Technical Report of Label Leakage and Protection in Federated Learning

Problem Setup

- In the settings of the split learning, the passive party (ByteDance) provides user embeddings, and the active party (advertisers such as JD, Alibaba etc) owns the label information and its own embeddings (if possible). The passive party sends the split layer's embedding to the active party and receives the corresponding gradients from the active party.
- The active party would like to protect their label information. However, sharing gradients definitely leaks the ground-truth labels.

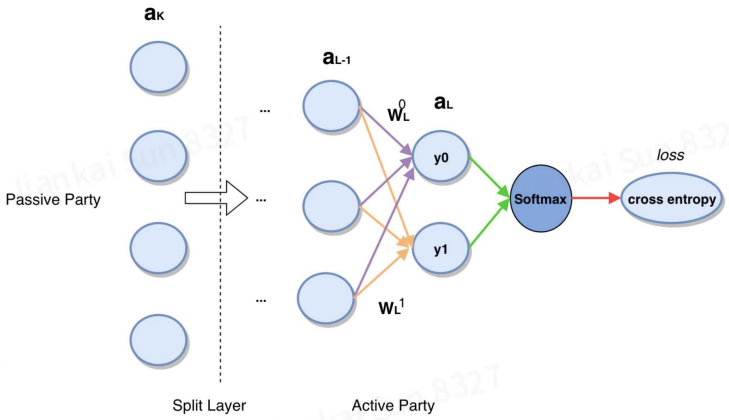


Figure 1: Illustration of the neural network.

Extracting Ground-truth Labels

We introduce a method that the passive party can use to extract the ground-truth labels from the active party via the shared gradients.

- The active party uses Softmax to calculate the loss

$$l(\mathbf{x}, c) = -log\frac{e^{y_c}}{\sum_j e^{y_j}} \qquad (1)$$

Where

- **x** is the input feature vectors
- c is the label

- **y** is the predicted logit score

$$p_i = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{4}$$

$$y_i = \mathbf{W}_L^{i\,T} \mathbf{a}_{L-1} + b_L^i \tag{5}$$

- p_i is the predicted probability of being label i

The gradient of the logit is:

$$g_i = \frac{\partial l(\mathbf{x}, c)}{\partial y_i} = -\frac{\partial log(e^{y_c}) - \partial log(\sum_j e^{y_j})}{\partial y_i}$$

$$= \begin{cases} -1 + \frac{e^{y_i}}{\sum_j e^{y_j}} & \text{if } i = c \\ \frac{e^{y_i}}{\sum_j e^{y_j}} & \text{else} \end{cases} \tag{2}$$

Then:

- given a positive instance: g_0 = p_0 = 1 - p1, g_1 = -1 + p1
- given a negative instance: g_0 = -1 + p_0 = -p1, g_1 = p1

The gradient of the activation layer:

$$\nabla \mathbf{a}_K = \sum_i \frac{\partial l(\mathbf{x}, c)}{\partial y_i} \cdot \frac{\partial y_i}{\partial \mathbf{a}_K} = \sum_i g_i \cdot \frac{\partial y_i}{\partial \mathbf{a}_K} = g_0 \cdot \frac{\partial y_0}{\partial \mathbf{a}_K} + g_1 \cdot \frac{\partial y_1}{\partial \mathbf{a}_K} \tag{12}$$

Hence, for positive instances:

$$\nabla \mathbf{a}_K^1 = (1 - p_1)(\frac{\partial y_0}{\partial \mathbf{a}_K} - \frac{\partial y_1}{\partial \mathbf{a}_K}) \tag{13}$$

For negative instances, we have:

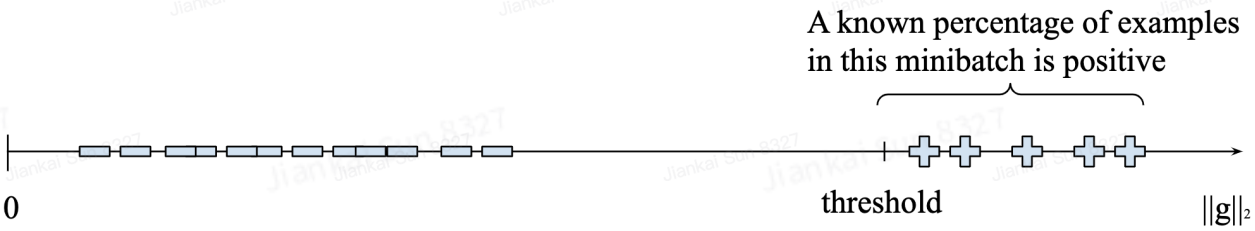
$$\nabla \mathbf{a}_K^0 = -p_1\left(\frac{\partial y_0}{\partial \mathbf{a}_K} - \frac{\partial y_1}{\partial \mathbf{a}_K}\right) \tag{14}$$

2-norm Attack

In the scenario of the online advertising, the number of positive instances are much smaller than the negative ones. For example, in the public Criteo dataset, the positive instance ratio is 25%. As a result, the gradient norm of the positive instances (||1-p1||) will be much larger than the negative ones (||p1||).

Thus we can uncover the label of examples in a minibatch by:

- ranking them according to the gradient norm
- take the top p percentage of the examples as positive



Empirically, we achieved very high leakage auc (~1.0) with the above 2-norm attacking method on different datasets.

Protection Methods

The fundamental idea of our protecting methods is we would like to add some perturbation to the gradients ($\mathbf{g}$) sent back to the passive party to make the attacker not distinguish the difference between positive and negative instances easily.

Previous training diagram (without protection):

Model Training (minibatch) (diagram)

$$\begin{array}{lccccccccc} \text{forward:} & X & \xrightarrow{\boldsymbol{f}} & \boldsymbol{f}(X) & \overset{\text{comm.}}{\Longrightarrow} & \boldsymbol{f}(X) & \xrightarrow{h} & \ell & \to & L \\ \text{Back:} & & & g & \overset{\text{comm.}}{\Longleftarrow} & g & \leftarrow & \frac{dL}{d\ell} = (\tilde{p_1} - y) & \leftarrow & 1 \\ & & & \downarrow & & & & \downarrow & & \\ & & & f\text{'s param} & & & & h\text{'s param} & & \end{array}$$

$$g = \underbrace{(\tilde{p_1} - y)}_{\in \mathbb{R}} \nabla_a h(a) \Big|_{a=\boldsymbol{f}(X)}, \text{ for every } (X, y) \text{ in the minibatch.}$$

The **g hat** is the perturbated **g** and **g hat** will be sent back to the passive party**:**

Model Training (minibatch) (diagram)

$$\begin{array}{lccccccccc} \text{forw.:} & X & \xrightarrow{\boldsymbol{f}} & \boldsymbol{f}(X) & \overset{\text{comm.}}{\Longrightarrow} & & & \boldsymbol{f}(X) & \xrightarrow{h} & \ell & \to & L \\ \text{back.:} & & & \tilde{g} & \overset{\text{comm.}}{\Longleftarrow} & \tilde{g} & \leftarrow & g & \leftarrow & \frac{dL}{d\ell} = (\tilde{p_1} - y) & \leftarrow & 1 \\ & & & \downarrow & & & & & & \downarrow & & \\ & & & f\text{'s param} & & & & & & h\text{'s param} & & \end{array}$$

Here, we provide two different ways as following to generate the perturbated **g hat**.

Max Norm Alignment

In this method, we would like to add some Gaussian noise to make the expected norm of the positive and negative instances be equal (un-distinguishable). For simplicity, the Gaussian noise has zero mean. Then the key question is what's the standard deviation of the Gaussian noise to add?

Next, we will show how to determine the standard deviation for each instance to add the Gaussian noise.

Alignment with the maximum gradient norm of the positive instances

Suppose in a minibatch, p_i is the minimum predicted probability among all the positive instances, according to equation 13, the corresponding instance will have the largest gradient norm among all positive instances.

Given a negative instance, suppose its predicted probability is p_j, the standard deviation \sigma_j of the Gaussian noise can be calculated by letting the expected norm of the pertubated gradient be equal with the p_i's. Then, we have,

$$
\begin{aligned}
E((1-p_i)^2) &= E(p_j^2(1+2\sigma_j+\sigma_j^2)) \\
E((1-p_i)^2) &= E(p_j^2(1+\sigma_j^2)) \\
\sigma_j^2 &= \frac{(1-p_i)^2}{p_j^2} - 1
\end{aligned}
\tag{28}
$$

Hence, the standard deviation of the Gaussian noise added to a negative instance will be:

$$\sqrt{\frac{(1-p_i)^2}{p_j^2} - 1}$$

The perturbated gradient g_j hat will be: g_j (1 + \sigma_j).

For other positive instances, their corresponding standard deviation will be calculated similarly as equation 28. Suppose a positive instance with predicted probability p_k (p_k > p_i), its corresponding standard deviation is:

$$\sqrt{\frac{(1-p_i)^2}{(1-p_k)^2} - 1}$$

The pertubated gradient g_k hat will be: g_k (1 + \sigma_k).

The corresponding algorithm to calculate the standard deviation and protect the label information is:

Algorithm 5 Preventing Label Leakage 1

Input: $\mathbf{g}$: gradient sending back to the passive party, n by d where n is the batch size and d is the number embedding dimension. $\mathbf{c}$: ground truth label, n by 1. $\mathbf{p}$: predicted probability, n by 1

Output: $\mathbf{g}'$: gradient with Gaussian noise

compute $p\_min = \min(\mathbf{p}_i), \forall i \in [1, n]$ and $\mathbf{c}_i = 1$

for $i \leftarrow 1$ *to* n **do**

  /* compute the standard deviation of the Gaussian noise */

  if $\mathbf{c}_i == 1$ **then**

    /* positive instance */

    $\sigma_i = \sqrt{\frac{(1-p\_min)^2}{(1-p_i)^2} - 1}$

  else

    /* negative instance */

    $\sigma_i = \sqrt{max(\frac{(1-p\_min)^2}{p_i^2} - 1, 0)}$

  $noise_i \sim \mathcal{N}(0, \sigma_i^2)$ // generate the zero mean Gaussian noise

  $\mathbf{g'_i} = \mathbf{g_i}(1 + noise_i)$ // add the generated noise to the gradient

Return $\mathbf{g}'$

Alignment with the maximum gradient norm of all the instances in the minibatch

In the above method, the active party has to distinguish the positive and negative instances and calculate the standard deviations respectively. For simplicity, we can align all the instances to have the same expected norm with the maximum norm among the mini-batch. It's worth mentioning that the instance which has the maximum norm is a positive one. Hence, it can achieve similar performance as the previous alignment method.

Suppose, instance i has the maximum gradient norm and its corresponding is g_i. Instance j's gradient is g_j. The standard deviation \sigma_j of the Gaussian noise to perturbate the gradient g_j to make the expected norm be equal, we have:

$$E(||g_i||^2) = E(||g_j||^2(1 + 2\sigma_j + \sigma_j^2))$$
$$E(||g_i||^2) = E(||g_j||^2(1 + \sigma_j^2))$$
$$\sigma_j^2 = \frac{||g_i||^2}{||g_j||^2} - 1 \tag{29}$$
$$\sigma_j = \sqrt{\frac{||g_i||^2}{||g_j||^2} - 1}$$

The perturbated gradient **g_j hat** is g_j (1 + \sigma_j).
The algorithm described above is:

Algorithm 6 Preventing Label Leakage 2

Input: $\mathbf{g}$: gradient sending back to the passive party; n by d where n is the batch size, d is the number embedding dimension.

Output: $\mathbf{g}'$: gradient with Gaussian noise

compute $max\_norm = \max(||\mathbf{g}_i||^2), \forall i \in [1, n]$

for $i \leftarrow 1$ *to* n **do**

  /* compute the standard deviation of the Gaussian noise */

  $\sigma_i = \sqrt{max(\frac{||max\_norm||^2}{||\mathbf{g}_i^2||} - 1, 0)}$

  $noise_i \sim \mathcal{N}(0, \sigma_i^2)$ // generate the zero mean Gaussian noise

  $\mathbf{g'}_i = \mathbf{g}_i(1 + noise_i)$ // add the generated noise to the gradient

Return $\mathbf{g}'$

The advantage of this one is that it requires less information than the previous one (not need to know the label information of each instance).

sumKL Minimization

The max norm alignment methods can prevent the passive party extracting the groud-truth labels via 2-norm attack. However, it may not be effective for some other attacks (unknown so far) which do not leverage gradient norms. Another drawback is it does not have a theoretical guarantee for the protection ability.

Here, we introduce a way to make the gradients of positive and negative instances undistinguishable from the gradient distribution perspective. We also give a theoretical guarantee for this protection method.

Optimization Setup

For the sake of simplicity, we first assume that

- $\mathcal{N}(g^{(1)}, \Sigma_+)$ is the distribution of g of the **positive** examples in the batch
- $\mathcal{N}(g^{(0)}, \Sigma_-)$ is the distribution of g of the **negative** examples in the batch

We add iid noise

- $\epsilon^{(1)} \overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \Sigma_1)$ to every positive gradient g
- $\epsilon^{(0)} \overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \Sigma_0)$ to every negative gradient g.

Thus over the entire batch,

- $\widetilde{P}^{(1)} := \mathcal{N}(g^{(1)}, \Sigma_+ + \Sigma_1)$ is the perturbed positive gradient distribution.
- $\widetilde{P}^{(0)} := \mathcal{N}(g^{(0)}, \Sigma_- + \Sigma_0)$ is the perturbed negative gradient distribution.

We also define a label recovering function which can determine the label of a target instance.

Consider a general class of label recovering functions $\{\mathbb{1}_A, A \subset \mathbb{R}^d\}$, where for any set $A \subset \mathbb{R}^d$,
the labelling function $\mathbb{1}_A : \mathbb{R}^d \to \{0, 1\}$, where $\mathbb{1}_A(x) = \mathbb{1}(x \in A)$.

The label recovering function of the 2-norm attack method we described before can be defined as follong:

This general class of labelling functions encompasses the 2-norm detection method.

$A = \{g \in \mathbb{R}^d : \|g\|_2 \geq t\}$ for some fixed threshold t.

We consider an instance as a positive one when its gradient norm is larger than a threshold t.

Given a perturbed gradient and some label recovering function, we use False Positive Rate （FPR） and False Negative Rate (FNR) to evaluate the protection (or leakage) quality.

$$
\begin{aligned}
\text{False Negative Rate (FNR)} &= \mathbb{P}_{\widetilde{g}\sim\tilde{P}^{(1)}}(\mathbb{1}_A(\widetilde{g})=0)\\
&= \mathbb{P}_{\widetilde{g}\sim\tilde{P}^{(1)}}(\widetilde{g}\in A^c)\\
&= \mathbb{P}_{\tilde{P}^{(1)}}(A^c)\\
\text{False Positive Rate (FPR)} &= \mathbb{P}_{\widetilde{g}\sim\tilde{P}^{(0)}}(\mathbb{1}_A(\widetilde{g})=1)\\
&= \mathbb{P}_{\widetilde{g}\sim\tilde{P}^{(0)}}(\widetilde{g}\in A)\\
&= \mathbb{P}_{\tilde{P}^{(0)}}(A).
\end{aligned}
$$

The attacker (passive party) would like to minimize the FNR and FPR. Hence, we can define a mixture error probability as following:

The adversary would equally want FNR and FPR to be as low as possible (not confuse pos as neg or neg as pos)
Hence, consider mixture error probability:

$$M(\tilde{P}^{(1)},\tilde{P}^{(0)},A)=0.5\cdot\text{FNR}+0.5\cdot\text{FPR}.$$

) If we know the fraction of positive examples is p, we can consider the objective

$$M_p(\tilde{P}^{(1)},\tilde{P}^{(0)},A)=p\cdot\text{FNR}+(1-p)\cdot\text{FPR}=p\,\mathbb{P}_{\tilde{P}^{(1)}}(A^c)+(1-p)\,\mathbb{P}_{\tilde{P}^{(0)}}(A).$$

The adversary would equally want FNR and FPR to be as low as possible (notconfuse pos as neg or neg as pos). The most malicious adversary chooses the perturbed distributions so that its error rate is thelowest(recover the most number of labels correctly).

Thus we define the worst case error probability as

$$\text{Error}(\tilde{P}^{(1)},\tilde{P}^{(0)})=\min_{A\subset\mathbb{R}^d}M(\tilde{P}^{(1)},\tilde{P}^{(0)},A).$$

Through some derivation, we have:

$$
\begin{aligned}
&\text{Error}_{0.5}(\tilde{P}^{(1)}, \tilde{P}^{(0)}) \\
=\ &\min_A M_{0.5}(\tilde{P}^{(1)}, \tilde{P}^{(0)}, A) \\
=\ &\min_A 0.5 \cdot \text{FNR} + 0.5 \cdot \text{FPR} \\
=\ &\min_A 0.5 \cdot \mathbb{P}_{\tilde{P}^{(1)}}(A^c) + 0.5 \cdot \mathbb{P}_{\tilde{P}^{(0)}}(A) \\
=\ &\min_A 0.5 \cdot (1 - \mathbb{P}_{\tilde{P}^{(1)}}(A)) + 0.5 \cdot \mathbb{P}_{\tilde{P}^{(0)}}(A) \\
=\ &\min_A 0.5 - 0.5 \cdot (\mathbb{P}_{\tilde{P}^{(1)}}(A)) - \mathbb{P}_{\tilde{P}^{(0)}}(A)) \\
=\ &0.5 - 0.5 \cdot \left[\max_A (\mathbb{P}_{\tilde{P}^{(1)}}(A)) - \mathbb{P}_{\tilde{P}^{(0)}}(A))\right] \\
=\ &0.5 - 0.5 \cdot \text{TV}(\tilde{P}^{(1)}, \tilde{P}^{(0)}),
\end{aligned}
$$

where $\text{TV}(P, Q)$ is the total variation distance between distribution P and Q. Based on this equality, our optimization problem with respect to Σ_1, Σ_0 becomes:

$$
\begin{aligned}
&\arg\max_{\Sigma_1, \Sigma_0} \text{Error}_{0.5}(\tilde{P}^{(1)}, \tilde{P}^{(0)}) \\
=\ &\arg\min_{\Sigma_1, \Sigma_0} \text{TV}(\tilde{P}^{(1)},\ \tilde{P}^{(0)}) \\
=\ &\arg\min_{\Sigma_1, \Sigma_0} \text{TV}(\mathcal{N}(g^{(1)}, \Sigma_+ + \Sigma_1),\ \mathcal{N}(g^{(0)}, \Sigma_- + \Sigma_0))
\end{aligned}
$$

In addition, we have a tighter bound of total variation using KL divergence by **Pinsker's inequalities**: $\text{TV}(P, Q) \leq \sqrt{\text{KL}(P \parallel Q)/2}$. Thus we can get an upper bound of total variation distance by symmetrized KL (TV is a distance and is symmetric) with Jensen's inequality:

$$
\text{TV}(P, Q) \leq \frac{\sqrt{\frac{\text{KL}(P \parallel Q)}{2}} + \sqrt{\frac{\text{KL}(Q \parallel P)}{2}}}{2} \leq \frac{\sqrt{\text{KL}(P \parallel Q) + \text{KL}(Q \parallel P)}}{2}.
$$

Thus our optimization objective becomes

$$
\min_{\Sigma_1, \Sigma_0} \text{KL}(\mathcal{N}(g^{(1)}, \Sigma_+ + \Sigma_1) \parallel \mathcal{N}(g^{(0)}, \Sigma_- + \Sigma_0)) + \text{KL}(\mathcal{N}(g^{(0)}, \Sigma_- + \Sigma_0) \parallel \mathcal{N}(g^{(1)}, \Sigma_+ + \Sigma_1))
$$

And then,

Let the minimal sum of KL be denoted as sumKL*. Then we know for our minimizing solution $(\widetilde{\Sigma}_1,\ \widetilde{\Sigma}_0)$, we have

$$
\begin{aligned}
&\text{Error}_{0.5}(\tilde{P}^{(1)}, \tilde{P}^{(0)}) \\
\geq\ &\frac{1}{2} - \frac{\sqrt{\text{sumKL}^*}}{4}.
\end{aligned}
$$

To maximize the error probability，we need minimize sumKL. One extreme case is that we add infinity noise for both negative and positive instances, then the sumKL will be 0. The error

probability will be 0.5 which's equivalent to the random guess. However, the model performance can drop a lot. It's better to add some constraints for the noise.

add the noise power constraint

$$p \cdot \mathrm{tr}(\Sigma_1) + (1-p) \cdot \mathrm{tr}(\Sigma_0) \leq P,$$

where p is the fraction of positive examples in the minibatch, and P is a tunable hyperparameter to control the amount of total variance.

Now the problem becomes:

$$\begin{aligned} \min_{\Sigma_1, \Sigma_0 \in \mathbb{S}_+^{d \times d}} & \mathrm{KL}(\mathcal{N}(g^{(1)}, \Sigma_+ + \Sigma_1) \parallel \mathcal{N}(g^{(0)}, \Sigma_- + \Sigma_0)) \\ & + \mathrm{KL}(\mathcal{N}(g^{(0)}, \Sigma_- + \Sigma_0) \parallel \mathcal{N}(g^{(0)}, \Sigma_- + \Sigma_0)) \\ \text{subject to} \quad & p \cdot \mathrm{tr}(\Sigma_1) + (1-p) \cdot \mathrm{tr}(\Sigma_0) \leq P \end{aligned}$$

The larger P value, the larger the search space we have. If the P is positive infinity, then the sumKL will become 0. The labeling function will work like a random guess. A larger P value can introduce more more noise to the instances, and it can hurt the model's performance more easily.

Minimizing the sumKL

Suppose u and v are standard deviations of the noise added to the negative and positive instance respectively:

Approximate $\Sigma_- \approx uI$ and $\Sigma_+ \approx vI$, we now have

$$\begin{aligned} \min_{\Sigma_1, \Sigma_0 \in \mathbb{S}_+^{d \times d}} & \mathrm{KL}(\mathcal{N}(g^{(1)}, vI + \Sigma_1) \parallel \mathcal{N}(g^{(0)}, uI + \Sigma_0)) \\ & + \mathrm{KL}(\mathcal{N}(g^{(0)}, uI + \Sigma_0) \parallel \mathcal{N}(g^{(0)}, vI + \Sigma_0)) \\ \text{subject to} \quad & p \cdot \mathrm{tr}(\Sigma_1) + (1-p) \cdot \mathrm{tr}(\Sigma_0) \leq P \end{aligned}$$

Under this assumption $\Sigma_- = uI_d$ and $\Sigma_+ = vI_d$ and $u, v > 0$, we express

$$\begin{aligned}\Sigma_0 =& Q^T \text{diag}(\lambda_1^{(0)}, \dots, \lambda_d^{(0)})Q \\ \Sigma_1 =& Q^T \text{diag}(\lambda_1^{(1)}, \dots, \lambda_d^{(1)})Q,\end{aligned}$$

where $Q \in \mathbb{R}^{d\times d}$ orthogonal matrix and the eigenvalues $\lambda_i^{(0)}, \lambda_i^{(1)}$ are nonnegative and decreasing in value.

Now the optimization becomes

$$\min_{\{\lambda_i^{(1)}\},\{\lambda_i^{(0)}\},Q} \sum_{i=1}^{d} \frac{\lambda_i^{(0)} + u}{\lambda_i^{(1)} + v} + \sum_{i=1}^{d} \frac{\lambda_i^{(1)} + v}{\lambda_i^{(0)} + u} + \left[Q(g^{(1)} - g^{(0)})\right]^T \text{diag}\left(\dots, \frac{1}{\lambda_i^{(0)} + u} + \frac{1}{\lambda_i^{(1)} + v}, \dots\right) Q(g^{(1)} - g^{(0)})$$

$$\begin{aligned}\text{subject to} \quad & p(\sum_{i=1}^{d} \lambda_i^{(1)}) + (1-p)(\sum_{i=1}^{d} \lambda_i^{(0)}) \le P \\ & -\lambda_i^{(1)} \le 0, \ \forall i \in [d] \\ & -\lambda_i^{(0)} \le 0, \ \forall i \in [d].\end{aligned}$$

Through simplication, we can convert this problem into a $\mathbf{4}$ variable optimization problem instead of $O(d^2)$. ($c \coloneqq \left\|g^{(1)} - g^{(0)}\right\|_2^2$)

$$\begin{aligned}\min_{\lambda_1^{(0)},\lambda_1^{(1)},\lambda_2^{(0)},\lambda_2^{(1)}} \quad & (d-1)\frac{\lambda_2^{(0)} + u}{\lambda_2^{(1)} + v} + (d-1)\frac{\lambda_2^{(1)} + v}{\lambda_2^{(0)} + u} + \frac{\lambda_1^{(0)} + u + c}{\lambda_1^{(1)} + v} + \frac{\lambda_1^{(1)} + v + c}{\lambda_1^{(0)} + u} \\ \text{subject to} \quad & \\ & p\lambda_1^{(1)} + p(d-1)\lambda_2^{(1)} + (1-p)\lambda_1^{(0)} + (1-p)(d-1)(\lambda_2^{(0)}) \le P \\ & -\lambda_1^{(1)} \le 0, \quad -\lambda_1^{(0)} \le 0, \quad -\lambda_2^{(1)} \le 0, \quad -\lambda_2^{(0)} \le 0 \\ & \lambda_2^{(1)} - \lambda_1^{(1)} \le 0 \\ & \lambda_2^{(0)} - \lambda_1^{(0)} \le 0.\end{aligned}$$

With insights from the KKT conditions, we can optimize this very quickly through an alternating minimization algorithm like SMO used in SVM solver.

After we solve all the lambdas, the covariance matrix of noise to be added to the negative instances are:

With solution to (10) $(\lambda_1^{(0)*}, \lambda_2^{(0)*}, \lambda_1^{(1)*}, \lambda_2^{(1)*})$, we see that

$$\begin{aligned}\Sigma_0^* &= Q^T \text{diag}(\lambda_1^{(0)*}, \lambda_2^{(0)*} \dots, \lambda_2^{(0)*})Q \\ &= \frac{\lambda_1^{(0)*} - \lambda_2^{(0)*}}{\|g^{(1)} - g^{(0)}\|_2^2}(g^{(1)} - g^{(0)})(g^{(1)} - g^{(0)})^T + Q^T \text{diag}(\lambda_2^{(0)*}, \lambda_2^{(0)*} \dots, \lambda_2^{(0)*})Q \\ &= \frac{\lambda_1^{(0)*} - \lambda_2^{(0)*}}{\|g^{(1)} - g^{(0)}\|_2^2}(g^{(1)} - g^{(0)})(g^{(1)} - g^{(0)})^T + \lambda_2^{(0)*} I_d\end{aligned}$$

We know that the distribution of $X \sim \mathcal{N}(\mathbf{0}, \Sigma_0^*)Y + Z$, where

$$Y \sim \mathcal{N}(\mathbf{0}, \frac{\lambda_1^{(0)*} - \lambda_2^{(0)*}}{\|g^{(1)} - g^{(0)}\|_2^2}(g^{(1)} - g^{(0)})(g^{(1)} - g^{(0)})^T),$$
$$Z \sim \mathcal{N}(\mathbf{0}, \lambda_2^{(0)*} I_d).$$

For $W = \sigma \epsilon g$, where $\epsilon \sim \mathcal{N}(0, 1)$. Then

$$\begin{aligned}\text{Cov}(W) =& \mathbb{E}[(W - \mathbb{E}[W])(W - \mathbb{E}[W])^T] \\ =& \mathbb{E}[\sigma^2 \epsilon^2 g g^T] \\ =& \sigma^2 \, \mathbb{E}[\epsilon^2] g g^T \\ =& \sigma^2 g g^T\end{aligned}$$

As a result, we can construct Y as

$$Y = \epsilon \left(\sqrt{\lambda_1^{(0)*} - \lambda_2^{(0)*}} \right) \frac{g^{(1)} - g^{(0)}}{\|g^{(1)} - g^{(0)}\|_2}, \ \epsilon \sim \mathcal{N}(0, 1)$$

And we can construct Z as

$$Z = \sqrt{\lambda_2^{(0)*}} \boldsymbol{\epsilon}, \ \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I_d)$$

Y, Z are the corresponding Gaussian noise we should add to perturbate the gradients of the negative instances.

Regarding positive instances,

Similarly, we have

$$\begin{aligned}\Sigma_1^* &= Q^T \text{diag}(\lambda_1^{(1)*}, \lambda_2^{(1)*}, \ldots, \lambda_2^{(1)*}) Q \\ &= \frac{\lambda_1^{(1)*} - \lambda_2^{(1)*}}{\|g^{(1)} - g^{(0)}\|_2^2}(g^{(1)} - g^{(0)})(g^{(1)} - g^{(0)})^T + \lambda_2^{(1)*} I_d\end{aligned}$$

Approximate Theoretical Guarantees

As a result of this observation, it is natural to require a lower bound L of $\text{Error}_{0.5}$ for our final perturbed gradient distribution $(\tilde{P}^{(1)}, \tilde{P}^{(0)})$. To achieve this, we require that

$$\text{Error}_{0.5}(\tilde{P}^{(1)}, \tilde{P}^{(0)}) \geq \frac{1}{2} - \frac{\sqrt{\text{sumKL}^*}}{4} \geq L$$

Given the lower bound L of the error probability, we introduce a way to find the best P value dynamically which can satisfy our optimization requirements.

tion of the learning problem. Hence the "best" P is the smallest value of P such that the covariance solution $\widetilde{\Sigma}_1$ and $\widetilde{\Sigma}_0$ satisfies exactly $\text{sumKL}^* = (2 - 4L)^2$. To approximately achieve this, we find P use a search algorithm:

$P \leftarrow g$.

If $\text{sumKL}^* > (2 - 4L)^2$:

$P \leftarrow 1.5 * P$.

Else:

Stop and use this value of P.

We can control our solution's sumKL* by changing the power constraint hyperparameter P that governs the amount of variance of the noise.

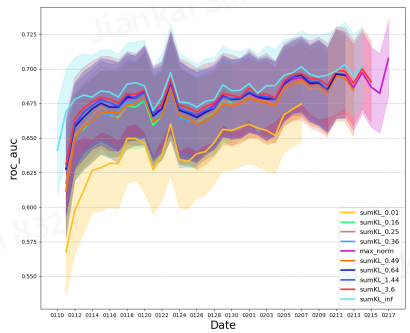
Thus we can very achieve different lower bounds of the adversary's error probability such as $40\%, 30\%, 20\%, 10\%$ etc by changing P.

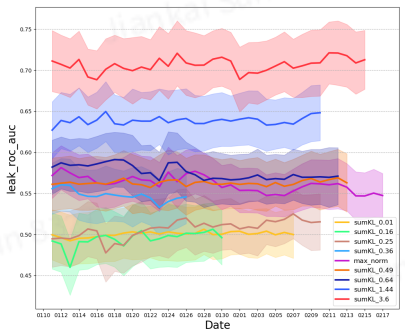
- Error lower bound by 40% requires $\text{sumKL}^* \leq 0.16$.
- Error lower bound by 30% requires $\text{sumKL}^* \leq 0.64$.

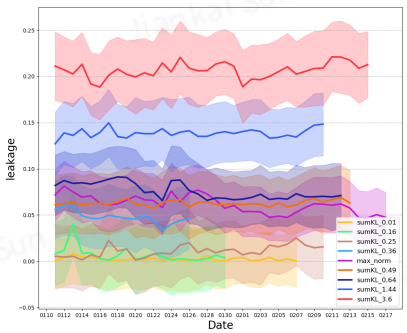
Experiments

Effectiveness

We tested above algorithms on a real CVR dataset which starts from 2020.01.11. We show the corresponding comparisons with confidence bands as follows:







- roc_auc: shows the mean performance auc per day with confidence bands
- leak_roc_auc: shows the mean leakage auc per day with confidence bands

- Leakage: shows |leak_roc_auc - 0.5|
- From the above figures, we can observe:
    - Performance: sumKL_inf > [sumKL_3.6 ~ max_norm ~ sumKL_1.44 ~ sumKL_0.64~sumKL_0.49] > sumKL_0.16 > sumKL_0.01.
        - The average diff between sumKL_inf and sumKL_0.01 is 0.04
        - The average diff between sumKL_inf and sumKL_0.16 is 0.014
        - The average diff between sumKL_inf and sumKL_0.49 is 0.012
    - Leakage (|leak_auc - 0.5|): sumKL_inf >> sumKL_3.6 >> >> sumKL_1.44 >> sumKL_0.64 > sumKL_0.49 > max_norm >> sumKL_0.16 > sumKL_0.01
- By varying the value of sumKL, we can see the trade off between training perforance and label protection effectiveness. A reasonable sumKL value can be set between 0.16 and 0.64.
    - The corresponding performance auc drop will be less than 0.02, and the leak_auc will be between [0.5, 0.6].

Efficiency

We also tested the efficiency of the sumKL minization method. Currently, we leverage tf.py_func (a little bit slow in comparison with using C++ ops) to implement the corresponding function to call the alternating minimization algorithm to compute lamdas. For example, when we set sumKL 0.04, the total time cost of computing lamdas is about 70ms.

Name	Wall Duration	Self time	Average Wall Duration	Occurrences	Event(s)	Link
PyFunc	68.124 ms	68.124 ms	68.124 ms	1	Incoming flow	input_layer/slot_1001_embedding/AsString
IteratorGetNext	61.912 ms	61.912 ms	61.912 ms	1	Incoming flow	input_layer/slot_100_embedding/AsString
RecvTensor	9.399 ms	9.118 ms	0.409 ms	23	Incoming flow	input_layer/slot_4_embedding/AsString
AsString	1.489 ms	1.489 ms	0.186 ms	8	Incoming flow	input_layer/slot_999_embedding/AsString
Const	1.467 ms	1.467 ms	0.025 ms	59	Incoming flow	input_layer/slot_900_embedding/AsString
GatherV2	1.378 ms	1.378 ms	0.020 ms	68	Incoming flow	input_layer/slot_73_embedding/slot_73_embedding_weights/embedding_lookup_sparse/Unique
Tile	1.374 ms	1.374 ms	0.060 ms	23	Incoming flow	input_layer/slot_73_embedding/slot_73_embedding_weights/embedding_lookup_sparse/Unique:1
					Incoming flow	EagerPyFunc_1

Conclusion

Summary

- We propose to use 2-norm attack method to extract the ground-truth labels of the the active party
- We propose two different methods to prevent the label leakage
    - Max norm alignment
        - Can prevent 2-norm attack
    - sumKL minimization
        - With approximate theoretical guarantees
- We conducted several experiments to demonstrate the effectiveness of our proposed methods
- Codes: https://code.byted.org/jiankai.sun/deep_ctr/tree/master/examples

Future work

1. Design some new attack methods which avoid to use the gradient norm information
2. Spped up the alternating minimization algorithm by converting it to C++ op if necessary

Acknowledgement

- We have applied patents for our two protection methods: max norm alignment and sumKL minimization
- We would like to express our thanks to all the collaborators. Without them, we cannot finish the project.
  - Main contributors in ByteDance: Jiankai Sun, Weihao Gao, Hongyi Zhang, Junyuan Xie, Liangchao Wu, Chong Wang
  - Main intern contributors: Runliang Li, Xin Yang

- If you have any questions/suggestions, please contact: jiankai.sun@bytedance.com

References

[1] iDLG: Improved Deep Leakage from Gradients [arXiv]