Generalizable Deep Learning with Differential Privacy: Using Gradient Compression and De-noising

Milad Nasr
University of Massachusetts Amherst
milad@cs.umass.edu

Reza Shokri
National University of Singapore
reza@comp.nus.edu.sg

Amir Houmansadr
University of Massachusetts Amherst
amir@cs.umass.edu

ABSTRACT

Deep learning models leak a significant amount of information about their training datasets. This is a serious privacy concern for the users and data owners. Existing technologies based on differential privacy can preserve privacy, however, with a significant loss in the accuracy of the model. To address this concern, we use gradient compression to minimize privacy leakage and improve the accuracy of differentially private models.

We show that gradient vectors can be compressed in a way that preserves important elements of the gradient but removes unnecessary components, as a result, the compressed vectors become more robust to the added noise by the privacy mechanism. Also, we take advantage of the post-processing property in differential privacy, and introduce the idea of noise correction schemes to extract useful information from the noisy gradients. Finally, we combined these methods and design a differentially private SGD optimizer for deep learning models. We show that our method improves the state-of-the-art utility for the exact same privacy guarantees.

1 INTRODUCTION

Recent advances in training complex (deep) neural network models and the availability of large amounts of datasets have drastically increased the adoption of deep learning in many applications and services. Deep learning models are being trained on various (potentially sensitive) data types including user speeches, images, medical records, financial data, social relationships, and location data points. The confidentiality of such data is crucial to the data owners. Even if the data owners trust the applications, the remaining major concern is if a model's computations (i.e., its predictions or model parameters) can be exploited to endanger the privacy of its sensitive training data.

Many applications use very high dimensional, deep learning models to achieve good performance for various tasks; this high capacity of deep learning models makes them susceptible to memorizing unnecessary details about the training datasets. This can be leveraged to leak unintended information from deep learning models which compromises users privacy. Recent works have investigated several privacy attacks on deep neural networks. Several works have shown that deep nets are vulnerable to membership [3, 8, 9, 11] and model inversion attacks [6]. To protect the privacy of training data, Abadi et al. [2] used the differential privacy framework to provide provable privacy guarantees for deep learning (DPSGD). DPSGD adds noises to model gradients to improve privacy. They also introduced the *moments accountant* method to provide a tight privacy analysis.

Contributions. In this paper, we improve the generalizability of private deep learning models to **gain a better utility** with having the **exact same privacy budget**.

Training a deep neural network model is an iterative procedure; in each step of training, we compute the gradient of model parameters using some training data and update the parameters based on the gradients. While gradients contain useful information about the dataset, they may also contain unintended private information about the training dataset and cause privacy leakage. We need to reduce such private information in the gradients. A common approach to removing such information is to add some calibrated noises to the gradients before updating the parameters. In this work, we suggest creating a sketch from gradient (a compressed version) which contains the general information about the training data but not the detailed information. We show that there are many unnecessary information in the gradient vector that can be removed without harming the accuracy of the model, therefore, reducing the chances of leaking private information. We improve the gradient clipping method by applying a *gradient discretization* mechanism.

To be able to provide formal privacy guarantees, we use Gaussian mechanisms [5] to add noise to the gradients. The noise is sampled from a Gaussian probability distribution. While most samples are concentrated, a large noise can be sampled and such large noise values can drastically demolish the learning performance. To solve this problem, we leverage the post-processing property of differential privacy [5]. The post-processing property allows us to modify the output of a differentially private method without weakening its privacy guarantees. We use post-processing to design a correction scheme which prevents the private gradient values (noisy gradients) to intensely change model parameters. Also, we take advantage of some general information about the gradient patterns to improve the accuracy of gradients. We construct a specific error correction scheme that uses hypothesis testing to improve the accuracy of gradient updates, and increase the overall performance of private learning. Finally, we design a customized stochastic gradient descent (SGD) optimizer that uses compression and error correction to improve the generalizability of differentially private deep learning models.

2 BACKGROUND

In this section, we overview basics of the differential privacy and deep learning.

2.1 Differential Privacy

Differential privacy [4, 5] introduced a strong definition for data privacy. Differential privacy designed to protect the output of aggregation mechanisms on databases. Formally, we define:

Definition 1. *A randomized mechanism $\mathcal{M}$ with domain $\mathcal{D}$ and range $\mathcal{R}$ is $(\epsilon, \delta)-$differential privacy for any two adjacent inputs*

$d, d' \in \mathcal{D}$ and for any subset $S \subseteq \mathcal{R}$, we have:

$$\Pr[\mathcal{M}(d) \in S] \leq e^{\epsilon} \Pr[\mathcal{M}(d') \in S] + \delta \quad (1)$$

We used a relaxed version of the original differential privacy introduced by Dwork et al. [5]. This let violation of $\epsilon-$differential privacy with probability of δ. We use a common approach to provide privacy guarantee which approximates any function $f(.)$ with a differential private mechanism by adding noise proportional to the function's sensitivity, Δf. In particular, we use Gaussian mechanism:

$$\mathcal{M}(d) = f(d) + \mathcal{N}(0, \Delta_2 f \sigma^2) \quad (2)$$

Gaussian mechanism on function f with $l2$-sensitivity of $\Delta_2 f$ is $(\epsilon, \delta)-$differential privacy if $\delta \geq \frac{4}{5} e^{\frac{-(\delta\epsilon)^2}{2}}$ and $\epsilon \in (0, 1)$ [5].

2.2 Deep Learning

Deep neural networks are showing impressive results on many machine learning tasks. To train a complex neural network we define a loss function $\mathbf{L}$ which is typically non-convex. We use mini-batch stochastic gradient descent (SGD) algorithm to optimize the network model parameters and reduces the loss. SGD is an iterative algorithm which selects a set of training data instances then approximate the model gradient by computing the average gradient of data points in the selected mini-batch. Then, update the model parameters in the opposite direction of the parameters' gradient.

3 DIFFERENTIALLY PRIVATE DISCRETE SGD

A differentially private mechanism consists of two steps. First, it needs to compute the influence (sensitivity) of each individual data point on the results, and then add some noise proportional to it [5]. As mentioned earlier, SGD is being used to train a deep learning model, which for each training data instance it computes the model's loss gradients w.r.t. model parameters and modifies the parameters accordingly. Therefore, the influence of each data instance in iterations of training can be computed using the gradient's size (∇W). In deep learning, there is no prior information on the size of gradients, and finding a general bound on gradient sizes is currently a challenge. Instead, Abadi et al. [2] suggested to limit the influence of individual instances by clipping the gradient norm and compute overall privacy cost of training a model using this approach.

In this work, we take a closer look at the clipping mechanism used in DPSGD [2]. As shown in Section 5.2, gradient values are concentrated around zero with a long tail. Therefore, the gradient norm will be impacted drastically by the long trail gradients. By naively clipping the gradient norm, the gradient values will be reduced to very small values. To make the gradients private we add some *calibrated noise* (based on sensitivity) to these values. However, the clipped gradient values are clamped and the added noise makes them indistinguishable.

Algorithm 1 presents the general steps of our approach. At each iteration of the algorithm, a subset (a batch) of random instances with size n from the training dataset will be selected (selection should be done with replacement, therefore, each instance has a probability $q = \frac{n}{N}$ to appear in each training batch). Then, we randomly divide them into smaller groups (μ-batches) with size η, and compute the gradient ∇L_θ for each group.

Algorithm 1 Differentially Private Discrete SGD

Require: learning rate η, μ-batchsize μ, batchsize n, noise scale σ, gradient threshold τ, gradient norm clip C

Initiate θ randomly
for $t \in \{T\}$ **do**
  $B_t \leftarrow$ Sample n instances from dataset randomly
  $\nabla_\theta^G \leftarrow \vec{0}$
  for all μ-batch $\in B_t$ **do**
    Compute gradient of micro-batch ∇_θ^μ
    $\nabla_\theta^\mu \leftarrow$Discretize gradients using Algorithm 2 given τ and ∇_θ^μ
    Clip gradient $\widetilde{\nabla_\theta^\mu} \leftarrow \nabla_\theta^\mu \times C/max(C, ||\nabla_\theta^\mu||_2)$
    $\nabla_\theta^G \leftarrow \nabla_\theta^G + \widetilde{\nabla_\theta^\mu}$
  $\widetilde{\nabla_\theta^G} \leftarrow \nabla_\theta^G + \mathcal{N}(0, \sigma^2 C^2 \mathbb{I})$
  $\widehat{\nabla_\theta^G} \leftarrow$Error correction on $\widetilde{\nabla_\theta^G}$
  $\theta \leftarrow \theta - \eta \widehat{\nabla_\theta^G}$
return output θ and privacy cost

Algorithm 2 Gradient Discretization

$\tau \leftarrow$ threshold
$\nabla \leftarrow$ parameters' gradients
for all v in ∇ **do**
  if $v > \tau$ **then**
    $v = 1$ {UP}
  else if $v < -\tau$ **then**
    $v = -1$ {DOWN}
  else
    $v = 0$ {STOP}
return ∇

Instead of using exact gradient values, we suggest to discretize the gradient values by a discretization scheme. In the next step, the discretized gradient of each group will be clipped to make the l_2 norm of each gradient vector equal to C. Now, we add up gradients of groups to compute an estimate of batch gradients. Since we are limiting the sensitivity of each data instance by clipping the $l2$-norm, we use Gaussian noise to make the aggregated gradients private. Finally, we design a correction scheme to improve the accuracy of private gradients. We apply the noise correction scheme (which only works on the private version of the gradients) and update model parameters.

In the rest of this section, we introduce our discretization scheme and error correction mechanisms.

3.1 Gradient Discretization

To improve over DPSGD's gradient norm clipping, we introduce the idea of *gradient discretization*. As explained earlier, to make the gradients private, the private learning algorithm adds some noises to gradients after clipping them. For a reasonable value of ϵ the added noise is very large compared to gradient values which

makes close gradient values indistinguishable. In other words, by having gradient value g_i and $g_i + \epsilon$ we only pay an unnecessary cost (contribution of ϵ to the gradient norm) and we cannot distinguish between g_i and $g_i + \epsilon$. Instead, we discretize the gradient and only allow the gradients to have a limited set of values, therefore, using this discretization we map gradient values to more distinguishable values. We designed our discretization method such that it does not increase gradient norms.

Specifically, we use the extreme case where we only have three possible values for the gradients: Up (+1), Down (-1), or Stop (0). We used Algorithm 2 to this aim. We see that using this approach does not impact utility drastically.

3.2 Error Correction

The benefit of using an error correction scheme is that, as long as the scheme only works on the noisy (private) version of the gradients (and maybe some prior knowledge) it does not use a privacy budget.

Lemma 1. *Given aggregated gradients which are $(\epsilon, \delta)-$ differentially private, applying an error correction scheme does not have any additional privacy cost and the result will be $(\epsilon, \delta)-$differentially private.*

Proof. Using the post-processing property of differential privacy we can easily see the error correction schemes which only uses private version data will not reduce privacy guarantee. □

In the rest of this section, we explain our error correction technique.

Independent Error Correction We model the privacy mechanism as follow:

$$Y = X + Z \tag{3}$$

$$Z \sim \mathcal{N}(0, \sigma^2 C^2 \mathbb{I}) \tag{4}$$

where X is the clipped gradients (not private), Z is the noise, and Y is the private gradient vectors. In this scheme, we have three hypotheses:

$$H_D : X < -\nu \tag{5}$$

$$H_U : X > \nu \tag{6}$$

$$H_S : |X| \leq \nu \tag{7}$$

We can compute the probability of each hypothesis given the observation of the private gradients and also the parameters of the added noise to gradient using the Bayes' theorem as follow:

$$\Pr(H_D|\nu, C, \sigma) = \frac{\Pr(\nu|H_D, C, \sigma)\Pr(H_D)}{\Pr(\nu, C, \sigma)} \tag{8}$$

$$\Pr(H_U|\nu, C, \sigma) = \frac{\Pr(\nu|H_U, C, \sigma)\Pr(H_U)}{\Pr(\nu, C, \sigma)} \tag{9}$$

$$\Pr(H_S|\nu, C, \sigma) = \frac{\Pr(\nu|H_S, C, \sigma)\Pr(H_S)}{\Pr(\nu, C, \sigma)} \tag{10}$$

The denominators in all cases are similar and can be removed. If we have access to some public data, then we can estimate the values of marginal probabilities of each hypotheses using maximum likelihood estimation. In the case of not having any public data, we assign equal values to the marginal probabilities. Also, the probability of seeing ν given a hypothesis can be computed analytically.

Algorithm 3 Independent Noise Correction

Require: Private gradients $\widetilde{\nabla_\theta^G}$, batch treshold ν, noise scale σ, gradient norm clip C

1: **for all** υ in $\widetilde{\nabla_\theta^G}$ **do**
2: **if** $\Pr(H_U|\upsilon, C, \sigma) > \Pr(H_D|\upsilon, C, \sigma)$ and $\Pr(H_S|\upsilon, C, \sigma)$ **then**
3: $\upsilon = \nu$
4: **else if** $\Pr(H_D|\upsilon, C, \sigma) > \Pr(H_U|\upsilon, C, \sigma)$ and $\Pr(H_S|\upsilon, C, \sigma)$ **then**
5: $\upsilon = -\nu$
6: **else**
7: $\upsilon = 0$
8: **return** corrected gradients

Therefore, we can compute the non-normalized probability of each hypothesis.

Finally, we update the gradient values based on the hypothesis with the highest non-normalized probability. Algorithm 3 summarizes this error correction scheme. Please note that the error correction method is used in Line 11 of Algorithm 1.

3.3 Privacy Analysis

Theorem 1. *Given noise magnitude σ and number of iterations T, Algorithm 1 is $(\epsilon, \delta)-$differentially private where:*

$$\epsilon = T(\min_\lambda \frac{\log(\delta) - \alpha(\lambda)}{\lambda}) \tag{11}$$

$$\alpha(\lambda) = \max(\mathbb{E}_{z\sim\mu_0}[(\mu_0(z)/\mu(z))^\lambda] \tag{12}$$

$$, \mathbb{E}_{z\sim\mu}[(\mu(z)/\mu_0(z))^\lambda])$$

$$\mu(z) = (1-q)\mu_0(z) + q\mu_1(z) \tag{13}$$

$$\mu_0(z) = PDF[\mathcal{N}(0, \sigma^2)](z) \tag{14}$$

$$\mu_1(z) = PDF[\mathcal{N}(1, \sigma^2)](z) \tag{15}$$

Proof. First, we limit the sensitivity of the algorithm by clipping $l2$-norm after applying discretization, therefore, line (6) does not have any privacy cost.

In Lemma 1, we show the error correction scheme which only uses the private version of gradients does not have any privacy costs. Hence, we only need to analyze the privacy cost without considering discretization and error correction.

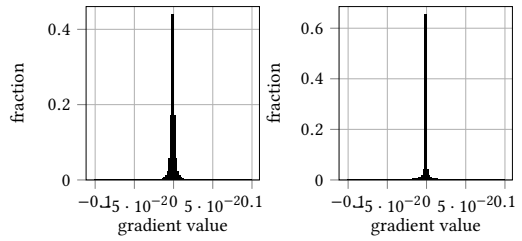
To analyze the privacy guarantees of Algorithm 1, we use momentum accountant method introduced by Abadi et al. [2]. To upper-bound the moments (i.e, $\alpha(\lambda)$) we use binomial expansion (similar to Lemma 3 in [2]). □

4 IMPLEMENTATION

We fully implemented our differentially private discretized SGD algorithm in PyTorch [10] and Tensorflow [1]. Unlike normal SGD, we need to compute the gradients per sample (or per $\mu-$batch). Computing gradients is computationally expensive. Since we are not updating the model parameters in each epoch, we can use parallelization techniques to speed up the process. We implemented a special version of the private optimizer in PyTorch which allows the use of parallel models to compute the gradient of several $\mu-$batches at the same time which we use to speed up the learning process.

5 EXPERIMENTS

In this section, we analyze the performance of our method. Please note that we used random sampling with replacement to create



(a) CIFAR10, max = 0.7 **(b) MNIST, max = 6**

Figure 1: Distribution of parameters' gradients

Table 1: Model accuracy for MNIST dataset, using $\delta = 10^{-5}$, batchsize = 256, $C = 1$ and $\nu = 0.01$ for different parameters and privacy budgets

LR	σ	μ–batch size	T	τ	ϵ	This work Acc	DPSGD [2] Acc
0.15	0.7	1	1.2×10^4	0.001	7.58	97.3 ± 0.1%	97.2 ± 0.1%
0.15	1.0	1	8×10^3	0.001	2.68	96.2 ± 0.1%	95.9 ± 0.1%
0.15	1.1	1	8×10^3	0.001	2.27	96.0 ± 0.1%	95.6 ± 0.2%
0.15	1.3	1	8×10^3	0.001	1.76	95.0 ± 0.1%	94.7 ± 0.1%

our training mini-batches (which is the correct way of creating the training batches).[1]

5.1 Setup

MNIST We perform experiments on the MNIST image dataset which consists of 70,000 instances of 28×28 handwritten grayscale digit images and the task is to recognize the digits. MNIST splits the training data to 60,000 images for training and 10,000 for the testing phase. We used a convolutional neural network with two convolutional layers and two fully connected layers for the MNIST experiments.

CIFAR We also use CIFAR10 to evaluate our method. CIFAR10 is a standard benchmark dataset which is consists of 32×32 RGB images. The learning task is to classify the images into 10 classes of different objects. This dataset partitioned into 50,000 for training and 10,000 testing instances. We used DenseNet [7] model to train our classifier for these experiments.

Error Correction For our error correction scheme (as explained in section 3.2) we did not use prior information about the training data in the experiments and we assumed each hypothesis occurs with equal probability.

Hyper-parameters For both CIFAR10 and MNIST, we used cross-entropy as the loss function. We chose MNIST model's parameters similar to the latest model used by Abadi et al. [2].[2] For CIFAR10, we used DenseNet with depth 40, growth rate 10, and compression rate 2. Other hyper-parameters have been tuned for each experiment.

5.2 Gradient Value Analysis

In Section 3, we talked about the effect of parameters' gradient distribution on clipping. Figure 1 presents the distribution of gradient values for datasets. As we see, values are mostly close to each other, however, the maximum value is far away from the mean which influences the $l2$–norm drastically.

Table 2: Model accuracy for CIFAR10 dataset, using $\delta = 10^{-5}$, $C = 1$ and $\nu = 0.01$ for different parameters and privacy budgets

LR	σ	batch size	μ–batch size	T	τ	ϵ	This work Acc	DPSGD [2] Acc
0.7	0.8	1024	8	6×10^3	0.01	16.20	50.5 ± 0.1%	45.2 ± 0.1%
0.7	1.0	512	4	1×10^4	0.01	7.65	46.8 ± 0.1%	41.0 ± 0.1%
0.7	1.3	512	4	6×10^3	0.01	3.80	38.2 ± 0.1%	33.6 ± 0.1%

5.3 Results

Table 1 summarizes the results of our private optimizer for the MNIST dataset task. We computed the accuracy of the classification task given different privacy budgets. Also, we trained the same model with DPSGD [2] and compared the results. As expected, by increasing the privacy budget, the accuracy of the classification task will increase. Also by comparing our results to DPSGD, we see that our method performs better and gains higher accuracy. We run all of the experiments 10 times and take the average.[3]

We also evaluated our method on CIFAR10 dataset. Unlike Abadi et al [2], we did not use the CIFAR100 dataset as public data and we trained a model only on CIFAR10. Table 2 shows the accuracy of the image classification on CIFAR10 dataset for different privacy budgets. We compared the performance of the DPSGD with the same privacy budget. Our method performs better for similar privacy parameters and noise profiles. Also, similar to the previous experiment, we see that by increasing the privacy budget the classification accuracy improves.

ACKNOWLEDGEMENTS

Milad Nasr is supported by a Google PhD Fellowship in Security and Privacy. This work was supported in part by the NSF grant CNS-1525642, as well as the Singapore Ministry of Education Academic Research Fund Tier 1, R-252-000-660-133. Reza Shokri would like to acknowledge the support of NVIDIA Corporation with the donation of a Titan Xp GPU which was used for this research.

REFERENCES

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283. https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 308–318.

[3] Nicholas Carlini, Chang Liu, Jernej Kos, Úlfar Erlingsson, and Dawn Song. 2018. The secret sharer: Measuring unintended neural network memorization & extracting secrets. *arXiv preprint arXiv:1802.08232* (2018).

[4] Cynthia Dwork. 2011. Differential privacy. *Encyclopedia of Cryptography and Security* (2011), 338–340.

[1] This is the reason why we see some inconsistencies between our results and results available at https://github.com/tensorflow/privacy which uses shuffling to create mini-batches.

[2] Available at https://github.com/tensorflow/privacy

[3] For DPSGD experiments we used hyper-parameters listed at https://github.com/tensorflow/privacy/tree/master/tutorials at the time of writing this paper

[5] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

[6] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1322–1333.

[7] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[8] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. IEEE.

[9] Milad Nasr, Reza Shokri, and Amir Houmansadr. [n.d.]. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In *Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning*. IEEE, 0.

[10] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic Differentiation in PyTorch. In *NIPS Autodiff Workshop*.

[11] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 1310–1321.