CS 860: Algorithms for Private Data Analysis Fall 2020

Lecture 13 — Private ML and Stats: Differentially Private ERM

Prof. Gautam Kamath *Scribe: Gautam Kamath*

Let's really get started on differentially private machine learning. But before we get to that, we need to lay some groundwork in the non-private setting.

A Quick Primer on Non-Private Machine Learning

Formulation. At its core, any machine learning problem (private or non-private) can be generically expressed as follows. We have a dataset D of (x, y) pairs, where the x's are the feature vectors, and the y's are the labels. There is a problem specific loss function ℓ, which takes in a parameter vector θ and a datapoint, and outputs a value:

$$\mathcal{L}(\theta, D) = \sum_{i=1}^{n} \ell(\theta, x_i, y_i).$$

If this is too abstract, you can think of $x_i, \theta \in \mathbb{R}^d, y_i \in \mathbb{R}$, and $\ell(\theta, x_i, y_i) = (\langle x_i, \theta \rangle - y_i)^2$ as the squared-loss for the classic problem of linear regression.

At its core, we will focus on the problem of empirical risk minimization (ERM). This fancy term just means that we want to find θ which minimizes $\mathcal{L}(\theta, D)$. The goal will be: given a dataset D of n (x, y) pairs from a universe $\mathcal{X}$, and a closed convex "parameter set" $\mathcal{C}$, minimize $\mathcal{L}(\theta, D)$ over $\theta \in \mathcal{C}$. The expected excess empirical risk of an algorithm is $\mathbf{E}[\mathcal{L}(\hat{\theta}, D) - \mathcal{L}(\theta^*, D)]$, where $\hat{\theta}$ is the parameter the algorithm outputs, and θ^* is the true minimizer of $\mathcal{L}(\theta, D)$ over $\theta \in \mathcal{C}$. The expectation is only over the randomness of the algorithm (i.e., the dataset is considered to be a fixed, deterministic input). There will be a corresponding classifier $f_\theta(x)$ which will be used to assign labels to new points, but since our only focus is empirical risk minimization, we will generally not consider this function. For example, in the linear regression problem, this function is $\langle x, \theta \rangle$.

As stated, this formulation is quite general, and trying to solve it without further restrictions would prove to be a fruitless endeavour. We will impose a few restrictions in order to avoid nastiness which would arise otherwise. In particular, we will assume the diameter of $\mathcal{C}$ is bounded, $\ell(\cdot, x_i, y_i)$ is convex and L-Lipschitz for all $(x_i, y_i) \in \mathcal{X}$.

Terminology. We explain a few terms before we proceed:

- The gradient of a function $\ell(\theta) : \mathbb{R}^d \to \mathbb{R}$ at a point $\tilde{\theta}$ is a vector $\nabla \ell \in \mathbb{R}^d$ where the ith coordinate is $\frac{\partial \ell(\tilde{\theta})}{\partial \theta_i}$.
- We denote the diameter of a set $\mathcal{C}$ as $\|\mathcal{C}\|_2$. This is the maximum distance between any two points in $\mathcal{C}$.
- A function $\ell : \mathcal{C} \to \mathbb{R}$ is convex if for all $x, y \in \mathcal{C}$, we have that for all $t \in [0, 1]$, $f(tx+(1-t)y) \leq tf(x) + (1-t)f(y)$.

- A function $\ell : \mathcal{C} \to \mathbb{R}$ is L-Lipschitz (in ℓ_2-norm) if for all $x, y \in \mathcal{C}$, we have $|\ell(x) - \ell(y)| \leq L\|x - y\|_2$. As an exercise, verify that this implies that $\|\nabla \ell\|_2 \leq L$. Hint: a gradient is just a multivariate derivative, so it might be easier to start with the univariate case.

Loss Functions. There are other restrictions which are common to impose on the loss function, including strong convexity and smoothness (essentially saying that, at all points, the function is lower and upper bounded by a quadratic function at every point). However, details on these are more advanced than required for this lecture, so we will not describe them further here (though we will mention them when required).

Let's see a few examples of common loss functions. In all the following examples, we have $x, \theta \in \mathbb{R}^d$.

1. Linear Regression: We have $y \in \mathbb{R}$, and $\ell(\theta, x, y) = (\langle x, \theta \rangle - y)^2$.
2. Logistic Regression: This is a classification problem, so $y \in \{\pm 1\}$, and $\ell(\theta, x, y) = \log(1 + e^{-y\langle x, \theta \rangle})$.
3. Geometric median: This is an "unsupervised" task, and y is not used in the loss function. We have the loss function $\ell(\theta, x, y) = \|\theta - x\|_2$. This is easily seen to be 1-Lipschitz. Note the related loss function $\ell(\theta, x, y) = \|\theta - x\|_2^2$, which is used for the mean.
4. Support Vector Machines (SVMs): Another classification task, where $y \in \{\pm 1\}$. The loss function here is the hinge loss $\ell(\theta, x, y) = \max(0, 1 - y\langle x, \theta \rangle)$, and it is L-Lipschitz if $\|x\|_2 \leq L$.

Optimization. Our entire focus during today's class is essentially an optimization question. How do we find the parameter $\hat{\theta}$ which minimizes the loss function? The most common and flexible method is gradient descent. We will describe it in more detail in the final part of this lecture, as we discuss how to privatize this method. However, until then, we will simply claim there exists some black-box algorithm which is capable of optimizing convex loss functions non-privately, and describe how they can be used to solve ERM problems privately.

Generalization. The focus of today's lecture is ERM – that is, we wish to find the parameter that minimizes the loss function on a given dataset. However, this is not typically the end goal of actual machine learning algorithms. In the real world, we often have training data generated from some distribution, and we want the learned classifier to perform well on new data generated from the same distribution. It is frequently the case that this will follow if one simply performs ERM on the training data – this phenomenon is known as *generalization*. That said, this is not the focus of this lecture, and we focus only on the ERM problem.

Privacy Considerations. The goal of differentially private machine learning is to output a parameter vector θ which is differentially with respect to the training dataset D. Unfortunately, naively solving an ERM problem may reveal sensitive information. While we have discussed this numerous times in previous lectures, let's use a few more technical examples. Consider the (geometric) median in 1 dimension: if the number of points n is odd, this will exactly output an element of the training dataset, clearly violating this privacy notion. Similar phenomena arise for SVMs: it is well known that the optimal parameter vector θ is entirely determined by the datapoints which are

closest to it (known as the support vectors). Removing one of these points or adding a new point closer than previous support vectors could dramatically shift the parameter vector, again violating differential privacy. Thus we can see that, even from a technical perspective, a naïve approach will not work.

In the following sections, we will investigate a few different methods of privatizing machine learning procedures, which apply noise in different stages of the pipeline: namely, output perturbation, objective perturbation, and gradient perturbation.

We note that another natural method for privatization is *input perturbation*, in which the input dataset is privatized and then by post-processing, we can do whatever non-private operations we want on the result. However, privatizing the input dataset is essentially synonymous with local differential privacy, which would necessitate significantly more data. As such, we do not cover it here.

Output Perturbation

The first approach, output perturbation (introduced in [RBHT12, CMS11]), is perhaps the most obvious and familiar method to guarantee differential privacy. We simply compute the non-private solution to an ERM problem, add appropriately calibrated noise, and release the result. However, as we alluded to in the above discussion on privacy considerations, the non-private solution might be highly sensitive to the input. We will employ *regularization*, a technique commonly used in machine learning in non-private settings as well.

Though we have already seen a few examples where the parameter vector θ is sensitive to the input dataset, we provide one more in the form of polynomial regression. Suppose we have a set of n points $(x_i, y_i) \in \mathbb{R}^2$, which we wish to fit using a polynomial. That is, we consider the square loss function $\ell(\theta, x, y) = \left(\sum_{j=0}^{k} \theta_j x^j - y\right)^2$, and associated predictor is $\sum_{j=0}^{k} \theta_j x^j$. Since we might not know what degree k is needed to appropriately fit the dataset, we could set $k = n$, thus learning the model using a degree-n polynomial. Without further restrictions, this will perfectly interpolate the dataset ("connect the dots") with a rather unnatural and extreme looking curve. This achieves an excess empirical risk of 0, but is associated with a couple of problems. With the motivation of privacy, the solution would clearly differ greatly on neighbouring databases: adding a point would cause the polynomial to reshape significantly to fit this new point. Another issue (if you look at it right, the same issue), and the motivation in classical machine learning, is that this "overfits" to the training dataset and fails to generalize well. Suppose you receive a new point which is not in the training dataset. Unless this happens to lie precisely on the curve, it is likely to be a very poor at predicting the correct label.

The standard way of getting around this is using a method called regularization – we add a term to the loss function which favours parameter vectors θ which are somehow "simple." In other words, we instead try to minimize the function

$$\mathcal{L}(\theta, D) = \sum_{i=1}^{n} \ell(\theta, x_i, y_i) + \lambda N(\theta),$$

where N is a function which depends only on the parameter vector θ and not the datapoint. λ is a parameter which indicates the emphasis on regularization. Setting $\lambda = 0$ corresponds to no

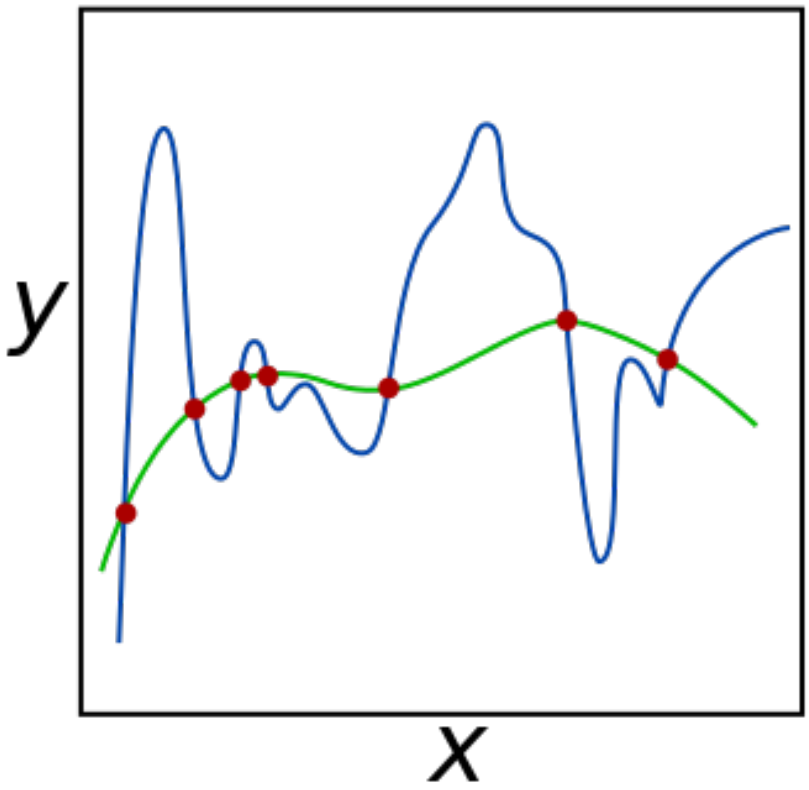

Figure 1: An illustration of regularization in action, by Wikipedia user Nicoguaro. The blue curve is unregularized, and overfits to the given dataset. The green curve is with regularization, and simultaneously seems less sensitive to each individual point, as well as more likely to fit new datapoints from the same distribution.

regularization, while $\lambda = \infty$ corresponds to ignoring the data entirely. In our polynomial regression example, a common choice is to regularize via the ℓ_2-norm of the parameter vector θ, corresponding to $N(\theta) = \sum_{j=0}^{k} \theta_j^2$. Intuitively, this prevents the solution from varying wildly by the addition of a new training example, as θ is held back by the requirement to also have a small ℓ_2-norm. This can be formalized, consider the following lemma in [CMS11].

Lemma 1 (Corollary 8 in [CMS11])**.** *If N is differentiable and 1-strongly convex, and ℓ is convex and 1-Lipschitz, then for all θ, the ℓ_2-sensitivity of* $\arg\min \mathcal{L}(\theta, D)$ *is at most $\frac{2n}{\lambda}$.*

The proof of this is not too hard, exploiting all the given properties of N and ℓ, and we don't cover it here. But with this piece in place, it is easy to complete the argument towards a differentially private algorithm. Simply compute

$$\tilde{\theta} = \arg\min_{\theta \in \mathcal{C}} \mathcal{L}(\theta, D)$$

and output $\hat{\theta} = \tilde{\theta} + b$, where b is an appropriate noise random variable. For instance, $b \sim N\left(0, O\left(\frac{n^2 \log(1/\delta)}{\lambda^2 \varepsilon^2}\right) \cdot I_{d \times d}\right)$ noise suffices for (ε, δ)-DP.

Objective Perturbation

The next approach is qualitatively different from most other approaches we've seen before. In objective perturbation, introduced in [CMS11] and developed in [KST12], we add noise to the objective function which we are optimizing. Consider a regularized objective function of the form

$$\mathcal{L}(\theta, D) = \sum_{i=1}^{n} \ell(\theta, x_i, y_i) + \frac{\lambda}{2}\|\theta\|_2^2.$$

Rather than solving this optimization problem, we instead solve a related problem where the components of the parameter vector θ are penalized or rewarded by a random amount:

$$\mathcal{L}^{\text{priv}}(\theta, D) = \sum_{i=1}^{n} \ell(\theta, x_i, y_i) + \frac{\lambda}{2}\|\theta\|_2^2 + \langle b, \theta \rangle,$$

where b is distributed according to either a Gamma or Gaussian distribution (depending on whether we want pure or approximate differential privacy). Indeed, it can be shown that non-private optimization of $\mathcal{L}^{\text{priv}}$ guarantees the appropriate privacy notion. The proof involves reasoning about the optimization landscape of the perturbed function under neighbouring datasets, and gets a bit technical, so we again omit it here.

One limitation of early works in this area [CMS11, KST12] is that they require the non-private optimization algorithm to find an *exact* minimum for the function in order to guarantee privacy. This highlights their impracticality for a number of reasons. For instance, numerical precision issues on finite computers might result in not finding an exact optimum. While this might seem contrived, attacks like this have been demonstrated for even the basic Laplace mechanism [Mir12]. Another issue is that most optimizers, such as gradient descent, are iterative in nature – they find a solution by taking steps towards an optimum, getting closer to (but never reaching) this point. More recent work has removed this restriction, only requiring that the non-private algorithm finds an *approximate* optimum [BFTT19, INS+19]. With improvements like this, variations of objective perturbation can be shown to be quite practical: [INS+19] gives some of the best empirical results for practical differentially private convex optimization. While this work requires that the loss function is convex to achieve both privacy and accuracy, more recent results have removed the requirement of convexity for proving privacy [NRVW20]. It remains to be seen whether these approaches might prove applicable for modern large scale machine learning settings.

Gradient Perturbation

The current prevailing approach for differentially private machine learning is gradient perturbation, in which we perform a noisy form of gradient descent. This approach was first suggested by [WM10] and later by [SCS13], but it was most developed by Bassily, Smith, and Thakurta [BST14] (whose work we will be covering today). There is more recent work towards making this approach practical and applying it on neural networks, but we will cover that in a future lecture.

Before we talk about noisy gradient descent, we must first talk about regular gradient descent, described in Algorithm 1. The idea is to iteratively take steps in the direction of the gradient of the loss function. By properties of convex functions, this will lead us towards the global minimum,

at which point the gradient will be 0. The function $\eta(t)$ is a "learning rate" (determining the size of the step in the direction of the gradient), and $\Pi_{\mathcal{C}}$ is the projection operator on the set $\mathcal{C}$ (if we take a step to a parameter which is outside the feasible set $\mathcal{C}$, move to the nearest $\theta \in \mathcal{C}$).

Algorithm 1: Projected Gradient Descent

Set $\theta_0 \in \mathcal{C}$ arbitrarily
for $t = 1$ *to* T **do**
  Compute $\theta_t = \Pi_{\mathcal{C}}\left(\theta_{t-1} - \eta(t)\nabla\mathcal{L}(\theta_{t-1}, D)\right)$
end
return $\hat{\theta} = \theta_T$

This algorithm is effective in fairly general settings, we state one such guarantee here. It will be slightly more general than we need for now, but this will come in handy later.

Theorem 2 (Theorem 2 from [SZ13])**.** *Let F be a convex function and let $\theta^* = \arg\min_{\theta\in\mathcal{C}} F(\theta)$. Let θ_0 be an arbitrary point in $\mathcal{C}$, and $\theta_{t+1} = \Pi_{\mathcal{C}}\left(\theta_t - \eta(t)G_t(\theta_t)\right)$, where $\mathbf{E}[G_t(\theta_t)] = \nabla F(\theta_t)$ and $\mathbf{E}[\|G_t(\theta_t)\|_2^2] \leq G^2$, and the learning rate function $\eta(t) = \frac{\|\mathcal{C}\|_2}{G\sqrt{t}}$. Then for any $T > 0$, we have the following:*

$$\mathbf{E}[F(\theta_t) - F(\theta^*)] = O\left(\frac{\|\mathcal{C}\|_2 G \log T}{\sqrt{T}}\right).$$

We can immediately apply this with parameter G equal to n times the Lipschitz constant L. This tells us that if we run Algorithm 1 for T iterations with learning rate function $\eta(t) = \frac{\|\mathcal{C}\|_2}{L\sqrt{t}}$, we achieve an expected excess empirical risk of $\tilde{O}\left(\frac{\|\mathcal{C}\|_2 nL}{\sqrt{T}}\right)$. Note that given enough time (i.e., a large enough T), this quantity can be taken arbitrarily small. In particular, if we take $T \geq \Omega\left(\frac{n^2L^2\|\mathcal{C}\|_2^2}{\alpha^2}\right)$, the expected excess empirical risk is bounded by α, for any value of n.

One downside is running time of this full gradient descent algorithm, which takes $Tn \approx n^3$ gradient calculations. Much more popular in practice is the stochastic gradient descent algorithm, described in Algorithm 2. The main difference is that, instead of computing the exact value of gradient for the entire dataset, we select a random point from the dataset and evaluate the gradient for this one point.

Algorithm 2: Stochastic Projected Gradient Descent

Set $\theta_0 \in \mathcal{C}$ arbitrarily
for $t = 1$ *to* T **do**
  Select $i \in [n]$ uniformly at random
  Compute $\theta_t = \Pi_{\mathcal{C}}\left(\theta_{t-1} - \eta(t)\left(n \cdot \nabla\ell(\theta_{t-1}, x_i, y_i)\right)\right)$
end
return $\hat{\theta} = \theta_T$

This algorithm has exactly the same accuracy guarantees as before. In particular, we apply Theorem 2 with $G_t(\theta_t) = n \cdot \nabla\ell(\theta_t, x_i, y_i)$, where $i \in [n]$ is chosen uniformly at random. This results in $\mathbf{E}[G_t(\theta_t)] = \nabla\mathcal{L}(\theta_t, D)$ and $\mathbf{E}[\|G_t(\theta_t)\|_2^2] \leq n^2L^2$, as before. This gives the exact same error we achieved in the full gradient descent setting: if $T \geq \Omega\left(\frac{n^2L^2\|\mathcal{C}\|_2^2}{\alpha^2}\right)$, the expected excess empirical risk is bounded by α, for any value of n. The main difference is that this only requires $T \approx n^2$ gradient computations, whereas before we needed $\approx n^3$.

Private Stochastic Gradient Descent

Let's now privatize SGD. As Algorithm 2 is so simple, there's really only one place to inject noise into the process: the gradient computation. We compute the gradient, and then add Gaussian noise to privatize it. Since the loss function is Lipschitz, this bounds the magnitude of the gradient and thus the sensitivity. Adding Gaussian noise and applying advanced composition guarantees privacy. The utility analysis is even simpler: we apply Theorem 2 again, with a slightly different expression for the second moment of the estimate of the gradient. We elaborate on both these arguments below.

Algorithm 3: Private Stochastic Projected Gradient Descent

Define $\sigma^2 \leftarrow \frac{32L^2n^2\log(n/\delta)\log(1/\delta)}{\varepsilon^2}$
Set $\theta_0 \in \mathcal{C}$ arbitrarily
for $t = 1$ *to* n^2 **do**
  Select $i \in [n]$ uniformly at random
  Compute $\theta_t = \Pi_{\mathcal{C}}\left(\theta_{t-1} - \eta(t)\left(n \cdot \nabla\ell(\theta_{t-1}, x_i, y_i) + b_{t-1}\right)\right)$, where $b_{t-1} \sim N(0, \sigma^2 \cdot I_{d\times d})$
end
return $\hat{\theta} = \theta_{n^2}$

Utility

Utility is straightforward using Theorem 2. This time, we use $G_t(\theta_t) = n \cdot \nabla\ell(\theta_t, x_i, y_i) + b_t$, where $i \in [n]$ is chosen uniformly at random and $b_{t-1} \sim N(0, \sigma^2 \cdot I_{d\times d})$. Once again, we have that $\mathbf{E}[G_t(\theta_t)] = \nabla\mathcal{L}(\theta_t, D)$ – this is the exact same as the non-private case, but the added noise has mean 0. For the second moment of G_t, we have

$$\mathbf{E}[\|G_t\|_2^2] = n^2\mathbf{E}[\|\nabla\ell(\theta_t, x_i, y_i)\|_2^2] + 2n\mathbf{E}[\langle\nabla\ell(\theta_t, x_i, y_i), b_t\rangle] + \mathbf{E}[\|b_t\|_2^2] \leq n^2L^2 + 0 + d\sigma^2.$$

The first term is bounded exactly as we have done before. The second term is equal to 0, since the gradient and the Gaussian noise are independent of each other and the latter has mean 0. The third term is the sum of the squares of d Gaussians, each with variance σ^2.

Substituting this into the statement of Theorem 2, we get that the expected excess empirical error is bounded as

$$\mathbf{E}[\mathcal{L}(\theta_t, D) - \mathcal{L}(\theta^*, D)] = \tilde{O}\left(\frac{\|\mathcal{C}\|_2\sqrt{n^2L^2 + d\sigma^2}}{\sqrt{n^2}}\right) = \tilde{O}\left(\frac{\|\mathcal{C}\|_2L\sqrt{d\log(1/\delta)}}{\varepsilon}\right).$$

Privacy

The last thing to do is argue that this is private. This will be by a combination of the Gaussian mechanism, amplification by subsampling, and advanced composition. Recall that we can bound the ℓ_2-sensitivity of $n \cdot \nabla\ell(\theta_{t-1}, x_i, y_i)$ by nL. Suppose for the time being that the choice of $i \in [n]$ for each time step t was fixed. Then the Gaussian mechanism would guarantee that for each t, the privacy loss random variable for $G_t(\theta_t)$ is $\leq \frac{\varepsilon}{2\sqrt{\log(1/\delta)}}$ with probability at least $1 - \delta/2$. However, we can strengthen this guarantee, using the inherent randomness in the sampling process: since

we pick a set of size γn with $\gamma = 1/n$, this randomness is amplified by a factor of $n/2$, using the following lemma.

Lemma 3. *Suppose an algorithm is $\varepsilon' < 1$ differentially private. If it is executed on a uniformly random subset of the dataset of size γn, then it will be $2\gamma\varepsilon'$-differentially private.*

Thus, for each iteration, the privacy loss random variable will be bounded by $\frac{\varepsilon}{n\sqrt{\log(1/\delta)}}$ with probability at least $1 - \delta/2$. Using advanced composition for all n^2 iterations, this multiplies the overall privacy loss by (roughly) n, giving (ε, δ)-DP overall.

References

[BFTT19] Raef Bassily, Vitaly Feldman, Kunal Talwar, and Abhradeep Guha Thakurta. Private stochastic convex optimization with optimal rates. In *Advances in Neural Information Processing Systems 32*, NeurIPS '19, pages 11282–11291. Curran Associates, Inc., 2019.

[BST14] Raef Bassily, Adam Smith, and Abhradeep Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '14, pages 464–473, Washington, DC, USA, 2014. IEEE Computer Society.

[CMS11] Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(29):1069–1109, 2011.

[INS+19] Roger Iyengar, Joseph P Near, Dawn Song, Om Thakkar, Abhradeep Thakurta, and Lun Wang. Towards practical differentially private convex optimization. In *Proceedings of the 40th IEEE Symposium on Security and Privacy*, SP '19, pages 299–316, Washington, DC, USA, 2019. IEEE Computer Society.

[KST12] Daniel Kifer, Adam Smith, and Abhradeep Thakurta. Private convex empirical risk minimization and high-dimensional regression. In *Proceedings of the 25th Annual Conference on Learning Theory*, COLT '12, pages 25.1–25.40, 2012.

[Mir12] Ilya Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 650–661, New York, NY, USA, 2012. ACM.

[NRVW20] Seth Neel, Aaron Roth, Giuseppe Vietri, and Zhiwei Steven Wu. Oracle efficient private non-convex optimization. In *Proceedings of the 37th International Conference on Machine Learning*, ICML '20. JMLR, Inc., 2020.

[RBHT12] Benjamin Rubinstein, Peter Bartlett, Ling Huang, and Nina Taft. Learning in a large function space: Privacy-preserving mechanisms for SVM learning. *The Journal of Privacy and Confidentiality*, 4(1):65–100, 2012.

[SCS13] Shuang Song, Kamalika Chaudhuri, and Anand D Sarwate. Stochastic gradient descent with differentially private updates. In *Proceedings of the 2013 IEEE Global Conference on Signal and Information Processing*, GlobalSIP '13, pages 245–248, Washington, DC, USA, 2013. IEEE Computer Society.

[SZ13] Ohad Shamir and Tong Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *Proceedings of the 30th International Conference on Machine Learning*, ICML '13, pages 71–79. JMLR, Inc., 2013.

[WM10] Oliver Williams and Frank McSherry. Probabilistic inference and differential privacy. In *Advances in Neural Information Processing Systems 23*, NIPS '10, pages 2451–2459. Curran Associates, Inc., 2010.