Technical Report of Embedding Protection in Federated Learning

Jiankai Sun, Weihao Gao, Chong Wang
Applied Machine Learning Group, ByteDance Inc.
{jiankai.sun, weihao.gao, chong.wang}@bytedance.com

Abstract

In this paper, we make the vertical federated learning more secure, by preventing the leakage of the participants' embedding information in a two-party split learning setting. Particularly, we propose several techniques to increase the difficulty of reconstructing the raw attributes/features from the communicated embedding between different participants. As a result, the adversary attacker cannot leverage the shared embedding for other unknown task. We then experimentally demonstrate the effectiveness and competitiveness of the proposed protection methods compared to other baselines.

1 Introduction

With the increasing concerns on data security and user privacy in machine learning, *federated learning* (FL) [20] becomes a promising solution to privacy and security challenges. Based on how sensitive data are distributed among various parties, FL can be classified into different categories [32], notable among which are *vertical (cross-silo) FL* and *horizontal (cross-device) FL*. In contrast to horizontal FL where the data are partitioned by examples, vertical FL assumes that the data are partitioned by different features (including labels). A typical example of vertical FL is when an online media platform A displays advertisements of an ecommerce company B to its users and charges B for each *conversion* (e.g., user clicking the ad and buying the product). In this case, both parties have different features for the same set of users: A has features on users' media viewing records and B has the users' product browsing records and conversion labels. The conversion labels are not available to A because users' buying processes happen entirely on B's website/mobile app.

If both parties want to train a deep learning model jointly to predict the conversion rate based on feature-partitioned data, they could sign legal agreements to allow them share the data with each other. However, due to privacy and business concerns, recently companies and other entities are unwilling to share their data with each other or other third parties. *Split learning* [15, 30] enables the joint training of deep learning models without sharing sensitive data by splitting the execution of a deep learning model between the parties on a layer-wise basis. In vanilla split learning, the party without labels (called the *passive party*) sends the computation results of the intermediate layer (called the *cut layer*) rather than the raw data to the party with labels (called *active party*). The active party then completes the rest of the forward step computation, computes the gradients based on the labels, and sends the gradient with respect to the cut layer back to the passive party. The passive party then completes the back propagation with the gradients of the cut layer using chain rule.

At first glance, the process of split learning seems private because no raw feature or label is communicated between the two parties. However, the computation results of the cut layer (named as intermediate embedding) still contain rich information which can breach users' privacy. For example, Mahendran et al. showed that an attacker can exploit the intermeidate embedding to reconstruct the raw image, and hence the person on the raw image can be re-identified from the reconstructed image [19]. In addition, Li et al. [18] demonstrated that an attacker can leverage the intermediate embedding to infer private attributes, such as gender and age. Recently, Zhu et al. (2019) demonstrated that in horizontal FL setting, the central server could recover the raw features and labels of a device using the model parameters and the gradient shared by that device.

In this work, we study the two-party split learning setting for vertical FL with an assumption that the adversary attacker can apply any deep neural network architecture in the adversary reconstructor design. The worst case can happen when the attacker knows the architecture of the passive party and based on that the attack can build the most powerful reconstructor. We propose a framework which can unite several protection techniques to hide sensitive information that can be exploited for reconstructing raw features and inferring private attributes, and still keep useful embeddings for downstream tasks such as conversation rate prediction.

2 Related work

Uncovering the raw data. Even though raw data are not shared across different parties in federated learning, they are still not secure when gradients and model parameters are shared among different parties. In the horizontal FL setting, Zhu et al. (2019) showed that an honest-but-curious server can uncover the raw features and labels of a participating device by knowing the model architecture, parameters, and the communicated gradient of the loss on the device's data.

Privacy protection methods Because sharing intermediate computation results (model parameters, representations, gradients) can leak the raw data, FL communications still need to proceed in a privacy-preserving manner. There are generally three classes of approaches to communicate in a privacy-preserving manner: **1)** cryptography-based methods such as *Secure Multi-party Computation* [2, 5, 9, 23] and *homomorphic encryption*[3, 26]; **2)** system-based methods such as *Trusted Execution Environments* [27, 29]; **3)** perturbation methods such as randomly perturbing the communicated message [1, 21], shuffling the messages [8, 10], reducing message's data-precision, compressing and sparsifying the message [33], etc. Many of the randomness-based privacy protection methods was proposed in the domain of horizontal FL where the aim is to protect the privacy of each example / each device's dataset [1, 4, 13, 21, 22, 24, 25, 28].

3 Embedding protection in split learning

In this section we first use the gradient-based two-party split learning problem for classification as an example to show the workflow of split learning. We then talk about the protection modules one by one.

3.1 Workflow of Split learning

For simplicity, we consider two parties split learning a model for a binary classification problem over the domain $\mathcal{X} \times \{0, 1\}$ as an example. Here the passive and active parties want to learn a composition model $h \circ f$ jointly, where the raw features X and $f : \mathcal{X} \to \mathbb{R}^d$ are stored the passive party side while the labels y and $h : \mathbb{R}^d \to \mathbb{R}$ is on the active party side. Let $\ell = h(f(X))$ be the logit of the positive class where the positive class's predicted probability is given by the sigmoid function as follows: $\widetilde{p}_1 = 1/(1 + \exp(-\ell))$. The loss of the model is given by the cross entropy $L = \log(1 + \exp(-\ell)) + (1 - y)\ell$. During the model inference, the passive party computes $f(X)$ and sends it to the active party who will then compute the rest of computation (Forward in Table 1). [1]

To train the model using gradient descent, the active party starts the gradient computation process by first computing the gradient of the loss with respect to the logit $\frac{dL}{d\ell} = (\widetilde{p}_1 - y)$. Using chain rule, the active party can then compute the gradient of L with respect to h's parameters through ℓ. In order to also allow the passive party to learn f, the active party also computes the gradient of L with respect to the input of the function h. We denote this gradient by $g := \nabla_{f(X)} L = (\widetilde{p}_1 - y) \nabla_a h(a)|_{a=f(X)} \in \mathbb{R}^d$ (last equality by chain rule). After receiving g sent from the active party, the passive party can compute the gradient of L with respect to f's parameters (Backward in Table 1).

[1]For the simplifying of the notation and derivation, we add no additional features in the active party to compute the logit.

$$\begin{tikzcd}
 & & & & & & y \arrow[d] \\
\text{Forward:} & X \arrow[r, "f"] & f(X) \arrow[r, Rightarrow, "\text{comm.}"] & f(X) \arrow[r, "h"] & \ell \arrow[rr] & & L \\
\text{Backward:} & & g \arrow[d] & g \arrow[l, Rightarrow, "\text{comm.}"'] & \frac{dL}{dl} = (\tilde{p_1} - y) \arrow[l] \arrow[d] & & 1 \arrow[ll] \\
 & & f\text{'s param} & & h\text{'s param} & &
\end{tikzcd}$$

Table 1: Computation and communication diagram of model training ($\leftarrow$ and $\downarrow$ represent gradient computation using chain rule)

When B examples are forwarded as a batch, the communicated features $f(X)$ and gradients g will both be matrices of shape $\mathbb{R}^{B \times d}$ with each row belonging to a specific example in the batch. It is important to note that here the gradients as rows of the matrix are gradients of the loss with respect to different examples' intermediate computation results but not the model parameters; therefore, no averaging over or shuffling of the rows of the matrix can be done prior to communication for the sake of correct gradient computation of f's parameters on the passive party side.

Previous work [18, 19, 33] have shown that $f(X)$ still contain rich information which can breach users' privacy. Adversary attackers can leverage $f(X)$ to recover the raw features and infer private attributes, such as gender and age.

3.2 Overview of our protection framework

We now describe our protection framework, which is illustrated in Figure 1. The framework consists of four independent modules: distance correlation minimizing, noise regularization, adversary reconstructor, and label prediction. They can not only work independently, but also can be united in one framework to prevent information leakage effectively.

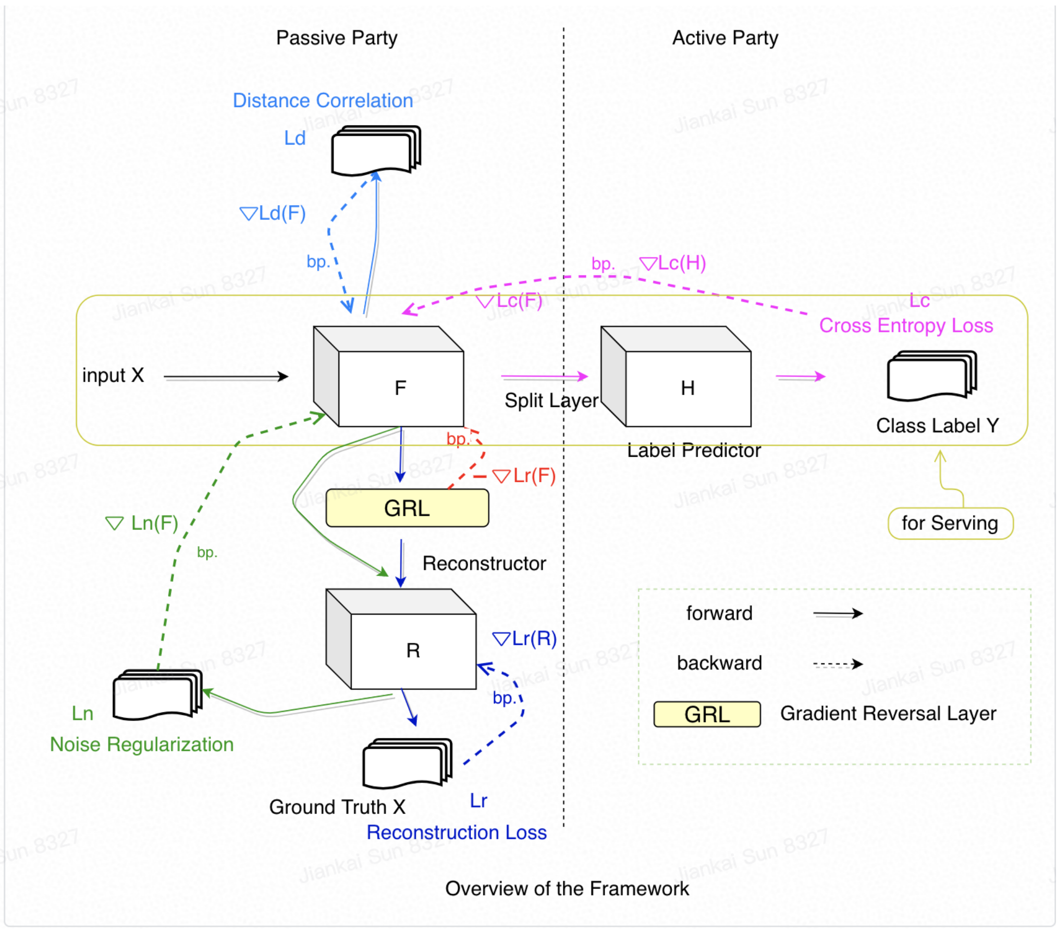


Figure 1: Overview of our protection framework

3.2.1 Minimizing Distance correlation

Distance correlation is a measure of non-linear (and linear) statistical dependence [31]. The distance correlation loss function for n samples of input data X, estimated split layer activations $F(X)$ is given by $L_d = DCOR(X, F(X))$. We reduce the (log of) distance correlation (L_d) between the raw data (X) and activations at the split layer ($F(X)$) during the training of the network, since minimizing this difference can be interpreted as X being a good proxy dataset to construct $F(X)$ but not as vice versa in terms of reconstructing X from $F(X)$ [31].

Disadvantages of using distance correlation

Since computing DCOR needs compute the pairwise distance between instances, the time complexity for computing DCOR between X and $F(X)$ is $O(n^2)$, where n is the batch size. There are some fast estimators of distance correlation requiring $O(nlogn)$ computational complexity for univariate [6] and $O(nKlogn)$ complexity for multivariate settings [16], where K is the number of random projections.

Another drawback is that the distance correlation is computed per batch. It can be sensitive to the batch size. To demonstrate above observation, we compute the distance correlation between X[2] and Normal distribution noise with different batch sizes, as shown in Figure 2. A larger batch size can give a more accurate estimation of the distance correlation, since X contains meaningful information and should have small distance correlation with random noise.

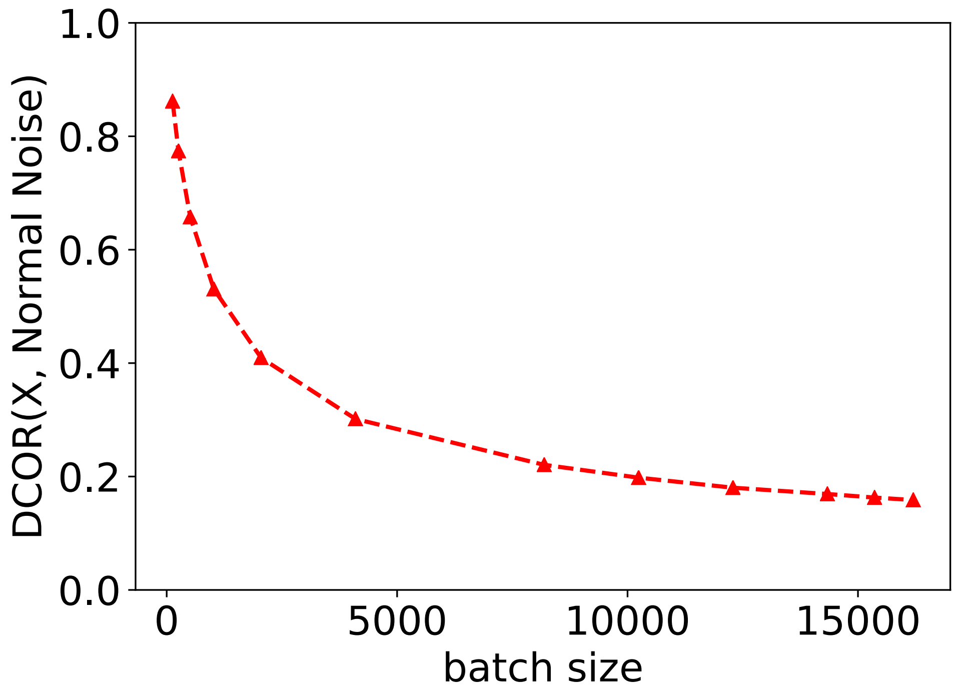


Figure 2: Distance correlation between X and Normal distribution noise with different batch sizes.

3.2.2 Adversary Reconstructor

When playing as an attacker, the adversary reconstructor is trained to optimize the quality of the reconstructed embedding $R(F(X))$ as close as the original X. Here, we use the reconstruction loss $L_r(X, R(F(X)))$ to measure the reconstruction quality during training. On the contrary, a defender expects to degrade the quality of the reconstructed embedding as much as possible (maximizing the

[2] X is a n by 384 matrix from a real CVR dataset. More details of the corresponding dataset can be seen at the Experimental section.

corresponding reconstruction loss). Li et al. [17] and Feutry et al. [11] proposed to adopt Generative Adversarial Networks (GAN) [14] to learn relevant properties (e.g., regular labels) of a dataset as a whole while protecting the privacy of the individual contributors (private labels which can identify a person).

Inspired by the previous work, we proposed to simulate an attacker in the adversarial training procedure by the Reconstructor R which aims to recover the raw X as much as possible. As shown in Figure 1, an encoder network F which is a feature extractor mapping input X to a representation $F(X)$, as well as two branch networks taking $F(X)$ as input, i.e. a predictor H for the ground-truth labels Y and an adversary reconstructor R. The encoder F is trained to help the predictor estimate Y as close as possible, it is also trained by R to prevent the attacker from extracting label relevant information from X by maximizing the reconstruction loss.

Inspired by the using of gradient reversal layer (GRL) in the network architecture of previous work [11, 12], we leverage GRL to help the adversarial training. GRL acts as an identity transform during the forward propagation. However, during the backpropagation, GRL multiplies the corresponding gradient by $-\lambda$ ($\lambda > 0$, not updated backpropogation.) and passes it to the preceding layer. As shown in Figure 1, the GRL is placed between the feature encoder F and the adversary reconstructor R. As the backpropagation process passes through the GRL, the partial derivatives of the reconstruction loss L_r w.r.t. the layer parameters $\frac{\partial L_r}{\partial F}$ is replaced by $-\lambda\frac{\partial L_r}{\partial F}$. Intuitively, during back propagation the output of GRL is basically leading to the opposite of gradient descent that is performing gradient ascent on the feature extractor F with respect to maximize the adversary reconstruction loss and achieving the goal of preventing the attacker from extracting the label relevant information from X.

3.2.3 Noise regularization

When playing as an attacker, the adversary reconstructor is trained to make each reconstructed feature be similar to X. As a defender, we would like to degrade the reconstruction quality.

Inspired by DeepObfuscator [18], we propose to use a noise regularization module aiming to degrade the reconstruction quality by making the reconstructed embedding be close to a random noise. We generate one random Gaussian[3] noise X_{noise} and define the noise regularization as $L_n = ||R(F(X)) - X_{noise}||_2^2$. It's worth mentioning that there is no GRL between F and R in the noise regularization module. As shown in Figure 1, $F(X)$ are feeded into R directly. And we only compute the gradients of the L_r regarding to the parameters of F (not including the reconstructor R) and update the parameters of F correspondingly.

By updating the parameters of F and changing the input of the adversary reconstructor R, the noise regularization module can make the adversary reconstructor be trained to make each reconstructed embedding similar to a random noise but different from X, and the performance of the label predictor should be maintained.

3.2.4 United in one framework

The loss function for the network is a combination of four losses of different modules: (log) distance correlation (L_d), noise regularization (L_n), adversarial reconstruction loss (L_r) and cross entropy loss (L_c). The categorical cross entropy (L_c) is optimized between predicted labels and ground-truth for classification.

$$L = L_c + \alpha_d L_d + \alpha_n L_n + \alpha_r L_r \tag{1}$$

Where $\alpha_d \geq 0$, $\alpha_n \geq 0$ and $\alpha_r \geq 0$ are weights for distance correlation module, noise regularization, and adversary reconstructor module respectively.

4 Experiments

In this section, we evaluate the proposed protection techniques on a real-world large-scale binary classification dataset (anonymous for privacy concerns) with millions of user click records in online advertising. We train a Wide&Deep model [7] where the passive party f consists of the embedding

[3]Random noise from other distributions such as Uniform distribution are effective too.

layer for the input features and several layers of ReLU activated multilayer perceptron (deep part) and the active party consists of the last logit layer of the deep part and the entire wide part of the model. In every iteration, the passive party sends a minibatch of 512 [4] examples' 64-dimensional vector to the active party and the active party sends the gradients of the loss with respect to these vectors back to the passive side. We also tested our methods on the FMINIST [5] dataset as an toy example.

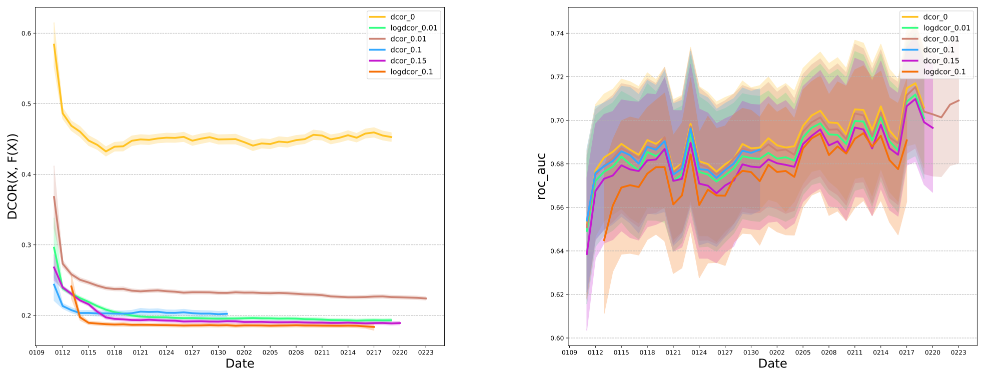
We analyze how our technique compares to a baseline method where i.i.d. isotropic Gaussian noise is added to every embedding of the cut layer in the communicated mini-batch.

This shows that our protection techniques have achieved a better privacy performance trade-off than adding isotropic Gaussian noise.

4.1 Minimizing Distance Correlation

In this section, we study the effects of minimizing distance correlation module specially. The loss function for the network in this experiment is $L = L_c + \alpha_d L_d$. By varying the value of α_d, we can study the effects/sensitivity of minimizing distance correlation, as shown in Figure 3. We can conclude that:

- The network itself in the baseline ($\alpha_d = 0$) can reduce the distance correlation between X and $F(X)$.
- With increasing α_d, we can observe that a smaller $DCOR(X, F(X))$ can be achieved. Comparing with the baseline, we can reduce the $DCOR(X, F(X))$ from 0.45 to 0.2 (when $\alpha_d = 0.1$), while the roc-auc drops less than 0.01. It indicates that minimizing distance correlation can achieve a privacy-performance trade-off with appropriate α_d.
- We also compare the difference between DCOR and log(DCOR) used in the loss function. It seems that the log(DCOR) is more robust than DCOR, since the gap of log(DCOR) between $\alpha_d = 0.01$ and $\alpha_d = 0.1$ is much smaller than DCOR.



(a)

(b)

Figure 3: Figure (a) shows the distance correlation between X and the split layer embedding $f(X)$ with different values of distance correlation weights ($\alpha_d = [0, 0.01, 0.1, 0.15]$). Figure (b) shows the corresponding model performance. $logdcor$ represents that we use $log(DCOR(X, F(X)))$ in the loss function.

To demonstrate that DCOR is sensitive to the batch size, we also show the DCOR(X, $F(X)$) with different batch sizes. Different batch sizes can still achieve the same conclusion as we describe before.

4.2 Noise Regularization

In this section, we study the effect of noise regularization module. Firstly, we investigate the sensitivity of the noise regularization module on types of the random noise. As shown in Figure 5, we compared

[4]The batch size is 512 if not specified.

[5]`https://deepobs.readthedocs.io/en/stable/api/datasets/fmnist.html`

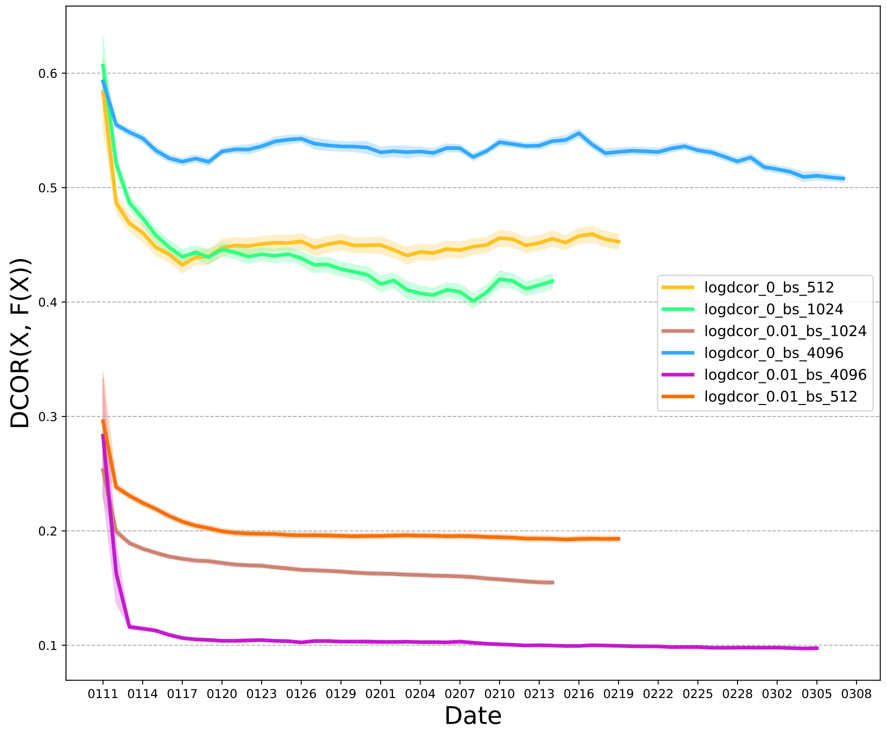


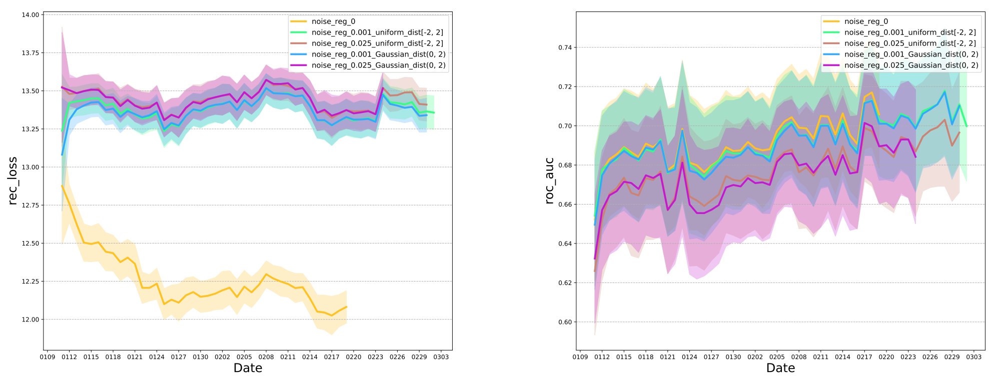
Figure 4: Distance correlation between X and the cut layer embedding $F(X)$ with different batch sizes. logdcor_α_d_bs_n represents that we use $log(DCOR(X, F(X)))$ in the loss function with corresponding weight α_d and the batch size is n.

two different types of random noise: Gaussian distribution noise with mean 0 and standard deviation 2 and Uniform distrubtion noise with min -2 and max 2. With the same α_r, we can observe that both random noise can achieve similar reconstruction loss and model performance. It indicates that the noise regularization module is not sensitive to the type of the random noise.

By varying the α_r, we can see the trade-off between protection and model performance. A larger α_r can cause a larger reconstruction loss (better protection) and a smaller roc_auc score (worse model performance).

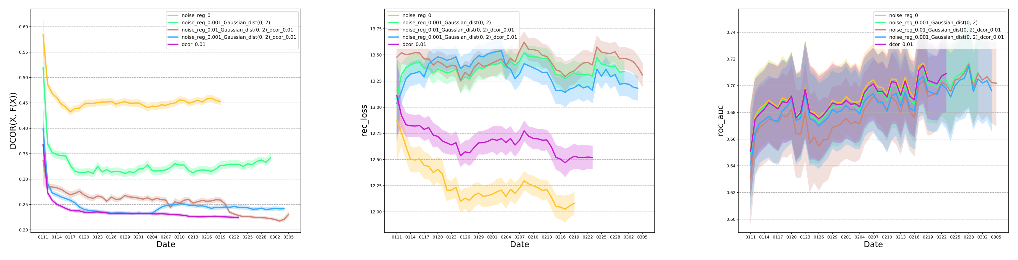
4.2.1 Combining optimizing distance correlation and noise regularization module

As shown in Figure 6, we also study the effects of optimizing distance correlation and noise regularization simultaneously. We can observe from Figure 6 (a) that noise regularization module itself can reduce more $DCOR(X, F(X))$ than the baseline, but performs worse than the distance correlation module, since distance correlation module is optimized specially to reduce the $DCOR(X, F(X))$. However, as shown in Figure 6 (b), distance correlation module has less impacts on maximizing the reconstruction loss than the noise regularization module, since the noise regularization module is designed to prevent recovering X from $F(X)$. Combining optimizing distance correlation and noise regularization module together can help gain the advantages of both modules: reducing the distance correlation $DCOR(X, F(X))$ and increasing the reconstruction loss. As a result, not surprisingly, it can hurt the model performance more than optimizing one of the two modules only as shown in Figure 5 (c).



(a) (b)

Figure 5: Figure (a) shows the reconstruction loss with different types of random noise with different weights ($\alpha_n = [0, 0.001, 0.025]$). Figure (b) shows the corresponding settings' performance evaluated by roc_auc.



(a) (b) (c)

Figure 6: Figure (a) shows the distance correlation between X and the cut layer embedding $F(X)$ of different settings including the baseline ($noise\_reg\_0$), minimizing the distance correlation only with $\alpha_d = 0.01$ ($dcor\_0.01$), Gaussian noise regularization with $\alpha_n = 0.001$, and optimizing distance correlation and noise regularly together with $\alpha_d = 0.01$ and $\alpha_r = 0.001, 0.01$. Figure (b) shows the corresponding settings' reconstruction loss. Figure (c) shows the corresponding settings' performance evaluated by roc_auc.

4.3 Adversarial Optimization

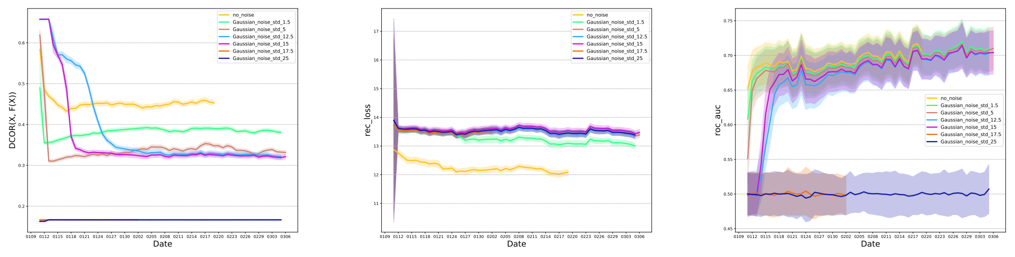
In this section, we conducted our experiments on FMNIST to demonstrate the effectiveness of our proposed protection modules. As shown in Table 2, we test 4 different settings. For example, by setting $\alpha_n = 0$, we turn off the noise regularization module. And the adversarial optimization is turned off if we set $\alpha_r = 0$. By combing adversarial optimization and noise regularization module together, we can achieve the best privacy protection (larger reconstruction loss is better) and the worst model performance (larger cross entropy loss is worse).

	Reconstruction loss	Cross entropy loss
$\alpha_r = 0$, $\alpha_n = 0$	7.949	0.01671
$\alpha_r = 1.0$, $\alpha_n = 0$	114.7	0.1759
$\alpha_r = 0$, $\alpha_n = 1.0$	132.1	0.3953
$\alpha_r = 1.0$, $\alpha_n = 1.0$	256.8	0.6133

Table 2: Protection and Pefromance of different settings on FMNIST

4.4 Add random noise to the split layer

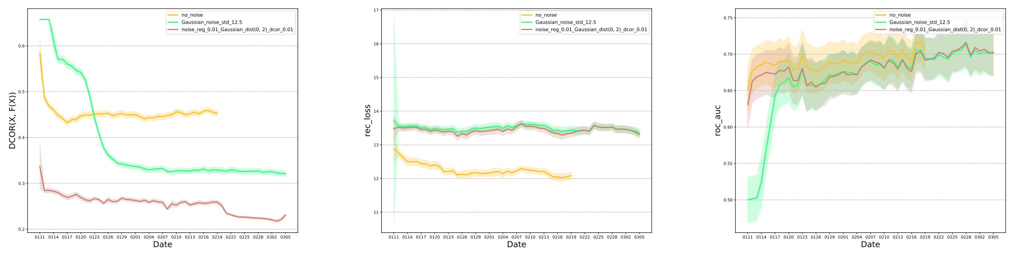
We also conduct several experiments to test a simple and straightforward protection method: adding random noise to the embedding ($F(X)$) of the split layer. Here, the random noise is generated from a Gaussian noise with zero mean. We only tune the standard deviation of the Gaussian noise to control the amount of noise added to $F(X)$. As shown in Figure 7, with increasing the amount of noise added to the split layer, we can get a better protection (smaller distance correlation and larger reconstruction loss) but worse model performance (smaller roc_auc). When the amount of noise is large enough (standard deviation ≥ 17.5), $F(X)$ is dominated by the Gaussian noise. As a result, the roc_auc of the model is 0.5, which is equivalent with a random guess.



(a) (b) (c)

Figure 7: Figure (a) shows the distance correlation between X and the cut layer embedding $F(X)$ of adding 0-mean Gaussian noise with different standard deviations such as $1.5, 5, 12.5, 15, 17.5, 25$. Figure (b) shows the corresponding settings' reconstruction loss. Figure (c) shows the corresponding settings' performance evaluated by roc_auc.

We also compare this straightforward method with our aforementioned modules such as optimizing distance correlation and noise regularization. The corresponding result is shown in Figure 8. Both can have achieve similar model performance and reconstruction loss, since both models can make the reconstructed embedding be similar to the mean of X. Adding Gaussian noise can reduce the distance correlation between X and $F(X)$, but be worse than optimizing distance correlation specially. Overall, with a good control of the amount of the random noise added to the split layer, it can be an effective protection strategy.



(a) (b) (c)

Figure 8: Figure (a) shows the distance correlation between X and the cut layer embedding $F(X)$ of different settings including the baseline (no_noise), and optimizing distance correlation and noise regularly together with $\alpha_d = 0.01$ and $\alpha_r = 0.01$. Figure (b) shows the corresponding settings' reconstruction loss. Figure (c) shows the corresponding settings' performance evaluated by roc_auc.

5 Conclusion

In this paper, we conducted extensive experiments to test several protection techniques in a united framework to prevent attackers from reconstructing the raw features and attributes in the split learning. Particularly, adding random noise to the embedding of the split layer, minimizing distance correlation, optimizing noise regularization, and adversarial reconstruction are demonstrated to be effective on prevent the embedding leakage problem in the federated learning.

Acknowledgements We would like to express our thanks to the team members of applied machine learning group at ByteDance Inc. for proving the collaborative federated learning framework Fedlearner[6]. Particularly, we would like to thank Liangchao Wu, Junyuan Xie, Di Wu,

[6]https://github.com/bytedance/fedlearner

and Xiaobing Liu for their discussions and technical supports. The code is availale at `https://github.com/bytedance/fedlearner/tree/master/example/privacy/`.

References

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pages 308–318, 2016.

[2] N. Agrawal, A. Shahin Shamsabadi, M. J. Kusner, and A. Gascón. Quotient: two-party secure neural network training and prediction. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pages 1231–1247, 2019.

[3] Y. Aono, T. Hayashi, L. Wang, S. Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. IEEE Transactions on Information Forensics and Security, 13(5): 1333–1345, 2017.

[4] A. Bhowmick, J. Duchi, J. Freudiger, G. Kapoor, and R. Rogers. Protection against reconstruction and its applications in private federated learning. arXiv preprint arXiv:1812.00984, 2018.

[5] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 1175–1191, 2017.

[6] A. Chaudhuri and W. Hu. A fast algorithm for computing distance correlation. Computational Statistics Data Analysis, 135:15 – 24, 2019. ISSN 0167-9473. doi: https://doi.org/10.1016/j.csda.2019.01.016. URL `http://www.sciencedirect.com/science/article/pii/S0167947319300313`.

[7] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al. Wide & deep learning for recommender systems. In Proceedings of the 1st workshop on deep learning for recommender systems, pages 7–10, 2016.

[8] A. Cheu, A. Smith, J. Ullman, D. Zeber, and M. Zhilyaev. Distributed differential privacy via shuffling. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 375–403. Springer, 2019.

[9] W. Du, Y. S. Han, and S. Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In Proceedings of the 2004 SIAM international conference on data mining, pages 222–233. SIAM, 2004.

[10] Ú. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, K. Talwar, and A. Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 2468–2479. SIAM, 2019.

[11] C. Feutry, P. Piantanida, Y. Bengio, and P. Duhamel. Learning anonymized representations with adversarial neural networks. 2018.

[12] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. In Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, page 1180–1189. JMLR.org, 2015.

[13] R. C. Geyer, T. Klein, and M. Nabi. Differentially private federated learning: A client level perspective. arXiv preprint arXiv:1712.07557, 2017.

[14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[15] O. Gupta and R. Raskar. Distributed learning of deep neural network over multiple agents. Journal of Network and Computer Applications, 116:1–8, 2018.

[16] C. Huang and X. Huo. A statistically and numerically efficient independence test based on random projections and distance covariance. 2017.

[17] A. Li, J. Guo, H. Yang, and Y. Chen. Deepobfuscator: Adversarial training framework for privacy-preserving image classification. ArXiv, abs/1909.04126, 2019.

[18] A. Li, J. Guo, H. Yang, and Y. Chen. Deepobfuscator: Adversarial training framework for privacy-preserving image classification. CoRR, abs/1909.04126, 2019. URL `http://arxiv.org/abs/1909.04126`.

[19] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 5188–5196, 2015. doi: 10.1109/CVPR.2015.7299155.

[20] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In Artificial Intelligence and Statistics, pages 1273–1282. PMLR, 2017.

[21] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning differentially private recurrent language models. arXiv preprint arXiv:1710.06963, 2017.

[22] H. B. McMahan, G. Andrew, U. Erlingsson, S. Chien, I. Mironov, N. Papernot, and P. Kairouz. A general approach to adding differential privacy to iterative training procedures. arXiv preprint arXiv:1812.06210, 2018.

[23] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In 2013 IEEE Symposium on Security and Privacy, pages 334–348. IEEE, 2013.

[24] V. Pichapati, A. T. Suresh, F. X. Yu, S. J. Reddi, and S. Kumar. Adaclip: Adaptive clipping for private sgd. arXiv preprint arXiv:1908.07643, 2019.

[25] A. Rajkumar and S. Agarwal. A differentially private stochastic gradient descent algorithm for multiparty classification. In N. D. Lawrence and M. Girolami, editors, Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics, volume 22 of Proceedings of Machine Learning Research, pages 933–941, La Palma, Canary Islands, 21–23 Apr 2012. PMLR. URL `http://proceedings.mlr.press/v22/rajkumar12.html`.

[26] S. S. Sathya, P. Vepakomma, R. Raskar, R. Ramachandra, and S. Bhattacharya. A review of homomorphic encryption libraries for secure computation. arXiv preprint arXiv:1812.02428, 2018.

[27] P. Subramanyan, R. Sinha, I. Lebedev, S. Devadas, and S. A. Seshia. A formal foundation for secure remote execution of enclaves. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 2435–2450, 2017.

[28] O. Thakkar, G. Andrew, and H. B. McMahan. Differentially private learning with adaptive clipping. arXiv preprint arXiv:1905.03871, 2019.

[29] F. Tramer and D. Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. arXiv preprint arXiv:1806.03287, 2018.

[30] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. arXiv preprint arXiv:1812.00564, 2018.

[31] P. Vepakomma, T. Swedish, R. Raskar, O. Gupta, and A. Dubey. No peek: A survey of private distributed deep learning. CoRR, abs/1812.03288, 2018. URL `http://arxiv.org/abs/1812.03288`.

[32] Q. Yang, Y. Liu, T. Chen, and Y. Tong. Federated machine learning: Concept and applications. ACM Transactions on Intelligent Systems and Technology (TIST), 10(2):1–19, 2019.

[33] L. Zhu, Z. Liu, and S. Han. Deep leakage from gradients. In Advances in Neural Information Processing Systems, pages 14774–14784, 2019.