Server-side feature inference attacks in split learning*

Xiaochen Zhu†
National University of Singapore
xzhu@u.nus.edu

Abstract

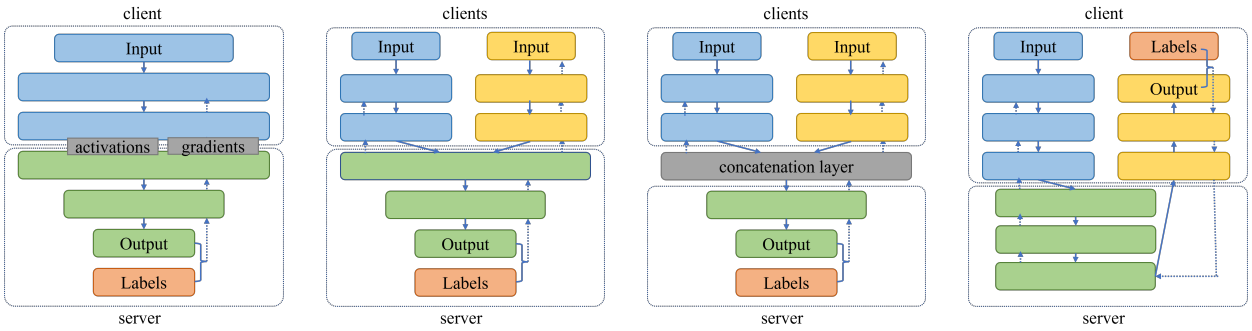
Split learning, a recently proposed protocol for distributed learning of neural networks, has drawn an emerging amount of attention for its performance, communication efficiency and preservation of privacy. This project evaluates the privacy of split learning and proposes a novel server-side passive feature inference attack against the protocol, namely PASIV. More specifically, we show that a malicious server, with the access to a dataset with similar distribution as the client's private data, is able to train an encoder that simulates the client's private model's behaviour by training a discriminator to distinguish the two encoders' outputs. Through experiments, we empirically demonstrate that PASIV can reach similar reconstruction performance compared with existing active attacks and can overcome recently proposed privacy-preserving improvements of split learning. Finally, we discuss some future research plans to further relax the assumptions of PASIV.

1 Introduction

Deep learning with neural networks has been widely adopted for its state-of-the-art performance in many applications, such as computer vision, natural language processing, robotics and recommendation systems. However, in real world settings, deep learning may be difficult to deploy because: (i) the training dataset is not centralized on one server but distributed on databases from multiple clients [28]; (ii) the high computational complexity makes data owners unable to deploy deep learning on their own [22]. In both scenarios, deep learning can be successfully deployed if all data is combined in a centralized server with powerful computational resources. However, data sources are generally reluctant to share its raw data with each other or to a centralized server because: (i) the data contains sensitive information for them to successfully operate or remain competitive; (ii) they have to abide by their internal privacy policies and external regulations such as GDPR [26]; (iii) it may not be feasible or economically efficient to transmit huge amount of data due to limited communication bandwidth between data owners and the server. To address these issues, federated learning (FL) [3, 5, 19, 27] has been proposed and extensively researched to provide secure privacy-preserving communication protocols such that multiple parties can collaboratively learn a shared machine learning (ML) model. Ususally, FL solutions achieve secure communication and collaboration via cryptography like homomorphic encryption and secure multiparty computation (MPC). Recently, another novel approach, split learning (SL) [4, 12, 24], which enables distributed learning of neural networks without directly revealing raw data of any data owner, has drawn attention because of its simple architecture, high communication efficiency and flexibility to multiple configurations [21]. However, split learning does not provide privacy guarentees as strong as federated learning as it is hard to apply encryption or MPC techniques [14]. There have been works [16, 25] developing privacy-preserving improvements of split learning, but a recently proposed server-side feature inference attack, Feature Space Hijacking Attack (FSHA) [20] has been shown to successfully reconstruct clients' private data even with these defense techniques present. However, FSHA is an active attack where the malicious server constantly hijacks the messages sent to the clients and the malicious server requires a dataset with similar distribution as the target private dataset to successfully attack. In reality, these assumptions are too strong to make because: (i) actively attacking private data of collaborating parties means no preservation of original training task and a high risk of being detected as violating agreements and facing serious consequences; (ii) it is usually not possible for the malicious server to find a public dataset with similar distribution as the target private data.

*This is an interim report submitted to Department of Computer Science, School of Computing, in partial fulfillment of the requirements for Undergraduate Research Opportunities Programme.

†Supervised by Prof Xiaokui Xiao, xkxiao@nus.edu.sg, National University of Singapore.



(a) Vanilla split learning (b) Horizontally partitioned (c) Vertically partitioned (d) U-shaped vanilla SL

Figure 1: Configurations of split learning

Therefore, the objectives of this project is to: (i) investigate the security and privacy of split learning via literature review; (ii) propose a novel passive approach of feature inference attack in split learning; (iii) research how the requirements on the public dataset can be relaxed; (iv) discuss the effects of existing privacy-preserving defenses on our newly proposed attack and further suggest possible countermeasures against our attack.

Our contribution so far is to propose a server-side passive feature inference attack that can successfully reconstruct client's private data while preserving the original training task. We also empirically demonstrate that our attack can overcome existing defenses while the malicious party fully complies with the intended protocol. At last, we discuss some future research plans to further relax the assumptions of the proposed attack.

2 Preliminaries and literature review

2.1 Split learning: distributed learning of neural networks

Terminology in deep learning Assume that a neural network (NN) model [15] $H : \mathcal{X}^* \to \mathcal{Y}^*$ is trained on the dataset $D = \{(x_t, y_t) \in \mathcal{X} \times \mathcal{Y} : t = 1, 2, \ldots, N\}$, where $\mathcal{X}$ is the feature space, $\mathcal{Y}$ is the label space of D and $A^* = \{A^n : n \in \mathbb{Z}_+\}$ for any set A. We define H over $\mathcal{X}^*$ because the model can evaluate a batch of examples from $\mathcal{X}$ (usually represented as a matrix). As a deep neural network model, H can be described by a sequence of n layers $L_0, L_1, \ldots, L_n$ such that $H = L_n \circ L_{n-1} \ldots \circ L_0$. In each iteration where the model sees input $X \in \mathcal{X}^*$ (i.e. X can be a single example or a batch of examples), the training of H contains two steps: (i) forward pass: one evaluates the output of the NN model by $\hat{Y} = H(X) = L_n(L_{n-1}(\ldots(L_0(X)))$ and then calculates the loss against actual labels $Y \in \mathcal{Y}^*$ by some loss function $\ell : (\mathcal{Y}^*)^2 \to \mathbb{R}$: $\mathcal{L} = \ell(Y, \hat{Y}) = \ell_Y(\hat{Y})$, where $\ell_Y(\cdot) = \ell(Y, \cdot)$; (ii) backpropogation: for any parameter w in layer L_k, to minimize $\mathcal{L}$, we calculate the gradient by the chain rule of derivatives

$$\begin{aligned} \nabla w = \frac{\partial \mathcal{L}}{\partial w} &= \frac{\partial \ell_Y(H(X))}{\partial H(X)} \frac{\partial L_n(L_{n-1}(\ldots(L_0(X))))}{\partial L_{n-1}(L_{n-2}(\ldots(L_0(X))))} \cdots \frac{\partial L_k(L_{k-1}(\ldots(L_0(X))))}{\partial w} \\ &= \ell'_Y L'_n L'_{n-1} \ldots L'_{k+1} \frac{\partial L_k(L_{k-1}(\ldots(L_0(X))))}{\partial w} \end{aligned} \tag{1}$$

to apply gradient descent updates [15]. Note that to calculate ∇w, we need to know the gradient the loss function, that of all the following layers and the *activations* returned by the previous layer (i.e. $L_{k-1}(\ldots(L_0(X)))$).

Split learning The key idea of split learning (SL) [12] is to *split* the execution of H by a *cut layer* (namely, L_k) and assign the first half including the cut layer to the client and second half to the server (Figure 1a). Assume the model owned by the client (i.e. the first half of the NN) is $f = L_k \circ \ldots \circ L_0$ and the model owned by the server (i.e. the second half of the NN) is $g = L_n \circ \ldots \circ L_{k+1}$, then $H = f \circ g$. On forward passes, the client sends activations returned by the cut layer $Z = f(X)$ (i.e. smashed data) to the server such that it can continues the evaluation by $H(X) = g(Z) = g(f(X))$. On backpropogation rounds, the server can update its parameters on g because it owns the gradients of the following layers and activations from the previous layers (sent from the client in forward passes), while the client can update its parameters on f after the server sends back the product of gradients $\ell'_Y L'_n \ldots L'_{k+1}$, which we simply denote as *gradients*.

Due to its simplicity, split learning can be easily adopted into various configurations. [24] demonstrates an application of SL on horizontally partitioned medical datasets (Figure 1b), while [4] presents that a simple concatenation layer can adapt SL to vertically partitioned data (Figure 1c). If label sharing is not allowed, SL can also be configured with a U-shaped architecture such that the labels are stored with the clients [12] (Figure 1d). However, since the focus of this project is the security and privacy of split learning, we mainly discuss the vanilla configuration, as our results should also work with other configurations.

Privacy vulnerabilities of split learning Split learning is claimed to be privacy preserving because the data owners do not share their *raw data* with other parties. However, the activations sent from the clients and gradients sent back from the server are transmitted in plaintext because complicated functions in neural networks and automatic differentiation are not computable under homomorphic encryption [2]. Hence, transmitted activations and gradients carry information that leads to the following vulnerabilities of split learning:

1. a malicious server may be able to train a model to recover original private data or infer private attributes from the activations;
2. the client relies on gradients transmitted from the server to update its model f, and in the case of a dishonest server, f can be forced to converge towards an undesirable result.

2.2 Privacy preserving improvements of split learning

Minimizing correlation between original and smashed data To mitigate the potential risk of the first vulnerability that a malicious server may train a model to recover the original dataset X from received activations Z, several improvements [16, 18, 25] have been proposed with the same key idea to minimize the correlation between X and Z. It is intuitively effective because if the correlation between X and Z is optimized to be low, it will be harder for a malicious party to reconstruct X from smashed data Z.

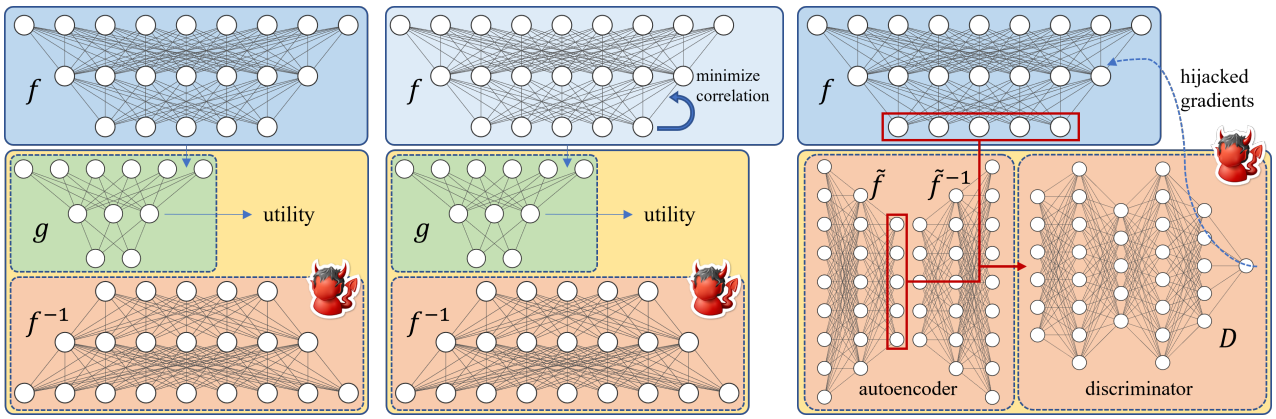
Proposed by [25], *NoPeek* minimizes the correlation by making the client update its neural network f with the optimization goal of minimizing

$$\alpha_1 \cdot \text{dCor}(X, Z) + \alpha_2 \cdot \mathcal{L} \tag{2}$$

, where $\text{dCor}(x, y)$ is the distance correlation [23] between x and y, and α_1, α_2 are hyperparameters controlling the tradeoff between data privacy and utility. Given that, the vanilla SL is simply a special case of the NoPeek approach where $\alpha_1 = 0, \alpha_2 = 1$, prioritizing utility over data privacy. Such replacement of optimization goal can be done because the first term in (2) only depends on f's parameters so the client can minimize the correlation term on its own, while the second term is just part of the regular split learning protocol (Figure 2b). There are other variants of NoPeek depending on different perspectives of data privacy we want to emphasize. For example, if one prioritizes the protection of some private attributes from property inference attacks, the correlation term in (2) can be replaced by the correlation between the private attributes and the activations.

While NoPeek takes a straightforward approach to minimize the correlation, *DeepObfuscator* [16] introduces two additional models to achieve the same goal via adversarial training: an adversary reconstructor that imitates a malicious server executing feature inference attacks, and an adversary classifier that imitates a malicious server executing property inference attacks. The client-side neural network f, other than being updated to minimize the intended loss $\mathcal{L}$, is also trained adversarially to degrade the performance of the adversary models. After introducing tradeoff parameters to control how much the client weights each optimization goal, DeepObfuscator is demonstrated to reduce information leakage while preserving the intended training task.

Differential privacy Currently, the state-of-the-art technique to quantify and reduce information disclosure about individuals is differential privacy (DP) [7, 9]. Assume $\mathcal{D}$ is the set of all possible datasets and $\mathcal{H}$ is the set of all possible learnt models. A randomized algorithm $\mathcal{A} : \mathcal{D} \to \mathcal{H}$ satisfies (ϵ, δ)-DP if and only if for any adjacant datasets $D, D' \in \mathcal{D}$ that differ in only one entry x (i.e. $D = D' \cup \{x\}$), and for any set of models $H_0 \subset \mathcal{H}$, we have $\Pr[\mathcal{A}(D) \in H_0] \leq e^{\epsilon} \Pr[\mathcal{A}(D') \in H_0] + \delta$. Hence, when choosing relatively small ϵ and δ, the probability of the original dataset being D or D' is indistinguishable, so one cannot tell from the learnt model about the existence of x. Usually, DP is achieved by injecting random noise to the data samples or gradients [1, 8], and it provides the worst-case privacy guarentees for membership inference attasks. However, the privacy guarentee we are interested in split learning is different from conventional DP, where we are more



(a) Straightforward attack (b) Decorrelation in NoPeek (c) Feature-space hijacking attack

Figure 2: Server-side feature inference attacks and defenses in split learning

interested in protecting the private features and information of any client-side entry from being reconstructed or inferred given a high dimensional representation of the corresponding entry (i.e. the activations).

2.3 Feature inference attacks against split learning

Utilizing the vulnerabilities introduced in Section 2.1, several server-side feature inference attacks [16, 20, 25] have been proposed to reconstruct private data from the client. These attacks share a common critical component, a *decoder* that aims to invert the model f and recover private original data X_{priv}[1] from received transmitted activations $Z = f(X_{priv})$ but they have different assumptions, threat models and architectures.

Straightforward model inversion attack A straightforward feature inference attack [16, 25] exploits the first vulnerability of split learning (as introduced in Section 2.1), directly training a decoder to reconstruct original data from Z (Figure 2a). To train such a decoder, the malicious server needs to collect leaked data $X_{leak} \subseteq X_{priv}$ and their associated smashed data $f(X_{leak})$ output from client's model f. Using $f(X_{leak})$ as its features and X_{leak} as its labels, a malicious server can train a model to recover any given X from $f(X)$. However, although it may be justifiable to assume that the server has a small portion of the original data in a (not-so-rare) event of data breach [17], it is very hard to justify why the server can also know the activation outputs $f(X_{leak})$ associated with the leaked data X_{leak} when it is not given any (not even black-box) access to client's model f. Besides, without knowing the structure of f, it is hard to for the server to determine the structure of its decoder model to behave like f^{-1}. At last, since the training of the decoder solely depends on X_{leak} and $f(X_{leak})$, the decorrelation defenses mentioned in Section 2.2 can effectively protect the client data from such attacks because the clients publish decorrelated X_{leak} to the server.

Hijacking backpropogated gradients Feature-space hijacking attack (FSHA) [20] improves from the previous attack by introducing an autoencoder [10] to invert server's own white-box encoder and exploiting the second vulnerability of SL via forcing the client-side model to simulate the behavior of server's own encoder. As shown in Figure 2c, the malicious server trains additional models to execute feature inference attacks:

(i) an autoencoder $(\tilde{f}, \tilde{f}^{-1})$ where the encoder and decoder are trained adversarially to minimize the loss between a given input x and decoded $\tilde{f}^{-1}(\tilde{f}(x))$;

(ii) a discriminator D that is trained to distinguish the smashed data output by f and $\tilde{f}$, i.e. D is trained to output 1 for smashed data output by f and 0 for smashed data output by $\tilde{f}$.

The key idea of FSHA is that, by hijacking the gradients sent back to the client, one can force the client's model f to converge to server-side encoder $\tilde{f}$, which can be then inverted by $\tilde{f}^{-1}$ in an autoencoder structure. This is achieved by hijacking the optimization goal of f to minimize the loss between $D(f(X))$ and $\mathbf{0}$, so that f is trained to have smashed data output indistinguishable with that of $\tilde{f}$. With the use of autoencoder

[1] X_{priv} is the same as X defined previously, but used this way to distinguish from other types of data introduced in later sections.

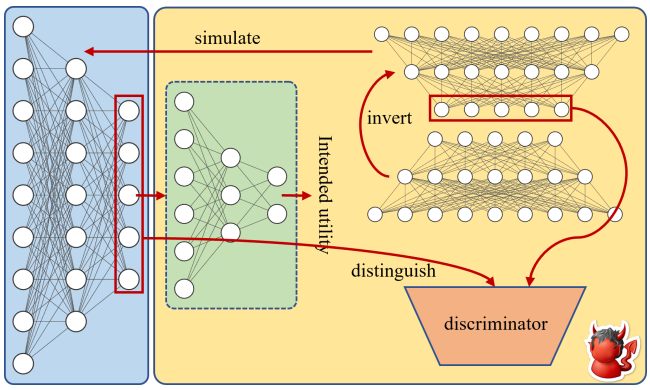


Figure 3: Architecture of PASIV

Figure 4: Preservation of intended task

and discriminator, one no longer needs the access to leaked data X_{leak} and their associated smashed data $f(X_{leak})$ to train the encoder, as the training of autoencoder is self-contained and the discriminator can learn to distinguish the range of f and $\tilde{f}$ without comparing $f(X)$ and $\tilde{f}(X)$ for the same input X. Therefore, FSHA does not require the server to own a leaked subset of the private dataset $X_{leak} \subset X_{priv}$. Instead, the server only needs to have access to a public dataset X_{pub} disjoint with X_{priv}, as long as they share a similar distribution. At last, FSHA can overcome the decorrelation defense in NoPeek because the server has control over the second term of (2) (i.e. $\alpha_2 \mathcal{L}$). The malicious server implementing FSHA is able to scale up hyperparameter α_2 and dominate the total loss, weakening the effect of the decorrelation defense.

Limitations of FSHA Compared to the straightforward model inversion attack, FSHA can achieve better reconstruction performance without knowing f's model structure, leaked data X_{priv} or their associated smashed data $f(X_{priv})$, and can overcome the NoPeek decorrelation defense. However, FSHA has the following inherent limitations:

1. The attack is active since it actively modifies the gradients sent back to the client, which is potentially detectable by the client as violating agreements and will possibly face serious consequences.
2. The original learning task is not preserved because f is not updated to minimize the intended loss $\mathcal{L}$.
3. The attack requires a public dataset sharing similar distribution with the private target data. While it may be feasible in image datasets, it is hardly justifiable for tabular datasets.

3 Our work and preliminary results

3.1 PASIV: A passive server-side feature inference attack

To address the first two limitations of FSHA, we propose a novel server-side feature attack. It is named PASIV because it is a passive attack via model simulation, inversion and adversarial training. Our attack is superior to FSHA because it is a passive attack where the attacker fully complies with the agreed protocols and the intended learning task can be preserved while attacking.

Threat model We assume that the attacker (i.e. the server) holds no information on the collaborating client other than received activations in the training and inference processes of split learning. The attacker does not have any access to client's model f (i.e. it cannot feed f with data of its favor and collect its output) and it has no information about the model structure or parameters other than the dimension of its output. Also, the attacker does not have any examples from client's private dataset X_{priv}, but only a disjoint public dataset X_{pub} of a similar distribution. For example, if X_{priv} contains photos of user's mobile phone gallery, X_{pub} should also contain photos of user's mobile phone gallery, but $X_{priv} \cap X_{pub} = \emptyset$. Most importantly, the attacker is assumed to fully comply with all the protocols in split learning as well as possible defense techniques, without modifying any of the messages it sends to the other parties. Therefore, our threat model is more restrictive than that of [16, 20, 25], making our proposed attack more feasible and robust in real world applications.

Figure 5: Attack performance of FSHA (original images in the first row while reconstructed ones in the second)

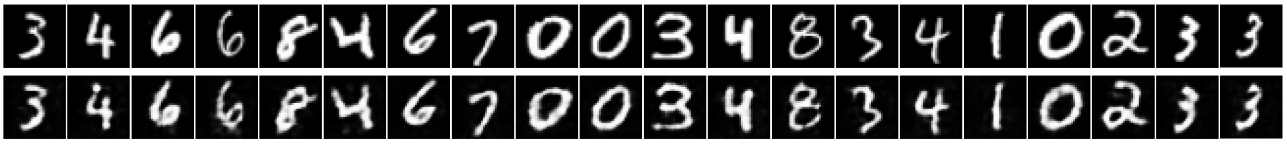


Figure 6: Attack performance of PASIV (original images in the first row while reconstructed ones in the second)

Attack architecture As per Figure 3, PASIV trains three additional models compared with the SL protocol:

(i) an encoder $\tilde{f}$, which is trained on X_{pub} to simulate the behavior of client's side f, i.e. to maximize the probability of being classified as $f(X_{priv})$ by the discriminator (refer to model in (iii));

(ii) a decoder $\tilde{f}^{-1}$. which is trained on $\tilde{X_{pub}}$ to invert $\tilde{f}$, i.e. to minimize the loss between X_{pub} and $\tilde{f}^{-1}(\tilde{f}(X_{pub}))$;

(iii) a discriminator D, which is trained on $\tilde{f}(X_{pub})$ and $f(X_{priv})$ to distinguish them.

After training these models, one can recover private data x given its associated activations $z = f(x)$ via inference of the decoder, i.e. $x \approx \tilde{f}^{-1}(z)$. Note that although the attacker has no information about f's model structure, it can just initialize its encoder $\tilde{f}$ to be a much more complicated model which can model the simpler f. After determining the structure of $\tilde{f}$, $\tilde{f}^{-1}$ can be then determined, as the transpose structure of $\tilde{f}$.

Comparisons with FSHA Although PASIV and FSHA share a similar architecture where the attack trains a encoder, decoder and a discriminator, the key difference between them is that FSHA forces the client-side encoder to converge to server's encoder which is trained adversarially with its decoder as an autoencoder, while PASIV simulates the behavior of client's model and trains a standalone decoder to perform model inversion on the learnt encoder. Therefore, by not training the encoder and decoder together in an autoencoder, one may be concerned about whether the decoder can successfully invert the encoder's model given that it's updated separately for a different goal. However, such issue should not be serious because the decoder is attempting model inversion to a model that it has white-box access to. Our preliminary results show that the decoder can still successfully converge, which will be presented in the following Section 3.2.

3.2 Experimental results and discussion

Implementation details We implement[2] the proposed attack PASIV in Python with TensorFlow and conduct the experiments on the dataset of MNIST handwritten digits [6]. X_{priv} and X_{pub} are disjoint subsects of the dataset. Models for the intended task, f and g, is a smaller version of ResNet following [13]. To demonstrate that the attacker does not know the model structure of f, $\tilde{f}$ is a neural network of convolutional layers (instead of a similar structure with f) and $\tilde{f}$ is the transpose model of $\tilde{f}$. For the discriminator D, we apply Wasserstein loss and gradient penalty for better performance [11].

Preservation of original task Note that the training of models $\tilde{f}, \tilde{f}^{-1}, D$ in PASIV does not disturb the training and inference of f and g. Hence, the attacker sends back honest gradients that aim to minimize the loss for the original task. In contrast, the attacker in FSHA sends back hijacked gradients such that f is updated to converge to $\tilde{f}$. Although the second half of the network g is still updated to minimize $\mathcal{L}$, it should not preserve the original task since f is being optimized for a different goal. As shown in Figure 4, PASIV has lower training loss of the original task than FSHA, almost overlapping with the case with an honest server.

[2]To make our work reproducible, we make the source code available at `https://github.com/zhxchd/sl-privacy`.

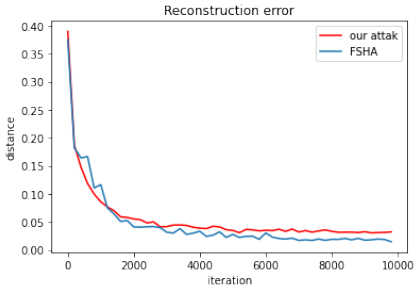


Figure 7: Attack performance

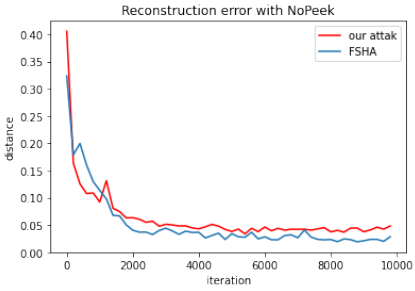


Figure 8: Attack against NoPeek

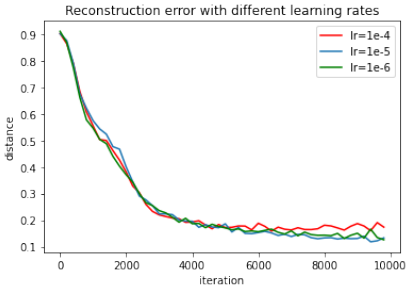


Figure 9: Effects of learning rates

Figure 10: Reconstruction results (second row) of PASIV against NoPeek decorrelation defense

Attack performance According to our experiments, PASIV is able to converge and reconstruct the private features with the same number of iterations as FSHA. This is presented in Figure 5 and 6 where both attacks are executed in 10000 epochs. It is shown in Figure 7 that PASIV is able to reach similar level of reconstruction performance compared with FSHA while complying with the learning protocol.

Against the decorrelation defense FSHA is able to overcome the NoPeek decorrelation defense because it has control over α_2 and can scale up α_2 to cancel the effects of the decorrelation optimization executed on the client. However, the network f after being decorrelated is still a neural network with some other parameters, which can still be simulated by $\tilde{f}$ in PASIV. Hence, PASIV is able to overcome the NoPeek defense without scaling up α_2, fully complying with the NoPeek and SL protocols. Experiments are conducted where $\alpha_1 = 100, \alpha_2 = 1$ and FSHA scales up α_2 by 50 times while PASIV complies with the given hyperparameters. Figure 8 shows that PASIV is able to achieve reasonable reconstruction error with NoPeek present and Figure 10 visually demonstrates the reconstruction performance.

On the choice of learning rate One potential issue with PASIV is that, $\tilde{f}$ is trained to simulate the behavior of f, which may be difficult when f has a high learning rate and changes too fast for $\tilde{f}$ to imitate, while the attacker has no control over the choice of learning rate of f. As per Figure 9, this issue is not serious as the attacks with learning rates $10^{-4}, 10^{-5}, 10^{-6}$ all manage to converge and reach a reasonable reconstruction error in the end. Indeed, the reconstruction error converges more slowly with a higher learning rate, but this can be addressed by continuing the PASIV attack during the inference time when the models f, g are fixed if needed.

4 Future research plans

As presented in Section 3, our current progress is to propose PASIV, a passive server-side feature inference attack that can preserve intended task and overcome the decorrelation defense. Based on that, we plan to study PASIV in greater depth, improve the attack and evaluate possible defense techniques:

1. Conduct more comprehensive experiments on PASIV on more datasets and discuss the effects of various implementation choices, such as the similarity between X_{priv} and X_{pub}.
2. Improve the proposed attack PASIV such that the attack performance is invariant of f's learning rate.
3. Further relax the assumption of PASIV by also addressing the third issue with FSHA. We are to research how the requirement of X_{pub} can be relaxed such that we can extend the application of PASIV to tabular datasets, where the existence of a similar dataset X_{pub} is no longer justifiable to assume. One possible

approach is to conduct weaker attacks like property membership attacks instead of feature inference attacks[3].

4. Propose a defense technique against FSHA (i.e. to detect the hijacked gradients) such that we can better demonstrate the limitations of the active attack and justify the necessity of a passive attack.
5. Propose and evaluate possible countermeasures against our proposed attack, PASIV, to examine how the attack is robust under various defenses.

Acknoledgements

Thanks to Prof Xiaokui Xiao for supervising and guiding the project, and to Yuncheng Wu, Xinjian Luo and Yangfan Jiang for valuable discussions. Thanks to Yiqiu Liu for her support and company.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[2] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Comput. Surv.*, 51(4), July 2018. ISSN 0360-0300. doi: 10.1145/3214303.

[3] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn 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.

[4] Iker Ceballos, Vivek Sharma, Eduardo Mugica, Abhishek Singh, Alberto Roman, Praneeth Vepakomma, and Ramesh Raskar. Splitnn-driven vertical partitioning. *arXiv preprint arXiv:2008.04137*, 2020.

[5] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems*, 2021.

[6] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[7] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[8] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[9] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 14 Autoencoders. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5769–5779, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[12] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018.

[3] This direction is currently under progress.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021. ISSN 1935-8237. doi: 10.1561/2200000083.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[16] Ang Li, Jiayi Guo, Huanrui Yang, Flora D. Salim, and Yiran Chen. Deepobfuscator: Obfuscating intermediate representations with privacy-preserving adversarial learning on smartphones. In *Proceedings of the International Conference on Internet-of-Things Design and Implementation*, IoTDI '21, page 28–39, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383547. doi: 10.1145/3450268.3453519.

[17] Liyuan Liu, Meng Han, Yan Wang, and Yiyun Zhou. Understanding data breach: A visualization aspect. In *International Conference on Wireless Algorithms, Systems, and Applications*, pages 883–892. Springer, 2018.

[18] Sicong Liu, Junzhao Du, Anshumali Shrivastava, and Lin Zhong. Privacy adversarial network: Representation learning for mobile data privacy. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 3(4), December 2019. doi: 10.1145/3369816.

[19] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[20] Dario Pasquini, Giuseppe Ateniese, and Massimo Bernaschi. Unleashing the tiger: Inference attacks on split learning. *arXiv preprint arXiv:2012.02670*, 2020.

[21] Abhishek Singh, Praneeth Vepakomma, Otkrist Gupta, and Ramesh Raskar. Detailed comparison of communication efficiency of split learning and federated learning. *arXiv preprint arXiv:1909.09145*, 2019.

[22] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[23] Gábor J. Székely, Maria L. Rizzo, and Nail K. Bakirov. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6):2769 – 2794, 2007. doi: 10.1214/009053607000000505.

[24] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

[25] Praneeth Vepakomma, Abhishek Singh, Otkrist Gupta, and Ramesh Raskar. Nopeek: Information leakage reduction to share activations in distributed deep learning. In *2020 International Conference on Data Mining Workshops (ICDMW)*, pages 933–942. IEEE, 2020.

[26] Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10:3152676, 2017.

[27] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. Privacy preserving vertical federated learning for tree-based models. *Proc. VLDB Endow.*, 13(12):2090–2103, July 2020. ISSN 2150-8097. doi: 10.14778/3407790.3407811.

[28] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.