtimes of the implementations on** Bismarck**. (a), (b), (c)**: Vary the number of epochs with mini-batch size = 10. **(d), (e), (f)**: Vary the mini-batch size for a single epoch. Only the results of Strongly Convex, (ε, δ)-DP are reported, and other settings have very similar trends. Noiseless is the regular mini-batch SGD in Bismarck. We fix $\varepsilon = 0.1$.

which is as expected because our algorithm only adds noise once at the end of all epochs. In contrast, both SCS13 and BST14 incur significant runtime overheads in all settings and datasets. In terms of runtime performance for 20 epochs and a batch size of 10, both SCS13 and BST14 are between 2x and 3x slower than our algorithm. The gap grows larger as the batch size is reduced: for a batch size of 1 and 1 epoch, both SCS13 and BST14 are up to 6x slower than our algorithm. This is expected since these algorithms invoke expensive random sampling code from sophisticated distributions for each mini-batch. When the batch size is increased to 500, the runtime gap between these algorithms practically disappears as the random sampling code is invoked much less often. Overall, we find that not only is our algorithm easier to integrate with existing popular RDBMSes, it can be significantly faster than the alternative algorithms.

5 Related Work

There has been much prior work on differentially private convex optimization. There are three main styles of algorithms – output perturbation [2, 4, 13, 22], objective perturbation [4, 16] and online algorithms [2, 6, 13, 25]. Output perturbation works by finding the exact convex minimizer and then adding noise to it, while objective perturbation *exactly* solves a randomly perturbed optimization problem. Unfortunately,

the privacy guarantees provided by both styles often assume that the exact convex minimizer can be found, which usually does not hold in practice.

There are also a number of online approaches. [13] provides an online algorithm for strongly convex optimization based on a *proximal algorithm* (see for example Parikh and Boyd [19]), which is more difficult to implement than SGD. They also provide an offline version (Algorithm 6) for the strongly convex case that is similar to our approach. Finally, SGD-style algorithms were provided by [2, 6, 13, 25].

6 Conclusion and Future Work

In-RDBMS and differentially private convex ERM have each received significant attention in the past decade. Unfortunately, little previous work has examined the private ERM problem for in-RDBMS data analytics. This paper takes a step to bridge this gap.

There are many intriguing future directions to pursue. An important direction is to have a better understanding of the convergence behavior of private SGD when we can only afford to do a constant number of passes over the data. BST14 [2] provides a convergence bound for private SGD when $O(m)$ passes are made over the data. SCS13 [25] does not provide a convergence proof; however, the work of Duchi, Jordan and Wainwright [6], which considers *local differential privacy*, a privacy model where data providers do not even trust the data collector, can be used to conclude convergence for SCS13, though at a very slow rate. Finally, while our method converges very well in practice with multiple passes, we can only prove convergence with *one pass*, and the bound is weaker than BST14, which uses m passes. Can we prove convergence bounds of our algorithms for multiple-passes and match the bounds of BST14?

References

[1] Apache Mahout. mahout.apache.org.

[2] R. Bassily, A. Smith, and A. Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *FOCS*, 2014.

[3] S. Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8(3-4):231–357, 2015.

[4] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12:1069–1109, 2011.

[5] K. Chaudhuri and S. A. Vinterbo. A stability-based validation procedure for differentially private machine learning. In *NIPS*, 2013.

[6] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Local privacy and statistical minimax rates. In *FOCS*, 2013.

[7] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.

[8] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[9] X. Feng, A. Kumar, B. Recht, and C. Ré. Towards a unified architecture for in-rdbms analytics. In *SIGMOD*, 2012.

[10] J. Gray et al. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Data Min. Knowl. Discov.*, 1(1):29–53, Jan. 1997.

[11] M. Hardt, B. Recht, and Y. Singer. Train faster, generalize better: Stability of stochastic gradient descent. *ArXiv e-prints*, Sept. 2015.

[12] J. Hellerstein et al. The MADlib Analytics Library or MAD Skills, the SQL. In *VLDB*, 2012.

[13] P. Jain, P. Kothari, and A. Thakurta. Differentially private online learning. In *COLT*, 2012.

[14] P. Jain and A. Thakurta. Differentially private learning with kernels. In *ICML*, 2013.

[15] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, 2013.

[16] D. Kifer, A. D. Smith, and A. Thakurta. Private convex optimization for empirical risk minimization with applications to high-dimensional regression. In *COLT*, 2012.

[17] A. Nemirovsky and D. Yudin. Problem complexity and method efficiency in optimization. 1983.

[18] Y. Nesterov. *Introductory lectures on convex optimization : a basic course*. Applied optimization. Kluwer Academic Publ., 2004.

[19] N. Parikh and S. P. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):127–239, 2014.

[20] B. T. Polyak. *Introduction to optimization*. Optimization Software, 1987.

[21] N. L. Roux, M. W. Schmidt, and F. R. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *NIPS*, 2012.

[22] B. I. P. Rubinstein, P. L. Bartlett, L. Huang, and N. Taft. Learning in a large function space: Privacy-preserving mechanisms for SVM learning. *CoRR*, abs/0911.5708, 2009.

[23] S. Shalev-Shwartz, O. Shamir, N. Srebro, and K. Sridharan. Learnability, stability and uniform convergence. *The Journal of Machine Learning Research*, 11:2635–2670, 2010.

[24] O. Shamir. Without-Replacement Sampling for Stochastic Gradient Methods: Convergence Results and Application to Distributed Optimization. *ArXiv e-prints*, Mar. 2016.

[25] S. Song, K. Chaudhuri, and A. D. Sarwate. Stochastic gradient descent with differentially private updates. In *GlobalSIP*, 2013.

[26] J. Zhang, X. Xiao, Y. Yang, Z. Zhang, and M. Winslett. Privgene: differentially private model fitting using genetic algorithms. In *SIGMOD*, 2013.

[27] J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett. Functional mechanism: Regression analysis under differential privacy. *PVLDB*, 2012.

[28] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, 2003.

[29] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li. Parallelized stochastic gradient descent. In *NIPS*, 2010.

A Proofs

A.1 Proof of Lemma 6

Proof. Let $T = km$, so we have in total T updates. Applying Lemma 3, Growth Recursion Lemma (Lemma 4), and the fact that the gradient operators are 1-expansive, we have:

$$\delta_t \leq \begin{cases} \delta_{t-1} + 2L\eta_t & \text{if } t = i^* + jm, \\ & j = 0, \ldots, k-1 \\ \delta_{t-1} & \text{otherwise.} \end{cases} \tag{7}$$

Unrolling the recursion completes the proof. □

A.2 Proof of Corollary 2

Proof. We have that

$$\sup_{S \sim S'} \sup_{r} \|A(r;S) - A(r;S')\| \leq \frac{4L}{\beta} \left(\sum_{j=0}^{k-1} \frac{1}{m^c + jm + 1} \right).$$

Therefore

$$\frac{4L}{\beta}\left(\sum_{j=0}^{k-1}\frac{1}{m^c+jm+1}\right) = \frac{4L}{\beta}\left(\frac{1}{m^c+1}+\sum_{j=1}^{k-1}\frac{1}{m^c+jm+1}\right)$$
$$\leq \frac{4L}{\beta}\left(\frac{1}{m^c}+\frac{1}{m}\sum_{j=1}^{k-1}\frac{1}{j}\right)$$
$$\leq \frac{4L}{\beta}\left(\frac{1}{m^c}+\frac{\ln k}{m}\right)$$

as desired. □

A.3 Proof of Lemma 7

Proof. Let $T = km$, so we have in total T updates. We have the following recursion

$$\delta_t \leq \begin{cases} (1-\eta\gamma)\delta_{t-1}+2\eta L & \text{if } t = i^*+jm, \\ & j = 0,1,\ldots,k-1 \\ (1-\eta\gamma)\delta_{t-1} & \text{otherwise.} \end{cases} \tag{8}$$

This is because at each pass different gradient update operators are encountered only at position i^* (corresponding to the time step $t = i^*+jm$), and so the two inequalities directly follow from the growth recursion lemma (Lemma 4). Therefore, the contribution of the differing entry in the first pass contributes $2\eta L(1-\eta\gamma)^{T-i^*}$, and generalizing this, the differing entry in the $(j+1)$-th pass $(j = 0,1,\ldots,k-1)$ contributes $2\eta L(1-\eta\gamma)^{T-i^*-jm}$. Summing up gives the first claimed bound.

For sensitivity, we note that for $j = 1,2,\ldots,k$, the j-th pass can only contribute at most $2\eta L\cdot(1-\eta\gamma)^{(k-j)m}$ to δ_T. Summing up gives the desired result. □

A.4 Proof of Lemma 8

Proof. From the Growth Recursion Lemma (Lemma 4) we know that in the γ-strongly convex case, with appropriate step size, in each iteration either we have a contraction of δ_{t-1}, or, we have a contraction of δ_{t-1} plus an additional additive term. In PSGD, in each pass the differing data point will only be encountered once, introducing an additive term δ^*, and is contracted afterwards.

Formally, let T be the number of updates, the differing data point is at location i^*. Let $\rho_t < 1$ be the expansion factor at iteration t. Then the first pass contributes $\delta^*\prod_{t=i^*+1}^{T}\rho_t$ to δ_T, the second pass contributes $\delta^*\prod_{t=i^*+m+1}^{T}\rho_t$ to δ_T. In general pass j contributes $\delta^*\prod_{t=i^*+(j-1)m+1}^{T}\rho_t$ to δ_T.

Let $\iota_j = \delta^*\prod_{t=i^*+(j-1)m+1}$ be the contribution of pass j to δ_T. We now figure out δ^* and ρ_t. Consider ι_1, we consider two cases. If $t \geq \frac{\beta}{\gamma}$, then $\eta_t \leq \frac{1}{\gamma t} \leq \frac{1}{\beta}$, and so G_t is $(1-\eta_t\gamma) = (1-\frac{1}{t})$ expansive. Thus if $i^* \geq \frac{\beta}{\gamma}$ then before i^* the gap is 0 and after i^* we can apply expansiveness such that

$$\frac{2L}{\gamma t}\cdot\prod_{i=t+1}^{km}\left(1-\frac{1}{i}\right) = \frac{2L}{\gamma t}\cdot\prod_{i=t+1}^{km}\frac{i-1}{i} = \frac{2L}{\gamma km},$$

The remaining case is when $i^* \leq \frac{\beta}{\gamma}-1$. In this case we first have 1-expansiveness due to convexity that the step size is bounded by $\frac{1}{\beta} < \frac{2}{\beta}$. Moreover we have $(1-\frac{1}{t})$-expansiveness for G_t when $\frac{\beta}{\gamma} \leq t \leq m$. Thus

$$2L\eta_{i^*}\cdot\prod_{j=\frac{\beta}{\gamma}}^{km}\left(1-\frac{1}{j}\right) \leq \frac{2L\eta_{i^*}\beta/\gamma}{km} = 2L\cdot\frac{1}{\beta}\cdot\frac{\beta}{\gamma km} = \frac{2L}{\gamma km},$$

Therefore $\Delta_1 \leq \frac{2L}{\gamma km}$. Finally, for $j = 2, \ldots, k$,

$$\Delta_j \leq \frac{2L}{\gamma((j-1)m + i^*)} \cdot \prod_{t=(j-1)m+i^*+1}^{km} \frac{t-1}{t} = \frac{2L}{\gamma km}.$$

Summing up gives the desired result. □

A.5 Proof of Theorem 8

Proof. The proof follows exactly the same argument as Theorem 1 of Zinkevich [28], except we change the step size in the final accumulation of errors. □

A.6 Proof of Theorem 10

Proof. The output of the private PSGD algorithm is $\tilde{w} = \bar{w}_T + \kappa$, where κ is distributed according to a Gamma distribution $\Gamma(d, \frac{\Delta_2}{\varepsilon})$. By Lemma 1, $\Delta_2 \leq 2L\eta = \frac{2R}{\sqrt{m}}$. Therefore by Lemma 11, $\mathbb{E}_\kappa[L_S(\tilde{w}) - L_S(\bar{w}_m)] \leq \frac{2dR}{\varepsilon\sqrt{m}}$, where we use the fact that the expectation of the Gamma distribution is $\frac{d\Delta_2}{\varepsilon}$. Summing up gives the bound. □