DEEP GRADIENT ATTACKS WITH STRONG DP-SGD LOWER BOUNDS FOR LABEL PRIVACY

Sen Yuan, Min Xue, Kaikai Wang & Milan Shen
Facebook
1601 Willow Road, Menlo Park, CA, United States
{yuansen xuemingrace, kaikaiwang, milanshen}@fb.com

ABSTRACT

The PPML designs such as Federated Learning (FL), Split Learning etc. usually require sharing gradients, which may cause label leakage. Most existing works for label privacy attacks are unable to reconstruct labels with both high accuracy and efficiency when gradients are revealed. In this work, we propose a deterministic attack to reconstruct training labels with 100% accuracy when aggregated gradients are revealed. We propose a novel algorithm to tightly estimate differential privacy lower bound for label privacy problem when differentially private SGD (DP-SGD) is used. To our best knowledge, our work is the first proposal about threat model specific label privacy analysis with DP lower bound.

1 INTRODUCTION

Conventional machine learning (ML) techniques require bringing all data to one machine for model training. This raises a number of concerns if either training labels or features contain sensitive information. Some recent studies show that sharing gradients is a risky practice which might reveal key information of training data. In particular, Deep Leakage from Gradients (DLG) Zhu et al. (2019) proves training data can be algorithmically reconstructed by obtaining all gradients. In this paper, we present an efficient deterministic attack to exactly reconstruct binary training labels with only aggregated gradients. Different from DLG, our attacks are deterministic based on solving linear equations which only require aggregated gradients, instead of running an iterative algorithm on individual gradients. There are two other recent noticeable works besides DLG that propose innovative methods to re-identify training labels. The improved DLG (iDLG) Zhao et al. (2020) presents an analytical formula for re-identifying training labels based on example level gradient and Norm based label uncovering (Norm) Li et al. (2020) exploits gradient norm to uncover training labels. Compared to those related works, our attacks are focused on binary label only and more powerful for binary label re-identification. One major advantage for our proposals is that our attacks are based on aggregated gradient over mini-batch samples whereas both iDLG and Norm require example level gradient. We develop novel DP lower bound estimation methods for label privacy protection problem and provide an analytical formula for calculating DP lower bound. Our DP lower bound algorithm is inspired by the method used in Jagielski et al. (2020). However, we significantly improve their approach by developing a closed form formula for calculating probability terms in lower bound estimation to replace the computational intensive Monte Carlo simulation. Our analytical formula also addresses statistical uncertainties challenge for probability estimation that is created by Monte Carlo. To our best knowledge, this work is the first proposal about threat model specific label privacy analysis.

2 GRADIENT ATTACKS

2.1 PRELIMINARY

We consider training a deep neural network with binary labels where $\mathcal{P}(y[r] = 1) = p[r]$ and $\mathcal{P}(y[r] = 0) = 1 - p[r]$. The loss function we use is the logarithmic loss $\mathcal{L}(y[r], p[r]) = -y[r]\log(p[r]) - (1 - y[r])\log(1 - p[r])$. We use $L = \frac{1}{N}\sum_{r=1}^{N}\mathcal{L}(y[r], p[r])$ to denote the average

loss across batch samples. We consider a general activation function unless otherwise specified. We display our notation conventions in both Table 1.

Symbol	Definition
$[r]$	Index for the rth data in the mini-batch.
W^L	Weight vector for last hidden layer to output.
B^L	Bias for output.
X_i^j	Neuron output at Layer j for Variable i.
Z_i^j	X_i^j =Activation(Z_i^j).
W_{ki}^j	Weight connecting Variable k in Layer j to Variable i in Layer $j+1$.
p_j	Number of neurons in Layer j.
lr	learning rate.

Table 1: Notation definitions

2.2 Attack Algorithms

We develop our attacks from a key observation based chain rule. Suppose W is a d dimensional weight vector from a certain layer (it does not matter which layer). The model output $\mathcal{P}(y[r]=1) = \frac{1}{1+\exp\{-Z[r]\}}$ where $Z[r] = \sum_k W_k^L X_k^L[r] + B^L$ is the final output before sigmoid transformation for the Sample r. Consider refactoring the gradient as $\frac{dL}{dW} = \sum_r \frac{dL}{dZ[r]} * \frac{dZ[r]}{dW}$. Simplifying this chain rule we get the following equation:

$$\frac{dL}{dW} = \frac{1}{N}\left[\frac{dZ[1]}{dW} \dots \frac{dZ[N]}{dW}\right] \begin{bmatrix} p[1]-y[1] \\ \vdots \\ p[N]-y[N] \end{bmatrix} \tag{1}$$

The Equation 1 establishes a linear equation system that allows us to solve for every $p[r]-y[r]$. Since our label $y[r]$ is either 1 or 0 and $p[r]$ is between 0 and 1, if $p[r]-y[r]>0$ then $y[r]$ must be 0 otherwise $y[r]$ equals 1. Based on this fact, to solve for $y[r]$ is equivalent to solve for $p[r]-y[r]$ for binary problem. The lefthand side of the equation is a d dimensional vector where d is the dimension of the weight vector W. The threat model for Algorithm 1 is that adversary is honest but curious and can access example level partial derivatives $\frac{dZ[r]}{dW}$ for a certain layer. Then we can reconstruct labels directly from Equation 1. This attack is general in the sense that we do not specify which layer of gradients and neurons are accessible by adversaries. Other types of specific attacks such as intermediate layer attack and input layer attack are explained in appendix.

Algorithm 1 HBC General Gradient Attack

Input: Partial derivatives of model output before Sigmoid transformation w.r.t. a certain d dimensional weight vectors $\mathbf{C} = [\frac{dZ[1]}{dW}, ..., \frac{dZ[N]}{dW}]$ and the weight difference $\Delta W = -lr * \frac{dL}{dW}$.
Output: $I((\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\Delta W > 0)$

3 DP Lower Bounds for Label Privacy

In the auditing DP-SGD paper Dwork et al. (2014), they propose a method to estimate DP-SGD lower bound based on backdoor attacks. The data under protection in their work is training data. However, our problem is focused on protecting labels only. We borrow the general principle from their paper based on which we develop our own algorithm for DP-SGD lower bound estimation for label privacy problem. Consider two adjacent datasets D_0 and D_1 that only disagree on one row. We can define (ϵ, δ)label DP by following definition from Dwork et al. (2014).
Definition (ϵ, δ)-DP: An algorithm $\mathcal{A}: \mathcal{D} \to \mathcal{R}$ is (ϵ, δ)-*differentially private* if for any two datasets

D_0, D_1 which differ on at most one row, and every set of outputs $\mathcal{O} \in \mathcal{R}$:

$$\mathcal{P}[\mathcal{A}(D_0) \in \mathcal{O}] \leq e^{\epsilon}\mathcal{P}[\mathcal{A}(D_1) \in \mathcal{O}] + \delta \tag{2}$$

The D_0 in our label privacy scenario is the original label vector whereas D_1 is the label vector with only one entry being flipped. We can rewrite the DP definition as

$$\epsilon \geq \log \sup_{D_0, D_1, \mathcal{O}} \frac{\mathcal{P}[\mathcal{A}(D_0) \in \mathcal{O}] - \delta}{\mathcal{P}[\mathcal{A}(D_1) \in \mathcal{O}]} \tag{3}$$

Then the ϵ lower bound estimation problem becomes finding a good combination of $(D_0, D_1, \mathcal{A}, \mathcal{O})$ to generate a high value for $\frac{\mathcal{P}[\mathcal{A}(D_0)\in\mathcal{O}]-\delta}{\mathcal{P}[\mathcal{A}(D_1)\in\mathcal{O}]}$. We define $\mathcal{O} = \{D_0[i] : D_0[i] \neq D_1[i]\}$ the single point set for the true label for the disagreement row and $\mathcal{A}$ as one of our gradient attack output for the disagreement row only. The Algorithm 2 describe our method of estimating DP lower bound based on a single batch. The technical details for our DP lower bound algorithm and efficient estimation based on all batches are in Appendix A.3.

Algorithm 2 DP-SGD Label Privacy Lower Bound

Input: DP-SGD parameters including noise multiplier σ, max norm clip C, batch size N. Define $s = \frac{\sigma C}{N}$. The weight vector W and coefficient matrix $\mathbf{C}$ for attack Equation 1.

for $r \in [N]$ **do**

1. Example level gradient $g_0(x[r]) = \frac{dL(p[r],y[r])}{dW}$
2. Clipping $\bar{g_0}(x[r]) = g_0(x[r]) / \max(1, \frac{||g_0(x[r])||_2}{C})$

end for

Let $\bar{G_0} = \frac{1}{N}\sum_r \bar{g_0}(x[r])$

for $r \in [N]$ **do**

1. Flip the rth row of D_0 to create $D_1[r]$
2. Re-calculate only $\bar{g_0}(x[r])$ with $D_1[r]$ to get $\bar{g_1}(x[r])$
3. $\bar{G_1}[r] = \bar{G_0} + \frac{1}{N}(\bar{g_1}(x[r]) - \bar{g_0}(x[r]))$
4. Let $R[r] = (0, ..., 1, ..., 0)^T$ only rth position 1
5. $\mathcal{P}_1[r] = \Phi(\frac{(-1)^{y[r]}R^T[r](\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\bar{G_1}[r]}{s\sqrt{R^T[r](\mathbf{C}^T\mathbf{C})^{-1}R[r]}})$
6. $\mathcal{P}_0[r] = \Phi(\frac{(-1)^{y[r]}R^T[r](\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\bar{G_0}}{s\sqrt{R^T[r](\mathbf{C}^T\mathbf{C})^{-1}R[r]}})$
7. LB[r]=$\log(\frac{\mathcal{P}_0[r]-\delta}{\mathcal{P}_1[r]})$

end for

Output: $\max_r$ LB$[r]$

4 EXPERIMENTS

The data we used for the experiments is Cervical Cancer Data Fernandes et al. (2017) and Bank Marketing Data Moro et al. (2014) from UCI ML Repository datasets Cancer Data and Bank Data. We choose these two datasets because both of them are binary classification datasets with sensitive training labels (patient cervical cancer biopsy outcomes and client subscription decisions). We build Multiplayer Perceptrons with two hidden layers for our experiments with both hidden layers having 200 neurons. Our choice of activation is ReLu. I used Opacus to run DP-SGD algorithm.

4.1 EXPERIMENT I: DP LOWER BOUNDS, UPPER BOUNDS FOR DIFFERENT NOISE MULTIPLIERS

In Figure 1, we show DP-SGD upper bounds as well as DP lower bound estimations based on both last hidden layer and second last hidden layer gradients. We perform the same numerical studies for both cancer and bank datasets. We set the gradient norm bound parameter C in DP-SGD as 1.0 to control this parameter. Our first observation is that both lower bound estimations exhibit the same

trend with DP-SGD upper bound as we increase noise multiplier, which is aligned with our intuitive understanding. Note the gap between lower bound and upper bound appears constant at log scale (our y-axis is log scale). This implies the absolute gap decreases significantly as we increase noise multiplier.

4.2 Experiment II: DP lower bounds, upper bounds for different gradient norm bounds

In this experiment, we fix noise multiplier as 0.5 and run against different choices of gradient norm bounds. The gradient norm bounds parameter in DP-SGD is used in gradient clipping but do not affect provable epsilon. However, it plays an important role in optimizing model prediction accuracy. As shown in Figure 2, increasing C would lead to smaller lower bounds, hence more private designs from the threat model analysis perspective. To achieve better privacy-utility tradeoff, a good practical strategy should be considering C as a tuning parameter to optimize model prediction performance. Among all candidate Cs with good model performance, pick the maximum one.

For both Experiment I and II, we notice that exposing last layer is far more risker than exposing the second last layer. The technical explanation for this key insight is included in the appendix.

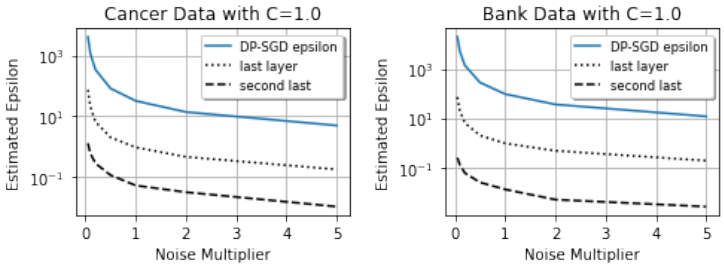

Figure 1: DP-SGD upper bound and our lower bound estimations for various noise multipliers.

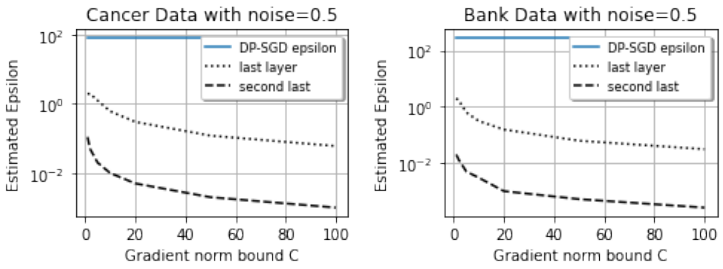

Figure 2: DP-SGD upper bound and our lower bound estimations for various gradient norm bounds.

5 Conclusion

In this paper, we propose our deterministic attack to reconstruct training labels with 100% accuracy based on aggregated gradients. Our attacks can be used for label reconstruction for a complete batch and do not require all gradients revealed to adversaries. We introduce a novel DP lower bound algorithm for DP-SGD algorithm that only protects training labels. Different from existing works, our gradient attack based lower bound estimation is designed for label privacy and has an efficient analytical formula. Our numerical experiments demonstrate the gap between DP-SGD upper bound and lower bound varies when noise multiplier, gradient norm bound or threat model changes. We believe our lower bound estimation method would provide practical guidance on selecting DP-SGD parameters for better privacy-utility trade-off.

REFERENCES

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

Kelwin Fernandes, Jaime S Cardoso, and Jessica Fernandes. Transfer learning with partial observability applied to cervical cancer screening. In *Iberian conference on pattern recognition and image analysis*, pp. 243–250. Springer, 2017.

Matthew Jagielski, Jonathan Ullman, and Alina Oprea. Auditing differentially private machine learning: How private is private sgd? *arXiv preprint arXiv:2006.07709*, 2020.

Oscar Li, Jiankai Sun, Weihao Gao, Hongyi Zhang, Xin Yang, Junyuan Xie, and Chong Wang. Label leakage and protection in two-party split learning. In *NeurIPS Workshop on Scalability, Privacy and Security in Federated Learning*, 2020.

S. Moro, Cortez P., and P. Rita. A data-driven approach to predict the success of bank telemarketing. In *Decision Support Systems*, pp. 62:22–31,. Elsevier, 2014.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.

Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, pp. 14774–14784, 2019.

A APPENDIX

A.1 MORE SPECIFIC ATTACKS

The purpose of studying more specific attacks is because for Algorithm 1, the coefficient matrix $\mathbf{C}$ might not be directly provided by the underlying deep learning designs. If so, we argue that the complete label leakage still exists even without know $\mathbf{C}$ directly for some important scenarios.

A.1.1 LAST HIDDEN LAYER ATTACK

As an immediate consequence of Algorithm 1, we develop HBC last hidden layer attack. To evolve from Algorithm 1 to Algorithm 3, one just need to notice $\frac{dZ[r]}{dW^L} = (X_1^L[r], ..., X_{p_L}^L[r])^T$.

Algorithm 3 HBC Last Hidden Layer Attack

Input: The weight difference ΔW for W^L. The last hidden neuron value matrix $\mathbf{X^L}$ with dimension $p_L \times N$ with the ith row corresponds to ith neuron in Layer L and jth column corresponds to the jth sample.

Output: $I((\mathbf{X^L}^T\mathbf{X^L})^{-1}\mathbf{X^L}^T\Delta W > 0)$

A.1.2 INTERMEDIATE LAYER ATTACK

We study the gradient attack based on information from second last hidden layer. To calculate the coefficient matrix $\mathbf{C}$ in Equation 1. Let us assume we select the weight vector that connect second last layer neurons to the Jth neuron in the last hidden layer. And we denote the activation derivate $U[r] = A'(Z_J^L[r])$. Then the coefficient matrix can be presented as Equation 4. An important observation from 4 is that there is a unknown scalar constant W_J^L. This number is unknown because we only know weights connecting second last hidden layer to last hidden layer if the last hidden layer is not revealed. The unknown sign of this constant brings us to add another minor assumption specific to this attack: the adversaries has prior knowledge about whether the positive label ratio is less than 0.5. For a wide spectrum of real problems, this quantity is known. For instance, loan approval, tumor detection, fraudulent seller etc. all present an extremely low positive label rate, which is essentially a prior information. We summarize our attack as Algorithm 4.

ReLu We highlight a special scenario of Algorithm 4 when ReLu is used as the activation function. As a consequence of using ReLu, the coefficient matrix may have columns with all zero entries. For example, $U[1] = A'(Z_J^L[1]) = I(Z_J^L[1] > 0)$ is either 1 or 0 for sample 1. If it is 0, then the first column of $\mathbf{C}$ is a zero vector. As a result, the $p[1] - y[1]$ can never be determined by this equation system because its coefficients are zeros in each equation in the system.

$$\mathbf{C} = W_J^L \begin{pmatrix} X_1^{L-1}[1]U[1] & \cdots & X_1^{L-1}[N]U[N] \\ X_2^{L-1}[1]U[1] & \cdots & X_2^{L-1}[N]U[N] \\ \vdots & \vdots & \vdots \\ X_{p_{L-1}}^{L-1}[1]U[1] & \cdots & X_{p_{L-1}}^{L-1}[N]U[N] \end{pmatrix} \tag{4}$$

Technical Proof We defined $Z[r]$ in Equation 1 as $Z[r] = \sum_k W_k^L X_k^L[r] + B^L$. Suppose the Jth neuron in layer L is selected therefore we consider the weights connecting second last hidden layer to neuron J in the last hidden layer as our attack weight vector as described in Algorithm 4. We denote this weight vector as $W = (W_{J1}^{L-1}, ..., W_{Jp_{L-1}}^{L-1})^T$. $X_J^L[r] = A(Z_J^L[r])$ where $Z_J^L[r] = \sum_i W_{Ji}^{L-1} X_i^{L-1}[r] + B_J^L$. Notice that among all neurons in Layer J, only $X_J^L[r]$ is a function W and by chain rule: $\frac{dZ[r]}{dW} = \frac{dZ[r]}{dX_J^L[r]} \frac{X_J^L[r]}{dZ_J^L[r]} \frac{dZ_J^L[r]}{dW}$

Since $\frac{dZ[r]}{dX_J^L[r]} = W_J^L$, $\frac{X_J^L[r]}{dZ_J^L[r]} = A'(Z_J^L[r]) = U[r]$ and $\frac{dZ_J^L[r]}{dW} = (X_1^{L-1}[r], ..., X_{p_{L-1}}^{L-1}[r])^T$.

$\frac{dZ[r]}{dW} = W_J^L (X_1^{L-1}[r]U[r], ..., X_{p_{L-1}}^{L-1}[r]U[r])^T$. Apply this example $[r]$ result to any examples $[1], ..., [N]$ then combine results in a matrix we get $\mathbf{C}$.

Algorithm 4 HBC Second Last Hidden Layer Attack

Input: Arbitrarily choose a $J \in \{1, .., p_L\}$. Consider the weight difference ΔW for the weight vector connecting all neurons in second last hidden layer with Neuron J in last hidden layer, prior knowledge about whether positive label ratio is less than 0.5.

1. One step forward prop Compute the last hidden layer pre-activated neurons $Z^L[r]$ according to forward propagation based on second last hidden layer weights and neurons.
2. Activation derivative Compute Jth activation derivates $U[r] = A'(Z_J^L[r])$ for $r = 1, ..., N$.
3. Update coefficient Matrix Define $\mathbf{X} = C/W_J^L$.

Output: Output either $I((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Delta W < 0)$ or $1 - I((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Delta W < 0)$ that matches prior knowledge of positive label ratio.

A.1.3 INPUT LAYER ATTACK

What if the adversary is only able to access the input layer gradients? We show that if we assume the adversary is malicious, label reconstruction is guaranteed to be 100% accurate. To avoid zero columns for coefficient matrix, the malicious adversary would not use ReLu as the activation. The basic idea for this malicious attack is to construct features such that for all neuron outputs are exactly equal across samples. Mathematically, we would have $\mathbf{W}X[1] + B = ... = \mathbf{W}X[N] + B$ where $\mathbf{W}$ is the input layer weight matrix, B is the bias vector and $X[r]$ is the feature vector for sample r. Creating a feature set to satisfy the previous equalities requires some conditions to be met, which is shown in Algorithm 5.

Technical Proof Consider the weight vector in Algorithm 5, connecting all input layer neurons to some neuron J in the second layer. The chain rule equation (1) can be written as:

$$\frac{dL(y[r], p[r])}{dW} = (p[r] - y[r]) \frac{dZ[r]}{dX^L[r]} \frac{dX^L[r]}{dZ^L[r]} \frac{dZ^L[r]}{dX^{L-1}[r]} ... \frac{dZ^2[r]}{dW} \tag{5}$$

Notice that each X^j and Z^j in the chain rule equation above are vectors and the derivative terms are Jacobian matrices. Let $G(r) = \frac{dZ[r]}{dX_J^L[r]} \frac{dX^L[r]}{dZ^L[r]} \frac{dZ^L[r]}{dX^{L-1}[r]} ....$ Based on the way we construct training

Algorithm 5 Malicious Input Layer Attack

Input: Only the input layer weights before and after mini-batch updates.

1. Structure setup Intentionally choose a large feature size p_1 and small first hidden layer size p_2.
2. Batch size Pick a mini-batch size $N \leq p_1 - p_2$ and randomly generate a non-zero vector $V$of dimension $p_2 \times 1$.
3. Construct Features The dimension of linear equation $\mathbf{W}X = V$ solution space is $p_1 - p_2 \geq N$. Choose the first N basis vectors from the equation solution space to construct the feature matrix $\mathbf{X}$ with dimension $p_1 \times N$.
4. Weight delta Choose an arbitrary neuron J in the second hidden layer and consider the weight vector W that connects neurons between input layer and neuron J. Denote weight difference as ΔW.

Output: Output either $I((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Delta W < 0)$ or $1 - I((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Delta W < 0)$ that matches prior knowledge of positive label ratio.

features for Algorithm 5 (Step 3), all neurons value in every layer are no long a function of samples. For different samples in the batch, all neurons are essentially the same across samples starting from Layer 2. Therefore, $G(r)$ is not a function of sample index $[r]$ since all neuron values are not functions of sample index $[r]$ by the way we construct training features. Replacing $G(r)$ with G, we can simplify the chain rule equation as:

$$\frac{dL(y[r], p[r])}{dW} = (p[r] - y[r])G\frac{dZ^2[r]}{dW} \tag{6}$$

$\frac{dZ^2[r]}{dW} = \frac{d(Z_1^2[r],...,Z_J^2[r],...Z_{p_2}^2[r])^T}{dW} = (0, .., \frac{dZ_J^2[r]}{dW}, ..., 0)^T = (0, .., X[r], ...0)^T$ because W connecting to the Jth neuron in Layer2 which implies only $Z_J^2[r]$ is a function of W. The $X[r]$ is the feature vector for sample r. Notice that the matrix G dimension is $1 \times p_2$, we can write row vector $G = (g_1, ..., g_J, ..., g_{p_2})$. Therefore, $G\frac{dZ_J^2[r]}{dW} = g_J X[r]$. Based on this fact, and we sum over samples $[r]$, we obtain

$$\sum_r \frac{dL(y[r], p[r])}{dW} = g_J \sum_r (p[r] - y[r])X[r] = g_J(X[1], ..., X[N])(p[1] - y[1], ..., p[N] - y[N])^T \tag{7}$$

Note that feature matrix is $\mathbf{X} = (X[1], ..., X[N])$, g_J is an unknown constant and $\Delta W = -lr/N * \sum_r \frac{dL(y[r], p[r])}{dW}$. Therefore, $(p[1] - y[1], ..., p[N] - y[N])^T = -N/(lr \times g_J)(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Delta W$. Since $p[r]$ is between 0 and 1, $y[r]$ is either 0 or 1. Then sign of the left hand side would imply $y[r]$. However, g_J is unknown, therefore the conclusion is: $(y[1], ..y[N])^T$ is either $I(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Delta W > 0)$ or $I(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Delta W < 0)$.

A.2 DP-SGD Lower Bound Technical Details

A.3 Proof of Algorithm refalg:dp analytical formula

We already perform gradient clipping in Algorithm 2. Therefore, the DP-SGD algorithm is updated according to the equation below

$$\Delta W = -lr * (\bar{G} + \epsilon) \tag{8}$$

We do not specify $\bar{G}_0$ or $\bar{G}_1$ because our reasoning applies to both. Our gradient attack is either $I((\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\Delta W > 0)$ or $1 - I((\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\Delta W > 0)$. Without loss of generality, we consider $I((\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\Delta W > 0)$ as our attack. Note that for rth iteration, we only focus on row r which is the row that D_0 and D_1 disagree. If we focus our attack on this row, it becomes $I(R^T(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\Delta W > 0)$ where $R = (0, ...1, ..., 0)^T$ with only rth position 1.

$I(R^T(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\Delta W > 0) = I(R^T(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\epsilon < -R^T(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\bar{G})$

Since $R^T(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\epsilon \sim Normal(0, R^T(\mathbf{C}^T\mathbf{C})^{-1}Rs^2)$ where $s = \sigma C/N$. Therefore, if the true label $y[r] = 1$, then our probability is measured by $P(I(R^T(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\epsilon < -R^T(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\bar{G}) = 1) = P(R^T(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\epsilon < -R^T(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\bar{G}) = \Phi(\frac{-R^T(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\bar{G}}{s\sqrt{R^T(\mathbf{C}^T\mathbf{C})^{-1}R}})$
Similarly, if the true label $y[r] = 0$, our probability is measured by $P(I(R^T(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\epsilon < -R^T(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\bar{G}) = 0) = \Phi(\frac{R^T(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\bar{G}}{s\sqrt{R^T(\mathbf{C}^T\mathbf{C})^{-1}R}})$.

A.3.1 DP LOWER BOUND BASED ON ALL BATCHES: AN EXTRAPOLATION THEOREM

Theorem Suppose the lower bound for each sample in the batch is $x[1], ..., x[N]$. The single batch based lower bound estimation is $LB(1) = \max_{1\le r\le N} x[r]$. Consider if we have M batches with total sample size $M.N$, then our lower bound estimation becomes $LB(M) = \max_{1\le r\le M.N} x[r]$. Let's assume all lower bounds for each sample are independently identically distributed as exponential distribution with C.D.F. $F(x) = 1 - \lambda e^{-\lambda x}$,

then we conclude that $\mathbf{E}(LB(M)) = (1 + \frac{\log M}{\log N})\mathbf{E}(LB(1))$ when $M.N$ gets large.

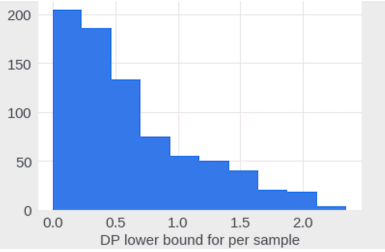

Figure 3: Distribution of lower bounds when sample size is 800 for cancer data

Proof The reason we made exponential distribution assumption are two folds: first, as displayed in Figure 3, the lower bounds are empirically distributed as exponentially distribution; second, it is convenient for us to derive analytical solution.

Denote max order statistics are $x(n)$ for any n. Then C.D.F of $x(n)$ is $P(x(n) < x) = P(x[1] < x, ..., x[n] < x) = P(x[1] < x)...P(x[n] < x) = P(x[1] < x)^n = F(x)^n$.

$$\mathbf{E}(x(n)) = \int_0^\infty x dF(x)^n = \int_0^\infty nx(1 - e^{-\lambda x})^{n-1}\lambda e^{-\lambda x}dx = n/\lambda \int_0^\infty x(1 - e^{-x})^{n-1}e^{-x}dx \tag{9}$$

Because $(1 - e^{-x})^{n-1} = \sum_{k=0}^{n-1}\binom{n-1}{k}(-1)^k e^{-xk}$, plug this into the equation above and notice that $\int_0^\infty xe^{-x(k+1)}dx = \frac{1}{k+1}$ we have

$$\mathbf{E}(x(n)) = n/\lambda \sum_{k=0}^{n-1}(-1)^k \frac{1}{(k+1)^2}\binom{n-1}{k} \tag{10}$$

Notice that $\frac{1}{(k+1)^2}\binom{n-1}{k} = \binom{n}{k+1}\frac{1}{n}\frac{1}{k+1}$. Therefore, we have
$\sum_{k=0}^{n-1}(-1)^k\frac{1}{(k+1)^2}\binom{n-1}{k} = \sum_{k=0}^{n-1}(-1)^k\binom{n}{k+1}\frac{1}{n}\frac{1}{k+1} = \frac{1}{n}\sum_{k=1}^{n}(-1)^{k-1}\binom{n}{k}\frac{1}{k}$

We observe the generating function of series $\sum_{k=1}^{n}(-1)^{k-1}\binom{n}{k}\frac{1}{k}x^{k-1}$ and the following equation

$$\sum_{k=1}^{n}(-1)^{k-1}\binom{n}{k}\frac{1}{k}x^{k-1} = \frac{1 - (1-x)^n}{x} \tag{11}$$

Integrate over $[0, 1]$ for both left and right hand side of the equation above we get:

$\sum_{k=1}^{n}(-1)^{k-1}\binom{n}{k}\frac{1}{k} = \int_0^1 \frac{1-(1-x)^n}{x}dx = \int_0^1 \frac{1-x^n}{1-x}dx = \int_0^1 \sum_{i=0}^{n-1} x^i dx = \sum_{i=0}^{n-1}\frac{1}{i+1} \approx \log(n)$

Combine all those results we have $\mathbf{E}(x(n)) = \frac{\log(n)}{\lambda}$. Therefore, $\frac{\mathbf{E}(x(M.N))}{\mathbf{E}(x(N))} = \frac{\log(M.N)}{\log N}$

The implication of this theorem is important. It implies that our lower bound estimation grows $\log(M)$ fast as the number of batches M grows. And this aligns with our observation in Figure 5 in the paper that lower bound grows slow as number of batches increases.

A.4 Additional details for experiments

A.4.1 Why exposing last hidden layer is riskier than the second last layer

Another interesting observation is that the second last hidden layer based attack generates a consistently smaller lower bound, which suggests revealing second last hidden layer is more private than revealing last hidden layer. The implication of this comparison is that threat model affects lower bound estimation sizably. The intuitive reason why revealing last layer is riskier is because last layer stays "closer" with training labels. However, there are more reasonable explanations. In Algorithm 2, those two probabilities $P_0[r]$ and $P_1[r]$ will be extremely close if gradients magnitudes $\bar{G}_1[r]$ and $\bar{G}_0$ are both small. The gradient magnitude for last layer can be significant higher than that of second last layer due to back propagation, which consequently causes lower bound for second last layer much closer to 0. To verify this, we show distribution of gradient absolute values of both last layer and second last layer in Figure 4. From the histogram, it is clear that second last layer gradient magnitude is much smaller.

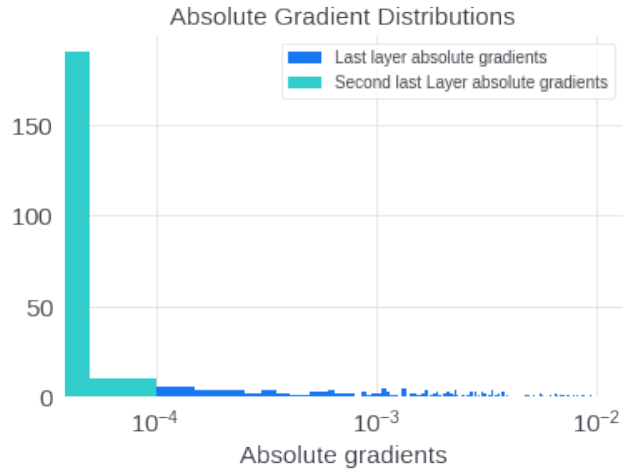

Figure 4: Absolute gradients distributions for last hidden layer vs. second last hidden layer

The Figure 4 shows magnitudes of gradients for second last layer is much smaller than that of the last layer. This leads to lower DP lower bound for the second last layer attack as explained in the paper. Here we briefly explain why magnitudes for the second last layer gradients are smaller. To see this, we compare gradients between those two attacks:

For second last layer: $\frac{dL(y[r],p[r])}{dW} = W_J^L(X_1^{L-1}[r]U[r], ..., X_{p_{L-1}}^{L-1}[r]U[r])^T(p[r] - y[r])$

For last layer: $\frac{dL(y[r],p[r])}{dW} = (X_1^L[r], ..., X_{p_L}^L[r])^T(p[r] - y[r])$

The key for the magnitude difference is because of additional component W_J^L in second last layer gradient due to longer derivative chain. In most of software such as pytorch, this quantity is initialize as a small value, e.g., in pytorch the number is generated from Uniform$[-\frac{1}{\sqrt{p_L}}, \frac{1}{\sqrt{p_L}}]$ where dimension of p_L is usually large in practical setups. As a result, the large the last layer size is, the bigger the gap will be for the gradient magitudes between those two attacks.

A.4.2 THE EFFECTS OF NUMBER OF BATCHES USED FOR DP LOWER BOUND ESTIMATION

The Algorithm 2 describes the lower bound estimation based on a single batch. Using more samples in principle improve lower bound. One can iterate through multiple batches or even all batches to obtain tighter bound than a single batch. However, running with more batches requires longer time. It is important to investigate how large the return is by increasing number of batches. We show our numerical results in Figure 5. For cancer data, the relative difference between a single batch and all batches is only about 11%. For bank data, even if we increase from a single batch to 80 batches, the lower bound only increases from 0.033 to 0.091. Consider its corresponding DP-SGD upper bound 282.24, this improvement perhaps does not change our views of the gap between upper bound and lower bound. Therefore, we suggest using a limited number of batches should be helpful in providing practical guidance for privacy decision. Our Extrapolation Theorem in A.3.1 suggests the growing factor (the DP lower bound based on all batches divided by the DP lower bound based on a single batch) for each datasets should be: (1+log(15)/log(50)) = 1.69 and (1 + log(80)/log(50)) = 2.12 respectively, which basically aligns with we observe in Figure 5.

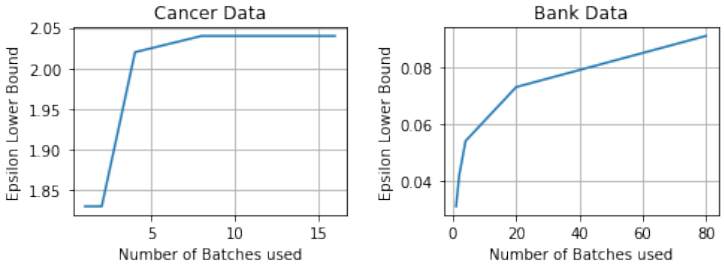

Figure 5: Lower bound estimations based on different numbers of batches.