FAST DIFFERENTIALLY PRIVATE-SGD VIA JL PROJECTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Differentially Private-SGD (DP-SGD) of Abadi et al. (2016) and its variations are the only known algorithms for private training of large scale neural networks. This algorithm requires computation of per-sample gradients norms which is extremely slow and memory intensive in practice. In this paper, we present a new framework to design differentially private optimizers called DP-SGD-JL and DP-Adam-JL. Our approach uses Johnson–Lindenstrauss (JL) projections to quickly approximate the per-sample gradient norms without exactly computing them, thus making the training time and memory requirements of our optimizers closer to that of their non-DP versions.

Our algorithms achieve state-of-the-art privacy-vs-accuracy tradeoffs on MNIST and CIFAR10 datasets while being significantly faster. Unlike previous attempts to make DP-SGD faster which work only on fully-connected or convolutional layers, our algorithms work for *any* network in a black-box manner which is the main contribution of this paper. To illustrate this, on IMDb dataset, we train a Recurrent Neural Network (RNN) to achieve good privacy-vs-accuracy tradeoff, whereas existing DP optimizers are either inefficient or inapplicable. On RNNs, our algorithms are orders of magnitude faster than DP-SGD for large batch sizes.

The privacy analysis of our algorithms is more involved than DP-SGD, we use the recently proposed f-DP framework of Dong et al. (2019). In summary, we design new differentially private training algorithms which are fast, achieve state-of-the-art privacy-vs-accuracy tradeoffs and generalize to all network architectures.

1 INTRODUCTION

Over the past decade, machine learning algorithms based on (deep) neural architectures have lead to a revolution in applications such as computer vision, speech recognition and natural language processing (NLP). An important factor contributing to this success is the abundance of data. For most of these applications, however, the training data comes from individuals, often containing personal and sensitive information about them. For example, natural language models for applications such as suggested replies for e-mails and dialog systems rely on the training of neural networks on email data of users (Chen et al. (2019); Deb et al. (2019)), who may be left vulnerable if personal information is revealed. This could happen, for example, when a model generates a sentence or predicts a word that can potentially reveal private information of users in the training set. Many studies have shown successful membership inference attacks on deep learning models (Shokri et al. (2017); Carlini et al. (2019)). Indeed, in a recent work, Carlini et al. (2019) show that "unintended memorization" in neural networks is both commonplace and hard to prevent. Such memorization is not due to overtraining (Tetko et al. (1995); Carlini et al. (2019)), and ad hoc techniques such as early-stopping, dropout etc., do not prevent the risk of privacy violations. Moreover, Feldman (2020) shows that memorization is in fact *necessary*, provably, for some learning tasks. Thus, to prevent unintended privacy breaches one needs a principled approach for private training of deep learning models. In this paper we study training neural networks with differential privacy, a mathematically rigorous notion of privacy introduced in the seminal work of Dwork et al. (2006), and focus on user level privacy.

Definition 1.1 ((ε, δ)-DP)**.** We say that an algorithm M is (ε, δ)-DP if for any two neighboring databases D, D' and any subset S of outputs, we have $\Pr[M(D) \in S] \leq e^{\varepsilon} \Pr[M(D') \in S] + \delta$.

Besides being a *provable* privacy notion, it has been shown that deep learning models trained with DP protect against leakage of sensitive information; we refer the readers to Carlini et al. (2019); Abadi et al. (2016) for more details.

In a highly influential paper, Abadi et al. (2016) introduced a differentially private version of stochastic gradient descent (DP-SGD) for training deep learning models, and showed that it is possible to achieve reasonable accuracy-vs-privacy tradeoff on common benchmarks such as MNIST and CIFAR10. Since then, there has been a vast body of work building on and extending the algorithm of Abadi et al. (2016); we refer the readers to McMahan et al. (2018); Bu et al. (2019);

Carlini et al. (2019); Thakkar et al. (2019); Augenstein et al. (2020); Zhou et al. (2020); Chen et al. (2020); Balle et al. (2020). The DP-SGD and its variations such as DP-Adam differ from their non-private counter parts in two crucial ways (see Algorithm 4):

- **Gradient Clipping:** In each iteration of DP-SGD, we clip *each* per-sample gradient to have ℓ_2-norm at most some fixed parameter C. This step ensures that the *sensitivity* of the average gradient is bounded, which is crucial for privacy analysis. Computing the norms of per-sample gradients is the most expensive step in the algorithm. Efficient implementations of backpropagation such as in TensorFlow and PyTorch only maintain the average of per-sample gradients across a batch by default. Therefore, getting the norm of each per-sample gradient requires significantly more time and memory.
- **Adding Noise:** Once clipped gradients are averaged across a batch, DP-SGD algorithm adds carefully calibrated noise, typically sampled from the Gaussian distribution, to ensure privacy.

The analysis of DP-SGD in Abadi et al. (2016) then follows from a careful tracking of privacy budget lost in each iteration, for which they introduced a novel technique called *moments accountant*, which was later generalized as Renyi differential privacy by Mironov (2017). This analysis was further refined and improved using the f-DP framework by Dong et al. (2019) and Bu et al. (2019), which lead to better privacy bounds. In this work, we use f-DP framework of Dong et al. (2019) for our privacy analysis.

While DP-SGD has been shown to achieve reasonable accuracy-vs-privacy tradeoff Bu et al. (2019); Abadi et al. (2016), and arguably is the only known algorithm for training deep neural networks, its use in real-world deep learning has been rather limited. One of the primary reasons for this is the training time of DP-SGD compared to SGD. In DP-SGD, per-sample gradients are computed at a heavy cost in runtime, especially for large deep learning models. The naive approach of setting the batch size to 1 is too slow to be practical as we completely lose the benefits of parallelization. This problem has attracted significant attention in the community and has been noted in popular implementations of DP-SGD including Tensorflow Privacy and Opacus (Pytorch DP). Many strategies have been proposed to circumvent this issue, and they fall broadly into the following categories:

- **Microbatching:** DP-SGD implementation in Tensorflow Privacy allows dividing a batch into several micro-batches and clipping the gradient at the microbatch level; the per-sample gradients in a microbatch are first averaged and then clipped, and finally these clipped gradients are averaged across microbatches. Thus, if each microbatch is of size L, then it gives a speedup of L over the usual DP-SGD. Unfortunately, the sensitivity goes up by a factor of L, so we need to add L times more noise. In our experiments, we observe that this often leads to poor accuracy even for moderate values of L.
- **Multiple method:** In this approach, proposed by Goodfellow and implemented as the vectorized DP-SGD in Tensorflow Privacy, one copies the model as many times as there are samples in a batch. As each copy of the model is used on only one example, we can compute the per-sample gradients in parallel. This approach improves speed at the cost of memory and is impractical for large models.
- **Outer product method:** This strategy was proposed by Goodfellow (2015) for fully-connected networks and later generalized by Rochette et al. (2019) to include convolutional networks. The norms of per-sample gradients are computed *exactly* using outer products between the activations and backpropagated gradients across adjacent layers. The biggest drawback of this approach is that it does not work for many network architectures (e.g., RNNs such as bidirectional LSTMs). Moreover, this method requires significantly more memory than DP-SGD and non-private SGD (see Section D.1).

Table 1: Summary of different methods for DP training of neural networks

Optimizers	Privacy	Speed	Memory	Generalizability
Non-DP SGD	✘	✔	✔	✔
DP-SGD-Vanilla	✔	✘	✔	✔
DP-SGD-Multiple	✔	✔	✘	✔
DP-SGD-Outer	✔	✔	✔	✘
DP-SGD-JL	✔	✔	✔	✔

As we can see, none of these approaches for speeding up DP-SGD completely solve the problem and fall short in at least one dimension. In this work, we propose a new *algorithmic framework* based on JL-projections for *fast* differentially private training of deep neural networks, which bypasses the expensive step of exactly computing per-sample gradient norms. We summarize this discussion in Table 1.

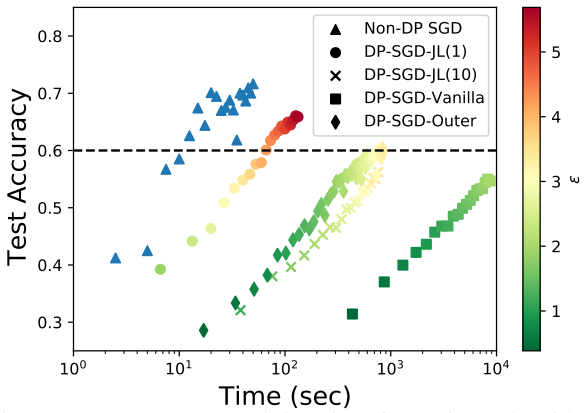

Figure 1: Accuracy vs Training time for various algorithms on CIFAR10 using the network from Papernot et al. (2020). Privacy loss ε is color coded for a fixed $\delta = 10^{-5}$.

Our Contributions and Techniques The main idea of our algorithm is to *approximate* the per-sample gradient norms instead of computing them exactly. Johnson–Lindenstrauss (JL) projections provide a convenient way to approximate the ℓ_2 norm of a vector; simply project the vector onto a uniformly random direction, the length of the projection (scaled appropriately) is a good approximation to the ℓ_2-norm of the vector. By doing more such projections and averaging, we get even better approximation to the true ℓ_2-norm. Moreover, there is an efficient way to obtain such projections using forward-mode auto-differentiation or Jacobian-vector product (`jvp`) (see Section 2.1 for details). `jvp` can be calculated during the forward pass making it very efficient. Since this makes the *sensitivity* itself a random variable, the privacy analysis is significantly harder than the traditional DP-SGD. We use the recently proposed f-DP framework of Dong et al. (2019) for our analysis. Intuitively, f-DP captures the entire collection of (ε, δ)-DP guarantees that each iteration of the algorithm satisfies and composes them optimally to find the best (ε, δ)-DP guarantee for the final algorithm.

To summarize, the key contributions of this paper are:

- Our algorithms DP-SGD-JL and DP-Adam-JL achieve training times comparable to their non-private counterparts, are significantly faster than previously known private training algorithms, and work for *all* network architectures. The privacy-vs-accuracy tradeoff achieved by our algorithms is comparable to the existing state-of-the-art DP-algorithms.
- Compared to DP-SGD, our analysis of privacy is more involved. Since we only approximate the per-sample gradient norms, we cannot precisely bound *sensitivity*. Therefore the analysis requires significantly new ideas and we use the recently developed f-DP framework of Dong et al. (2019).
- We demonstrate these improvements by performing extensive experiments on MNIST, CIFAR10, and IMDb datasets using various network architectures such as convolution layers, recurrent layers, embedding layers etc.. See a glimpse of our results in Figure 1. Our experiments in Section 3 show that our algorithms achieve state-of-the-art privacy-vs-accuracy tradeoffs, sometimes even improving them, while being $20\times$ faster on MNIST, $8\times$ faster on CIFAR10, and $350\times$ faster on IMDb for a bidirectional LSTM network.
- Our algorithms introduce a new *knob*, the dimension of JL-projection, which allows us to do a tradeoff between training-time and privacy, which was not possible in earlier algorithms. All hyperparameters being the same, smaller JL dimension will give much better running time with a slight increase in privacy budget.

Estimating per-sample gradients norms is also useful in other applications such as optimization based on importance sampling (Zhao & Zhang (2015)). We believe that our JL-projection technique will be useful in speeding up all such applications.

2 DP-SGD-JL Algorithm

In this section we describe our new differentially private optimizers. We will first present DP-SGD-JL, and DP-Adam-JL follows in the same lines and is presented in Appendix A. We begin with an introduction to "Jacobian-vector product" (`jvp`) which is crucial for our algorithm.

2.1 Jacobian-vector product (`jvp`)

Given a function $f : \mathbb{R}^d \to \mathbb{R}$, the gradient of f with respect to θ, denoted by $\nabla_\theta f$, is:

$$\nabla_\theta f = \left(\frac{\partial f}{\partial \theta_1}, \frac{\partial f}{\partial \theta_2}, \ldots, \frac{\partial f}{\partial \theta_d} \right).$$

Let $F : \mathbb{R}^d \to \mathbb{R}^m$ be some function given by $F(\theta) = (F_1(\theta), F_2(\theta), \ldots, F_m(\theta))$. The Jacobian of $F(\theta)$, denoted by $\nabla_\theta F$, is the matrix:

$$\nabla_\theta F = \left[\frac{\partial F_i}{\partial \theta_j}\right]_{ij} = \begin{bmatrix} \nabla_\theta F_1 \\ \nabla_\theta F_2 \\ \vdots \\ \nabla_\theta F_m \end{bmatrix}.$$

Most auto-differentiation packages allow for calculating the vector-Jacobian product (`vjp`) given by $u^T \nabla_\theta F = \sum_{i=1}^m u_i \nabla_\theta F_i$ for any $u \in \mathbb{R}^m$ efficiently using reverse-mode auto-differentiation, which is the familiar 'backpropagation'.[1] One can also calculate the Jacobian-vector product (`jvp`) given by

$$\nabla_\theta F \cdot v = \begin{bmatrix} \langle \nabla_\theta F_1, v \rangle \\ \langle \nabla_\theta F_2, v \rangle \\ \vdots \\ \langle \nabla_\theta F_m, v \rangle \end{bmatrix}$$

efficiently using forward-mode auto-differentiation (i.e., the required derivatives are calculated during the forward pass of the network). This is implemented in the recent tensorflow versions.[2] Unfortunately, PyTorch doesn't have an implementation of forward-mode auto-differentiation. Instead, one can compute `jvp` using two calls to `vjp`, this is called the 'double `vjp` trick' (see Townsend). Define $G(\alpha) = \alpha^T \nabla_\theta F$, which can be calculated using `vjp`. Note that $\nabla_\alpha G = (\nabla_\theta F)^T$. Now we can use `vjp` again on G to calculate

$$v^T \nabla_\alpha G = v^T (\nabla_\theta F)^T = (\nabla_\theta F \cdot v)^T.$$

In our experiments, we use `jvp` to compute the Jacobian-vector products as double `vjp` trick is significantly slower.

2.2 Algorithm

The main idea of our algorithm is to approximate ℓ_2-norms of per-sample gradient norms instead of computing them exactly. And the key tool to achieve this is JL projections.

Proposition 2.1 (JL projections)**.** Let $y \in \mathbb{R}^d$ be any vector. If $v_1, v_2, \ldots, v_r \sim \mathcal{N}(0, I_d)$ are independent standard Gaussian vectors, then $M_r = \sqrt{\sum_{i=1}^r \frac{1}{r} \langle y, v_i \rangle^2}$ has the same distribution as $\|y\|_2 \sqrt{\frac{1}{r}\chi_r^2}$.[3] In particular $\mathbb{E}[M_r^2] = \|y\|_2^2$.

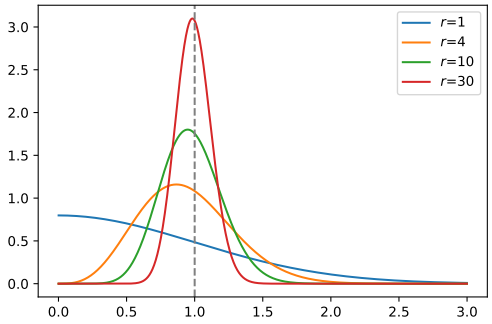

Figure 2: Distribution of $\sqrt{\frac{1}{r}\chi_r^2}$ for different r.

Proof. By the properties of the standard Gaussian distribution, $\langle y, v_i \rangle$ has the distribution of $\|y\|_2 \mathcal{N}(0, 1)$. And $\langle y, v_i \rangle$ are independent for $i = 1$ to r. Therefore $\sum_{i=1}^r \langle y, v_i \rangle^2$ has the same distribution as $\|y\|_2^2 \chi_r^2$. □

As shown in Figure 2, as r grows larger, the distribution of $\sqrt{\frac{1}{r}\chi_r^2}$ concentrates more around 1 and therefore M_r becomes a better estimate of $\|y\|_2$. Using `jvp`, we can compute projections of per-sample gradients on to standard Gaussian vectors quickly and therefore get good approximations to their norms. This is the main idea of DP-SGD-JL (Algorithm 1). The privacy analysis of our algorithm is quite involved and we will get back to it in Section 4 after discussing our experiments.

[1] In PyTorch, $u^T \nabla_\theta F$ can be calculated as `autograd.grad(F,θ,grad_outputs=u)`. In Tensorflow, this is `tf.GradientTape().gradient(F,θ,output_gradients=u)`.

[2] Supported in tf-nightly≥2.4.0.dev20200924 as `tf.autodiff.ForwardAccumulator(θ,v).jvp(F)`.

[3] χ_r^2 is the chi-square distribution with r degrees of freedom which is the distribution of sum of squares of r standard Gaussians.

Algorithm 1 Differentially private SGD using JL projections (DP-SGD-JL)

Input: Examples $\{x_1, x_2, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \mathbb{E}_{i\in[N]}[\mathcal{L}(\theta; x_i)]$, initialization θ_0.
Parameters: number of iterations T, learning rates $(\eta_1, \eta_2, \ldots, \eta_T)$, noise scale σ, batch size B, clipping norms $(C_1, C_2, \ldots, C_T)$, number of JL projections r.

for $t = 1$ to T **do**
  Sample $S_t = \{X_1, X_2, \ldots, X_B\} \subset \{x_1, x_2, \ldots, x_N\}$ uniformly at random.
  Define $F(\theta) = (\mathcal{L}(\theta; X_1), \mathcal{L}(\theta; X_2), \ldots, \mathcal{L}(\theta; X_B))$

  Sample $v_1, v_2, \ldots, v_r \leftarrow \mathcal{N}(0, I_d)$ (where $\theta \in \mathbb{R}^d$) ▷ JL projections to estimate per-sample gradient norm
  for $j = 1$ to r **do**
    $(P_{1j}, P_{2j}, \ldots, P_{Bj}) \leftarrow \nabla_\theta F(\theta_{t-1}) \cdot v_j$ (using `jvp`) ▷ Note that $P_{ij} = \langle \nabla_\theta \mathcal{L}(\theta_{t-1}; X_i), v_j \rangle$
  end for
  for $i = 1$ to B **do**
    Set $M_i = \sqrt{\frac{1}{r}\sum_{j=1}^{r} P_{ij}^2}$ ▷ M_i is an estimate for $\|\nabla_\theta \mathcal{L}(\theta_{t-1}; X_i)\|_2$.
  end for
  Define $\widetilde{\mathcal{L}}(\theta) = \frac{1}{B}\sum_{i\in B} \min\{1, \frac{C_t}{M_i}\} \cdot \mathcal{L}(\theta; X_i)$. ▷ Clip the per-sample gradients

  $\tilde{g}_t \leftarrow \nabla_\theta \widetilde{\mathcal{L}}(\theta_{t-1}) + \frac{\sigma \cdot C_t}{B} \cdot \mathcal{N}(0, I_d)$ ▷ Add noise to the average of clipped gradients

  Update $\theta_t \leftarrow \theta_{t-1} - \eta_t \tilde{g}_t$
end for
Output: $\theta_0, \theta_1, \theta_2, \ldots, \theta_T$

3 EXPERIMENTS

We experimentally evaluate and compare our new algorithms for training deep neural networks to previously algorithms on multiple classification tasks, such as MNIST (LeCun & Cortes (2010)), CIFAR10 (Krizhevsky et al. (2009)) and IMDb Dataset (Maas et al. (2011)). We demonstrate that compared to existing implementations of DP-SGD with exact per-sample gradient clipping, our optimizers have significant advantages in speed and memory cost while achieving comparable accuracy-vs-privacy tradeoff. Moreover our algorithms perform well on a variety of network architectures.

In the following, we write 'Non-DP-SGD' as the standard non-private SGD. We note that the original DP-SGD described in Abadi et al. (2016) can be implemented in two ways: we denote the 'DP-SGD-Vanilla' as the one implemented in Tensorflow Privacy and the mathematically equivalent but faster implementation in Opacus as 'DP-SGD-Outer' since it uses the 'outer product' trick discussed in the introduction. We note that DP-SGD-Vanilla can be applied to any network architecture (see TFissue) including recurrent layers such as LSTM and GRU, at the cost of slow computation.

We can implement DP-SGD-JL and DP-Adam-JL in two different ways: 1) via `jvp` directly (in Tensorflow) and 2) via double `vjp` trick (in Pytorch and Tensorflow). Both implementations are mathematically equivalent and result in the same accuracy and privacy, but they differ in computation time. We note that using `jvp` is usually much faster than the double `vjp` trick. In Appendix D.1, we analyze the memory footprints for various algorithms. In Appendix D.2, we demonstrate the scalability of our algorithms by training a large neural network VGG13. In this section, we focus on accuracy-vs-privacy-vs-time tradeoffs while fixing the memory.

Notation: We denote the noise multiplier as σ, clipping norm as C, batch size as B, learning rate as η and the number of epochs as $E = BT/N$. We fix the privacy parameter $\delta = 10^{-5}$, as done by prior work. We denote the JL dimension used by each optimizer in the parentheses. We use one Tesla P100 16GB GPU for all experiments.

3.1 MNIST EXPERIMENTS

Figure 3 displays the performance of various algorithms in training the 2-layer CNN from Bu et al. (2019); Tensorflow Privacy; Opacus on MNIST. We use the same hyper-parameters as in Bu et al. (2019): $B = 256, \sigma = 1.3, C = 1.5, \eta = 0.25, E = 15$.

Fixing the accuracy at 95%, DP-SGD-JL(1) requires less than 10 seconds and ε of 2.4. On the other hand, DP-SGD-Outer requires more than 100 seconds; however, it achieves an ε of less than 1. So, while DP-SGD-JL(1) gives a significant speedup, it does require higher privacy budget. However, by increasing the JL dimension, we can tradeoff

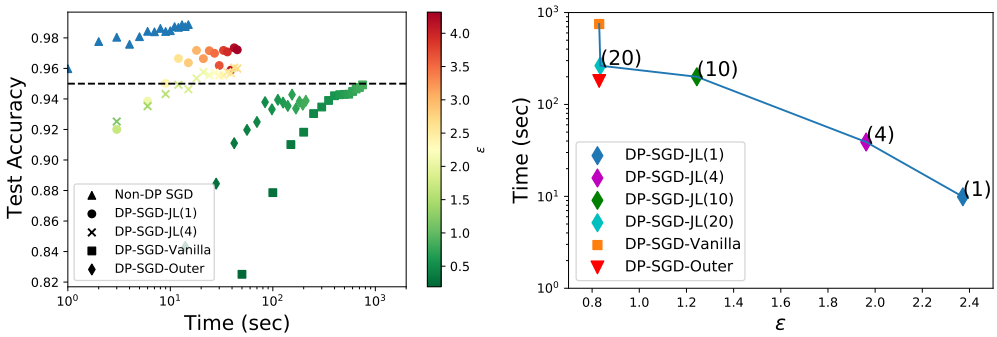

Figure 3: Left: Performance of various algorithms training a 2-layer CNN on MNIST. Right: Fixing accuracy at 95%, plotting speed-vs-privacy tradeoff.

the speed-vs-privacy. These experiments show another important strength of JL optimizers: our algorithms give a new *knob* – the dimension of JL projections – to tune privacy-vs-speed tradeoff. On the other hand, DP-SGD-Vanilla would take roughly the same time per epoch regardless of the value of ε.

3.2 CIFAR10 Experiments

Next, we conduct experiments on a more difficult image dataset, CIFAR10, using a neural network architecture proposed in a recent work of Papernot et al. (2020). This paper achieves an accuracy of 66.2% at $\varepsilon = 7.53$ without data-augmentation or pre-training, in contrast to previous works Abadi et al. (2016); Xiang et al. (2019); Phan et al. (2017). In Table 2, the DP-SGD implemented by Pytorch uses double `vjp` trick to compute `jvp` and the one by Tensorflow uses `jvp` directly.

	Non-DP SGD	DP-SGD-Vanilla	DP-SGD-Outer	DP-SGD-JL(1)	DP-SGD-JL(10)
Pytorch	12	-	17	78	657
Tensorflow	2.5	434	-	6.6	38

Table 2: Seconds per epoch to train the network from Papernot et al. (2020) (7 convolutional layers and 1 fully-connected layers) with 623,146 parameters. We set $\sigma = 0.9, C = 2, B = 256, \eta = 0.1, E = 20$.

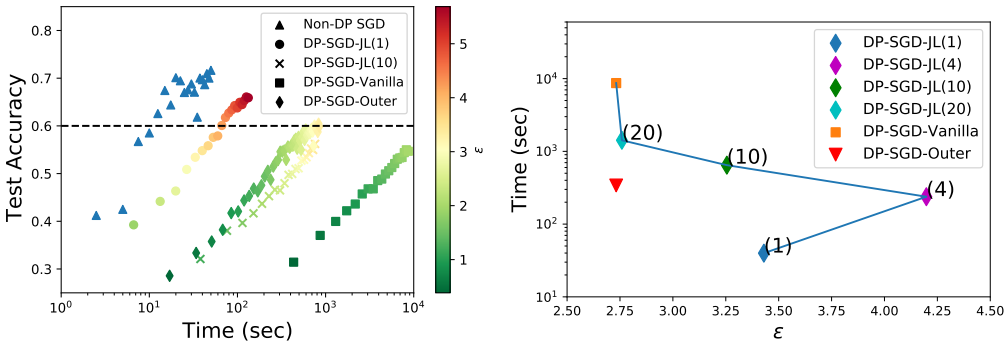

Figure 4: Left: Performance of various algorithms training the network in Papernot et al. (2020) on CIFAR10. Right: Privacy-vs-Training time tradeoff fixing accuracy at 55%.

In Figure 4, we see that DP-SGD-JL(1) matches the state-of-the-art[4] accuracy of 66.1% at $\varepsilon = 5.55$ while clearly beating all other algorithms in training time. If we fix an accuracy of 55%, DP-SGD-JL1 is about $8\times$ faster than DP-SGD-Outer while losing only a little bit in privacy. We also note that DP-SGD-JL(10) closely matches the performance of DP-SGD-Outer.

It is interesting that DP-SGD-JL(1) is simultaneously more private and faster than DP-SGD-JL(4). One possible explanation is that DP-SGD-JL(1) learns faster than larger JL dimensions since it might be taking larger steps compared to others. Another possibility is the non-monotonicity of the privacy loss w.r.t JL dimension (see Appendix C).

[4]We could not reproduce the exact results of Papernot et al. (2020), which states that an accuracy of 66.2% can be achieved using $\varepsilon = 7.53$.

3.3 IMDB EXPERIMENTS

The goal of these experiments is to show that our algorithms work well on RNNs for which existing techniques do not work. We train a bidirectional LSTM with an embedding layer on the IMDb dataset for sentiment analysis. We remark that simpler networks *not* based on RNNs can achieve good accuracy-vs-privacy tradeoff as shown in Bu et al. (2019) and Pytorch. However, LSTM models based on RNN architectures are widely used in NLP applications, and hence efficient private training of such models remains an important challenge in this area (McMahan et al. (2018)). Moreover, as we noted in the introduction, methods such as the outer product trick do not work for bidirectional LSTMs. [5].

Non-DP Adam	DP-Adam-Vanilla	DP-Adam-JL(1)	DP-Adam-JL(4)	DP-Adam-JL(10)	DP-Adam-JL(30)
12	13931	38	100	243	669

Table 3: Seconds per epoch to train our RNN with 598,274 parameters. We set $\beta_1 = 0.9, \beta_2 = 0.999, \sigma = 0.6, C = 1, B = 256, \eta = 0.001, E = 20$.

We implement DP-Adam-JL in Algorithm 3 using `jvp` method. We train the same single-layer bidirectional LSTM as in the Tensorflow, using the same IMDb dataset with 8k vocabulary. The dataset has binary labels and is preprocessed by zero-padding the sequences to a length of 150 before being fed into the embedding layer.

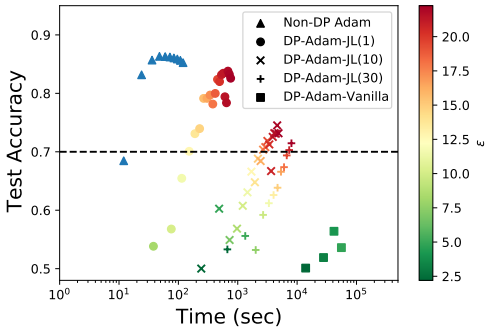

Figure 5: Performance of RNN on IMDb dataset with different Adam.

Table 3 shows a speedup of about $350\times$ for DP-Adam-JL(1) over DP-Adam-Vanilla. This is unexpected as the speedup should not be more than batch size (256); this may be due to memory and implementation issues. Moreover, DP-Adam-JL(1) achieves an accuracy of 83.2% with $\varepsilon = 21$. This is very close to the accuracy achieved by Non-DP-Adam.

In Figure 6, fixing test accuracy at 70%, we see that DP-SGD-JL(1) is much faster and more private than other JL dimensions up to 10. DP-Adam-Vanilla is extremely slow, so we could only run it for 4 epochs in our experiments and we couldn't train it until it reaches an accuracy of 70%. But we expect the accuracy-vs-privacy tradeoff of DP-Adam-Vanilla to be very close that of DP-Adam-JL(30) but much slower. Finally, fixing the training time at 720 seconds, we observe a trade-off between accuracy and privacy, with smaller JL dimensions achieving higher accuracy at the cost of more privacy loss. DP-Adam-Vanilla doesn't even finish one epoch.

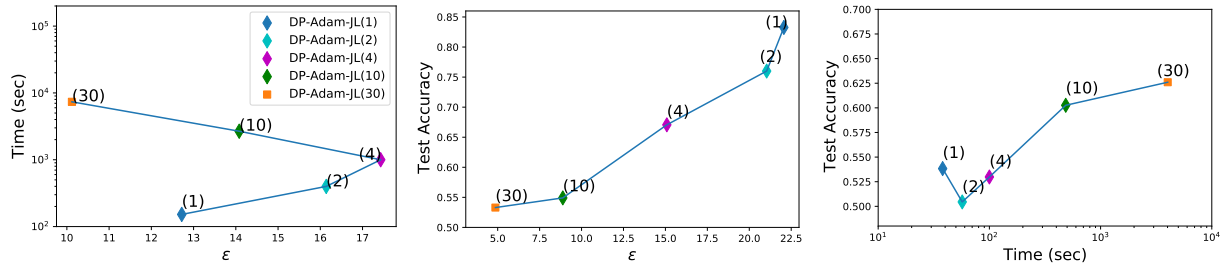

Figure 6: Accuracy, privacy and training time trade-off for training RNN on IMDb dataset. Left: Fixing accuracy at 70%, Middle: Fixing training time at 720 seconds, Right: Fixing $\varepsilon = 8$.

4 PRIVACY ANALYSIS

4.1 f-DP PRELIMINARIES

We use the recently proposed f-DP framework of Dong et al. (2019) for our privacy analysis. The f-DP framework allows us to reason about a collection of (ε, δ)-privacy guarantees simultaneously which can then be composed to get a much better (ε, δ)-privacy for the final algorithm. We will first define the notion of (ε, δ)-DP formally and then define

[5] In Pytorch Opacus github, the LSTM layer is only partially supported, e.g. single directional, single LSTM layer, no dropout layer; other recurrent layers such as GRU are not supported (see Opacus).

the notion of f-DP. We then state a proposition from Dong et al. (2019) which shows that these two notions are dual to each other.

Definition 4.1 ((ε, δ)-DP)**.** We say that an algorithm M is (ε, δ)-DP if for any two neighboring databases D, D' and any subset S of outputs, we have $\Pr[M(D) \in S] \leq e^{\varepsilon} \Pr[M(D') \in S] + \delta$.

For each value of $\varepsilon \geq 0$, there exists some $\delta \in [0, 1]$ such that M is (ε, δ)-DP. We can represent all these privacy guarantees by a function $\delta(\varepsilon) : \mathbb{R}^{\geq 0} \to [0, 1]$ and say that M is $(\varepsilon, \delta(\varepsilon))$-DP for each $\varepsilon \geq 0$. We will now see that there is a dual way to represent the function $\delta(\varepsilon)$ called f-DP. To introduce this, we will need the notion of a tradeoff function.

Definition 4.2 (Tradeoff function)**.** Let P, Q be two distributions over some domain X. Let $\phi : X \to [0, 1]$ be a prediction rule which given a sample predicts which distribution the sample came from. The type I error is defined as $\alpha = \mathbb{E}_{x \sim P}[\phi(x)]$ and the type II error is defined as $\beta = \mathbb{E}_{x \sim Q}[1 - \phi(x)]$. The tradeoff function $T(P||Q) : [0, 1] \to [0, 1]$ is defined as:

$$T(P||Q)(\alpha) = \inf_{\phi} \{1 - \mathbb{E}_Q[\phi] : \mathbb{E}_P[\phi] \leq \alpha\}. \tag{1}$$

Given two random variables X, Y, we define $T(X||Y)$ to be $T(P||Q)$ where P, Q are the distributions of X, Y respectively.

Note that if $T(P, Q) = f$, then $T(Q, P) = f^{-1}$. A tradeoff curve f is called symmetric if $f^{-1} = f$. Given two functions f, g on the same domain, we say $f \preceq g$ if $f(x) \leq g(x)$ for all x in the domain. $f \succeq g$ is similarly defined.

Definition 4.3 (f-DP)**.** We say an algorithm M is f-differentially private if for every two neighboring databases D, D', we have $T(M(D)||M(D')) \succeq f$.

Proposition 4.1 (Duality of f-DP and (ε, δ)-DP from Dong et al. (2019))**.** If M satisfies f-DP where f is symmetric (i.e., $f^{-1} = f$). Let α^* be such that $f(\alpha^*) = \alpha^*$. Then M satisfies $(\varepsilon(\alpha), \delta(\alpha))$-DP for every $0 \leq \alpha \leq \alpha^*$ where:

$$\varepsilon(\alpha) = \log(-f'(\alpha)), \ \delta(\alpha) = 1 - f(\alpha) + \alpha f'(\alpha).$$

So the tangent to f at α has slope $-\exp(\varepsilon(\alpha))$ and y-intercept $1 - \delta(\alpha)$ (see Figure 7).

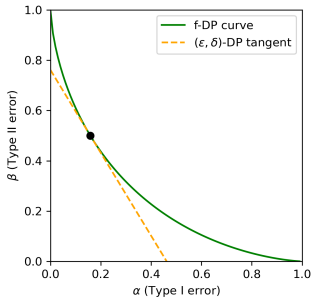

Figure 7: Duality of f-DP and (ε, δ)-DP. Every tangent to the f-DP curve gives an (ε, δ)-DP guarantee where the slope is $-e^{\varepsilon}$ and y-intercept is $1 - \delta$.

Note that by convexity of f, $f' : [0, 1] \to \mathbb{R}$ is increasing in $[0, 1]$ and by symmetry $f'(\alpha^*) = 0$. Therefore Proposition 4.1 covers the entire range of $\varepsilon \geq 0$.

Definition 4.4 (Parametrization of a tradeoff curve)**.** A parametrization of a tradeoff curve f is given by $(\alpha(t), \beta(t))$ for some parameter $t \in \mathbb{R}$ such that $f(\alpha(t)) = \beta(t)$. A symmetric parametrization is a parametrization such that $(\alpha(-t), \beta(-t)) = (\beta(t), \alpha(t))$

If we have a symmetric parametrization of the tradeoff curve given by $f(\alpha(t)) = \beta(t)$, then we can find ε, δ as:

$$\varepsilon(t) = \log\left(-\frac{\beta'(t)}{\alpha'(t)}\right), \ \delta(t) = 1 - \beta(t) + \alpha(t)\beta'(t)/\alpha'(t) \tag{2}$$

for $t \geq 0$. Note that in a symmetric parametrization, $\alpha^* = \alpha(0) = \beta(0)$.

Proposition 4.2 (Post-processing)**.** Let X, Y be two random variables supported on A and let $M : A \to B$ is some randomized function, then $T(X||Y) \preceq T(M(X)||M(Y))$.

Proposition 4.3. Let (X_1, Y_1) and (X_2, Y_2) be pairs of random variables such that $X_1|_{Y_1=y}$ has the same distribution as $X_2|_{Y_2=y}$ for all y. Then $T(X_1, Y_1||X_2, Y_2) = T(Y_1||Y_2)$.

Proof. By post-processing (Proposition 4.2), $T(X_1, Y_1||X_2, Y_2) \preceq T(Y_1||Y_2)$. Let $X(y)$ be a random variable which has the distribution of $X_1|_{Y_1=y}$ and $X_2|_{Y_2=y}$. Let $M(y) = (X(y), y)$. Then $M(Y_1) = (X_1, Y_1)$ and $M(Y_2) = (X_2, Y_2)$. Therefore, by post-processing (Proposition 4.2), we have the inequality in the other direction. □

Proposition 4.4 (Composition Dong et al. (2019))**.** Let $X_1, X_2, \ldots, X_m$ be independent random variables and let $Y_1, Y_2, \ldots, Y_m$ be independent random variables. Then

$$T(X_1, X_2, \ldots, X_m||Y_1, Y_2, \ldots, Y_m) = T(X_1||Y_1) \otimes T(X_2||Y_2) \otimes \cdots \otimes T(X_m||Y_m)$$

where $\otimes$ is a commutative, associative operation on functions from $[0, 1] \to [0, 1]$.

For any random variable X, we have $T(X||X) \equiv \mathrm{Id}$ where $\mathrm{Id} : [0,1] \to [0,1]$ is defined as $\mathrm{Id}(\alpha) = 1 - \alpha$. The function Id is identity for $\otimes$ operation i.e. $f \otimes \mathrm{Id} = f$ for all f.

We will the need the following proposition which explains how subsampling affects the privacy curve.

Proposition 4.5 (Dong et al. (2019))**.** Let $p \in (0,1)$ and let $f = T(P||Q)$. Then $T(P||(1-p)P + pQ) = p \cdot f + (1-p) \cdot \mathrm{Id}$.

We will now present two crucial lemmas which are important for our privacy analysis. The proofs of the lemmas are presented in Appendix B.

Lemma 4.1. Let Z be some random variable and let $f = T(Z, N(Z,1)||Z, N(0,1))$. Then f can be parametrized as $f(\alpha(t)) = \beta(t)$ where:

$$\alpha(t) = \mathbb{E}_Z\left[\Phi\left(-\frac{t}{Z} - \frac{Z}{2}\right)\right],\ \beta(t) = \mathbb{E}_Z\left[\Phi\left(\frac{t}{Z} - \frac{Z}{2}\right)\right]$$

and Φ is the CDF of standard Gaussian. We also have:

$$\alpha'(t) = -\frac{\exp(-t/2)}{\sqrt{2\pi}}\mathbb{E}_Z\left[\frac{1}{Z}\exp\left(-\frac{t^2}{2Z^2} - \frac{Z^2}{8}\right)\right],\ \beta'(t) = \frac{\exp(t/2)}{\sqrt{2\pi}}\mathbb{E}_Z\left[\frac{1}{Z}\exp\left(-\frac{t^2}{2Z^2} - \frac{Z^2}{8}\right)\right]$$

and $\beta'(t)/\alpha'(t) = -\exp(t)$.

Lemma 4.2. Let $Z, \widetilde{Z}$ be two random variables such that there exists some coupling $(Z, \widetilde{Z})$ with $Z \geq \widetilde{Z} \geq 0$. Then $T(Z, N(Z,1)||Z, N(0,1)) \preceq T(\widetilde{Z}, N(\widetilde{Z},1)||\widetilde{Z}, N(0,1))$.

4.2 Proof of privacy for Algorithm 4

Algorithm 2 Subroutine of Algorithm 1

Input: Vectors $\{g_1, g_2, \ldots, g_N\} \subset \mathbb{R}^d$, clipping norm C, noise scale σ, number of JL projections r.

Sample $v_1, v_2, \ldots, v_r \leftarrow \mathcal{N}(0, I_d)$ ▷ JL projections to estimate per-sample gradient norm

For $i \in [N]$, set $M_i = \sqrt{\frac{1}{r}\sum_{j=1}^{r}\langle g_i, v_j\rangle^2}$ ▷ M_i is an estimate for $\|g_i\|_2$

$\tilde{g} \leftarrow \left(\sum_{i=1}^{N} \min\{1, \frac{C}{M_i}\} \cdot g_i\right) + \sigma \cdot C \cdot \mathcal{N}(0, I_d)$

Output: $\tilde{g}$

We will first analyze the privacy of a crucial subroutine used in Algorithm 4 which is shown in Algorithm 2.

Lemma 4.3. Algorithm 2 is f-DP with

$$f = T(Z_r, \mathcal{N}(Z_r/\sigma, 1)||Z_r, \mathcal{N}(0,1))$$

where $Z_r = \frac{1}{\sqrt{\frac{1}{r}\chi^2_r}}$. Moreover f can be parametrized as $f(\alpha(t)) = \beta(t)$ for $t \in \mathbb{R}$ where:

$$\alpha(t) = \mathbb{E}_{Z_r}\left[\Phi\left(-\frac{t\sigma}{Z_r} - \frac{Z_r}{2\sigma}\right)\right],\ \beta(t) = \mathbb{E}_{Z_r}\left[\Phi\left(\frac{t\sigma}{Z_r} - \frac{Z_r}{2\sigma}\right)\right].$$

Proof. Let X be the output of Algorithm 2 with input $\{g_0, g_1, \ldots, g_N\}$ and let Y be the output of Algorithm 2 with input $\{g_1, \ldots, g_N\}$. We want to show that $T(X||Y) \succeq f$. We have

$$X = \left(\sum_{i=0}^{N} \min\left\{1, \frac{C}{M_i}\right\} \cdot g_i\right) + \sigma \cdot C \cdot \mathcal{N}(0, I_d),\ \ Y = \left(\sum_{i=1}^{N} \min\left\{1, \frac{C}{M_i}\right\} \cdot g_i\right) + \sigma \cdot C \cdot \mathcal{N}(0, I_d).$$

By post-processing property (Proposition 4.2), we have

$$T(X||Y) \succeq T(v_1, v_2, \ldots, v_r, M_0, X'||v_1, v_2, \ldots, v_r, M_0, Y')$$

where

$$X' = \min\left\{1, \frac{C}{M_0}\right\} \cdot g_0 + \sigma \cdot C \cdot \mathcal{N}(0, I_d),\ \ Y' = \sigma \cdot C \cdot \mathcal{N}(0, I_d).$$

By Proposition 4.3,

$$T(v_1, v_2, \ldots, v_r, M_0, X' || v_1, v_2, \ldots, v_r, M_0, Y') = T(M_0, X' || M_0, Y').$$

Let U be a rotation matrix which rotates g_0 to $\|g_0\|_2 \cdot e_1 \in \mathbb{R}^d$ where $e_1 = (1, 0, 0, \ldots, 0)$. Let $X'' = UX'$ and $Y'' = UY'$. Since U is a fixed bijective map,

$$T(M_0, X' || M_0, Y') = T(M_0, X'' || M_0, Y'').$$

Because of rotation invariance, $U \cdot \mathcal{N}(0, I_d)$ has the same distribution as $\mathcal{N}(0, I_d)$. So,

$$X'' = \min\left\{1, \frac{C}{M_0}\right\} \cdot \|g_0\|_2 \, e_1 + \sigma \cdot C \cdot \mathcal{N}(0, I_d),\ Y'' = \sigma \cdot C \cdot \mathcal{N}(0, I_d).$$

The coordinates $(X_i'')_{i \geq 2}$ are independent of each other and M_0, X_1''. Similarly the coordinates $(Y_i'')_{i \geq 2}$ are also independent of each other and M_0, Y_1''. Moreover X_i'' and Y_i'' has the same distribution for $i \geq 2$. Therefore by Proposition 4.4,

$$T(M_0, X'' || M_0, Y'') = T(M_0, X_1'' || M_0, Y_1'') \otimes \mathrm{Id} \otimes \mathrm{Id} \otimes \cdots \otimes \mathrm{Id} = T(M_0, X_1'' || M_0, Y_1''),$$

$$\text{where } X_1'' = \min\left\{1, \frac{C}{M_0}\right\} \cdot \|g_0\|_2 + \sigma \cdot C \cdot \mathcal{N}(0, 1) \text{ and } Y_1'' = \sigma \cdot C \cdot \mathcal{N}(0, 1).$$

Let $f(M_0) = \min\left\{\frac{\|g_0\|_2}{\sigma C}, \frac{\|g_0\|_2}{\sigma M_0}\right\}$. We can further simplify this using Proposition 4.3 as:

$$\begin{aligned}
&T(M_0, X_1'' || M_0, Y_1'') \\
&= T\left(M_0, f(M_0) + \mathcal{N}(0, 1) \,||\, M_0, \mathcal{N}(0, 1)\right) \\
&= T\left(M_0, f(M_0), f(M_0) + \mathcal{N}(0, 1) \,||\, M_0, f(M_0), \mathcal{N}(0, 1)\right) \\
&= T\left(f(M_0), f(M_0) + \mathcal{N}(0, 1) \,||\, f(M_0), \mathcal{N}(0, 1)\right)
\end{aligned}$$

We can simplify $f(M_0)$ further by using the fact that $\frac{\|g_0\|}{M_0} = \|g_0\|_2 \,/\, \left(\sqrt{\frac{1}{r}\sum_{j=1}^{r} \langle g_0, v_j \rangle^2}\right)$ has the same distribution as $1/\sqrt{\frac{1}{r}\chi_r^2}$. Therefore $f(M_0)$ has the same distribution as

$$\tilde{Z} = \min\left\{\frac{\|g_0\|_2}{\sigma C}, \frac{1}{\sigma\sqrt{\frac{1}{r}\chi_r^2}}\right\}.$$

Let $Z_r = \frac{1}{\sqrt{\frac{1}{r}\chi_r^2}}$ and so $\tilde{Z} = \min\left\{\frac{\|g_0\|_2}{\sigma C}, \frac{Z_r}{\sigma}\right\}$. By Lemma 4.2,

$$T(\tilde{Z}, \mathcal{N}(\tilde{Z}, 1) || \tilde{Z}, \mathcal{N}(0, 1)) \succeq T(Z_r, \mathcal{N}(Z_r/\sigma, 1) || Z_r, \mathcal{N}(0, 1)).$$

Therefore, this proves that $T(X||Y) \succeq T(Z_r, \mathcal{N}(Z_r/\sigma, 1) || Z_r, \mathcal{N}(0, 1))$ where $Z_r = \frac{1}{\sqrt{\frac{1}{r}\chi_r^2}}$. The parametrization follows from Lemma 4.1. □

Theorem 4.1. Let $p = B/N$ where B is the batch size and N is the total number of samples. Then Algorithm 1 is g-DP with $g = \min\left\{f_p^{\otimes T}, (f_p^{\otimes T})^{-1}\right\}^{**}$ where $f_p = pf + (1-p)\mathrm{Id}$ and $f = T(Z_r, \mathcal{N}(Z_r/\sigma, 1) || Z_r, \mathcal{N}(0, 1))$.[6]

Proof. Algorithm 1 can be thought of as adaptive composition of T iterations of Algorithm 2, but where the inputs to the Algorithm 2 in each iteration is subsampled from the entire input with sampling probability $p = B/N$. And we already showed in Lemma 4.3 that Algorithm 2 satisfies f-DP with f as claimed. The rest of the proof is very similar to Theorem 3 in Bu et al. (2019) which itself builds on a similar theorem in Dong et al. (2019). It proceeds by applying Proposition 4.5 to understand the effect of subsampling and an adaptive version of the composition in Proposition 4.4 to compose the privacy curves in all the T iterations. □

One could hope to use the central limit theorem for composition from Dong et al. (2019); Bu et al. (2019) (see Proposition B.3) to find an approximate closed form expression for the final privacy curve. Unfortunately, these central limit theorems do not apply in our setting (see Proposition B.5). Instead, we numerically compute the final privacy curve obtained in Theorem 4.1.

[6] h^{**} is the double convex conjugate of h (i.e., the greatest convex lower bound for h).

REFERENCES

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

Sean Augenstein, H. Brendan McMahan, Daniel Ramage, Swaroop Ramaswamy, Peter Kairouz, Mingqing Chen, Rajiv Mathews, and Blaise Agüera y Arcas. Generative models for effective ML on private, decentralized datasets. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SJgaRA4FPH.

Borja Balle, Peter Kairouz, H. Brendan McMahan, Om Thakkar, and Abhradeep Thakurta. Privacy amplification via random check-ins. *CoRR*, abs/2007.06605, 2020. URL https://arxiv.org/abs/2007.06605.

Zhiqi Bu, Jinshuo Dong, Qi Long, and Weijie J. Su. Deep learning with gaussian differential privacy, 2019.

Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pp. 267–284, 2019.

Mia Xu Chen, Benjamin N. Lee, Gagan Bansal, Yuan Cao, Shuyuan Zhang, Justin Lu, Jackie Tsay, Yinan Wang, Andrew M. Dai, Zhifeng Chen, Timothy Sohn, and Yonghui Wu. Gmail smart compose: Real-time assisted writing. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (eds.), *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pp. 2287–2295. ACM, 2019. doi: 10.1145/3292500.3330723. URL https://doi.org/10.1145/3292500.3330723.

Xiangyi Chen, Zhiwei Steven Wu, and Mingyi Hong. Understanding gradient clipping in private SGD: A geometric perspective. *CoRR*, abs/2006.15429, 2020. URL https://arxiv.org/abs/2006.15429.

Budhaditya Deb, Peter Bailey, and Milad Shokouhi. Diversifying reply suggestions using a matching-conditional variational autoencoder. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

Jinshuo Dong, Aaron Roth, and Weijie J. Su. Gaussian differential privacy, 2019.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, pp. 265–284. Springer, 2006.

Vitaly Feldman. Does learning require memorization? a short tale about a long tail. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy (eds.), *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pp. 954–959. ACM, 2020. doi: 10.1145/3357713.3384290. URL https://doi.org/10.1145/3357713.3384290.

Ian Goodfellow. https://github.com/tensorflow/tensorflow/issues/4897#issuecomment-290997283.

Ian Goodfellow. Efficient per-example gradient computations. *arXiv preprint arXiv:1510.01799*, 2015.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.

H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=BJ0hF1Z0b.

Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pp. 263–275. IEEE, 2017.

Library Opacus.
github.com/facebookresearch/pytorch-dp.

Nicolas Papernot, Abhradeep Thakurta, Shuang Song, Steve Chien, and Úlfar Erlingsson. Tempered sigmoid activations for deep learning with differential privacy. *arXiv preprint arXiv:2007.14191*, 2020.

NhatHai Phan, Xintao Wu, Han Hu, and Dejing Dou. Adaptive laplace mechanism: Differential privacy preservation in deep learning. In *2017 IEEE International Conference on Data Mining (ICDM)*, pp. 385–394. IEEE, 2017.

Team Pytorch. Dp training on imdb.
https://github.com/pytorch/opacus/blob/master/examples/imdb_README.md.

Gaspar Rochette, Andre Manoel, and Eric W Tramel. Efficient per-example gradient computations in convolutional neural networks. *arXiv preprint arXiv:1912.06015*, 2019.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 3–18. IEEE, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Team Tensorflow. Text classification with an rnn.
https://www.tensorflow.org/tutorials/text/text_classification_rnn.

Library Tensorflow Privacy.
github.com/tensorflow/privacy.

Igor V. Tetko, David J. Livingstone, and Alexander I. Luik. Neural network studies, 1. comparison of overfitting and overtraining. *J. Chem. Inf. Comput. Sci.*, 35(5):826–833, 1995. doi: 10.1021/ci00027a006. URL https://doi.org/10.1021/ci00027a006.

86 TFissue. Fast per example gradient support.
https://github.com/tensorflow/privacy/issues/86.

Om Thakkar, Galen Andrew, and H. Brendan McMahan. Differentially private learning with adaptive clipping. *CoRR*, abs/1905.03871, 2019. URL http://arxiv.org/abs/1905.03871.

Jamie Townsend. A new trick for calculating Jacobian vector products.
https://j-towns.github.io/2017/06/12/A-new-trick.html.

Liyao Xiang, Jingbo Yang, and Baochun Li. Differentially-private deep learning from an optimization perspective. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 559–567. IEEE, 2019.

Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *international conference on machine learning*, pp. 1–9, 2015.

Yingxue Zhou, Zhiwei Steven Wu, and Arindam Banerjee. Bypassing the ambient dimension: Private SGD with gradient subspace identification. *CoRR*, abs/2007.03813, 2020. URL https://arxiv.org/abs/2007.03813.

A DP-SGD AND DP-ADAM-JL

For completeness, we provide pseudo-code for DP-SGD and DP-Adam-JL used in our experiments. DP-Adam-JL satisfies exactly the same privacy bounds as DP-SGD-JL.

Algorithm 3 Differentially private Adam using JL projections (DP-Adam-JL)

Input: Examples $\{x_1, x_2, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \mathbb{E}_{i\in[N]}[\mathcal{L}(\theta; x_i)]$, initialization θ_0.
Parameters: number of iterations T, momentum parameters (β_1, β_2), noise scale σ, batch size B, clipping norms $(C_1, C_2, \ldots, C_T)$, number of JL projections r.

for $t = 1$ to T **do**
  Sample $S_t = \{X_1, X_2, \ldots, X_B\} \subset \{x_1, x_2, \ldots, x_N\}$ uniformly at random.
  Define $F(\theta) = (\mathcal{L}(\theta; X_1), \mathcal{L}(\theta; X_2), \ldots, \mathcal{L}(\theta; X_B))$

  Sample $v_1, v_2, \ldots, v_r \leftarrow \mathcal{N}(0, I_d)$ (where $\theta \in \mathbb{R}^d$) ▷ JL projections to estimate per-sample gradient norm
  for $j = 1$ to r **do**
    $(P_{1j}, P_{2j}, \ldots, P_{Bj}) \leftarrow \nabla_\theta F(\theta_{t-1}) \cdot v_j$ (using `jvp`) ▷ Note that $P_{ij} = \langle \nabla_\theta \mathcal{L}(\theta_{t-1}; X_i), v_j \rangle$
  end for
  for $i \in B_t$ **do**
    $M_i \leftarrow \sqrt{\frac{1}{r}\sum_{j=1}^{r} P_{ij}^2}$ ▷ M_i is an estimate for $\|\nabla_\theta \mathcal{L}(\theta_{t-1}; X_i)\|_2$.
  end for
  Define $\widetilde{\mathcal{L}}(\theta) = \frac{1}{B}\sum_{i\in B} \min\{1, \frac{C_t}{M_i}\} \cdot \mathcal{L}(\theta; X_i)$. ▷ Clip the per-sample gradients

  $\tilde{g}_t \leftarrow \nabla_\theta \widetilde{\mathcal{L}}(\theta_{t-1}) + \frac{\sigma \cdot C_t}{B} \cdot \mathcal{N}(0, I_d)$ ▷ Add noise to the average of clipped gradients

  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)\,\tilde{g}_t$
  $u_t \leftarrow \beta_2 u_{t-1} + (1-\beta_2)\,\tilde{g}_t^2$
  $\eta_t \leftarrow m_t / (\sqrt{u_t} + 10^{-8})$
  Update $\theta_t \leftarrow \theta_{t-1} - \eta_t \tilde{g}_t$
end for
Output: $\theta_0, \theta_1, \theta_2, \ldots, \theta_T$

Algorithm 4 Differentially private SGD (DP-SGD) from Abadi et al. (2016)

Input: Examples $\{x_1, x_2, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \mathbb{E}_{i\in[N]}[\mathcal{L}(\theta; x_i)]$, initialization θ_0.
Parameters: number of iterations T, learning rates $(\eta_1, \eta_2, \ldots, \eta_T)$, noise scale σ, batch size B, clipping norms $(C_1, C_2, \ldots, C_T)$, number of JL projections r.

for $t = 1$ to T **do**
  Sample $S_t = \{X_1, X_2, \ldots, X_B\} \subset \{x_1, x_2, \ldots, x_N\}$ uniformly at random.
  for $i = 1$ to B **do**
    $\hat{g}_i \leftarrow \min\{1, \frac{C_t}{\|\nabla_\theta \mathcal{L}(\theta_{t-1}; X_i)\|_2}\} \cdot \nabla_\theta \mathcal{L}(\theta_{t-1}; X_i)$ ▷ Clip the per-sample gradients to ℓ_2-norm at most C_t
    $\tilde{g}_t \leftarrow \frac{1}{B}\left(\sum_{i\in B_t} \hat{g}_i\right) + \frac{\sigma C_t}{B} \cdot \mathcal{N}(0, I_d)$ ▷ Average the clipped gradients and add noise
    Update $\theta_t \leftarrow \theta_{t-1} - \eta_t \tilde{g}_t$
  end for
end for
Output: $\theta_0, \theta_1, \ldots, \theta_T$

B PRIVACY ANALYSIS: MISSING PROOFS

In this section, we will prove Lemma 4.1 and Lemma 4.2. Before we start the proofs, we will need some preliminaries.

Though Eqn (1) defining the tradeoff curve requires us to take an infimum over a large class of tests, Neyman-Pearson lemma gives a very simple form for the optimal tests to use.

Proposition B.1 (Neyman-Pearson lemma)**.** Let P, Q be two continuous distributions over some domain X. The type I error vs type II error tradeoff function between P and Q is attained by the Neyman-Pearson tests which are a single parameter family of tests of the form $\phi_t : X \to [0, 1]$ defined as:

$$\phi_t(x) = \begin{cases} 1 & \text{if } P(x) \leq tQ(x) \\ 0 & \text{if } P(x) > tQ(x). \end{cases}$$

Proposition B.2 (Dong et al. (2019))**.** For $\mu \geq 0$, define $G_\mu(\alpha) = T(\mathcal{N}(\mu, 1)||\mathcal{N}(0, 1)) = T(\mathcal{N}(0, 1)||\mathcal{N}(\mu, 1))$. Then

$$G_\mu(\alpha) = \Phi(-\mu + \Phi^{-1}(1 - \alpha)) = \Phi(-\mu - \Phi^{-1}(\alpha))$$

where Φ is the CDF of standard Gaussian.

Corollary B.1. Let $\mu \geq \mu' \geq 0$. Then $T(N(\mu, 1)||N(0, 1)) \preceq T(N(\mu', 1)||N(0, 1))$.

Corollary B.2. For $\mu \geq 0$, $G_\mu(\alpha) = T(\mathcal{N}(\mu, 1)||\mathcal{N}(0, 1))$ can be parametrized by $t \in \mathbb{R}$ as $G_\mu(\alpha(t)) = \beta(t)$ where:

$$\alpha(t) = \Phi\left(-\frac{t}{\mu} - \frac{\mu}{2}\right),\ \beta(t) = \Phi\left(\frac{t}{\mu} - \frac{\mu}{2}\right).$$

Proof. This follows from Proposition B.2.

$$\begin{aligned} &\Phi^{-1}(\alpha(t)) = -\frac{t}{\mu} - \frac{\mu}{2} \\ \Rightarrow &-\mu - \Phi^{-1}(\alpha(t)) = \frac{t}{\mu} - \frac{\mu}{2} \\ \Rightarrow &\Phi\left(-\mu - \Phi^{-1}(\alpha(t))\right) = \Phi\left(\frac{t}{\mu} - \frac{\mu}{2}\right) := \beta(t). \end{aligned}$$

□

Note that Corollary B.2 is a special case of Lemma 4.1 which we will prove now using the Neyman-Pearson lemma.

Proof of Lemma 4.1. Denote $P := Z \times N(0, 1), Q := Z \times N(Z, 1)$. From Neyman-Pearson lemma (Proposition B.1), the type I/II errors are

$$\begin{aligned} \alpha(t) &= \Pr_{(z,x)\sim P}\left[\frac{Q(z, x)}{P(z, x)} \geq e^t\right] \\ &= \Pr_{z\sim Z, x\sim\mathcal{N}(0,1)}\left[\exp\left(\frac{x^2}{2} - \frac{(x - z)^2}{2}\right) \geq e^t\right] \\ &= \Pr_{z\sim Z, x\sim\mathcal{N}(0,1)}\left[\frac{x^2}{2} - \frac{(x - z)^2}{2} \geq t\right] \\ &= \Pr_{z\sim Z, x\sim\mathcal{N}(0,1)}\left[zx - z^2/2 - t \geq 0\right] \\ &= \mathbb{E}_Z\left[\Phi\left(-\frac{t}{Z} - \frac{Z}{2}\right)\right] \end{aligned}$$

and

$$\begin{aligned} \beta(t) &= \Pr_{(z,x)\sim Q}\left[\frac{Q(z, x)}{P(z, x)} < e^t\right] \\ &= \Pr_{z\sim Z, x\sim\mathcal{N}(z,1)}\left[\exp\left(\frac{x^2}{2} - \frac{(x - z)^2}{2}\right) < e^t\right] \\ &= \Pr_{z\sim Z, x\sim\mathcal{N}(z,1)}\left[\frac{x^2}{2} - \frac{(x - z)^2}{2} < t\right] \\ &= \Pr_{z\sim Z, x\sim\mathcal{N}(0,1)}\left[\frac{(x + z)^2}{2} - \frac{x^2}{2} < t\right] \\ &= \Pr_{z\sim Z, x\sim\mathcal{N}(0,1)}\left[zx + z^2/2 - t < 0\right] \\ &= \mathbb{E}_Z\left[\Phi\left(\frac{t}{Z} - \frac{Z}{2}\right)\right]. \end{aligned}$$

Since $\Phi'(z) = \phi(z) = \frac{1}{\sqrt{2\pi}} \exp(-z^2/2)$ which is the Gaussian PDF, the derivatives are given by

$$\alpha'(t) = \mathbb{E}_Z\left[-\frac{1}{Z}\phi\left(-\frac{t}{Z} - \frac{Z}{2}\right)\right] = -\frac{\exp(-t/2)}{\sqrt{2\pi}}\mathbb{E}_Z\left[\frac{1}{Z}\exp\left(-\frac{t^2}{2Z^2} - \frac{Z^2}{8}\right)\right]$$
$$\beta'(t) = \mathbb{E}_Z\left[\frac{1}{Z}\phi\left(\frac{t}{Z} - \frac{Z}{2}\right)\right] = \frac{\exp(t/2)}{\sqrt{2\pi}}\mathbb{E}_Z\left[\frac{1}{Z}\exp\left(-\frac{t^2}{2Z^2} - \frac{Z^2}{8}\right)\right].$$

□

Proof of Lemma 4.2. Let $f_z = T(N(z,1)||N(0,1))$ be a family of privacy curves parametrized by $z \in \mathbb{R}^{\geq 0}$. By Corollary B.1, $f_z \preceq f_{\widetilde{z}}$ if $z \geq \widetilde{z} \geq 0$. Let $\alpha_z(t), \beta_z(t)$ be the parametrization of $f_z(t)$ given by:

$$\alpha_z(t) = \Phi\left(-\frac{t}{z} - \frac{z}{2}\right),\ \beta_z(t) = \Phi\left(\frac{t}{z} - \frac{z}{2}\right).$$

By Lemma 4.1 and Equation (2), we have $\varepsilon_z(t) = t$ and $\delta_z(t) = 1 - \beta_z(t) - \alpha_z(t)e^t$. Therefore $\delta_z(\varepsilon) = 1 - \beta_z(\varepsilon) - \alpha_z(\varepsilon)e^{\varepsilon}$. If $\delta_Z(\varepsilon)$ be the (ε, δ) privacy curve for $T(Z, N(Z,1)||Z, N(0,1))$, by Lemma 4.1, we have

$$\delta_Z(\varepsilon) = 1 - \mathbb{E}_{z\sim Z}[\beta_z(\varepsilon)] - \mathbb{E}_{z\sim Z}[\alpha_z(\varepsilon)]e^{\varepsilon} = \mathbb{E}_{z\sim Z}[\delta_z(\varepsilon)].$$

When $z \geq \widetilde{z}$, we have $f_z \preceq f_{\widetilde{z}}$ which implies that $\delta_z(\varepsilon) \leq \delta_{\widetilde{z}}(\varepsilon)$. Therefore by using the coupling $(Z, \widetilde{Z})$ with $Z \geq \widetilde{Z}$, we have $\mathbb{E}_{z\sim Z}[\delta_z(\varepsilon)] \geq \mathbb{E}_{\widetilde{z}\sim\widetilde{Z}}[\delta_{\widetilde{z}}(\varepsilon)]$. Therefore $\delta_Z(\varepsilon) \geq \delta_{\widetilde{Z}}(\varepsilon)$ which further implies that $T(Z, N(Z,1)||Z, N(0,1)) \preceq T(\widetilde{Z}, N(\widetilde{Z},1)||\widetilde{Z}, N(0,1))$. □

B.1 Central limit theorem for privacy curve composition

In this section, we show that we cannot apply the central limit theorems from Dong et al. (2019); Bu et al. (2019) to find a closed form expression for the privacy of our algorithms.

Proposition B.3 (Bu et al. (2019))**.** Suppose f is a trade-off function such that $f(0) = 1$, $f(x) > 0$ for all $0 \leq x < 1$ and $\int_0^1 (f'(\alpha) + 1)^4 d\alpha < \infty$. Let $f_p = pf + (1-p)\mathrm{Id}$. Furthermore, assume $p\sqrt{T} \to \nu$ as $T \to \infty$ for some constant $\nu > 0$. Then we have the uniform convergence

$$f_p^{\otimes T} \to G_{\nu\sqrt{\chi^2(f)}}$$

as $T \to \infty$ where $\chi^2(f) = \int_0^1 f'(\alpha)^2 d\alpha - 1$.

Proposition B.4. The PDF of $Z_r = \frac{1}{\sqrt{\frac{1}{r}\chi^2_r}}$ is given by

$$Z_r(a) = \frac{1}{C_r} \cdot \frac{1}{a^{r+1}} \cdot \exp\left(-\frac{r}{2a^2}\right)$$

where $C_r = \int_0^\infty \frac{1}{a^{r+1}} \cdot \exp\left(-\frac{r}{2a^2}\right) = \left(\frac{2}{r}\right)^{r/2} \cdot \frac{\Gamma(r/2)}{2}$ is the normalization constant and

$$\Gamma(r/2) = \begin{cases} \left(\frac{r}{2} - 1\right)! & \text{if } r \text{ is even,} \\ \frac{\sqrt{\pi}(r-1)!}{2^{r-1}\cdot\left(\frac{r-1}{2}\right)!} & \text{if } r \text{ is odd.} \end{cases}$$

Proposition B.5. Let $f = T(Z_r, \mathcal{N}(Z_r/\sigma, 1)||Z_r, \mathcal{N}(0,1))$. Then $\chi^2(f) = \infty$.

Proof. Using the parametrization of f from Lemma 4.1, we can compute $\chi^2(f)$ as follows.

$$
\begin{aligned}
&\chi^2(f) \\
&= \int_0^1 f'(\alpha)^2 d\alpha - 1 \\
&= -\int_{-\infty}^{\infty} \exp(2t)\alpha'(t)dt - 1 \\
&= \int_{-\infty}^{\infty} \exp(2t) \cdot \frac{\exp(-t/2)}{\sqrt{2\pi}} \mathbb{E}_{Z_r}\left[\frac{\sigma}{Z_r}\exp\left(-\frac{t^2\sigma^2}{2Z_r^2} - \frac{Z_r^2}{8\sigma^2}\right)\right] dt - 1 \\
&= \mathbb{E}_{Z_r} \int_{-\infty}^{\infty} \frac{\exp(3t/2)}{\sqrt{2\pi}} \cdot \frac{\sigma}{Z_r}\exp\left(-\frac{t^2\sigma^2}{2Z_r^2} - \frac{Z_r^2}{8\sigma_r^2}\right) dt - 1 \\
&= \mathbb{E}_{Z_r}\left[\exp\left(\frac{Z_r^2}{\sigma^2}\right)\right] - 1 \\
&= \int_{a=0}^{\infty} \frac{1}{C_r \cdot a^{r+1}} \exp\left(-\frac{r}{2a^2}\right) \cdot \exp\left(\frac{a^2}{\sigma^2}\right) da - 1. \\
&= \infty
\end{aligned}
$$

since the integrand tends to infinity as $a \to \infty$. □

C Effect of JL Dimension on Privacy

In this section, we analyze how JL dimension effects the privacy guarantees of DP-SGD-JL (or DP-Adam-JL as they have the same privacy guarantee). Since the per-sample gradient norm estimations get more accurate with JL dimension, it is clear that the privacy of DP-SGD-JL should converge to that of DP-SGD-Vanilla for large JL dimension. This can be see from Figure 8. Another interesting observation is that the privacy loss is not monotone w.r.t JL dimension. This is happening because it is not always true that $T(Z_r, \mathcal{N}(Z_r/\sigma, 1)||Z_r, \mathcal{N}(0,1)) \succeq T(Z_{r'}, \mathcal{N}(Z_{r'}/\sigma, 1)||Z_{r'}, \mathcal{N}(0,1))$ for $r > r'$. Thus sometimes, smaller JL dimension can be more private and faster than larger JL dimensions.

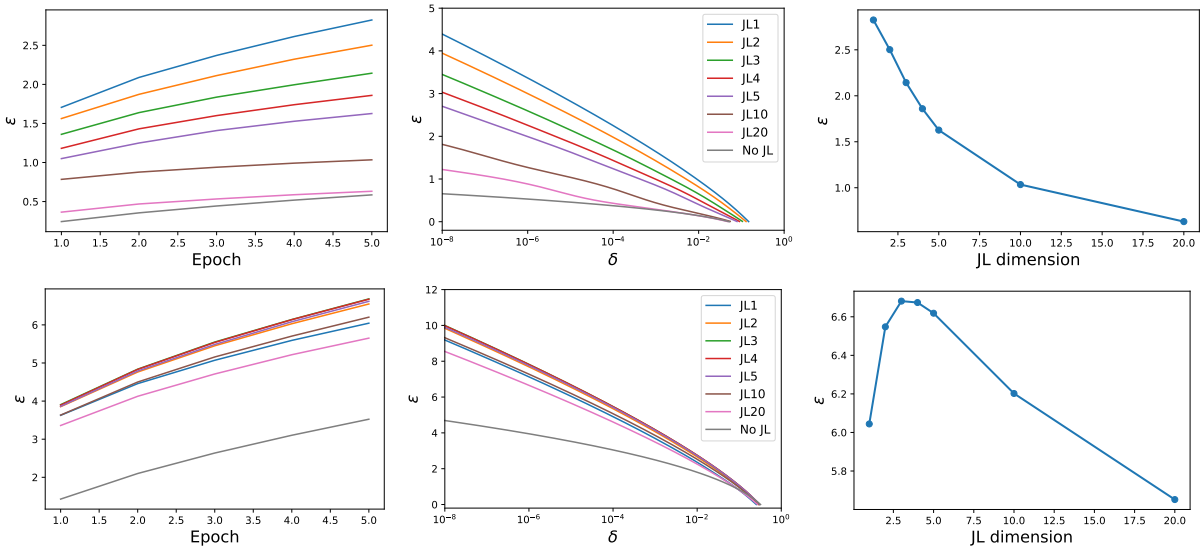

Figure 8: Privacy loss of DP-SGD-JL up to 5 epochs. Here 'No JL' represents the original the privacy loss of DP-SGD-Vanilla. We set $N = 60000, B = 256, \sigma = 1.3$) for the upper panel, and $N = 25000, B = 64, \sigma = 0.5$ for the lower panel.

D ADDITIONAL EXPERIMENTS

D.1 MEMORY ANALYSIS

In this section, we focus our attention to the memory footprint for various algorithms discussed in the paper. From Table 4, we see that even for small neural networks, DP-SGD-Outer may require too much memory to be deployable. In contrast, DP-SGD-JL only needs as much memory as non-DP SGD.

Optimizer	2-layer CNN	AlexNet	
	Sec/epoch	Sec/epoch	Memory (of 16GB)
Non-DP SGD	1	14	12%
DP-SGD-Vanilla[7]	50	332	38%
DP-SGD-Outer[8]	12	85	75%
DP-SGD-JL(1) (`jvp`)	3	43	15%
DP-SGD-JL(10) (`jvp`)	20	182	15%
DP-SGD-JL(1) (`double vjp`)	14	26	22%
DP-SGD-JL(10) (`double vjp`)	22	105	22%

Table 4: 2-layer CNN in OpacusTensorflow Privacy with 26,010 parameters and batch size of 256. AlexNet in Krizhevsky et al. (2012) (5 convolutional layers and 3 fully-connected layers) with 21,598,922 parameters and batch size of 64. The `double vjp` implementation is in PyTorch and the rest of them are in Tensorflow.

D.2 LARGE MODELS

To illustrate the scalability of our algorithms, we perform experiments on CIFAR10 with a large neural network, VGG13 (Simonyan & Zisserman (2014)), and a large batch size. In contrast to previous works Abadi et al. (2016); Xiang et al. (2019); Phan et al. (2017), we do not use data-augmentation nor pre-training.

	Non-DP SGD	DP-SGD-Vanilla	DP-SGD-Outer	DP-SGD-JL(1)	DP-SGD-JL(10)
Pytorch	13	-	Out of Memory	212	1946
Tensorflow	12	1625	-	20	106

Table 5: VGG13 (10 convolutional layers and 3 fully-connected layers). This network contains 32,394,562 parameters. During training, $\sigma = 0.9, R = 3, B = 256, \eta = 0.001, E = 20$.

For for VGG13, we observe that all existing acceleration methods fail: DP-SGD-Outer cannot fit in the memory. But unfortunately, overall no algorithm achieves reasonable privacy-vs-accuracy tradeoff; almost all the algorithms achieve less than 20% accuracy. Yet, these experiments are serve to demonstrate that our algorithms can indeed provide significant speed-up on large models.

Finally we note the difference in implementing DP-SGD-JL with `jvp` or `vjp`. When training with DP-SGD-JL, Tensorflow `jvp` takes 11 seconds per JL dimension per epoch but Pytorch double `vjp` takes about 180 seconds. Therefore, it is preferred to adopt `jvp` on large neural networks.