Deep Learning with Differential Privacy

Martín Abadi* Andy Chu* Ian Goodfellow†
H. Brendan McMahan* Ilya Mironov* Kunal Talwar*
Li Zhang*

ABSTRACT

Machine learning techniques based on neural networks are achieving remarkable results in a wide variety of domains. Often, the training of models requires large, representative datasets, which may be crowdsourced and contain sensitive information. The models should not expose private information in these datasets. Addressing this goal, we develop new algorithmic techniques for learning and a refined analysis of privacy costs within the framework of differential privacy. Our implementation and experiments demonstrate that we can train deep neural networks with non-convex objectives, under a modest privacy budget, and at a manageable cost in software complexity, training efficiency, and model quality.

1. INTRODUCTION

Recent progress in neural networks has led to impressive successes in a wide range of applications, including image classification, language representation, move selection for Go, and many more (e.g., [55, 30, 57, 40, 15]). These advances are enabled, in part, by the availability of large and representative datasets for training neural networks. These datasets are often crowdsourced, and may contain sensitive information. Their use requires techniques that meet the demands of the applications while offering principled and rigorous privacy guarantees.

In this paper, we combine state-of-the-art machine learning methods with advanced privacy-preserving mechanisms, training neural networks within a modest ("single-digit") privacy budget. We treat models with non-convex objectives, several layers, and tens of thousands to millions of parameters. (In contrast, previous work obtains strong results on convex models with smaller numbers of parameters, or treats complex neural networks but with a large privacy loss.) For this purpose, we develop new algorithmic techniques, a refined analysis of privacy costs within the framework of differential privacy, and careful implementation strategies:

1. We demonstrate that, by tracking detailed information (higher moments) of the privacy loss, we can obtain much tighter estimates on the overall privacy loss, both asymptotically and empirically.
2. We improve the computational efficiency of differentially private training by introducing new techniques. These techniques include efficient algorithms for computing gradients for individual training examples, subdividing tasks into smaller batches to reduce memory footprint, and applying differentially private principal projection at the input layer.
3. We build on the machine learning framework TensorFlow [3] for training models with differential privacy. We evaluate our approach on two standard image classification tasks, MNIST and CIFAR-10. We chose these two tasks because they are based on public datasets and have a long record of serving as benchmarks in machine learning. Our experience indicates that privacy protection for deep neural networks can be achieved at a modest cost in software complexity, training efficiency, and model quality.

Machine learning systems often comprise elements that contribute to protecting their training data. In particular, regularization techniques, which aim to avoid overfitting to the examples used for training, may hide details of those examples. On the other hand, explaining the internal representations in deep neural networks is notoriously difficult, and their large capacity entails that these representations may potentially encode fine details of at least some of the training data. In some cases, a determined adversary may be able to extract parts of the training data. For example, Fredrikson et al. demonstrated a model-inversion attack that recovers images from a facial recognition system [26].

While the model-inversion attack requires only "black-box" access to a trained model (that is, interaction with the model via inputs and outputs), we consider adversaries with additional capabilities, much like Shokri and Shmatikov [52]. Our approach offers protection against a strong adversary with full knowledge of the training mechanism and access to the model's parameters. This protection is attractive, in particular, for applications of machine learning on mobile phones, tablets, and other devices. Storing models on-device enables power-efficient, low-latency inference, and may contribute to privacy since inference does not require communicating user data to a central server; on the other hand, we must assume that the model parameters themselves may be exposed to hostile inspection. Furthermore, when we are

*Google.
†OpenAI. Work done while at Google.

CCS'16 October 24-28, 2016, Vienna, Austria

ACM ISBN 978-1-4503-4139-4/16/10.
DOI: http://dx.doi.org/10.1145/2976749.2978318

concerned with preserving the privacy of one record in the training data, we allow for the possibility that the adversary controls some or even all of the rest of the training data. In practice, this possibility cannot always be excluded, for example when the data is crowdsourced.

The next section reviews background on deep learning and on differential privacy. Sections 3 and 4 explain our approach and implementation. Section 5 describes our experimental results. Section 6 discusses related work, and Section 7 concludes. Deferred proofs appear in the full version of the paper [4].

2. BACKGROUND

In this section we briefly recall the definition of differential privacy, introduce the Gaussian mechanism and composition theorems, and overview basic principles of deep learning.

2.1 Differential Privacy

Differential privacy [21, 18, 22] constitutes a strong standard for privacy guarantees for algorithms on aggregate databases. It is defined in terms of the application-specific concept of adjacent databases. In our experiments, for instance, each training dataset is a set of image-label pairs; we say that two of these sets are adjacent if they differ in a single entry, that is, if one image-label pair is present in one set and absent in the other.

Definition 1. A randomized mechanism $\mathcal{M}\colon \mathcal{D} \to \mathcal{R}$ with domain $\mathcal{D}$ and range $\mathcal{R}$ satisfies (ε, δ)-differential privacy if for any two adjacent inputs $d, d' \in \mathcal{D}$ and for any subset of outputs $S \subseteq \mathcal{R}$ it holds that

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

The original definition of ε-differential privacy does not include the additive term δ. We use the variant introduced by Dwork et al. [19], which allows for the possibility that plain ε-differential privacy is broken with probability δ (which is preferably smaller than $1/|d|$).

Differential privacy has several properties that make it particularly useful in applications such as ours: composability, group privacy, and robustness to auxiliary information. Composability enables modular design of mechanisms: if all the components of a mechanism are differentially private, then so is their composition. Group privacy implies graceful degradation of privacy guarantees if datasets contain correlated inputs, such as the ones contributed by the same individual. Robustness to auxiliary information means that privacy guarantees are not affected by any side information available to the adversary.

A common paradigm for approximating a deterministic real-valued function $f\colon \mathcal{D} \to \mathbb{R}$ with a differentially private mechanism is via additive noise calibrated to f's *sensitivity* S_f, which is defined as the maximum of the absolute distance $|f(d) - f(d')|$ where d and d' are adjacent inputs. (The restriction to a real-valued function is intended to simplify this review, but is not essential.) For instance, the Gaussian noise mechanism is defined by

$$\mathcal{M}(d) \triangleq f(d) + \mathcal{N}(0, S_f^2 \cdot \sigma^2),$$

where $\mathcal{N}(0, S_f^2 \cdot \sigma^2)$ is the normal (Gaussian) distribution with mean 0 and standard deviation $S_f\sigma$. A single application of the Gaussian mechanism to function f of sensitivity S_f satisfies (ε, δ)-differential privacy if $\delta \geq \frac{4}{5}\exp(-(\sigma\varepsilon)^2/2)$ and $\varepsilon < 1$ [22, Theorem 3.22]. Note that this analysis of the mechanism can be applied *post hoc*, and, in particular, that there are infinitely many (ε, δ) pairs that satisfy this condition.

Differential privacy for repeated applications of additive-noise mechanisms follows from the basic composition theorem [19, 20], or from advanced composition theorems and their refinements [24, 34, 23, 11]. The task of keeping track of the accumulated privacy loss in the course of execution of a composite mechanism, and enforcing the applicable privacy policy, can be performed by the *privacy accountant*, introduced by McSherry [42].

The basic blueprint for designing a differentially private additive-noise mechanism that implements a given functionality consists of the following steps: approximating the functionality by a sequential composition of bounded-sensitivity functions; choosing parameters of additive noise; and performing privacy analysis of the resulting mechanism. We follow this approach in Section 3.

2.2 Deep Learning

Deep neural networks, which are remarkably effective for many machine learning tasks, define parameterized functions from inputs to outputs as compositions of many layers of basic building blocks, such as affine transformations and simple nonlinear functions. Commonly used examples of the latter are sigmoids and rectified linear units (ReLUs). By varying parameters of these blocks, we can "train" such a parameterized function with the goal of fitting any given finite set of input/output examples.

More precisely, we define a loss function $\mathcal{L}$ that represents the penalty for mismatching the training data. The loss $\mathcal{L}(\theta)$ on parameters θ is the average of the loss over the training examples $\{x_1, \ldots, x_N\}$, so $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Training consists in finding θ that yields an acceptably small loss, hopefully the smallest loss (though in practice we seldom expect to reach an exact global minimum).

For complex networks, the loss function $\mathcal{L}$ is usually nonconvex and difficult to minimize. In practice, the minimization is often done by the mini-batch stochastic gradient descent (SGD) algorithm. In this algorithm, at each step, one forms a batch B of random examples and computes $\mathbf{g}_B = 1/|B| \sum_{x \in B} \nabla_\theta \mathcal{L}(\theta, x)$ as an estimation to the gradient $\nabla_\theta \mathcal{L}(\theta)$. Then θ is updated following the gradient direction $-\mathbf{g}_B$ towards a local minimum.

Several systems have been built to support the definition of neural networks, to enable efficient training, and then to perform efficient inference (execution for fixed parameters) [31, 13, 3]. We base our work on TensorFlow, an open-source dataflow engine released by Google [3]. TensorFlow allows the programmer to define large computation graphs from basic operators, and to distribute their execution across a heterogeneous distributed system. TensorFlow automates the creation of the computation graphs for gradients; it also makes it easy to batch computation.

3. OUR APPROACH

This section describes the main components of our approach toward differentially private training of neural networks: a differentially private stochastic gradient descent (SGD) algorithm, the moments accountant, and hyperparameter tuning.

3.1 Differentially Private SGD Algorithm

One might attempt to protect the privacy of training data by working only on the final parameters that result from the training process, treating this process as a black box. Unfortunately, in general, one may not have a useful, tight characterization of the dependence of these parameters on the training data; adding overly conservative noise to the parameters, where the noise is selected according to the worst-case analysis, would destroy the utility of the learned model. Therefore, we prefer a more sophisticated approach in which we aim to control the influence of the training data during the training process, specifically in the SGD computation. This approach has been followed in previous works (e.g., [53, 8]); we make several modifications and extensions, in particular in our privacy accounting.

Algorithm 1 outlines our basic method for training a model with parameters θ by minimizing the empirical loss function $\mathcal{L}(\theta)$. At each step of the SGD, we compute the gradient $\nabla_\theta \mathcal{L}(\theta, x_i)$ for a random subset of examples, clip the ℓ_2 norm of each gradient, compute the average, add noise in order to protect privacy, and take a step in the opposite direction of this average noisy gradient. At the end, in addition to outputting the model, we will also need to compute the privacy loss of the mechanism based on the information maintained by the privacy accountant. Next we describe in more detail each component of this algorithm and our refinements.

Algorithm 1 Differentially private SGD (Outline)

Input: Examples $\{x_1, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Parameters: learning rate η_t, noise scale σ, group size L, gradient norm bound C.

Initialize θ_0 randomly

for $t \in [T]$ **do**

Take a random sample L_t with sampling probability L/N

Compute gradient

For each $i \in L_t$, compute $\mathbf{g}_t(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$

Clip gradient

$\bar{\mathbf{g}}_t(x_i) \leftarrow \mathbf{g}_t(x_i) / \max\left(1, \frac{\|\mathbf{g}_t(x_i)\|_2}{C}\right)$

Add noise

$\tilde{\mathbf{g}}_t \leftarrow \frac{1}{L}\sum_i (\bar{\mathbf{g}}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I}))$

Descent

$\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t$

Output θ_T and compute the overall privacy cost (ε, δ) using a privacy accounting method.

Norm clipping: Proving the differential privacy guarantee of Algorithm 1 requires bounding the influence of each individual example on $\tilde{\mathbf{g}}_t$. Since there is no *a priori* bound on the size of the gradients, we *clip* each gradient in ℓ_2 norm; i.e., the gradient vector $\mathbf{g}$ is replaced by $\mathbf{g}/\max\left(1, \frac{\|\mathbf{g}\|_2}{C}\right)$, for a clipping threshold C. This clipping ensures that if $\|\mathbf{g}\|_2 \le C$, then $\mathbf{g}$ is preserved, whereas if $\|\mathbf{g}\|_2 > C$, it gets scaled down to be of norm C. We remark that gradient clipping of this form is a popular ingredient of SGD for deep networks for non-privacy reasons, though in that setting it usually suffices to clip after averaging.

Per-layer and time-dependent parameters: The pseudocode for Algorithm 1 groups all the parameters into a single input θ of the loss function $\mathcal{L}(\cdot)$. For multi-layer neural networks, we consider each layer separately, which allows setting different clipping thresholds C and noise scales σ for different layers. Additionally, the clipping and noise parameters may vary with the number of training steps t. In results presented in Section 5 we use constant settings for C and σ.

Lots: Like the ordinary SGD algorithm, Algorithm 1 estimates the gradient of $\mathcal{L}$ by computing the gradient of the loss on a group of examples and taking the average. This average provides an unbiased estimator, the variance of which decreases quickly with the size of the group. We call such a group a *lot*, to distinguish it from the computational grouping that is commonly called a *batch*. In order to limit memory consumption, we may set the batch size much smaller than the lot size L, which is a parameter of the algorithm. We perform the computation in batches, then group several batches into a lot for adding noise. In practice, for efficiency, the construction of batches and lots is done by randomly permuting the examples and then partitioning them into groups of the appropriate sizes. For ease of analysis, however, we assume that each lot is formed by independently picking each example with probability $q = L/N$, where N is the size of the input dataset.

As is common in the literature, we normalize the running time of a training algorithm by expressing it as the number of *epochs*, where each epoch is the (expected) number of batches required to process N examples. In our notation, an epoch consists of N/L lots.

Privacy accounting: For differentially private SGD, an important issue is computing the overall privacy cost of the training. The composability of differential privacy allows us to implement an "accountant" procedure that computes the privacy cost at each access to the training data, and accumulates this cost as the training progresses. Each step of training typically requires gradients at multiple layers, and the accountant accumulates the cost that corresponds to all of them.

Moments accountant: Much research has been devoted to studying the privacy loss for a particular noise distribution as well as the composition of privacy losses. For the Gaussian noise that we use, if we choose σ in Algorithm 1 to be $\sqrt{2\log\frac{1.25}{\delta}}/\varepsilon$, then by standard arguments [22] each step is (ε, δ)-differentially private with respect to the lot. Since the lot itself is a random sample from the database, the privacy amplification theorem [35, 9] implies that each step is $(q\varepsilon, q\delta)$-differentially private with respect to the full database where $q = L/N$ is the sampling ratio per lot. The result in the literature that yields the best overall bound is the strong composition theorem [24].

However, the strong composition theorem can be loose, and does not take into account the particular noise distribution under consideration. In our work, we invent a stronger accounting method, which we call the moments accountant. It allows us to prove that Algorithm 1 is $(O(q\varepsilon\sqrt{T}), \delta)$-differentially private for appropriately chosen settings of the noise scale and the clipping threshold. Compared to what one would obtain by the strong composition theorem, our bound is tighter in two ways: it saves a $\sqrt{\log(1/\delta)}$ factor in the ε part and a Tq factor in the δ part. Since we expect δ to be small and $T \gg 1/q$ (i.e., each example is examined multiple times), the saving provided by our bound is quite significant. This result is one of our main contributions.

THEOREM 1. *There exist constants c_1 and c_2 so that given the sampling probability $q = L/N$ and the number of steps T, for any $\varepsilon < c_1 q^2 T$, Algorithm 1 is (ε, δ)-differentially private for any $\delta > 0$ if we choose*

$$\sigma \geq c_2 \frac{q\sqrt{T\log(1/\delta)}}{\varepsilon}\,.$$

If we use the strong composition theorem, we will then need to choose $\sigma = \Omega(q\sqrt{T\log(1/\delta)\log(T/\delta)}/\varepsilon)$. Note that we save a factor of $\sqrt{\log(T/\delta)}$ in our asymptotic bound. The moments accountant is beneficial in theory, as this result indicates, and also in practice, as can be seen from Figure 2 in Section 4. For example, with $L = 0.01N$, $\sigma = 4$, $\delta = 10^{-5}$, and $T = 10000$, we have $\varepsilon \approx 1.26$ using the moments accountant. As a comparison, we would get a much larger $\varepsilon \approx 9.34$ using the strong composition theorem.

3.2 The Moments Accountant: Details

The moments accountant keeps track of a bound on the moments of the privacy loss random variable (defined below in Eq. (1)). It generalizes the standard approach of tracking (ε, δ) and using the strong composition theorem. While such an improvement was known previously for composing Gaussian mechanisms, we show that it applies also for composing Gaussian mechanisms with random sampling and can provide much tighter estimate of the privacy loss of Algorithm 1.

Privacy loss is a random variable dependent on the random noise added to the algorithm. That a mechanism $\mathcal{M}$ is (ε, δ)-differentially private is equivalent to a certain tail bound on $\mathcal{M}$'s privacy loss random variable. While the tail bound is very useful information on a distribution, composing directly from it can result in quite loose bounds. We instead compute the log moments of the privacy loss random variable, which compose linearly. We then use the moments bound, together with the standard Markov inequality, to obtain the tail bound, that is the privacy loss in the sense of differential privacy.

More specifically, for neighboring databases $d, d' \in \mathcal{D}^n$, a mechanism $\mathcal{M}$, auxiliary input aux, and an outcome $o \in \mathcal{R}$, define the privacy loss at o as

$$c(o; \mathcal{M}, \mathsf{aux}, d, d') \triangleq \log \frac{\Pr[\mathcal{M}(\mathsf{aux}, d) = o]}{\Pr[\mathcal{M}(\mathsf{aux}, d') = o]}. \qquad (1)$$

A common design pattern, which we use extensively in the paper, is to update the state by sequentially applying differentially private mechanisms. This is an instance of *adaptive composition*, which we model by letting the auxiliary input of the k^{th} mechanism $\mathcal{M}_k$ be the output of all the previous mechanisms.

For a given mechanism $\mathcal{M}$, we define the λ^{th} moment $\alpha_{\mathcal{M}}(\lambda; \mathsf{aux}, d, d')$ as the log of the *moment generating function* evaluated at the value λ:

$$\alpha_{\mathcal{M}}(\lambda; \mathsf{aux}, d, d') \triangleq \log \mathbb{E}_{o\sim\mathcal{M}(\mathsf{aux},d)}[\exp(\lambda c(o; \mathcal{M}, \mathsf{aux}, d, d'))]. \qquad (2)$$

In order to prove privacy guarantees of a mechanism, it is useful to bound all possible $\alpha_{\mathcal{M}}(\lambda; \mathsf{aux}, d, d')$. We define

$$\alpha_{\mathcal{M}}(\lambda) \triangleq \max_{\mathsf{aux}, d, d'} \alpha_{\mathcal{M}}(\lambda; \mathsf{aux}, d, d')\,,$$

where the maximum is taken over all possible aux and all the neighboring databases d, d'.

We state the properties of α that we use for the moments accountant.

THEOREM 2. *Let $\alpha_{\mathcal{M}}(\lambda)$ defined as above. Then*

1. ***[Composability]*** *Suppose that a mechanism $\mathcal{M}$ consists of a sequence of adaptive mechanisms $\mathcal{M}_1, \ldots, \mathcal{M}_k$ where $\mathcal{M}_i\colon \prod_{j=1}^{i-1} \mathcal{R}_j \times \mathcal{D} \to \mathcal{R}_i$. Then, for any λ*

$$\alpha_{\mathcal{M}}(\lambda) \leq \sum_{i=1}^{k} \alpha_{\mathcal{M}_i}(\lambda)\,.$$

2. ***[Tail bound]*** *For any $\varepsilon > 0$, the mechanism $\mathcal{M}$ is (ε, δ)-differentially private for*

$$\delta = \min_{\lambda} \exp(\alpha_{\mathcal{M}}(\lambda) - \lambda\varepsilon)\,.$$

In particular, Theorem 2.1 holds when the mechanisms themselves are chosen based on the (public) output of the previous mechanisms.

By Theorem 2, it suffices to compute, or bound, $\alpha_{\mathcal{M}_i}(\lambda)$ at each step and sum them to bound the moments of the mechanism overall. We can then use the tail bound to convert the moments bound to the (ε, δ)-differential privacy guarantee.

The main challenge that remains is to bound the value $\alpha_{\mathcal{M}_t}(\lambda)$ for each step. In the case of a Gaussian mechanism with random sampling, it suffices to estimate the following moments. Let μ_0 denote the probability density function (pdf) of $\mathcal{N}(0, \sigma^2)$, and μ_1 denote the pdf of $\mathcal{N}(1, \sigma^2)$. Let μ be the mixture of two Gaussians $\mu = (1-q)\mu_0 + q\mu_1$. Then we need to compute $\alpha(\lambda) = \log\max(E_1, E_2)$ where

$$E_1 = \mathbb{E}_{z\sim\mu_0}[(\mu_0(z)/\mu(z))^\lambda]\,, \qquad (3)$$

$$E_2 = \mathbb{E}_{z\sim\mu}\ [(\mu(z)/\mu_0(z))^\lambda]\,. \qquad (4)$$

In the implementation of the moments accountant, we carry out numerical integration to compute $\alpha(\lambda)$. In addition, we can show the asymptotic bound

$$\alpha(\lambda) \leq q^2\lambda(\lambda+1)/(1-q)\sigma^2 + O(q^3/\sigma^3)\,.$$

Together with Theorem 2, the above bound implies our main Theorem 1. The details can be found in the full version of the paper [4].

3.3 Hyperparameter Tuning

We identify characteristics of models relevant for privacy and, specifically, hyperparameters that we can tune in order to balance privacy, accuracy, and performance. In particular, through experiments, we observe that model accuracy is more sensitive to training parameters such as batch size and noise level than to the structure of a neural network.

If we try several settings for the hyperparameters, we can trivially add up the privacy costs of all the settings, possibly via the moments accountant. However, since we care only about the setting that gives us the most accurate model, we can do better, such as applying a version of a result from Gupta et al. [29] (see the full version of the paper for details [4]).

We can use insights from theory to reduce the number of hyperparameter settings that need to be tried. While differentially private optimization of convex objective functions is best achieved using batch sizes as small as 1, non-convex learning, which is inherently less stable, benefits from aggregation into larger batches. At the same time, Theorem 1

suggests that making batches too large increases the privacy cost, and a reasonable tradeoff is to take the number of batches per epoch to be of the same order as the desired number of epochs. The learning rate in non-private training is commonly adjusted downwards carefully as the model converges to a local optimum. In contrast, we never need to decrease the learning rate to a very small value, because differentially private training never reaches a regime where it would be justified. On the other hand, in our experiments, we do find that there is a small benefit to starting with a relatively large learning rate, then linearly decaying it to a smaller value in a few epochs, and keeping it constant afterwards.

4. IMPLEMENTATION

We have implemented the differentially private SGD algorithms in TensorFlow. For privacy protection, we need to "sanitize" the gradient before using it to update the parameters. In addition, we need to keep track of the "privacy spending" based on how the sanitization is done. Hence our implementation mainly consists of two components: `sanitizer`, which preprocesses the gradient to protect privacy, and `privacy_accountant`, which keeps track of the privacy spending over the course of training.

Figure 1 contains the TensorFlow code snippet (in Python) of `DPSGD_Optimizer`, which minimizes a loss function using a differentially private SGD, and `DPTrain`, which iteratively invokes `DPSGD_Optimizer` using a privacy accountant to bound the total privacy loss.

In many cases, the neural network model may benefit from the processing of the input by projecting it on the principal directions (PCA) or by feeding it through a convolutional layer. We implement differentially private PCA and apply pre-trained convolutional layers (learned on public data).

Sanitizer. In order to achieve privacy protection, the sanitizer needs to perform two operations: (1) limit the sensitivity of each individual example by clipping the norm of the gradient for each example; and (2) add noise to the gradient of a batch before updating the network parameters.

In TensorFlow, the gradient computation is batched for performance reasons, yielding $\mathbf{g}_B = 1/|B| \sum_{x \in B} \nabla_\theta \mathcal{L}(\theta, x)$ for a batch B of training examples. To limit the sensitivity of updates, we need to access each individual $\nabla_\theta \mathcal{L}(\theta, x)$. To this end, we implemented `per_example_gradient` operator in TensorFlow, as described by Goodfellow [27]. This operator can compute a batch of individual $\nabla_\theta \mathcal{L}(\theta, x)$. With this implementation there is only a modest slowdown in training, even for larger batch size. Our current implementation supports batched computation for the loss function $\mathcal{L}$, where each x_i is singly connected to $\mathcal{L}$, allowing us to handle most hidden layers but not, for example, convolutional layers.

Once we have the access to the per-example gradient, it is easy to use TensorFlow operators to clip its norm and to add noise.

Privacy accountant. The main component in our implementation is `PrivacyAccountant` which keeps track of privacy spending over the course of training. As discussed in Section 3, we implemented the moments accountant that additively accumulates the log of the moments of the privacy loss at each step. Dependent on the noise distribution, one can compute $\alpha(\lambda)$ by either applying an asymptotic bound, evaluating a closed-form expression, or applying numerical

```
class DPSGD_Optimizer():
  def __init__(self, accountant, sanitizer):
    self._accountant = accountant
    self._sanitizer = sanitizer

  def Minimize(self, loss, params,
               batch_size, noise_options):
    # Accumulate privacy spending before computing
    # and using the gradients.
    priv_accum_op =
        self._accountant.AccumulatePrivacySpending(
            batch_size, noise_options)
    with tf.control_dependencies(priv_accum_op):
      # Compute per example gradients
      px_grads = per_example_gradients(loss, params)
      # Sanitize gradients
      sanitized_grads = self._sanitizer.Sanitize(
          px_grads, noise_options)
      # Take a gradient descent step
      return apply_gradients(params, sanitized_grads)

def DPTrain(loss, params, batch_size, noise_options):
  accountant = PrivacyAccountant()
  sanitizer = Sanitizer()
  dp_opt = DPSGD_Optimizer(accountant, sanitizer)
  sgd_op = dp_opt.Minimize(
      loss, params, batch_size, noise_options)
  eps, delta = (0, 0)
  # Carry out the training as long as the privacy
  # is within the pre-set limit.
  while within_limit(eps, delta):
    sgd_op.run()
    eps, delta = accountant.GetSpentPrivacy()
```

Figure 1: Code snippet of `DPSGD_Optimizer` and `DPTrain`.

integration. The first option would recover the generic advanced composition theorem, and the latter two give a more accurate accounting of the privacy loss.

For the Gaussian mechanism we use, $\alpha(\lambda)$ is defined according to Eqs. (3) and (4). In our implementation, we carry out numerical integration to compute both E_1 and E_2 in those equations. Also we compute $\alpha(\lambda)$ for a range of λ's so we can compute the best possible (ε, δ) values using Theorem 2.2. We find that for the parameters of interest to us, it suffices to compute $\alpha(\lambda)$ for $\lambda \leq 32$.

At any point during training, one can query the privacy loss in the more interpretable notion of (ε, δ) privacy using Theorem 2.2. Rogers et al. [49] point out risks associated with adaptive choice of privacy parameters. We avoid their attacks and negative results by fixing the number of iterations and privacy parameters ahead of time. More general implementations of a privacy accountant must correctly distinguish between two modes of operation—as a privacy odometer or a privacy filter (see [49] for more details).

Differentially private PCA. Principal component analysis (PCA) is a useful method for capturing the main features of the input data. We implement the differentially private PCA algorithm as described in [25]. More specifically, we take a random sample of the training examples, treat them as vectors, and normalize each vector to unit ℓ_2 norm to form the matrix A, where each vector is a row in the matrix. We then add Gaussian noise to the covariance matrix $A^T A$ and compute the principal directions of the noisy covariance matrix. Then for each input example we apply the

projection to these principal directions before feeding it into the neural network.

We incur a privacy cost due to running a PCA. However, we find it useful for both improving the model quality and for reducing the training time, as suggested by our experiments on the MNIST data. See Section 4 for details.

Convolutional layers. Convolutional layers are useful for deep neural networks. However, an efficient per-example gradient computation for convolutional layers remains a challenge within the TensorFlow framework, which motivates creating a separate workflow. For example, some recent work argues that even random convolutions often suffice [48, 14, 51, 56, 16].

Alternatively, we explore the idea of learning convolutional layers on public data, following Jarrett et al. [32]. Such convolutional layers can be based on GoogLeNet or AlexNet features [55, 37] for image models or on pretrained word2vec or GloVe embeddings in language models [43, 46].

5. EXPERIMENTAL RESULTS

This section reports on our evaluation of the moments accountant, and results on two popular image datasets: MNIST and CIFAR-10.

5.1 Applying the Moments Accountant

As shown by Theorem 1, the moments accountant provides a tighter bound on the privacy loss compared to the generic strong composition theorem. Here we compare them using some concrete values. The overall privacy loss (ε, δ) can be computed from the noise level σ, the sampling ratio of each lot $q = L/N$ (so each epoch consists of $1/q$ batches), and the number of epochs E (so the number of steps is $T = E/q$). We fix the target $\delta = 10^{-5}$, the value used for our MNIST and CIFAR experiments.

In our experiment, we set $q = 0.01$, $\sigma = 4$, and $\delta = 10^{-5}$, and compute the value of ε as a function of the training epoch E. Figure 2 shows two curves corresponding to, respectively, using the strong composition theorem and the moments accountant. We can see that we get a much tighter estimation of the privacy loss by using the moments accountant. For examples, when $E = 100$, the values are 9.34 and 1.26 respectively, and for $E = 400$, the values are 24.22 and 2.55 respectively. That is, using the moments bound, we achieve $(2.55, 10^{-5})$-differential privacy, whereas previous techniques only obtain the significantly worse guarantee of $(24.22, 10^{-5})$.

5.2 MNIST

We conduct experiments on the standard MNIST dataset for handwritten digit recognition consisting of 60,000 training examples and 10,000 testing examples [38]. Each example is a 28×28 size gray-level image. We use a simple feedforward neural network with ReLU units and softmax of 10 classes (corresponding to the 10 digits) with cross-entropy loss and an optional PCA input layer.

Baseline model.

Our baseline model uses a 60-dimensional PCA projection layer and a single hidden layer with 1,000 hidden units. Using the lot size of 600, we can reach accuracy of 98.30% in about 100 epochs. This result is consistent with what can be achieved with a vanilla neural network [38].

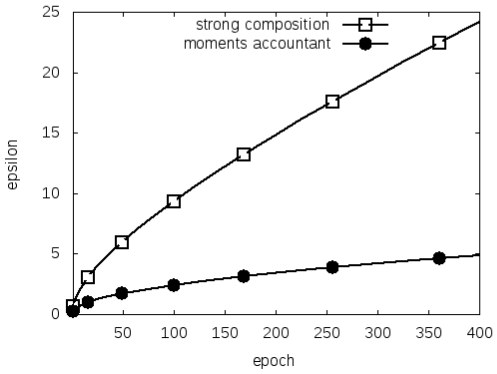


Figure 2: The ε value as a function of epoch E for $q = 0.01$, $\sigma = 4$, $\delta = 10^{-5}$, using the strong composition theorem and the moments accountant respectively.

Differentially private model.

For the differentially private version, we experiment with the same architecture with a 60-dimensional PCA projection layer, a single 1,000-unit ReLU hidden layer, and a lot size of 600. To limit sensitivity, we clip the gradient norm of each layer at 4. We report results for three choices of the noise scale, which we call small ($\sigma = 2, \sigma_p = 4$), medium ($\sigma = 4, \sigma_p = 7$), and large ($\sigma = 8, \sigma_p = 16$). Here σ represents the noise level for training the neural network, and σ_p the noise level for PCA projection. The learning rate is set at 0.1 initially and linearly decreased to 0.052 over 10 epochs and then fixed to 0.052 thereafter. We have also experimented with multi-hidden-layer networks. For MNIST, we found that one hidden layer combined with PCA works better than a two-layer network.

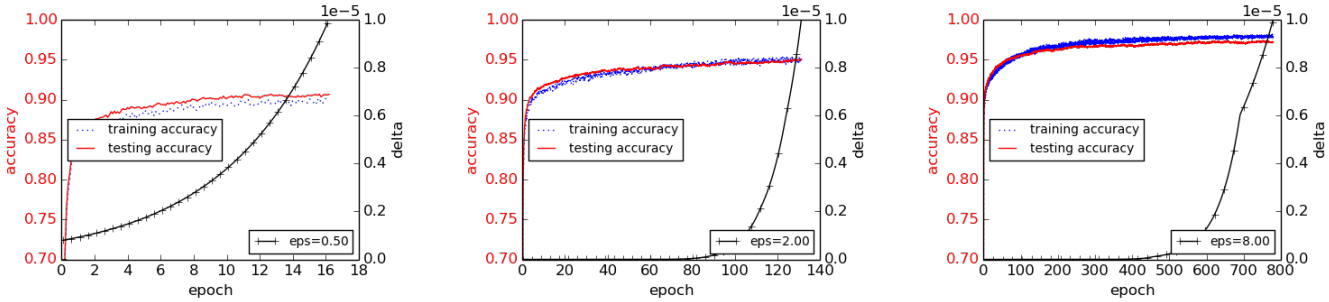
Figure 3 shows the results for different noise levels. In each plot, we show the evolution of the training and testing accuracy as a function of the number of epochs as well as the corresponding δ value, keeping ε fixed. We achieve 90%, 95%, and 97% test set accuracy for $(0.5, 10^{-5})$, $(2, 10^{-5})$, and $(8, 10^{-5})$-differential privacy respectively.

One attractive consequence of applying differentially private SGD is the small difference between the model's accuracy on the training and the test sets, which is consistent with the theoretical argument that differentially private training generalizes well [7]. In contrast, the gap between training and testing accuracy in non-private training, i.e., evidence of overfitting, increases with the number of epochs.

By using the moments accountant, we can obtain a δ value for any given ε. We record the accuracy for different (ε, δ) pairs in Figure 4. In the figure, each curve corresponds to the best accuracy achieved for a fixed δ, as it varies between 10^{-5} and 10^{-2}. For example, we can achieve 90% accuracy for $\varepsilon = 0.25$ and $\delta = 0.01$. As can be observed from the figure, for a fixed δ, varying the value of ε can have large impact on accuracy, but for any fixed ε, there is less difference with different δ values.

Effect of the parameters.

Classification accuracy is determined by multiple factors



(1) Large noise (2) Medium noise (3) Small noise

Figure 3: Results on the accuracy for different noise levels on the MNIST dataset. In all the experiments, the network uses 60 dimension PCA projection, 1,000 hidden units, and is trained using lot size 600 and clipping threshold 4. The noise levels (σ, σ_p) for training the neural network and for PCA projection are set at (8, 16), (4, 7), and (2, 4), respectively, for the three experiments.

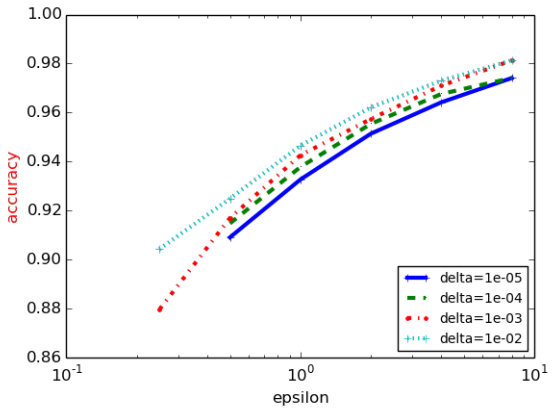


Figure 4: Accuracy of various (ε, δ) privacy values on the MNIST dataset. Each curve corresponds to a different δ value.

that must be carefully tuned for optimal performance. These factors include the topology of the network, the number of PCA dimensions and the number of hidden units, as well as parameters of the training procedure such as the lot size and the learning rate. Some parameters are specific to privacy, such as the gradient norm clipping bound and the noise level.

To demonstrate the effects of these parameters, we manipulate them individually, keeping the rest constant. We set the reference values as follows: 60 PCA dimensions, 1,000 hidden units, 600 lot size, gradient norm bound of 4, initial learning rate of 0.1 decreasing to a final learning rate of 0.052 in 10 epochs, and noise σ equal to 4 and 7 respectively for training the neural network parameters and for the PCA projection. For each combination of values, we train until the point at which $(2, 10^{-5})$-differential privacy would be violated (so, for example, a larger σ allows more epochs of training). The results are presented in Figure 5.

PCA projection. In our experiments, the accuracy is fairly stable as a function of the PCA dimension, with the best results achieved for 60. (Not doing PCA reduces accuracy by about 2%.) Although in principle the PCA projection layer can be replaced by an additional hidden layer, we achieve better accuracy by training the PCA layer separately. By reducing the input size from 784 to 60, PCA leads to an almost 10× reduction in training time. The result is fairly stable over a large range of the noise levels for the PCA projection and consistently better than the accuracy using random projection, which is at about 92.5% and shown as a horizontal line in the plot.

Number of hidden units. Including more hidden units makes it easier to fit the training set. For non-private training, it is often preferable to use more units, as long as we employ techniques to avoid overfitting. However, for differentially private training, it is not a priori clear if more hidden units improve accuracy, as more hidden units increase the sensitivity of the gradient, which leads to more noise added at each update.

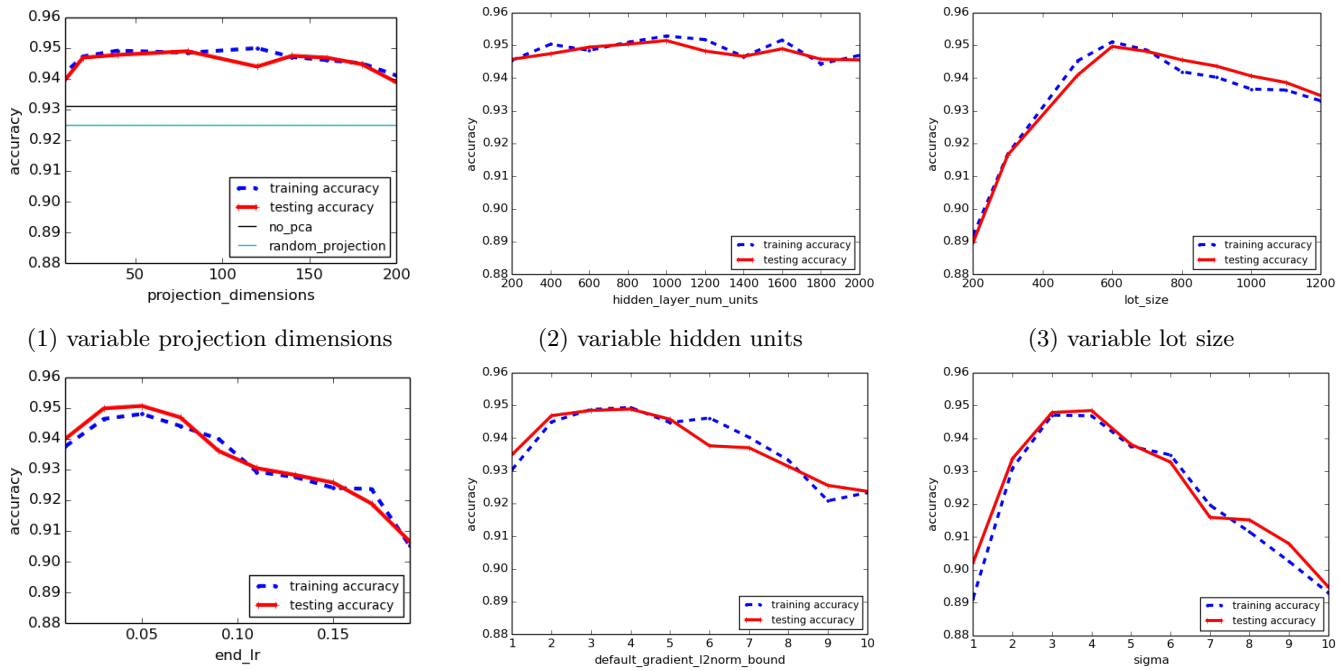
Somewhat counterintuitively, increasing the number of hidden units does not decrease accuracy of the trained model. One possible explanation that calls for further analysis is that larger networks are more tolerant to noise. This property is quite encouraging as it is common in practice to use very large networks.

Lot size. According to Theorem 1, we can run N/L epochs while staying within a constant privacy budget. Choosing the lot size must balance two conflicting objectives. On the one hand, smaller lots allow running more epochs, i.e., passes over data, improving accuracy. On the other hand, for a larger lot, the added noise has a smaller relative effect.

Our experiments show that the lot size has a relatively large impact on accuracy. Empirically, the best lot size is roughly $\sqrt{N}$ where N is the number of training examples.

Learning rate. Accuracy is stable for a learning rate in the range of $[0.01, 0.07]$ and peaks at 0.05, as shown in Figure 5(4). However, accuracy decreases significantly if the learning rate is too large. Some additional experiments suggest that, even for large learning rates, we can reach similar levels of accuracy by reducing the noise level and, accordingly, by training less in order to avoid exhausting the privacy budget.

Clipping bound. Limiting the gradient norm has two opposing effects: clipping destroys the unbiasedness of the gradient estimate, and if the clipping parameter is too small, the average clipped gradient may point in a very different



(1) variable projection dimensions (2) variable hidden units (3) variable lot size

(4) variable learning rate (5) variable gradient clipping norm (6) variable noise level

Figure 5: MNIST accuracy when one parameter varies, and the others are fixed at reference values.

direction from the true gradient. On the other hand, increasing the norm bound C forces us to add more noise to the gradients (and hence the parameters), since we add noise based on σC. In practice, a good way to choose a value for C is by taking the median of the norms of the unclipped gradients over the course of training.

Noise level. By adding more noise, the per-step privacy loss is proportionally smaller, so we can run more epochs within a given cumulative privacy budget. In Figure 5(5), the x-axis is the noise level σ. The choice of this value has a large impact on accuracy.

From the experiments, we observe the following.

1. The PCA projection improves both model accuracy and training performance. Accuracy is quite stable over a large range of choices for the projection dimensions and the noise level used in the PCA stage.
2. The accuracy is fairly stable over the network size. When we can only run smaller number of epochs, it is more beneficial to use a larger network.
3. The training parameters, especially the lot size and the noise scale σ, have a large impact on the model accuracy. They both determine the "noise-to-signal" ratio of the sanitized gradients as well as the number of epochs we are able to go through the data before reaching the privacy limit.

Our framework allows for adaptive control of the training parameters, such as the lot size, the gradient norm bound C, and noise level σ. Our initial experiments with decreasing noise as training progresses did not show a significant improvement, but it is interesting to consider more sophisticated schemes for adaptively choosing these parameters.

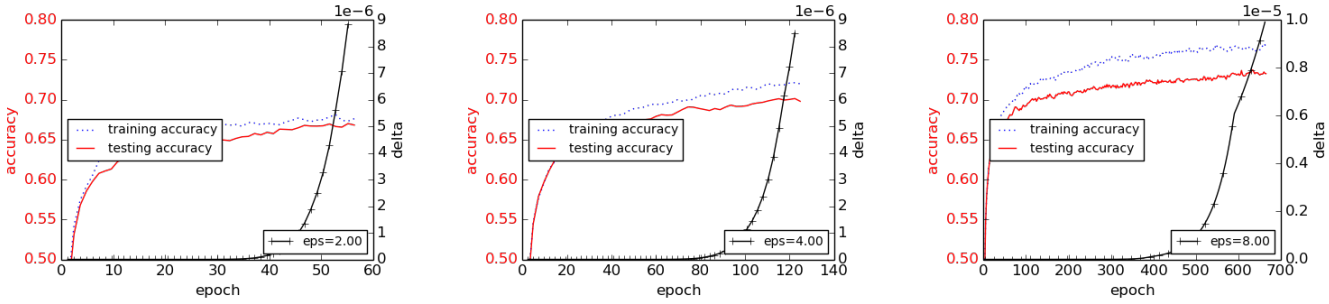
5.3 CIFAR

We also conduct experiments on the CIFAR-10 dataset, which consists of color images classified into 10 classes such as ships, cats, and dogs, and partitioned into 50,000 training examples and 10,000 test examples [1]. Each example is a 32×32 image with three channels (RGB). For this learning task, nearly all successful networks use convolutional layers. The CIFAR-100 dataset has similar parameters, except that images are classified into 100 classes; the examples and the image classes are different from those of CIFAR-10.

We use the network architecture from the TensorFlow convolutional neural networks tutorial [2]. Each 32×32 image is first cropped to a 24×24 one by taking the center patch. The network architecture consists of two convolutional layers followed by two fully connected layers. The convolutional layers use 5×5 convolutions with stride 1, followed by a ReLU and 2×2 max pools, with 64 channels each. Thus the first convolution outputs a $12 \times 12 \times 64$ tensor for each image, and the second outputs a $6 \times 6 \times 64$ tensor. The latter is flattened to a vector that gets fed into a fully connected layer with 384 units, and another one of the same size.

This architecture, non-privately, can get to about 86% accuracy in 500 epochs. Its simplicity makes it an appealing choice for our work. We should note however that by using deeper networks with different non-linearities and other advanced techniques, one can obtain significantly better accuracy, with the state-of-the-art being about 96.5% [28].

As is standard for such image datasets, we use *data augmentation* during training. For each training image, we generate a new distorted image by randomly picking a 24×24 patch from the image, randomly flipping the image along the left-right direction, and randomly distorting the brightness and the contrast of the image. In each epoch, these



(1) $\varepsilon = 2$ (2) $\varepsilon = 4$ (3) $\varepsilon = 8$

Figure 6: Results on accuracy for different noise levels on CIFAR-10. With δ set to 10^{-5}, we achieve accuracy 67%, 70%, and 73%, with ε being 2, 4, and 8, respectively. The first graph uses a lot size of 2,000, (2) and (3) use a lot size of 4,000. In all cases, σ is set to 6, and clipping is set to 3.

distortions are done independently. We refer the reader to the TensorFlow tutorial [2] for additional details.

As the convolutional layers have shared parameters, computing per-example gradients has a larger computational overhead. Previous work has shown that convolutional layers are often transferable: parameters learned from one dataset can be used on another one without retraining [32]. We treat the CIFAR-100 dataset as a public dataset and use it to train a network with the same architecture. We use the convolutions learned from training this dataset. Retraining only the fully connected layers with this architecture for about 250 epochs with a batch size of 120 gives us approximately 80% accuracy, which is our non-private baseline.

Differentially private version.

For the differentially private version, we use the same architecture. As discussed above, we use pre-trained convolutional layers. The fully connected layers are initialized from the pre-trained network as well. We train the softmax layer, and either the top or both fully connected layers. Based on looking at gradient norms, the softmax layer gradients are roughly twice as large as the other two layers, and we keep this ratio when we try clipping at a few different values between 3 and 10. The lot size is an additional knob that we tune: we tried 600, 2,000, and 4,000. With these settings, the per-epoch training time increases from approximately 40 seconds to 180 seconds.

In Figure 6, we show the evolution of the accuracy and the privacy cost, as a function of the number of epochs, for a few different parameter settings.

The various parameters influence the accuracy one gets, in ways not too different from that in the MNIST experiments. A lot size of 600 leads to poor results on this dataset and we need to increase it to 2,000 or more for results reported in Figure 6.

Compared to the MNIST dataset, where the difference in accuracy between a non-private baseline and a private model is about 1.3%, the corresponding drop in accuracy in our CIFAR-10 experiment is much larger (about 7%). We leave closing this gap as an interesting test for future research in differentially private machine learning.

6. RELATED WORK

The problem of privacy-preserving data mining, or machine learning, has been a focus of active work in several research communities since the late 90s [6, 39]. The existing literature can be broadly classified along several axes: the class of models, the learning algorithm, and the privacy guarantees.

Privacy guarantees. Early works on privacy-preserving learning were done in the framework of secure function evaluation (SFE) and secure multi-party computations (MPC), where the input is split between two or more parties, and the focus is on minimizing information leaked during the joint computation of some agreed-to functionality. In contrast, we assume that data is held centrally, and we are concerned with leakage from the functionality's output (i.e., the model).

Another approach, k-anonymity and closely related notions [54], seeks to offer a degree of protection to underlying data by generalizing and suppressing certain identifying attributes. The approach has strong theoretical and empirical limitations [5, 10] that make it all but inapplicable to de-anonymization of high-dimensional, diverse input datasets. Rather than pursue input sanitization, we keep the underlying raw records intact and perturb derived data instead.

The theory of differential privacy, which provides the analytical framework for our work, has been applied to a large collection of machine learning tasks that differed from ours either in the training mechanism or in the target model.

The moments accountant is closely related to the notion of Rényi differential privacy [44], which proposes (scaled) $\alpha(\lambda)$ as a means of quantifying privacy guarantees. In a concurrent and independent work Bun and Steinke [11] introduce a relaxation of differential privacy (generalizing the work of Dwork and Rothblum [22]) defined via a linear upper bound on $\alpha(\lambda)$. Taken together, these works demonstrate that the moments accountant is a useful technique for theoretical and empirical analyses of complex privacy-preserving algorithms.

Learning algorithm. A common target for learning with privacy is a class of convex optimization problems amenable to a wide variety of techniques [20, 12, 36]. In concurrent work, Wu et al. achieve 83% accuracy on MNIST via convex empirical risk minimization [58]. Training multi-layer neural networks is non-convex, and typically solved by an application of SGD, whose theoretical guarantees are poorly understood.

For the CIFAR neural network we incorporate differentially private training of the PCA projection matrix [25], which is used to reduce dimensionality of inputs.

Model class. The first end-to-end differentially private system was evaluated on the Netflix Prize dataset [41], a version of a collaborative filtering problem. Although the problem shared many similarities with ours—high-dimensional inputs, non-convex objective function—the approach taken by McSherry and Mironov differed significantly. They identified the core of the learning task, effectively sufficient statistics, that can be computed in a differentially private manner via a Gaussian mechanism. In our approach no such sufficient statistics exist.

In a recent work Shokri and Shmatikov [52] designed and evaluated a system for *distributed* training of a deep neural network. Participants, who hold their data closely, communicate sanitized updates to a central authority. The sanitization relies on an additive-noise mechanism, based on a sensitivity estimate, which could be improved to a hard sensitivity guarantee. They compute privacy loss per parameter (not for an entire model). By our preferred measure, the total privacy loss per participant on the MNIST dataset exceeds several thousand.

A different, recent approach towards differentially private deep learning is explored by Phan et al. [47]. This work focuses on learning autoencoders. Privacy is based on perturbing the objective functions of these autoencoders.

7. CONCLUSIONS

We demonstrate the training of deep neural networks with differential privacy, incurring a modest total privacy loss, computed over entire models with many parameters. In our experiments for MNIST, we achieve 97% training accuracy and for CIFAR-10 we achieve 73% accuracy, both with $(8, 10^{-5})$-differential privacy. Our algorithms are based on a differentially private version of stochastic gradient descent; they run on the TensorFlow software library for machine learning. Since our approach applies directly to gradient computations, it can be adapted to many other classical and more recent first-order optimization methods, such as NAG [45], Momentum [50], AdaGrad [17], or SVRG [33].

A new tool, which may be of independent interest, is a mechanism for tracking privacy loss, the moments accountant. It permits tight automated analysis of the privacy loss of complex composite mechanisms that are currently beyond the reach of advanced composition theorems.

A number of avenues for further work are attractive. In particular, we would like to consider other classes of deep networks. Our experience with MNIST and CIFAR-10 should be helpful, but we see many opportunities for new research, for example in applying our techniques to LSTMs used for language modeling tasks. In addition, we would like to obtain additional improvements in accuracy. Many training datasets are much larger than those of MNIST and CIFAR-10; accuracy should benefit from their size.

8. ACKNOWLEDGMENTS

We are grateful to Úlfar Erlingsson and Dan Ramage for many useful discussions, and to Mark Bun and Thomas Steinke for sharing a draft of [11].

9. REFERENCES

[1] CIFAR-10 and CIFAR-100 datasets. www.cs.toronto.edu/~kriz/cifar.html.

[2] TensorFlow convolutional neural networks tutorial. www.tensorflow.org/tutorials/deep_cnn.

[3] TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. *CoRR*, abs/1607.00133, 2016.

[5] C. C. Aggarwal. On k-anonymity and the curse of dimensionality. In *VLDB*, pages 901–909, 2005.

[6] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *SIGMOD*, pages 439–450. ACM, 2000.

[7] R. Bassily, K. Nissim, A. Smith, T. Steinke, U. Stemmer, and J. Ullman. Algorithmic stability for adaptive data analysis. In *STOC*, pages 1046–1059. ACM, 2016.

[8] R. Bassily, A. D. Smith, and A. Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *FOCS*, pages 464–473. IEEE, 2014.

[9] A. Beimel, H. Brenner, S. P. Kasiviswanathan, and K. Nissim. Bounds on the sample complexity for private learning and private data release. *Machine Learning*, 94(3):401–437, 2014.

[10] J. Brickell and V. Shmatikov. The cost of privacy: Destruction of data-mining utility in anonymized data publishing. In *KDD*, pages 70–78. ACM, 2008.

[11] M. Bun and T. Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. *CoRR*, abs/1605.02065, 2016.

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

[13] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A Matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[14] D. D. Cox and N. Pinto. Beyond simple features: A large-scale feature search approach to unconstrained face recognition. In *FG 2011*, pages 8–15. IEEE, 2011.

[15] D. Silver, A. Huang, C. J. Maddison et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[16] A. Daniely, R. Frostig, and Y. Singer. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. *CoRR*, abs/1602.05897, 2016.

[17] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Machine Learning Research*, 12:2121–2159, July 2011.

[18] C. Dwork. A firm foundation for private data analysis. *Commun. ACM*, 54(1):86–95, Jan. 2011.

[19] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT*, pages 486–503. Springer, 2006.

[20] C. Dwork and J. Lei. Differential privacy and robust statistics. In *STOC*, pages 371–380. ACM, 2009.

[21] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284. Springer, 2006.

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

[23] C. Dwork and G. N. Rothblum. Concentrated differential privacy. *CoRR*, abs/1603.01887, 2016.

[24] C. Dwork, G. N. Rothblum, and S. Vadhan. Boosting and differential privacy. In *FOCS*, pages 51–60. IEEE, 2010.

[25] C. Dwork, K. Talwar, A. Thakurta, and L. Zhang. Analyze Gauss: Optimal bounds for privacy-preserving principal component analysis. In *STOC*, pages 11–20. ACM, 2014.

[26] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *CCS*, pages 1322–1333. ACM, 2015.

[27] I. Goodfellow. Efficient per-example gradient computations. *CoRR*, abs/1510.01799v2, 2015.

[28] B. Graham. Fractional max-pooling. *CoRR*, abs/1412.6071, 2014.

[29] A. Gupta, K. Ligett, F. McSherry, A. Roth, and K. Talwar. Differentially private combinatorial optimization. In *SODA*, pages 1106–1125, 2010.

[30] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *ICCV*, pages 1026–1034. IEEE, 2015.

[31] R. Ierusalimschy, L. H. de Figueiredo, and W. Filho. Lua—an extensible extension language. *Software: Practice and Experience*, 26(6):635–652, 1996.

[32] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *ICCV*, pages 2146–2153. IEEE, 2009.

[33] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, pages 315–323, 2013.

[34] P. Kairouz, S. Oh, and P. Viswanath. The composition theorem for differential privacy. In *ICML*, pages 1376–1385. ACM, 2015.

[35] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. D. Smith. What can we learn privately? *SIAM J. Comput.*, 40(3):793–826, 2011.

[36] D. Kifer, A. D. Smith, and A. Thakurta. Private convex optimization for empirical risk minimization with applications to high-dimensional regression. In *COLT*, pages 25.1–25.40, 2012.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[38] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

[39] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *CRYPTO*, pages 36–54. Springer, 2000.

[40] C. J. Maddison, A. Huang, I. Sutskever, and D. Silver. Move evaluation in Go using deep convolutional neural networks. In *ICLR*, 2015.

[41] F. McSherry and I. Mironov. Differentially private recommender systems: Building privacy into the Netflix Prize contenders. In *KDD*, pages 627–636. ACM, 2009.

[42] F. D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *SIGMOD*, pages 19–30. ACM, 2009.

[43] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[44] I. Mironov. Rényi differential privacy. Private communication, 2016.

[45] Y. Nesterov. *Introductory Lectures on Convex Optimization. A Basic Course*. Springer, 2004.

[46] J. Pennington, R. Socher, and C. D. Manning. GloVe: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[47] N. Phan, Y. Wang, X. Wu, and D. Dou. Differential privacy preservation for deep auto-encoders: an application of human behavior prediction. In *AAAI*, pages 1309–1316, 2016.

[48] N. Pinto, Z. Stone, T. E. Zickler, and D. Cox. Scaling up biologically-inspired computer vision: A case study in unconstrained face recognition on Facebook. In *CVPR*, pages 35–42. IEEE, 2011.

[49] R. M. Rogers, A. Roth, J. Ullman, and S. P. Vadhan. Privacy odometers and filters: Pay-as-you-go composition. *CoRR*, abs/1605.08294, 2016.

[50] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, Oct. 1986.

[51] A. Saxe, P. W. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Ng. On random weights and unsupervised feature learning. In *ICML*, pages 1089–1096. ACM, 2011.

[52] R. Shokri and V. Shmatikov. Privacy-preserving deep learning. In *CCS*, pages 1310–1321. ACM, 2015.

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

[54] L. Sweeney. k-anonymity: A model for protecting privacy. *International J. of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[55] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9. IEEE, 2015.

[56] S. Tu, R. Roelofs, S. Venkataraman, and B. Recht. Large scale kernel learning using block coordinate descent. *CoRR*, abs/1602.05310, 2016.

[57] O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. E. Hinton. Grammar as a foreign language. In *NIPS*, pages 2773–2781, 2015.

[58] X. Wu, A. Kumar, K. Chaudhuri, S. Jha, and J. F. Naughton. Differentially private stochastic gradient descent for in-RDBMS analytics. *CoRR*, abs/1606.04722, 2016.