Automatic Software Testing to detect model poisoning attacks in Federated Learning

Arthur Nijdam

28 August 2023

Most conventional Machine Learning (ML) frameworks require clients to share their data to a central server, such that a single global model can be trained on the aggregated dataset of all clients. This model can then be deployed on all client's devices. This pipeline is not suitable for privacy-sensitive data, since it requires user data to be broadcasted to and stored on a server, making (the traffic to and from) the server vulnerable to adversarial attacks. Federated Learning (FL) [1] is often proposed as a solution to combat privacy attacks by "bringing the code to the data, instead of bringing the data to the code". Here, each client i receives an initial model w from the server, fine-tunes it based on their local data X_i, y_i and sends the resulting weight update Δw_i to the server. The server then computes the average of all weight updates and adjusts its global model accordingly: $w \leftarrow w + \frac{1}{N} \sum_{i=0}^{N-1} \Delta w_i$. Although FL greatly improves the privacy of user's data as compared to centralized learning, it is not an end-all be-all solution against adversaries. In fact, it is even possible to recover the user's original input data based off just the weight update w_i [2]. Moreover, one can think of various other attack models, where honest-but-curious or even actively adversarial clients or servers corrupt the training process and gain access to user data.

Additionally, one could argue that FL complicates the normal Machine Learning pipeline, as it distributes the training process over a set of clients. This does not only influence neural network training, but also affects debugging, model testing and hyperparameter tuning [3].

My PhD project focuses on Privacy-Preserving Machine Learning, which tackles both:

1. How FL itself can be made more secure, through e.g. secure multiparty computation [4] and differential privacy [5].

2. How secondary issues that arise due to the distributed training process in FL can be addressed, such that FL does not degrade the performance or functionality as compared to centralized learning.

In this essay, I address how we can take inspiration from Software Engineering (SE) to improve the security and privacy of FL, specifically the resilience to model poisoning attacks.

Firstly, it is important to note the differences between **security and privacy**. Security is a broader concept that entails measures and practices to protect both data and software systems from unauthorized access. Privacy specifically relates to protecting personal information and ensuring that people have control over what happens to their data. In FL, privacy often refers to ensuring that the client data samples x are not exposed to any honest-but-curious observer that observes the data streams or the process within the server. Security then corresponds to protecting client's data and software against malicious adversaries. In this essay, we focus on Model poisoning attacks, that are a commonly researched attack for FL. In model poisoning, the attacker changes the parameters of the global model, causing errors in the global model or leaving a backdoor. For example, there can be malicious clients, who change their model parameters to degrade the global model's performance.

In SE, **automated software testing** protocols, where the code is tested based on generated test cases, form an important basis to verify that code performs the computation that it is supposed to compute. Since model poisoning relies on perturbing models that are being sent to the server, *SE testing techniques could help detect model poisoning.* Note that in FL, when the performance of the global model decreases, this can be due to model poisoning or other issues in any of the clients, such that finding the exact client where this problem occurred and the round at which the problem started arising, is difficult. Therefore, automated software testing for FL is non-trivial, as analyzed by e.g. the authors of FedDebug [6].

Since automated software testing relies on the quality of generated test cases, generating appropriate test cases is an active area of research. One way to automatically generate test cases is **boundary value testing**. Here, the intuition is that correct software has the correct boundaries, such that boundary cases should behave as expected. Exploration of boundary cases can thus reveal undesired behaviour. This idea is used to detect backdoor attacks in AEVA [7]. Here, we have access to a batch of legitimate samples X_i for each class i and create adversarial examples for testing by, for each class t, pushing the samples of other samples towards the decision boundary.

Machine Learning models often suffer from **the oracle problem**: It is not always clear what the desired output $f(x)$ of an input x to a machine learning model should be. This is especially true for unlabeled datasets, or for datasets with unreliable annotations. Instead, one can check whether the output of two inputs that are similar to each other, are also similar. The difference between outputs can then be quantified in terms of some sort of Euclidean distance metric. This technique is used to detect model poisoning attacks in [8]. The difference between a client's update at time $t-1$ and t is bounded by the difference between the global model at $t-1$ and t, such that an expectation can be made of the client's update at t. Zhang et al [8] then compare the predicted model update of each client to the actual model update of said client using the Euclidean distance to detect malicious clients. The computation of the predicted model update requires estimation of a Hessian, which can be subject to inaccuracies. Further reseach could focus on improving the prediction of the client's model update.

Instead of a malicious client poisoning the global model, we can also think of an attack model where there is another actor that had access to the server and poisoned the model. In this case, it is important to detect *where* the model was poisoned. Most techniques discussed thusfar are black-box testing techniques, which means that we can only detect that a model has been perturbed, not where it was altered. In contrast, in a **white-box testing** context, we have access to all lines of code corresponding to the model during test time and focus on analyzing on which line of code undesired behaviour occurs. AUDEE [9] can detect inconsistencies, crashes and NaN errors in machine learning models and report the neural network layer where the bug occurs. Test scenarios were created using a genetic algorithm, so that the inputs and weights used would trigger bugs more easily. The buggy layer was localized using the intuition that buggy layers will change the output of the layer more than other layers, so the layer with the highest change rate is outputted as the buggy layer. Although I haven't been able to find concrete examples of such white-box testing being used to detect adversarial server operators, I think it could be used for that purpose. Furthermore, one could also think of using such a process in addition to the other methods to detect perturbations to the global model that have gone unnoticed.

Lastly, I want to highlight the importance of **data validation** when applying measures to improve the robustness of FL systems against model poisoning attacks. As mentioned in Ditto [10], improving the robustness against poisoning attacks often goes hand in hand with compromising on the fairness of neural networks: Robust methods might filter out updates from

rare clients and incorrectly label these clients as adversaries, resulting in unfairness. Ditto is a personalized FL method that identifies heterogeneity in the dataset, which is one of the main causes of the apparent trade-off between robustness and fairness. I think it could be interesting to analyze to which extent the other methods mentioned in this paper introduce degradation when it comes to fairness of the model: How often do they mark statistically heterogeneous clients as malicious adversaries?

In general, I believe that lessons learned from Software Engineering can help build FL systems that are more resilient against model poisoning attacks. In addition, I think there are quite a lot of open questions within SE for FL that have been undervalued, such as how to properly debug FL systems, evaluate them, and how to detect errors during deployment. This is not only important to create better maintainable FL systems, but also to detect adversarial attacks at any stage of the development. In my opinion, such matters have to be fixed properly before FL can be used more widely, and some of the solutions could extend beyond FL and teach ML engineers in general valuable lessons on ML code maintenance.

REFERENCES

References

[1] 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*, pp. 1273–1282, PMLR, 2017.

[2] W. Wei, L. Liu, M. Loper, K.-H. Chow, M. E. Gursoy, S. Truex, and Y. Wu, "A framework for evaluating gradient leakage attacks in federated learning," *arXiv preprint arXiv:2004.10397*, 2020.

[3] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[4] 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*, pp. 1175–1191, 2017.

[5] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020.

[6] W. Gill, A. Anwar, and M. A. Gulzar, "Feddebug: Systematic debugging for federated learning applications," *arXiv preprint arXiv:2301.03553*, 2023.

[7] J. Guo, A. Li, and C. Liu, "Aeva: Black-box backdoor detection using adversarial extreme value analysis," *arXiv preprint arXiv:2110.14880*, 2021.

[8] Z. Zhang, X. Cao, J. Jia, and N. Z. Gong, "Fldetector: Defending federated learning against model poisoning attacks via detecting malicious clients," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2545–2555, 2022.

[9] Q. Guo, X. Xie, Y. Li, X. Zhang, Y. Liu, X. Li, and C. Shen, "Audee: Automated testing for deep learning frameworks," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pp. 486–498, 2020.

[10] T. Li, S. Hu, A. Beirami, and V. Smith, "Ditto: Fair and robust federated learning through personalization," in *International Conference on Machine Learning*, pp. 6357–6368, PMLR, 2021.