Analysis of Differentially Private Stochastic Gradient Descent Algorithms

George Zhang
CS227r, Harvard University, Spring 2017
https://github.com/gzhang01/CS227FinalProject

As more and more fields adopt machine learning algorithms, data privacy becomes increasingly important. A common method of training these algorithms is gradient descent, which uses the data to minimize some objective function. However, using the data in this way potentially allows an adversary to learn about the dataset itself, thus violating privacy of the data. Differentially private algorithms attempt to produce useful statistics, such as the gradient we need for gradient descent, while providing a guarantee on the privacy of the dataset. Here we look at two differentially private stochastic gradient descent algorithms and compare their utility (as measured by accuracy on a test set) to the utility of the non-private algorithm. We find that for small datasets, the amount of privacy we can guarantee for the same level of accuracy is minimal, and there is a significant utility hit as we increase our privacy guarantee. In addition, we observe that the differentially private algorithms take longer to converge and are more vulnerable to suboptimal termination conditions.

1 Introduction

Gradient descent (GD) is a common method of finding minima of loss functions in machine learning (ML) algorithms. As more and more data become easily accessible, these algorithms are increasingly being implemented in a variety of domains, from autonomous vehicles and artificial "chatbots" to resume screening and medical diagnoses. Some of these applications, such as those in healthcare, use highly sensitive data to train and learn on, revealing a major privacy issue in these algorithms.

The field of differential privacy attempts to address some of these issues. The general purpose of differentially private algorithms is to release aggregate statistics on a dataset without violating the privacy of any individual in the dataset. More specifically, it seeks to provide a guarantee that any particular individual's inclusion in the dataset will not significantly affect the output of the algorithm. This guarantee implies that given an output, it is difficult for an adversary to determine whether a particular individual is in the dataset, and thus the privacy of the individual is preserved [1].

This paper attempts to analyze the utility of such algorithms in comparison to their non-private versions. We shall specifically look at the problem of two-class classification via logistic regression using private stochastic gradient descent (pSGD). We shall look at two specific algorithms for pSGD and examine how these algorithms perform with respect to non-private SGD (npSGD).

1.1 Classification

The premise of classification problems is simple: given a set of elements x and a set of classes y, assign each element $x_i \in x$ to one class $y_i \in y$ such that elements in the same class are more similar to each other than to elements in other classes. Often these elements are represented by a set of features, and based on the scores along each feature, we can assign each element to a point on an d-dimensional feature space. The goal of classification, then, becomes to find hyperplanes that adequately cluster these points.

1.1.1 Logistic Regression

A common approach to supervised two-class classification is logistic regression (LR). In LR, we train on a dataset $x = ((x_1, y_1), \ldots, (x_n, y_n))$, where $x_i \in \mathbb{R}^d$ represents the scores for element x_i on the d-dimensional feature space and $y_i \in \{-1, 1\}$ represents the class of x_i. We wish to find some weight vector $w \in C$ representing the hyperplane that separates the two classes. Our loss function, then, can be given by:

$$\mathcal{L}(w, x) = \frac{1}{n} \sum_{i=1}^{n} \left[\log \left(1 + e^{-y_i (w^T x_i)} \right) \right] + \lambda w^T w$$

Notice that the sign of the expression $w^T x_i$ gives us our estimated class of x_i and that if our estimation is correct, the loss contributed by that element is very small and if our estimation is incorrect, then the loss is large. The optimal w is exactly the argmin of $\mathcal{L}(w, x)$. We also add in a regularizer, which adds a penalty as the weights become large.

1.2 Gradient Descent

Often, finding the exact argmin is difficult and costly. Gradient descent allows us to incrementally find a local minimum of the loss function. The idea behind gradient descent is that the loss function decreases fastest in the direction opposite the gradient. Thus we have the following algorithm[1]:

Algorithm 1: Gradient Descent

Data: dataset x, loss function $\mathcal{L}$, step size η, number iterations t
Result: weight vector $w = \text{argmin}(\mathcal{L})$
$w_0 \leftarrow$ any element of C;
for $i \leftarrow 1$ **to** t **do**
  $w_i \leftarrow \Pi_C[w_{i-1} - \eta_i \cdot \nabla\mathcal{L}(w_{i-1}, x)]$;
end
return w_t

The algorithm as written is commonly known as batch gradient descent, as it calculates the gradient over the entire dataset. However, this is often impractical in most ML applications, as the datasets tend to be very large, and iterating over the entire dataset every time is infeasible. Stochastic gradient descent solves this problem by calculating the gradient for a single datapoint, allowing for any suboptimal updates to be "smoothed" over many iterations. Minibatching combines the two approaches by calculating the gradient for a subsample of the datapoints, thus ensuring a more optimal update at every iteration while avoiding heavy runtime.

1.3 Differential Privacy

Notice that in gradient descent, we are revealing information about each datapoint every time we calculate the gradient, as we utilize all the features of the datapoint itself as well as its class. We avoid this by introducing differentially private algorithms [1]. Given some algorithm $\mathcal{M}$, we can say that $\mathcal{M}$ is (ε, δ)-differentially private if both of the following are true:

$$\Pr[\mathcal{M}(x) = \xi] \leq e^{\varepsilon} \Pr[\mathcal{M}(x') = \xi] + \delta$$

$$\Pr[\mathcal{M}(x') = \xi] \leq e^{\varepsilon} \Pr[\mathcal{M}(x) = \xi] + \delta$$

Here, x and x' are two adjacent datasets, meaning they are identical except for the inclusion of one additional row in x'. Intuitively, this definition says that the probability of getting any particular output from our differentially private algorithm should be similar regardless of whether any particular individual is in the dataset. Thus given an output, it should be difficult to determine which dataset produced it, and thus whether or not any given individual is in the dataset. Since we cannot even determine whether a person is in the dataset, we have effectively guaranteed his privacy.

2 Differentially Private SGD

We now turn our attention to algorithms for differentially private stochastic gradient descent. We shall examine two such algorithms. The first adds noise via the Gaussian mechanism[2]to the gradient at every iteration: [2]

[1] Note that Π_C represents projection back onto C if the update takes us out of the domain of w

Algorithm 2: Private SGD

Data: dataset x, loss function $\mathcal{L}$ (with Lipschitz constant L), ε, δ, step size η
Result: weight vector $w = \text{argmin}(\mathcal{L})$
Set $\sigma \leftarrow c^2 L^2 n^2 \log\left(\frac{n^2}{\delta}\right) \log\left(\frac{1}{\delta}\right) / \varepsilon^2$;
$\widetilde{w}_0 \leftarrow$ any element of C;
for $i \leftarrow 1$ **to** n^2 **do**
  Choose $z \sim_{unif} x$ (with replacement);
  $w_i \leftarrow \Pi_C[w_{i-1} - \eta_i \cdot (\nabla \mathcal{L}(w_{i-1}, z) + b_t)]$ with $b_t \sim \mathcal{N}(0, I_p \sigma^2), p = |w|$;
end
return w_t

The privacy guarantee in this algorithm comes directly from the privacy of the Gaussian mechanism and composition of DP mechanisms (both of which we shall defer to Dwork, Roth [1]). The second algorithm adds noise not to the gradient, but to the objective function itself: [3]

Algorithm 3: Objective Perturbation

Data: dataset x, loss fn $\mathcal{L}$, ε, step size η
Result: weight vector $w = \text{argmin}(\mathcal{L})$
Pick $b \in \mathbb{R}^d$ with density $h(b) \propto e^{-(\varepsilon/2)||b||}$ by choosing direction uniformly and norm of vector from $\Gamma(d, 2/\varepsilon)$;
return $w^* = argmin_w \left(\mathcal{L}(w, x) + \frac{b^T w}{n}\right)$

Notice here that we want to return the argmin of the noisy loss function. We shall do this via normal non-private SGD, with new noise added to the objective function at every iteration. We defer privacy analysis to Shamir, Zhang [3].

3 Implementation

Since differentially private algorithms trade off utility for privacy, a logical question is to ask exactly how much utility we lose for the privacy guarantee we want. Both Bassily et al. and Shamir & Zhang give theoretical analyses, but our goal will be to implement these algorithms and empirically find the utility. All code discussed here can be found in the code/ directory in the Github project.

For ease of visualization, we generated data points in $\mathbb{R}^2$, with the goal of finding a line separating the two classes. However the implementation handles arbitrary dimensions, as can be seen in generate.py. We decided to generate our own data so we could focus on the algorithms themselves rather than feature selection. The three algorithms (npSGD, pSGD, objective perturbation) are each in their own files, and plot.py produces the data we desire. Instructions on running the code can be found in the readme.

We decided to focus on two scenarios, one in which we generate perfectly separated data (non-noisy) and one in which we generate imperfectly separated data (noisy). For each, we looked at the learning process over the first few iterations, the final learned threshold, and the loss during learning. We also subjected the learned threshold to a test dataset to see how accurate our learned thresholds are. Finally, we provide a brief analysis on how privacy affects utility by examining the c^2 term in the noise variance of the private algorithm.

4 Results and Analysis

All collected data can be found in the data/ directory in the Github project, which contains videos of the learning process that we shall refer to below.

[2]For details of the Gaussian mechanism, see Dwork, Roth [1]

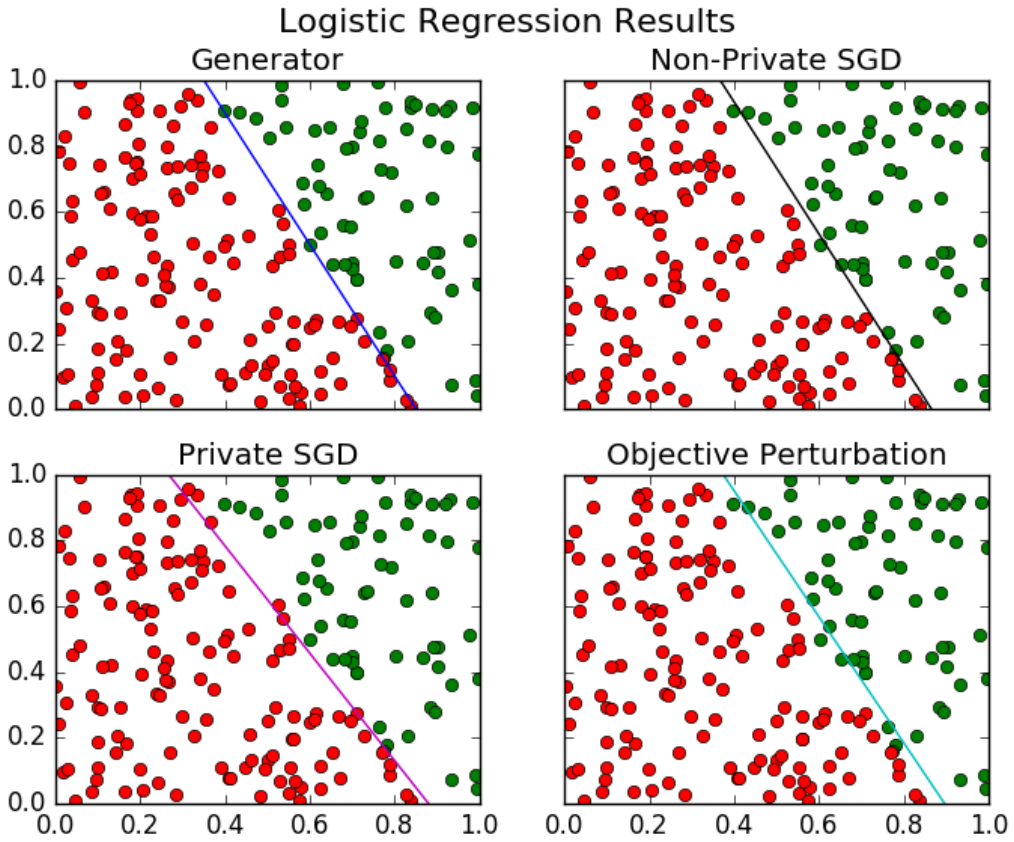


Figure 1: Final Threshold for Non-noisy Data

4.1 Non-noisy Threshold

The final learned thresholds when we generate non-noisy data[3]can be seen in figure 1, and the loss values during the learning process is in figure 2. We see that our final thresholds all resemble the generating threshold, as we would expect. The loss from npSGD and objective perturbation are similar, which is not particularly surprising, since both were generated through normal gradient descent. However, we find that pSGD resulted in much lower loss after 10,000 iterations. This is indeed unexpected, since we hypothesized that increased privacy should result in lower utility. However, we can explain this result by considering the coefficient of the noise variance. Bassily et al. used a constant $c^2 = 32$ in their algorithm [2]. In our tests, however, using this value for c^2 along with our parameters resulted in $\sigma^2 \approx 10^6$, meaning the noise we added completely overwhelmed the gradient we calculated. For our tests, we decreased c^2 to $1/400$, which means we have much more privacy loss, but better utility. We shall explore more about how c^2 affects utility below.

Despite the decreased c^2 parameter, we saw in the learning process[4]that pSGD took much longer to approach the true threshold, and the updates it made were much noisier than either npSGD and objective perturbation. As a result, we suspect the stoppage condition is more important for pSGD than npSGD. In batch gradient descent, we could easily run it to convergence (and the stochastic / mini-batch solution is similar), but the noisy updates of pSGD make determining a convergence point difficult. Stopping when an update noisily shifts the threshold to a particularly suboptimal location would significantly affect the utility of the mechanism.

We also looked at how the final learned thresholds did on a test dataset composed of 5000 newly gen-

[3]Here we used $n = 200$ data points, a learning parameter $\eta = 0.2$, regularization parameter $\lambda = 0.0005$, privacy parameters $\varepsilon = 1, \delta = 0.1$, and a mini-batch size (for npSGD and objective perturbation) of 20. The coefficient of the noise parameter in pSGD was set to $c^2 = 1/400$.

[4]https://github.com/gzhang01/CS227FinalProject/blob/master/data/%5B2%2C1%2C-1'7%5D200c0'0025Vid.mp4

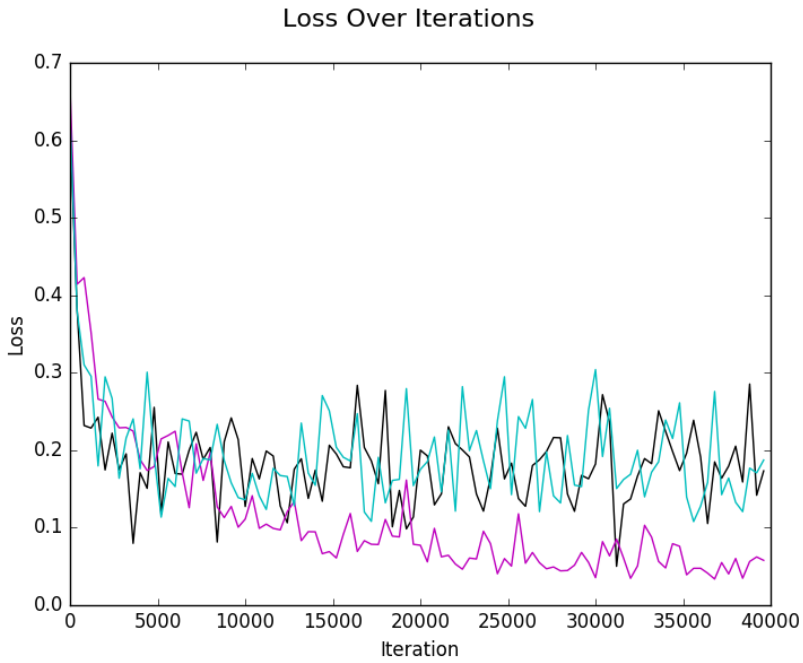


Figure 2: Loss over Iterations for Non-noisy Data

erated points. For each method, we compared our predicted class to the actual class using the same dataset. The results are shown in table 1.

Method	Num Correct	Num Total	Accuracy
npSGD	4787	5000	95.74%
pSGD	4779	5000	95.58%
OP	4777	5000	95.54%

Table 1: Accuracy on Test Set for Non-noisy Data

We see here that the accuracy is comparable for each of the methods. Given what we've discussed above, this should not be terribly surprising. Since we used npSGD to calculate the argmin for the perturbed loss function, we expect the results for those methods to be similar. In addition, our scaled down c^2 gives us a very low privacy guarantee, so we would also expect it to be similar to the non-private version.

4.2 Noisy Threshold

The noisy data was generated based on a sigmoid function with distance from the threshold as its parameter. Thus points closer to the threshold tend to be noisier than points far from the threshold. The final noisy threshold results[5]can be seen in figure 3 and the loss over iterations in figure 4. We again see that npSGD and objective perturbation are similar in their loss functions, but curiously, pSGD has higher loss with the noisy data. In addition, we do not quite see a drop in loss that we saw in the non-noisy version. We suspect this is simply due to the way we calculated the loss. In both versions, we only calculate the loss with respect to the samples in the mini-batch. This means that we are not normalizing over the whole set, and so it is possible that the noise near the threshold is significant enough to skew the loss. We suspect that a larger batch size for loss calculation should still show a decreasing loss over time.

[5]The conditions here are the same as those in the non-noisy version

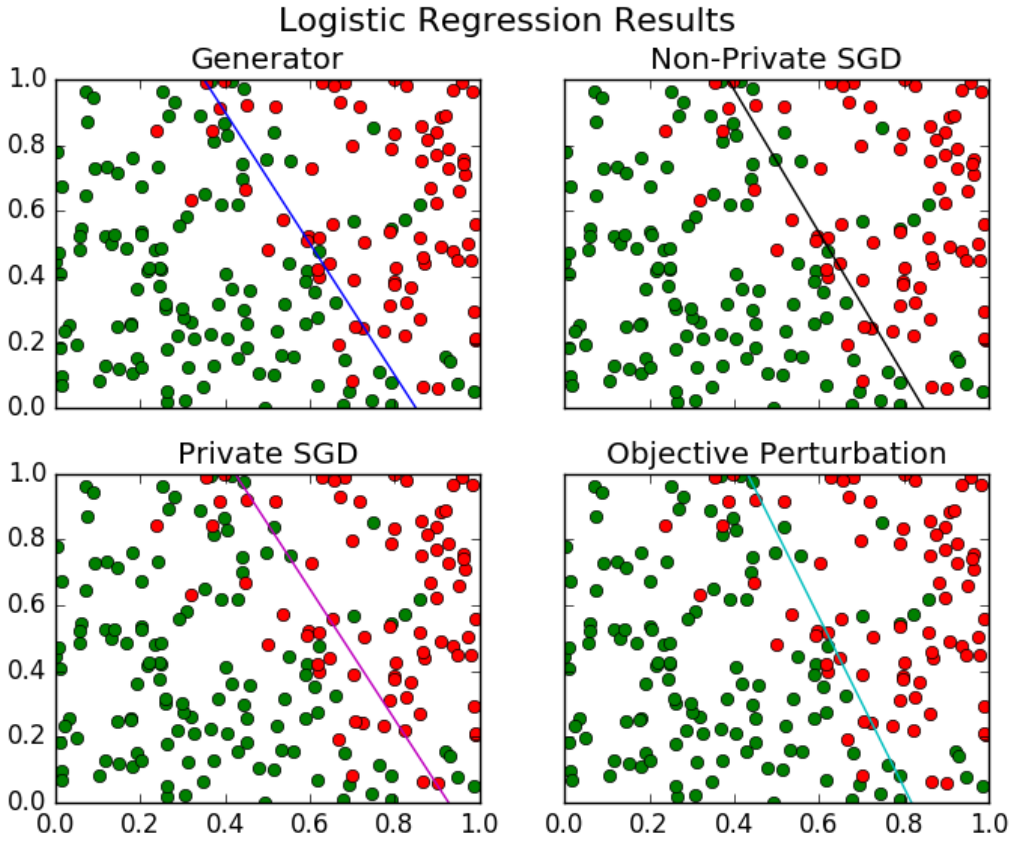


Figure 3: Final Threshold for Noisy Data

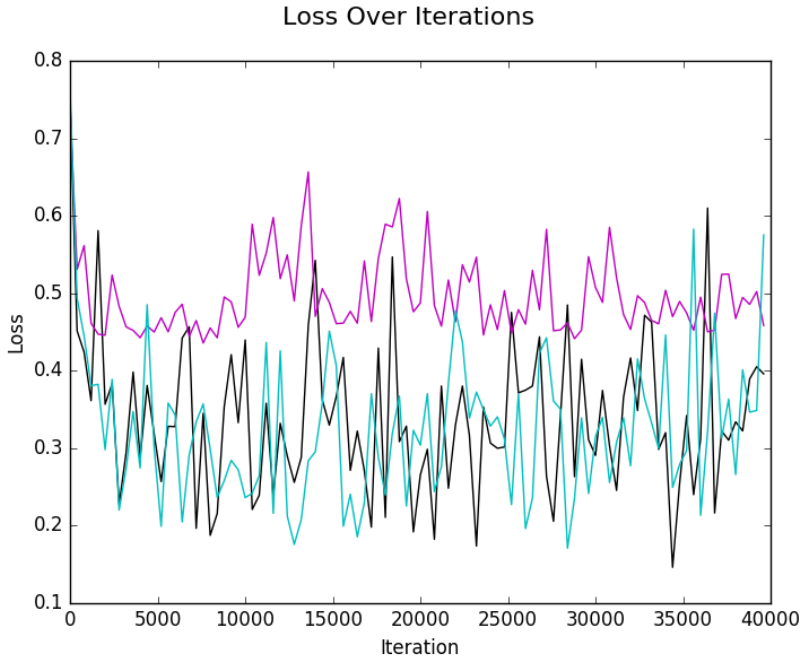


Figure 4: Loss over Iterations for Noisy Data

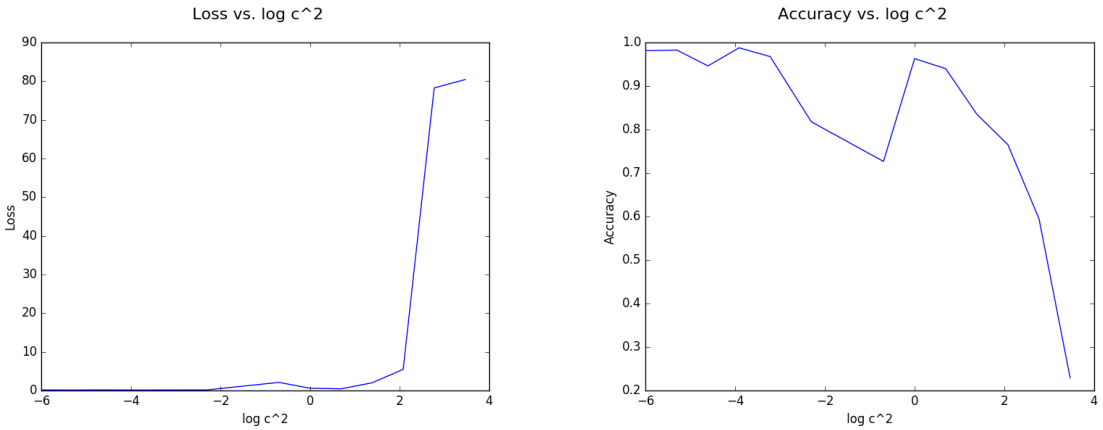
We also tested the final threshold on a test set of 5000 data points. The results are shown in table 2. Note that the test data was also noisy, so the lower accuracy is expected. More importantly, we see that the relative similarity between the private and non-private algorithms remains in the noisy data, suggesting that the algorithms are not significantly affected by noisy data.

Method	Num Correct	Num Total	Accuracy
npSGD	4337	5000	86.74%
pSGD	4289	5000	85.78%
OP	4340	5000	86.80%

Table 2: Accuracy on Test Set for Noisy Data

4.3 Analysis of Utility with Respect to Noise Variance

We noted above that Bassily et al. had a constant of $c^2 = 32$ in their noise variance for pSGD whereas we used a c^2 value of $1/400$. Here we shall look at how accuracy is affected in pSGD when we vary c^2. Note that we could also have analyzed changes to ε and δ, though we expect the trends to be similar. Figure 5 shows the final loss values after learning and the test set accuracy as we vary c^2 (note that the figures plot $\log c^2$ on the x-axis). The raw data for these figures is shown in appendix A.



(a) Final Loss Value

(b) Accuracy on Test Set

Figure 5: Loss and Accuracy vs. $\log c^2$

We can see that as we increase the privacy, our accuracy and loss both suffer. When we use the $c^2 = 32$ that Bassily et al. used, we got a test accuracy of about 20%. Thus there is a significant tradeoff between accuracy and privacy, as we would expect. However, it is important to note that accuracy is not strictly decreasing as we increase our privacy guarantee. This suggests that the stochasticity of the process is important to the final threshold found, which reinforces the idea that suboptimal stoppage conditions can severely affect utility. Here we see that we got about 70-80% accuracy with $c^2 = 0.1$ and $c^2 = 0.5$ but over 90% accuracy with $c^2 = 1$ and $c^2 = 2$. It's unclear whether the former was particularly suboptimal or the latter was particularly optimal, but we can see that convergence is difficult in the private version.

5 Conclusions and Further Work

As more and more of our personal data become available online, data privacy becomes more and more important as well, especially as machine learning algorithms spread into different fields. Whether we are included in train and test sets, these ML algorithms must operate in a way that protects the data it is being trained on. Differentially private algorithms provide a framework for accomplishing such a goal, and we explored two such algorithms for stochastic gradient descent.

We noticed that when we add noise to the gradient, we are more prone to error and are vulnerable to suboptimal termination. The algorithm we used ran for n^2 iterations; however, stopping after a set

number of iterations like this exposes us to potentially terminating after a particularly noisy gradient. This would cause our learned threshold to be suboptimal. Further inquiry should be done on different stoppage conditions to address this issue. Another way we could approach this problem would be to use a time dependent step size, and so noise later in the process has less of an affect on the update. This would allow us to converge as well, though it remains to be seen whether we would converge optimally.

With respect to the objective perturbation method, we added noise based on Shamir & Zhang's algorithm. However, it should be noted that the algorithm itself required us to find the argmin, not necessarily by gradient descent. Since we used GD to calculate this argmin, we have effectively queried the dataset many times, and so we have much more privacy loss (via composition). This suggests that we would have needed to add more noise on every iteration to ensure that our total privacy loss is bounded. Future work may investigate whether this additional noise causes the objective perturbation method to more closely resemble the noisy gradient algorithm, and whether the same convergence issues apply.

Finally, we note that differential privacy was developed to focus on datasets on the scale of the internet. Thus our tests, in which we used 200 data points, is terribly small compared to its intended usage. We expect that increasing the size of the data set will allow us to guarantee more privacy (as the noise will not overwhelm the gradient) as well as lead to better convergence. Given more compute power, we would have liked to see how these algorithms perform on much larger training sets.

References

[1] Dwork Roth. The algorithmic foundations of differential privacy. 2014. Available at: https://www.cis.upenn.edu/~aaroth/Papers/privacybook.pdf.

[2] Bassily Smith Thakurta. Differentially private empirical risk minimization: Efficient algorithms and tight error bounds. 2014. Available at: https://arxiv.org/pdf/1405.7085.pdf.

[3] Shamir Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. 2012. Available at: https://arxiv.org/pdf/1212.1824.pdf.

A Raw Data for c^2 Analysis

The raw data for figure 5 is shown below. Note that loss was calculated at end of learning iterations. Accuracy was taken with respect to a new test set of 5000 data points. Train and test sets were the same between values of c^2.

c^2	Loss	Accuracy
0.0025	0.141270496395	0.9812
0.005	0.117124628915	0.9826
0.01	0.157473363562	0.9462
0.02	0.113144777989	0.9878
0.04	0.136133337182	0.9676
0.1	0.151801335169	0.8178
0.5	2.10054494103	0.7266
1	0.588453341682	0.963
2	0.427375000576	0.9398
4	1.99662027479	0.8352
8	5.46429786396	0.765
16	78.2551784547	0.5948
32	80.4207324954	0.229

Table 3: Raw Data for c^2 Analysis