Private Ad Modeling with DP-SGD

Carson Denison[2], Badih Ghazi[1,*], Pritish Kamath[1], Ravi Kumar[1], Pasin Manurangsi[1], Krishna Giri Narra[1], Amer Sinha[1], Avinash Varadarajan[1] and Chiyuan Zhang[1]

[1]*Google*

[2]*Anthropic. Work done while at Google.*

Abstract

A well-known algorithm in privacy-preserving ML is differentially private stochastic gradient descent (DP-SGD). While this algorithm has been evaluated on text and image data, it has not been previously applied to ads data, which are notorious for their high class imbalance and sparse gradient updates. In this work we apply DP-SGD to several ad modeling tasks including predicting click-through rates, conversion rates, and number of conversion events, and evaluate their privacy-utility trade-off on real-world ads datasets. Our work is the first to empirically demonstrate that DP-SGD can provide both privacy and utility for ad modeling tasks.

Keywords

differential privacy, model training, ad models, CTR

1. Introduction

With increasing focus on privacy on the Web and mobile apps, and given the signal loss due to cookie deprecation by several platforms, there has been a great need for privacy-preserving ML methods applied to ad prediction tasks. The most widely used predictive models in digital advertising are typically trained on user data pertaining to one or multiple sites/apps, and are used by ad technology providers (Ad Techs) to optimize the placement of digital ads.

Differential Privacy (DP) [1, 2] has emerged as a popular notion of privacy that is extensively studied in the research community and widely deployed in industrial applications, especially for training ML models with provable privacy guarantees. The main goal of DP training in ad modeling is to mitigate the privacy risks. For instance, the training examples for ad prediction models often depend on cross-site information. In the absence of privacy guardrails, the weights of the trained model could reveal, e.g., the browsing history of users. Such leakage of user information present in the training data is mitigated by DP training methods. Intuitively, DP achieves a trade-off between privacy and utility by allowing statistical analysis and learning based on population-wide properties, while limiting the influence of (private) information from any individual user on the final output or model.

A training algorithm takes the set of examples as the input and produces the (trained weights of the) model as the output. For deep learning, various algorithms were proposed to privatize a learning pipeline, such as PATE [3, 4] and DP-FTRL [5]. But the most widely used generic algorithm is DP stochastic gradient descent (DP-SGD) [6], which goes back to the work of Abadi et al. [7]. We focus on DP-SGD in this paper. At a high level, DP-SGD works by clipping the norm of the per-example gradient to limit the influence of each example, and then adding Gaussian noise to the mini-batch averaged gradient to achieve DP. Since most deep neural network models are trained using SGD or variants such as Adam, DP-SGD can be adapted to any existing training pipeline with minimum modification by just replacing the optimizer.

However, a direct application of DP-SGD can lead to significant utility (i.e., accuracy) loss and a large computational overhead, in practice. In fact, until recently, it was not clear if DP-SGD was suitable for large-scale deep learning. Recent studies and success stories mostly focused on vision [8, 9] and text [10, 11] problems.

In this work, we present a systematic study of DP-SGD on ad prediction tasks. These tasks have highly unbalanced label distributions, categorical features with extremely sparse signals, and models with large embedding layers. Such properties make ads prediction more challenging than many other tasks. These difficulties meant that DP-SGD was commonly considered infeasible except for trivially large privacy budgets. In contrast, we demonstrate that it is possible to train private models with DP-SGD with only a small utility drop even in the high privacy regime. For example, for a click-through prediction task, the AUC loss is increased by only 15.8% relative to a non-private baseline ($0.1943 \rightarrow 0.2250$) even at a privacy budget of $\varepsilon = 0.5$. Furthermore, we show that with care, the computation and memory overheads of DP-SGD can be made almost identical to that of non-private training.

AdKDD'23

badihghazi@gmail.com (B. Ghazi); pritishk@google.com (P. Kamath); ravi.k53@gmail.com (R. Kumar); pasin@google.com (P. Manurangsi); krishnanarra@google.com (K. G. Narra); amersinha@google.com (A. Sinha); avaradar@google.com (A. Varadarajan); chiyuan@google.com (C. Zhang)

We will discuss the ideas that make these results possible, such as large batch training, efficient per-example gradient norm bounding, and improved privacy accounting. We also provide a comparison of DP-SGD to LabelDP [12, 13], a DP notion that protects only the labels and not the features. To the best of our knowledge, ours is the first systematic study on training large deep neural networks privately for ad prediction tasks. We hope our results serve as an optimistic example towards DP training of large ad prediction neural networks. We also hope our detailed studies provide useful information for practitioners to improve the utility and minimize the overhead of DP training.

2. Background

Let $\mathcal{A}$ be a (stochastic) training algorithm that produces a model given a labeled training set. We call two training sets *neighboring* if they differ on a single labeled example.

Definition 1 (Differential Privacy)**.** *Let $\varepsilon \geq 0, \delta \in [0, 1]$. A randomized training algorithm $\mathcal{A}$ is* (ε, δ)-differentially private ((ε, δ)-DP) *if for all $S \subseteq \mathrm{Range}(\mathcal{A})$ and all neighboring training sets D, D', it holds that*

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

DP-SGD [7] is the most widely used DP training algorithm for a deep learning pipeline. Let f_θ be a neural network with trainable weights θ, and $\{(x_1, y_1), \ldots, (x_B, y_B)\}$ be a random mini-batch of training examples. Let $\ell_i = \ell(f_\theta(x_i), y_i)$ be the loss on the ith example and let $\bar{\ell} = \frac{1}{B}\sum_{i=1}^{B} \ell_i$ be the average loss. Standard training algorithms compute the *average gradient* $\nabla_\theta \bar{\ell}$ and update θ with an optimizer such as SGD or Adam. In DP-SGD, the per-example gradients $\nabla_\theta \ell_i$ are computed, and then rescaled to have a maximum ℓ_2-norm C. The average of the norm-bounded per-example gradients is perturbed by adding independent Gaussian noise to each coordinate, and is subsequently passed to the optimizer. The privacy parameters ε and δ depend on the noise multiplier, and other parameters such as the batch size and training steps; they can be estimated via privacy accounting [7].

3. Summary of Main Results

We focus on three common predictions tasks for which Ad Techs build ML models.

- pCTR: predict the click-through rate for an ad.
- pCVR: predict the conversion rate for an ad click; here, only whether a conversion takes place matters, regardless of the number of conversions.

Table 1
DP-SGD results (average over five runs) on three different ads prediction datasets. Each row show the results under a specific privacy budget ε. It is common [7, 14, 15] to set $\delta = \mathcal{O}(1/N)$, where N is the number of training examples. In this paper, we fix $\delta = 1/N$. The percentages are relative loss increment calculated as $(L_\varepsilon - L_\infty)/L_\infty$, where L_ϵ is the loss for the (ϵ, δ)-DP model and L_∞ is the loss for the non-private ($\varepsilon = \infty$) baseline. We use AUC loss (i.e., $1 - \mathrm{AUC}$) for pCTR and pCVR, and Poisson log loss for pConvs.

Privacy Budget (ε)	Relative Loss Increment (%)		
	pCTR	pCVR	pConvs
0.5	16.11	9.99	97.04
1.0	13.58	9.51	85.71
3.0	8.77	8.55	68.19
5.0	7.40	7.84	67.14
10.0	6.27	7.28	60.64
30.0	5.67	6.45	46.00
50.0	5.56	5.84	41.20

- pConvs: predict the expected number of conversions after an ad click; this is a regression problem against integer count labels.

We evaluate pCTR on the public Criteo dataset [16], and pCVR, pConvs on a proprietary dataset. We train the binary classification problems on pCTR and pCVR with the binary cross entropy loss and report the test *AUC loss* (i.e., $1 - \mathrm{AUC}$), and the regression problem on pConvs with the Poisson log loss (PLL), and report the test PLL. Let $f_\theta(x)$ be the scalar prediction (i.e., the logit value) of the neural network, and y be the integer counting label. *PLL* is defined as $\ell(f_\theta, (x, y)) := \exp(f_\theta(x)) - y f_\theta(x)$. In all our experiments with (ε, δ)-DP, we set δ to be $1/N$, where N is the number of training examples in the dataset.

The percentage increases in loss at various privacy budgets are presented relative to a non-private baseline[1] in Table 1. We found DP-SGD can properly train the models for all three tasks with a reasonable loss gap, even for very high privacy (e.g., $\varepsilon < 1$) regimes. Furthermore, when implemented carefully, the computation and memory overheads can be minimized, allowing a training throughput similar to the non-private baseline. Next, we present in detail the techniques that enabled optimal privacy-utility trade-off and minimum memory/computation overhead, respectively.

[1] Our non-private AUC loss for the pCTR task is 0.1943. We are unable to report the absolute non-private baseline losses for pCVR and pConvs on the proprietary datasets due to confidentiality.

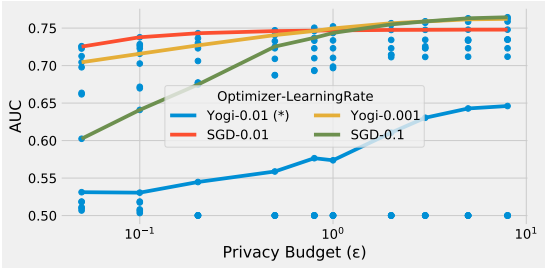


Figure 1: AUC under different (optimizer, learning rate) combinations on Criteo pCTR. Each dot is the average test AUC of 5 random runs with a specific combination of ε, optimizer, and learning rate. We evaluate the following optimizers: SGD, Yogi [17], Adagrad [18], Adam [19], AdamW [20]; and learning rates: 0.001, 0.01, 0.1. The blue line (Yogi-0.01) is with the optimal hyperparameters used in the non-private baseline.

4. Privacy-Utility Trade-off

One of the main obstacles for adopting DP-SGD in real-world deep learning pipelines is the potential deterioration of the model performance. Norm-rescaling and Gaussian noise introduce, respectively, bias and variance to gradient estimation. As a result, the trained models have lower performance. In this section, we study these challenges in detail by using the training setup from the non-private baseline as the starting point, and describe various improvements that lead to our main results. We state our results for Criteo pCTR, but the key points also hold on other prediction tasks.

4.1. Hyperparameter Tuning

The hyperparameters for training the non-private baseline models in real-world applications are typically extensively tuned. However, those hyperparameters are not necessarily the best for training with DP. Figure 1 plots the AUC of DP trained models under different optimizer and learning rate configurations. Specifically, the blue line shows that directly reusing the optimal hyperparameters for the non-private baseline leads to significant utility gap comparing to the best hyperparameters re-tuned under DP training. Note that the optimal hyperparameters also depend on ε: SGD with learning rate 0.1 performs better than with learning rate 0.01 in the low privacy (large ε) regime but worse in the high privacy (small ε) regime. For simplicity, we choose a single configuration (SGD-0.01) for the rest of the study; this already improves the AUC significantly compared to using the non-private hyperparameters[2].

[2]Ideally, hyperparameter tuning should be performed with DP, e.g., using [21]. However, following most prior work on DP training, we ignore the DP cost for hyperparameter tuning.

Table 2
AUC under different per-example norm bound (C) and privacy budget (ε) on Criteo pCTR.

ε	$C = 1.0$	$C = 3.0$	$C = 30.0$
0.5	$.7498 \pm .0011$	$.7441 \pm .0014$	$.5000 \pm .0000$
3.0	$.7524 \pm .0010$	$.7610 \pm .0010$	$.6971 \pm .0021$
8.0	$.7528 \pm .0010$	$.7629 \pm .0010$	$.7473 \pm .0009$

4.2. Bias-Variance Trade-off

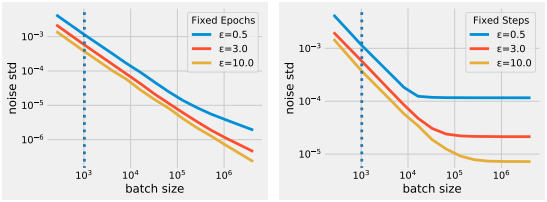
With fixed noise multiplier (and other hyperparameters like the training steps), the norm bound C for each per-example gradient allows a bias-variance trade-off in the gradient estimation for a fixed ε. Specifically, increasing C reduces bias but increases the variance in gradient estimation due to the addition of more noise. A proper choice of C improves the quality of trained model. For example, Table 2 shows that in a low privacy regime (where the noise multiplier is smaller), increasing C leads to better model performance because the added noise is already small. On the other hand, a large C can hurt performance in the high privacy regime by requiring large amounts of noise. Adaptively choosing the norm bound C has been studied in the literature [22, 23].

4.3. Large Batch Training

Batch size is an important hyperparameter that affects different aspects of DP training. Specifically, increasing the batch size leads to a larger subsampling ratio, which implies larger noise multiplier for the same ε. On the other hand, increasing the batch size also reduces the effective noise that is added to the average gradient because the noised sum of gradients are divided by the batch size [7, Algorithm 1]. Moreover, when fixing the number of training epochs, varying the batch size also changes the number of training steps, which affects both the privacy accounting and model utility.

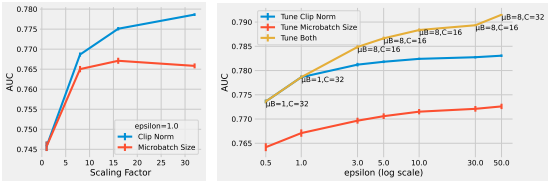
In Figure 2, we plot the relationship between the Gaussian noise standard deviation (std) required to guarantee a certain ε, when the batch size changes. In Figure 2(a), we fix the total number of passes (epochs) over the data; the noise std continues to decrease as the batch size increases beyond 10^6. However, this is not very realistic, because with fixed training epochs, increased batch sizes lead to decreased number of training steps. Modern neural networks trained with stochastic optimizers usually require a minimum number of steps to fit the data well. In Figure 2(b), we plot the relation under the condition of fixed number of training steps. In this case, the benefit of large batch sizes plateaus.

Since SGD optimization is itself stochastic, reducing the noise std might not improve the model *ad infini-*



(a) Fixed epochs (b) Fixed steps

Figure 2: Relationship between batch size and noise std for fixed **(a)** training epochs and **(b)** training steps. The dotted line shows the batch size (1024) in the non-private baseline.



(a) $\varepsilon = 1.0$ (b) Tuning for each ε

Figure 3: The impact on utility by tuning the norm bound C and microbatch size B_μ individually, and jointly, for Criteo pCTR. The search range for both C and B_μ are $\{1, 8, 16, 32\}$.

Table 3
AUC with large batch size and training epochs on Criteo pCTR. The results can be compared with Table 2 (batch size 1024, 8 epochs).

ε	batch size	epochs	C	AUC
0.5	16,384	64	30	0.7740 ± 0.0003
0.5	8,192	32	30	0.7689 ± 0.0003
3.0	16,384	64	30	0.7810 ± 0.0003
3.0	8,192	32	30	0.7782 ± 0.0003
8.0	16,384	64	30	0.7818 ± 0.0003
8.0	8,192	32	30	0.7799 ± 0.0004

tum. However, larger batches reduce noise, which allows the norm bound C to be increased while keeping the total added noise and privacy level fixed. Increasing C can yield performance gains. Specifically, Table 3 shows the model performances on Criteo pCTR with increased batch sizes, training epochs, and norm bound C. Comparing with Table 2, large batch sizes lead to significant performance boost. Note that for $\varepsilon = 0.5$, the model cannot properly train (AUC $= 0.5$) for $C = 30$, but reaches AUC $= 0.77$ when the batch size increases $16\times$. We note that large batches have previously been used to improve DP-SGD for vision and language models [10, 8, 9].

4.4. Microbatching

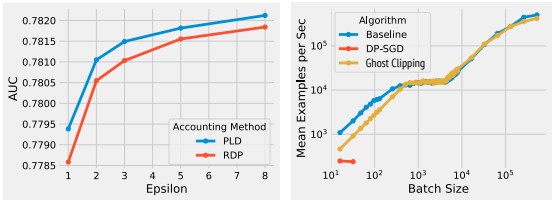
Microbatching was originally used to mitigate the computation and memory overhead of vanilla DP-SGD implementations [7]. It works by partitioning each mini-batch of B training examples into "microbatches" of size B_μ, and performing ℓ_2-norm rescaling on the average gradient of each microbatch (as oppose to the per-example gradient). With large B_μ, even a vanilla DP-SGD implementation could be quite efficient because microbatch average gradients can be computed with standard backpropagation API. However, because group norm bounding changes the sensitivity of the mean gradient query, the magnitude of Gaussian noises scale up by a factor of B_μ under the same privacy guarantee. As a result, large B_μ generally leads to worse model utility.

In our case, with an efficient implementation (Section 5), we do not need to use microbatching. However, we empirically found that small B_μ could improve the model utility. One potential reason is that this allows a different kind of bias-variance trade-off from changing the norm bound C (Section 4.2). The increased noise leads to higher variance in gradient estimation. On the other hand, averaging the gradients within the microbatch before clipping could potentially reduce the bias when there are cancellations. For example, averaging B_μ i.i.d. Gaussian vectors reduces the expected norm by a factor of $1/\sqrt{B_\mu}$. The cancellation of gradient vectors is hard to characterize theoretically, but as shown in Figure 3(a), increasing B_μ moderately indeed improve the utility for Criteo pCTR. Similar to C, the maximum tolerable value before it starts hurting the utility increases with ε. On the other hand, C and B_μ seem to be helping with bias reduction in different ways, because increasing each of them by the same scaling factor leads to different AUC, even though the Gaussian noise scale (i.e., the variance) would be increased by the same amount. As a result, we could combine the two factors to further boost the utility, as verified in Figure 3(b).

4.5. Tighter Privacy Accounting

Privacy accounting estimates the privacy budget ε for a DP-SGD trained model according to the specific hyperparameters such as the noise std, the norm bound C, the number of training steps, etc. Rényi Differential Privacy (RDP) accounting has been the most widely used approach in DP-SGD since the original paper [7]. With RDP, ε can be computed using only the number of epochs, the batch size, and the noise std.

In this paper, we explore latest advances in privacy accounting to provide tighter estimates. Specifically, several recent works [24, 25] studied numerical methods for estimating the privacy parameters of a DP mechanism to an arbitrary accuracy using the notion of *privacy loss distributions* (PLD). A crucial property is that the PLD of a composition of multiple mechanisms is the convolution



(a) Accounting (b) Computation

Figure 4: (a) Comparison between RDP and PLD accounting and **(b)** comparison of computational efficiency. Measured on Criteo pCTR with identical model architecture, hyperparameters one a single Nvidia® Tesla® P100 GPU.

of their individual PLDs. Thus, Koskela et al. [26] used the Fast Fourier Transform (FFT) in order to speed up the computation of the PLD of the composition; faster algorithms and more accurate algorithms have been proposed in subsequent work [27, 28], and have been the basis of multiple open-source implementations from both industry and academia including [29, 30, 31]. In particular, in this paper, we use the so-called "connect-the-dots" algorithm of Doroshenko et al. [32] for privacy accounting.

Figure 4(a) compares RDP accounting to the improved PLD accounting. Because the PLD estimation is tighter, a given ε requires smaller noise than implied by RDP accounting. We observe consistent improvements of model utility across all privacy regimes, with a larger gap for smaller ε's.

5. Computation & Memory Overhead

Another obstacle to using DP-SGD in real-world deep learning systems is the potential computation and memory overhead. A naive implementation of DP-SGD that explicitly computes each per-example gradient can lead to several orders of magnitude more memory consumption and computational cost. Therefore, efficient implementations of DP-SGD have been studied from various angles, including micro-batching [7], layer-specific algorithms [33, 34], just-in-time compilation [35], and approximation via random projections [36].

Here we demonstrate that when implemented with care, DP-SGD can be run with small computation and memory overheads for ads prediction models. This is enabled by the following observations: 1. To bound sensitivity, we only need per-example gradient norms, *not* per-example gradient vectors. 2. Once per-example gradient norms are computed, the norm-bounded average gradients can be computed using standard backpropagation with reweighted loss. 3. Most ads prediction models

Table 4
Comparison of memory consumption via the maximum batch size that can be trained on Criteo pCTR with one Nvidia® Tesla® P100 GPU ('-' indicates that the training process ran out of memory.)

Batch Size	Number of steps per second		
	Baseline	DP-SGD	Ghost Clipping
32	$62.73 \pm .08$	$7.48 \pm .08$	$28.57 \pm .61$
64	$63.91 \pm .71$	-	$28.05 \pm .33$
256	$41.60 \pm .70$	-	$27.52 \pm .35$
1,024	$14.17 \pm .83$	-	$14.07 \pm .09$
4,096	$3.77 \pm .16$	-	$3.88 \pm .26$
16,384	$3.10 \pm .34$	-	$3.27 \pm .28$
65,536	$2.91 \pm .13$	-	$2.62 \pm .24$
524,288	$0.96 \pm .05$	-	$0.80 \pm .06$
1,048,576	-	-	-

only use embedding and linear layers. As first noted by Goodfellow [33], the per-example gradient norms for fully connected layers can be efficiently estimated with standard backpropagation *sans* materializing the per-example gradient vectors. Since an embedding layer is equivalent to a fully connected layer on one-hot encoded inputs, Goodfellow's observation can be used in DP-SGD.

Based on these observations, we implement a two-pass algorithm that computes per-example gradient norms in the first pass and the norm-bounded average gradient in the second pass. Since most of the computation and memory overhead comes from materializing per-example gradient vectors, our implementation is able to reduce these significantly. This technique is usually called "ghost clipping", and is also effective in private training for computer vision [37] and natural language processing [14, 15]. We implement this algorithm in JAX [38], and compare to a baseline implementation using JAX just-in-time (JIT) compilation, which is already faster than a vanilla implementation as reported in Subramani et al. [35], as well as the non-private baseline (called Baseline).

Figure 4(b) plots the number of examples per second using different training algorithms and batch sizes. We observe that Fast-DP-SGD scales similarly to the non-private baseline, and the computation overhead is negligible for intermediate to large batch sizes. On the other hand, the JIT based implementation is not only slower but also incapable of handling batch sizes larger than 64 because its memory overhead grows linearly with the batch size. The maximum batch size that each algorithm can handle on a single GPU is documented in Table 4. We can see that the batch size cap is the same for Fast-DP-SGD and the non-private baseline, demonstrating that the memory overhead is negligible.

The experiments above were done on a single GPU. The batch size can be easily scaled beyond the maximum

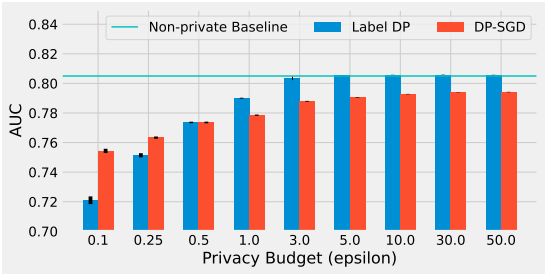


Figure 5: Comparing LabelDP (Randomized Response) with DP-SGD on Criteo under the same privacy budget ε.

value in Table 4 by using multi-device data parallelism, and gradient accumulation across multiple backpropagation steps.

6. Comparison to Label DP

Label differential privacy (LabelDP) [39] is a notion where the features are public and only the labels need privacy protection. It has been recently studied in deep learning [12, 13], and more specifically in ad modeling [40]. In this section, we compare DP-SGD with LabelDP algorithms under the same privacy budget ε. (Note that this is not an apples-to-apples comparison since unlike DP-SGD, LabelDP protects only the labels; furthermore, since we use randomized response as our LabelDP algorithm, we have $\delta = 0$.)

From Figure 5, we observe that LabelDP generally provides higher utility in low privacy (large ε) regimes, while DP-SGD outperforms it in high privacy (small ε) regimes. The behavior in high privacy regimes is counter-intuitive because DP-SGD has stronger privacy guarantees yet provides better utility.

7. Conclusions

In this work we showed that it is possible to privately train ad models using DP-SGD, while neither significantly sacrificing utility nor incurring computational cost. An interesting research direction is to develop new private training algorithms for "hybrid DP"—an interpolation between vanilla DP and Label DP in which some but not all of the features are public. Another line of work would be to study if the low-rank nature / sparsity of the gradients can be exploited to reduce the noise needed for private training.

Acknowledgments

We thank Silvano Bonacina and Samuel Ieong for many useful discussions and Christina Ilvento and Andrew Tomkins for their comments.

References

[1] C. Dwork, F. McSherry, K. Nissim, A. D. Smith, Calibrating noise to sensitivity in private data analysis, in: TCC, 2006, pp. 265–284.

[2] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, M. Naor, Our data, ourselves: Privacy via distributed noise generation, in: EUROCRYPT, 2006, pp. 486–503.

[3] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow, K. Talwar, Semi-supervised knowledge transfer for deep learning from private training data, arXiv preprint arXiv:1610.05755 (2016).

[4] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, Ú. Erlingsson, Scalable private learning with pate, arXiv preprint arXiv:1802.08908 (2018).

[5] P. Kairouz, B. McMahan, S. Song, O. Thakkar, A. Thakurta, Z. Xu, Practical and private (deep) learning without sampling or shuffling, in: International Conference on Machine Learning, PMLR, 2021, pp. 5213–5225.

[6] N. Ponomareva, H. Hazimeh, A. Kurakin, Z. Xu, C. Denison, H. B. McMahan, S. Vassilvitskii, S. Chien, A. Thakurta, How to DP-fy ML: A practical guide to machine learning with differential privacy, J. Artif. Intell. Res. 77 (2023) 1113–1201.

[7] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, L. Zhang, Deep learning with differential privacy, in: CCS, 2016, pp. 308–318.

[8] A. Kurakin, S. Chien, S. Song, R. Geambasu, A. Terzis, A. Thakurta, Toward training at ImageNet scale with differential privacy, arXiv preprint arXiv:2201.12328 (2022).

[9] S. De, L. Berrada, J. Hayes, S. L. Smith, B. Balle, Unlocking high-accuracy differentially private image classification through scale, arXiv preprint arXiv:2204.13650 (2022).

[10] R. Anil, B. Ghazi, V. Gupta, R. Kumar, P. Manurangsi, Large-scale differentially private BERT, in: EMNLP (Findings), 2022.

[11] N. Ponomareva, J. Bastings, S. Vassilvitskii, Training text-to-text transformers with privacy guarantees, in: ACL (Findings), 2022, pp. 2182–2193.

[12] B. Ghazi, N. Golowich, R. Kumar, P. Manurangsi, C. Zhang, Deep learning with label differential privacy, in: NeurIPS, 2021, pp. 27131–27145.

[13] M. Malek Esmaeili, I. Mironov, K. Prasad, I. Shilov, F. Tramer, Antipodes of label differential privacy: PATE and ALIBI, in: NeurIPS, 2021, pp. 6934–6945.

[14] X. Li, F. Tramer, P. Liang, T. Hashimoto, Large language models can be strong differentially private learners, in: ICLR, 2022.

[15] J. He, X. Li, D. Yu, H. Zhang, J. Kulkarni, Y. T. Lee, A. Backurs, N. Yu, J. Bian, Exploring the limits of differentially private deep learning with group-wise clipping, in: ICLR, 2022.
[16] Criteo-Labs, Display advertising challenge: Predict click-through rates on display ads, 2014. URL: https://www.kaggle.com/c/criteo-display-ad-challenge.
[17] M. Zaheer, S. Reddi, D. Sachan, S. Kale, S. Kumar, Adaptive methods for nonconvex optimization, in: NIPS, 2018.
[18] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization., JMLR 12 (2011).
[19] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: ICLR, 2015.
[20] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, in: ICLR, 2019.
[21] A. Gupta, K. Ligett, F. McSherry, A. Roth, K. Talwar, Differentially private combinatorial optimization, in: SODA, 2010, pp. 1106–1125.
[22] O. Thakkar, G. Andrew, H. B. McMahan, Differentially private learning with adaptive clipping, in: NeurIPS, 2021, pp. 17455–17466.
[23] V. Pichapati, A. T. Suresh, F. X. Yu, S. J. Reddi, S. Kumar, AdaCliP: adaptive clipping for private SGD, arXiv preprint arXiv:1908.07643 (2019).
[24] S. Meiser, E. Mohammadi, Tight on budget? Tight bounds for r-fold approximate differential privacy, in: CCS, 2018, pp. 247–264.
[25] D. M. Sommer, S. Meiser, E. Mohammadi, Privacy loss classes: The central limit theorem in differential privacy, PoPETS (2019) 245–269.
[26] A. Koskela, J. Jälkö, A. Honkela, Computing tight differential privacy guarantees using FFT, in: AISTATS, 2020, pp. 2560–2569.
[27] S. Gopi, Y. T. Lee, L. Wutschitz, Numerical composition of differential privacy, in: NeurIPS, 2021, pp. 11631–11642.
[28] B. Ghazi, P. Kamath, R. Kumar, P. Manurangsi, Faster privacy accounting via evolving discretization, in: ICML, 2022, pp. 7470–7483.
[29] L. Prediger, A. Koskela, Code for computing tight guarantees for differential privacy., https://github.com/DPBayes/PLD-Accountant, 2020.
[30] Google's DP Library., DP Accounting Library, https://github.com/google/differential-privacy/tree/main/python/dp_accounting, 2020.
[31] Microsoft., A fast algorithm to optimally compose privacy guarantees of differentially private (DP) mechanisms to arbitrary accuracy., https://github.com/microsoft/prv_accountant, 2021.
[32] V. Doroshenko, B. Ghazi, P. Kamath, R. Kumar, P. Manurangsi, Connect the dots: Tighter discrete approximations of privacy loss distributions, PoPETS (2022) 552–570.
[33] I. Goodfellow, Efficient per-example gradient computations, arXiv preprint arXiv:1510.01799 (2015).
[34] J. Lee, D. Kifer, Scaling up differentially private deep learning with fast per-example gradient clipping, PoPETS (2021) 128–144.
[35] P. Subramani, N. Vadivelu, G. Kamath, Enabling fast differentially private SGD via just-in-time compilation and vectorization, in: NeurIPS, 2021, pp. 26409–26421.
[36] Z. Bu, S. Gopi, J. Kulkarni, Y. T. Lee, H. Shen, U. Tantipongpipat, Fast and memory efficient differentially private-SGD via JL projections, in: NeurIPS, 2021, pp. 19680–19691.
[37] Z. Bu, J. Mao, S. Xu, Scalable and efficient training of large convolutional neural networks with differential privacy, in: NeurIPS, 2022.
[38] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, Q. Zhang, JAX: composable transformations of Python+NumPy programs, 2018. URL: http://github.com/google/jax.
[39] K. Chaudhuri, D. Hsu, Sample complexity bounds for differentially private learning, in: COLT, 2011, pp. 155–186.
[40] B. Ghazi, P. Kamath, R. Kumar, E. Leeman, P. Manurangsi, A. Varadarajan, C. Zhang, Regression with label differential privacy, in: ICLR, 2023.
[41] I. Loshchilov, F. Hutter, SGDR: stochastic gradient descent with warm restarts, in: ICLR, 2017.

A. Training Details

The Criteo pCTR dataset [16] contains around 40 million examples. The raw dataset comes with a labeled training set and an unlabeled test set. We split the raw training set chronologically into a 80/10/10 partition of train/validation/test sets. The reported metrics are on this labeled test split. Each example consists of 13 integer features, 26 categorical features, and one binary label. We preprocess each integer feature with a log transformation.

The neural network consists of six layers. In the first layer, each categorical feature is mapped into a dense feature vector via an embedding layer. The embedding dimension is decided via a heuristic rule as $\mathsf{int}[2V^{0.25}]$, where V is the number of unique tokens in each categorical feature. The dense features are then concatenated with the log-transformed integer features to form the first layer representation. This representation are fed into four fully connected layers, each with an output dimension of 598 and a ReLU activation function. Finally, a fully connected layer is used to compute the scalar prediction (the logit). There are around 78M trainable parameters in this neural network model.

The network is trained with binary cross entropy loss. Unless otherwise specified, we train the network for five epochs, and we scale the base learning rate with a cosine decay [41]. In the non-private baseline, we use the Yogi optimizer [17] with a base learning rate of 0.01, and a batch size of 1024.

Directly reusing the same hyperparameters for private training leads to suboptimal results. We discuss how to achieve better privacy-utility trade-off by adjusting different hyperparameters in Section 4. Even though the analysis shows that optimal hyperparameters could be different for different range of privacy budget ε's, for simplicity, we use a fixed value for most of hyperparameters in the final results of Criteo pCTR in Table 1: SGD optimizer with base learning rate 0.01, momentum 0.9, batch size 65,536, and training for 150 epochs. We only tune the norm bound $C \in \{1.0, 10.0, 50.0, 100.0\}$ and microbatch size $B_\mu \in \{1, 4, 8\}$ for each ε separately.

The model and training setup for the pCVR and pConvs are similar, except that the pConvs come with integer labels, thus are trained and evaluated with a Poisson log loss (PLL). Specifically, let $f_\theta(x)$ be the scalar prediction from the final layer of the neural network, interpreting $\exp(f_\theta(x))$ as predicting the mean of a Poisson distribution, then the log-likelihood of the integer label y is

$$y f_\theta(x) - \exp(f_\theta(x)) - \log(y!).$$

The maximum (log-)likelihood training objective is thus equivalent to minimizing the following PLL function:

$$\ell(f_\theta, (x, y)) = \exp(f_\theta(x)) - y f_\theta(x). \tag{1}$$

Note the $\log(y!)$ term is dropped since it is independent of the trainable parameters θ. Therefore this is an *unnormalized* PLL.