Learning to Attack Distributionally Robust Federated Learning

Wen Shen   Henger Li   Zizhan Zheng
Dept. of Computer Science, Tulane University
{wshen9, hli30, zzheng3}@tulane.edu

Abstract

We propose a two-stage attack framework that leverages the power of distribution matching and deep reinforcement learning to learn attack policies against federated learning. Our two-stage attack effectively learns an attack policy that minimizes the robustness levels of distributionally robust federated models, and substantially jeopardizes the performance of the federated learning systems even when the server imposes defense mechanisms. Our work brings new insights into how to attack federated learning systems with model-based reinforcement learning.

1 Introduction

Federated learning is a powerful paradigm that has the potential to train machine learning models among different devices in a distributed fashion without diminishing user experiences or jeopardizing users' privacy. However, federated learning models have also been shown to be vulnerable to threats [11] such as model poisoning attacks [5][2][3], data poisoning attacks [3][6][7], and inference attacks [12][8][18]. Even though various defense mechanisms (e.g., coordinate-wise median [17], trimmed mean [17], Krum [4], and Bulyan[13]) have been proposed to defend federated learning systems, delicate attack still can break the defending systems. For instance, the local model poisoning attacks [5] and the inner product manipulation attack [16] can compromise federated learning systems even when the server is equipped with robust aggregation rules such as coordinate-wise median or Krum.

The local model poisoning attack (LMPA) method crafts a myopic attack by poisoning local models on compromised workers such that the aggregated global model deviates the most towards the inverse of the global model when there is no attacks [5]. When the server imposes the Krum defense mechanism and the aggregation rule is known to the attacker, the LMPA method performs reasonably well for both cases when the attacker has full information about other normal workers' local models, and when it only has partial knowledge. However, when the server uses the coordinate-wise median defense, the LMPA method with partial knowledge performs substantially worse than the full knowledge case. Further, the LMPA method performs significantly worse when the aggregation rule is unknown to the attacker. The inner product manipulation (IPM) method implements an attack by manipulating the distance between the robust estimator and the correct mean to be negative [16] . In order to compute the attack policy, the IPM method requires that the attackers know the average of the normal workers' gradients. Instead of crafting a myopic attack policy as in the previous studies, we propose a two-stage attack framework that first learns the distribution of aggregated data using distribution matching and then learns a non-myopic attack policy using the data samples generated from the distribution. The proposed attack framework does not assume that the attacker knows benign workers' local models. It performs consistently well even if the aggregation rule is unknown to the attacker.

Previous attacks on federated learning models typically aim at compromising the performance of the trained models under normal testing data that follow the same distribution as the training data, but not its robustness against data shift. In particular, it is unclear if and to what extent the

NeurIPS-20 Workshop on Scalability, Privacy, and Security in Federated Learning

training-stage attacks can diminish the performance of federated learning models that are designed for providing *certified* robustness in the inference stage. To this end, we extend the distributionally robust optimization (DRO) with adversarial training framework of [15] to federated learning setting. The DRO method provides certified loss guarantees for data perturbations that are within a distance-bounded Wasserstein ball. To attack the federated training of a DRO model, we apply our two-stage attack framework that initially learns the distribution of the aggregated data using the attacker's local data and the parameters it receives from the server. Using the data generated from the learned distribution, the attacker performs model-based reinforcement learning [1] to train an attack policy that minimizes the robustness levels.

Our work advances the state-of-the-art in adversarial attacks against federated learning in the following aspects. First, we propose a two-stage attack framework that first learns the distribution of the aggregated data using local data and the parameters received from the server and then uses the distribution to learn a non-myopic attack policy with deep reinforcement learning. Second, we show how the proposed framework can attack distributionally robust federated learning models effectively in both a white-box (when the aggregation rule is known to the attacker) and a black-box (when the aggregation rule is unknown to the attacker) manner. Our experiments on both synthetic and real-world datasets demonstrate the advantages of the two-stage attack method compared with baseline attack methods such as random attacks and inner product manipulation attacks.

2 Distributionally Robust Federated Learning

We consider a distributed computing model with one server and m worker machines. Without loss of generality, we assume that each worker holds n data samples, and all the mn data samples are $i.i.d.$ drawn from a distribution P_0 supported on a space $\mathcal{Z} \subseteq \mathbb{R}^d$. Let $z_i(j)$ denote the i-th training sample on worker j, and $\widehat{P} := \frac{1}{mn}\sum_{j=1}^{m}\sum_{i=1}^{n}\delta_{z_i(j)}$ the empirical distribution generated by the data samples, where $\delta_{z_i(j)}$ denotes the Dirac point mass at $z_i(j)$.

The distributionally robust optimization (DRO) with adversarial training framework in [15] provides a training procedure that aims to choose the model parameter $\theta \in \Theta$ to minimize the worst-case expected loss defined as $\sup_{P\in\mathcal{P}} \mathbb{E}_{Z\sim P}[l(\theta; Z)]$ where the loss function $l(\theta; z)$ is generally non-convex in θ, and $\mathcal{P}$ is a class of distributions. We will consider $\mathcal{P} := \{P : W_c(P, P_0) \leq \rho\}$, where $W_c(\cdot,\cdot)$ is the Wasserstein metric. The worst-case formulation yields models with guaranteed performance under adversary Wasserstein perturbations up to level ρ. Following [15], we consider minimizing the relaxed dual objective $F(\theta) := \sup_P\{\mathbb{E}_{Z\sim P}[l(\theta; Z)] - \gamma W_c(P, \widehat{P})\} = \mathbb{E}_{Z_0\sim\widehat{P}} \sup_{z\in\mathcal{Z}}[l(\theta; z) - \gamma c(z, Z_0)]$, where γ is a fixed dual variable and $c(\cdot,\cdot)$ is the transportation cost that defines the Wasserstein metric. When the loss function $l(\theta; z)$ is smooth in z and γ is relatively large, the minimax problem becomes a nonconvex-strongly-concave optimization problem, and can be solved by stochastic gradient descent [15]. We observe that by setting $\gamma = \infty$, the DRO objective reduces to the standard objective of empirical risk minimization.

We extend the DRO framework to a federated learning setting similar to [14], where the server and workers work together to solve the minimax problem (see Algorithm 1). We call it *Distributionally Robust Federated Learning* (DRFL). In each time step t, each normal worker samples a minibatch from its local data and solves an optimization problem separately using the current model θ^{t-1}. This involves finding a (nearly) optimal $\hat{z}(z_0)$ to the inner problem for each data sample z_0 in the minibatch, and then computing the gradient of loss with respect to θ. The workers then forward the gradients to the server. The server applies projected gradient descent to the aggregated gradients to update the global model. The server may apply a simple aggregation rule such as average or a more robust rule such as Krum or coordinate-wise median. Note that rather than sending correct gradient information as a normal worker does, an adversary can send arbitrary information to the server.

Given a model θ (such as the one computed by Algorithm 1 within T epochs), let $P^*(\theta)$ denote the distribution that maximizes the dual objective and $\widehat{\rho}(\theta)$ its Wasserstein distance from $\widehat{P}$: $P^*(\theta) := \arg\max_P\{\mathbb{E}_{Z\sim P}[l(\theta; Z)] - \gamma W_c(P, \widehat{P})\} = \frac{1}{mn}\sum_{j=1}^{m}\sum_{i=1}^{n}\delta_{T_\gamma(\theta; z_i(j))}$, and $\widehat{\rho}(\theta) := W_c(P^*(\theta), \widehat{P}) = \mathbb{E}_{Z\sim\widehat{P}}[c(T_\gamma(\theta; Z), Z)]$, where $T_\gamma(\theta; z_0) := \arg\max_{z\in\mathcal{Z}}\{l(\theta; z) - \gamma c(z, z_0)\}$ is the transportation map or Monge map, and δ_z is the point mass at data z. It is observed in [15] that $(\mathbb{E}_{Z\sim P^*}[l(\theta; Z)], \widehat{\rho}(\theta))$ provides a data-driven robust certificate in the sense that the worst-case loss is upper bounded by $\mathbb{E}_{Z\sim P^*}[l(\theta; Z)]$ for Wasserstein perturbations up to level $\widehat{\rho}(\theta)$.

Algorithm 1: Distributionally Robust Federated Learning (DRFL)

```
Initialization: $\theta^0$, $m$ workers each with $n$ data samples , step size $\eta$;
Output: $\theta^T$
  for $t = 1$ to $T$ do
    Each Worker $j$:
    Sample a minibatch $B^t(j)$
    for $z_0 \in B^t(j)$ do
      $\hat{z}(z_0) = \arg\max_{z \in Z} l(\theta^{t-1}; z) - \gamma c(z, z_0)$
    end for
    $g_j^t = \frac{1}{|B^t(j)|} \sum_{z_0 \in B^t(j)} \nabla_\theta l(\theta^{t-1}; \hat{z}(z_0))$
    Send $g_j^t$ to the server
    Server:
    $g^t = Aggr(g_1^t, g_2^t \ldots, g_m^t)$
    $\theta^t = proj_\Theta(\theta^{t-1} - \eta g^t)$
    Broadcast $\theta^t$ to the workers
  end for
```

3 Two-Stage Attack Framework for Federated Learning

Threat Model We consider a single active attacker (as a worker) in the DRFL system. We make the weakest information assumption about the attacker (same as a normal worker): it only knows the global model parameters received from the server, in particular, $\{\theta^t\}$, γ, η, loss function $l(\cdot, \cdot)$, the local training algorithm and its local data. In addition, we assume that the attacker obtains information about the number of workers in the system and the size of data samples for each worker. We observe from our experiment that the attack method is still effective even if the attacker only knows a lower bound of the number of workers. We consider both the cases when the attacker knows the aggregation rule (white-box attack), and when this information is unavailable (black-box attack).

We consider a realistic attacker that sends crafted gradients to the server with the objective of maximizing the worst-case surrogate loss, i.e., $\sup_P\{\mathbb{E}_{Z\sim P}[l(\theta; Z)] - \gamma W_c(P, \widehat{P})\} = \mathbb{E}_{Z\sim P^*}[l(\theta; Z)] - \gamma\widehat{\rho}(\theta)$. It is convenient to maximize the expected loss or minimize the robust level. In our work, we consider minimizing the robust level $\widehat{\rho}$ as the attacker's objective.

The Reinforcement Learning Problem We formulate the attacker's optimization problem as a reinforcement learning problem. We represent it as a tuple (S, A, q, r, β), where

- $S = \{s_t\}$ is a continuous set of states. Here, $s_t := \theta^t \in \Theta$.
- $A = \{a_t\}$ denotes the action space of the attacker where $a_t := \tilde{g}_i^t \in \mathbb{R}^d$ is the gradient that attacker i sends to the server at time step t in Algorithm 1. Note that the action space is continuous. Our framework naturally supports multiple attackers. However, our experiments show that a single attacker is sufficient.
- $q(s, a, s')$ is the transition function that represents the probability of reaching a state $s' \in S$ from the state $s \in S$ when the attacker chooses action $a \in A$. The probability is jointly determined by the distribution $\widehat{P}$ of the aggregated data, the number of workers m, the number of data samples n used by each worker, the algorithm used by each worker, the aggregation rule used by the server, and the attack action a used by the attacker. The original distribution $\widehat{P}$ is fixed but unknown to the attacker. We assume that m and n are known to the attacker. Since the attacker is a worker in the federated learning system, it has knowledge of the algorithm used by each worker. We consider both the scenarios when the aggregation rule is known and when it is unknown to the attacker.
- $r : S \times A \times S \to \mathcal{R}$ is the reward function, where $\mathcal{R} \subseteq \mathbb{R}_{\geq 0}$ is a continuous set of rewards. We define the reward at time step t as $r_t := \widehat{\rho}(\theta^{t-1}) - \widehat{\rho}(\theta^t)$. To compute r_t, the attacker again needs the information on $\widehat{P}$, m, n, the algorithm used by each worker, and the aggregation rule used by the server.
- β is the discount factor for future rewards.

The attacker's objective is to find a stationary policy μ that maximizes the expected discounted total rewards over T time steps, where $\mu : S \to A$ denotes a stationary policy that maps the current model θ^{t-1} to the next attack action a_t. By setting the discount factor $\beta = 1$ and using the definition of r_t, this objective is equivalent to finding a policy μ that minimizes $\mathbb{E}[\widehat{\rho}(\theta^T)]$. A key step to compute the probability function $q(s, a, s')$ and the reward function is to compute $\widehat{P}$. To this end, we propose to first learn the distribution $\widehat{P}$. After that, the attacker is ready to compute the transition probability for triple (s, a, s') and the associated rewards. It then applies dynamic programming to find the optimal policy. However, this dynamic programming approach is inefficient in practice due to the large state and action spaces. We propose to use deep reinforcement learning to learn an attack policy instead.

The proposed attack method (See Fig. 1) consists of two stages: distribution learning and policy learning, both of which happen during the training phase of the DRFL process. In stage one, the attacker learns $\tilde{P}$ as an approximation of the true aggregated data distribution $\widehat{P}$ from epoch 1 to epoch k (determined by a threshold value that will be described later) using the attacker's local data distribution $\widehat{P}_a$ and the model parameters $\{\theta^0, \theta^1, ..., \theta^k\}$ it receives from the server. In stage two, the attacker utilizes the learned distribution $\tilde{P}$ to learn an attack policy μ. Starting from epoch $k + 1$ until epoch T, the attacker uses the same learned policy μ to generate actions (i.e., gradients) given a state as the input. It then sends the generated gradients to the server.

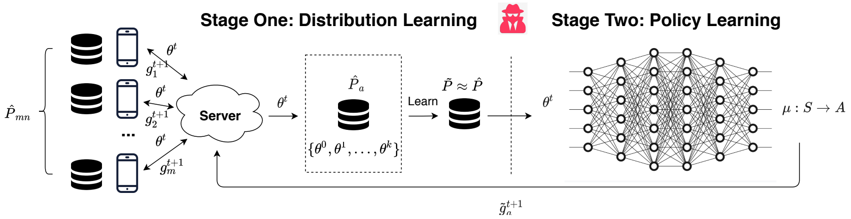


Figure 1: An overview of the two-stage attack framework for federated learning.

Stage One: Distribution Learning Initially, the attacker does not perform attacks. Instead, it learns the distribution of aggregated data using the *deep leakage from gradients* (DLG) method [18]. The DLG method utilizes only model updates (gradients) and learns the joint data-label pairs from dummy data inputs. It is also computationally efficient. The DLG method takes the neural network model, the gradients, and the dummy data as the input. Unlike the original DLG method, we use the attacker's local data instead of randomly generated data samples as the dummy data to improve the quality of the approximated distribution. In each time step t, we use DLG to compute a distribution $\tilde{P}^t$ with the pooled gradients up to time step t, which involves multiple iterations of gradient matching. We measure the Wasserstein distance of the approximated distributions between any two consecutive time steps, $\tilde{P}^t$ and $\tilde{P}^{t+1}$. The algorithm terminates once the Wasserstein distance is below a threshold.

In our experiments, we set the threshold as 0.005 and the number of iterations for gradient matching as 300 (same as [18]), and learn the data distribution from scratch. If the attacker has partial information about $\widehat{P}$ (e.g., its neighbors' data distributions), it can learn the distribution starting from that partial information instead of from scratch. In this way, our algorithm allows the attacker to learn and implement attacks at any time during the training of the federated learning systems.

Stage Two: Policy Learning In this stage, the attacker learns an attack policy that minimizes the robustness level $\widehat{\rho}$. The attacker's original optimization problem can be converted to a reinforcement learning problem that aims to maximize the total discounted rewards over T time steps. We use the *deep deterministic policy gradient* (DDPG) method [10] to learn the attack policy μ. The policy network takes a state s as the input and outputs the action directly. We adopt OpenAI Stable Baselines3 for DDPG implementation and use the default setting for the hyper-parameters in our experiments.

When we train a small neural network with the DRFL system, it is natural to use θ^t as the state, and the gradient $\tilde{g}_i^t$ as the action. When we use the DRFL framework to train a large neural network, however, this approach does not scale as it results in extremely high search space that requires both large runtime memory and long training time. To solve this problem, we propose to approximate the state θ^t and the action g_i^t for high-dimensional data. To approximate the state θ^t, we use parameters of the last hidden layer of the current neural network model as the state. We further define the action a_t as a one-dimension scaling factor $a_t \in [-1, 1]$, and set $\tilde{g}_i^t = a_t \times g_i^t$, which is used to compute the next state and rewards.

We rescale the gradients to reduce the chance that the gradients are filtered out by the defense mechanisms. In particular, when computing the gradients to be sent to the server, the attack method first calculates the maximum value $\zeta_{\max}$ and the minimum value $\zeta_{\min}$ that occur in any dimension of $\nabla_\theta l(\theta^{t-1}; z)$ for any z in the data samples generated in the first stage. The original scaling factor a_t is in $[-1, 1]$. The rescaled scaling factor $a_t' = a_t \times \frac{\zeta_{\max} - \zeta_{\min}}{|\zeta_{\max}|} \times 0.5 + \frac{\zeta_{\max} + \zeta_{\min}}{\zeta_{\max}} \times 0.5$.

The computational complexity of the two-stage attack framework mainly comes from learning an accurate estimation of the aggregated data distribution $\widehat{P}$, and learning an attack policy μ using the learned distribution. The training time for policy learning in stage two largely depends on the number of training epochs used to simulate the server's behavior, and the training episodes used by the attacker. The larger of the two factors, the more training time is required. In practice, the attacker can run training episodes in parallel with GPUs or other parallel computing devices. The attacker needs to make tradeoff among the time spent on distribution learning, simulating server's behavior and attack execution. Before performing the attacks, it takes time (usually several epochs) for the attacker to learn the aggregated distribution and train an attack policy. Once a policy has been trained, the attacker can utilize the same policy to implement the attacks without extra computation required.

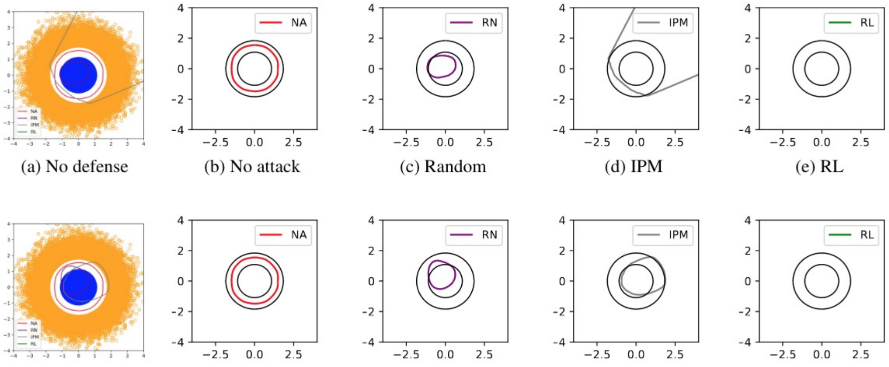
4 Experiments

We conduct two groups of experiments. In the first group, we visualize the benefits and advantages of learning attack policies with deep deterministic policy gradient on a synthetic dataset. In the second group of experiments, we test the performance the proposed method on testing datasets with different level of perturbations on the MNIST dataset [9]. For both groups, there are 10 workers in the federated learning system with one attacker using the two-stage attacks.

Synthetic Dataset We generate synthetic data according to the method described in [15]. In particular, we generate 50,000 data instances $Z = (X, Y) \sim P_0$ with $X_i \in \mathbb{R}^2 \sim N(0, I)$ and labels $Y_i = \text{sign}(\|X_i\|_2 - 2)$. Each worker has 5,000 data instances. We train a small neural network with two hidden layers of size four and two, respectively. We use ELU activations for all layers. In our experiments, we let $\gamma = 2$. To train the reinforcement learning model, we use θ^t as the state, and the gradient as the action. We train the DRFL model for 30 epochs. It takes 3 epochs for the attacker to learn the distribution $\widehat{P}$. Training the attack model takes 2 minutes, which is equivalent to 1 epoch of training in DRFL.

Fig. 2 illustrates the class boundaries for four DRFL training methods (no attack, random attack, inner product manipulation attack and two-stage reinforcement learning attack) over the ELU-activated models. Without attacks, similar to the original distributationaly robust learning described in [15], the DRFL provides a certified level of robustness as evidenced by an axisymmetric classification boundary that is resilient to adversarial perturbations in all directions (see Fig. 2b). When there are attacks in the federated training, the trained model no longer maintains the certified robustness (see Figs. 2c, 2d, and 2e). Of all the three attack models, our proposed two-stage attack using reinforcement learning performs the best (as evidenced by the missing classification boundary in 2e) by learning a non-myopic policy that minimizes the level of robustness $\widehat{\rho}$. The inner product manipulation attack outperforms the random attack due to the fact that it uses the information of the normal agents' average gradients. When the server employs the coordinate-wise median defense, the random attack and the inner product manipulation attack are still effective. However, their ability to deviate the robustness of the DRFL decreases (see Figs. 2h and 2i). This is also true when the server uses the Krum defense mechanism (not shown in the figure). Even when the server employs a defense mechanism, the two-stage reinforcement learning attack can compromise federated learning to the largest extent among all the attack methods (see Fig. 2j).

MNIST Dataset We test the performance of our two-stage reinforcement learning attack on a real-world supervised-learning benchmark - the MNIST dataset with 70,000 data instances. Each worker has 7,000 data instances. Our experiment procedure is similar to that in [15]. We train a neural network classifier that consists of 8×8, 6×6, and 5×5 convolutional filter layers with ELU activations and a fully connected layer and softmax output. In our experiment, we set $\gamma = 0.04\mathbb{E}_{\widehat{P}}[\|X\|_2]$, where X is the transformed image vector. To train the reinforcement learning model, we use parameters of the last hidden layer as the state. For each dimension of the state, the space is $[-inf, inf]$. We define the action as a scaling factor of the original gradients. The action space is $[-1, 1]$. We train the DRFL



(a) No defense (b) No attack (c) Random (d) IPM (e) RL

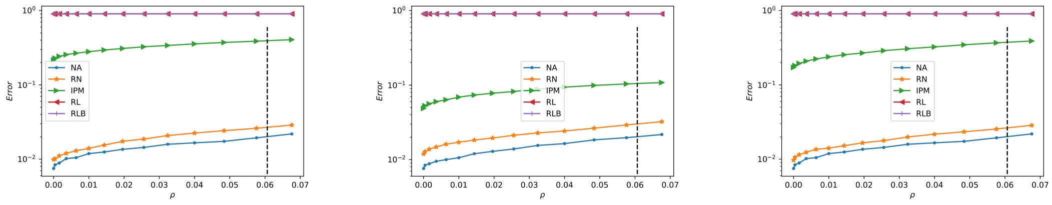
(f) Median (g) No attack (h) Random (i) IPM (j) RL

Figure 2: Results on synthetic data with no defense and coordinate-wise median based defense. Training data are shown in blue and orange. Classification boundaries are shown in red, purple, gray, and green for no attack (NA), random attack (RN), inner product manipulation (IPM), and reinforcement learning (RL) attack.

model for 30 epochs. It takes 5 epochs for the attacker to learn the distribution $\widehat{P}$. Training the attack model takes 1 minutes, which is equivalent to 1.4 epochs in DRFL training.

We test the performance of five federated models with testing datasets that have different levels of data shifts (in terms of Wasserstein perturbations) from the original data distribution. We measure the testing errors for different perturbation level ρ under three conditions for the server: no defense, coordinate-wise median, and Krum. In this experiment, we introduce a variant of the two-stage attack - blackbox reinforcement learning attack, where the attacker does not know the aggregation rule used by the server and simply uses the average rule to predict the next state in stage two.

Fig. 3a shows that the two-stage reinforcement learning attack outperforms both the random attack and the inner product manipulation attack substantially for all the settings in terms of the testing errors. The performance of the proposed two-stage attack remains stable even if the perturbation level ρ changes. This observation demonstrates the advantages of model-based reinforcement learning attacks in federated learning. When the server employs defense mechanisms such as coordinate-wise median and Krum, both the white-box and black-box reinforcement learning attacks consistently outperform the other two attack methods (see Figs. 3b and 3c). The experimental results indicate that even if defense mechanisms are employed by the server, the two-stage reinforcement learning attacks still can effectively compromise the federated learning models. This result raises security concerns for training distributionally robust models in federated learning.



(a) No Defense (b) Median (c) Krum

Figure 3: A comparison of misclassification errors (including both false positive and false negative) for different perturbation levels ρ with five DRFL models: NA, RN, IPM, RL and reinforcement learning-blackbox (RLB) when the server uses no defense, coordinate-wise median, and Krum, respectively. The vertical line indicates the robustness level $\widehat{\rho}$ of the trained model without attack.

5 Conclusion and Future Work

We propose a two-stage attack framework that combines distribution matching and deep reinforcement learning to learn a non-myopic attack policy that can effectively compromise the robustness of federated learning systems. Our experiments show that the proposed two-stage attack can effectively degrade the performance of distributionally robust federated learning models. Our work opens up new exciting revenues for further study. A rewarding direction would be to extend the current framework to federated learning systems with non-i.i.d. data. A key step towards the solution is to develop novel methods to effectively learn the distribution of the aggregated data in stage one. In our work, we assume that the attacker is always selected by the server. However, if the server adopts subsampling, the attacker might not be selected for some rounds. As a result, the attacker might not be able to learn an accurate distribution $\widehat{P}$. In addition, the attacker might not be able to always implement the attack. A systematic study to investigate the effectiveness of the proposed two-stage attack method under such condition is needed. The proposed method requires the attacker to first learn an accurate model $\widehat{P}$ and then learn an attack policy using deep reinforcement learning before it performs attacks. It would be interesting to study novel online methods to allow the attacker to learn the attack policy immediately after learning an approximate (not necessarily accurate) distribution.

Acknowledgements

This work was supported in part by NSF grant CNS-1816495. We would like to thank three anonymous reviewers for their constructive feedback.

References

[1] Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine 34*, 6 (2017), 26–38.

[2] Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., and Shmatikov, V. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics* (2020), PMLR, pp. 2938–2948.

[3] Bhagoji, A. N., Chakraborty, S., Mittal, P., and Calo, S. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning* (2019), PMLR, pp. 634–643.

[4] Blanchard, P., Guerraoui, R., Stainer, J., et al. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems* (2017), pp. 119–129.

[5] Fang, M., Cao, X., Jia, J., and Gong, N. Local model poisoning attacks to byzantine-robust federated learning. In *29th USENIX Security Symposium* (2020), pp. 1605–1622.

[6] Fung, C., Yoon, C. J., and Beschastnikh, I. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866* (2018).

[7] Gu, T., Dolan-Gavitt, B., and Garg, S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733* (2017).

[8] Hitaj, B., Ateniese, G., and Perez-Cruz, F. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), pp. 603–618.

[9] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[10] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[11] Lyu, L., Yu, H., and Yang, Q. Threats to federated learning: A survey. *arXiv preprint arXiv:2003.02133* (2020).

[12] Melis, L., Song, C., De Cristofaro, E., and Shmatikov, V. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy* (2019), pp. 691–706.

[13] Mhamdi, E. M. E., Guerraoui, R., and Rouault, S. The hidden vulnerability of distributed learning in byzantium. *arXiv preprint arXiv:1802.07927* (2018).

[14] Sadeghi, A., Wang, G., Ma, M., and Giannakis, G. B. Learning while respecting privacy and robustness to distributional uncertainties and adversarial data. *arXiv preprint arXiv:2007.03724* (2020).

[15] Sinha, A., Namkoong, H., and Duchi, J. Certifying some distributional robustness with principled adversarial training. In *International Conference on Learning Representations* (2018).

[16] Xie, C., Koyejo, O., and Gupta, I. Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation. In *Uncertainty in Artificial Intelligence* (2020), PMLR, pp. 261–270.

[17] Yin, D., Chen, Y., Ramchandran, K., and Bartlett, P. Byzantine-robust distributed learning: Towards optimal statistical rates. *arXiv preprint arXiv:1803.01498* (2018).

[18] Zhu, L., Liu, Z., and Han, S. Deep leakage from gradients. In *Advances in Neural Information Processing Systems* (2019), pp. 14774–14784.