Model poisoning attacks against distributed machine learning systems

Richard Tomsett[a], Kevin Chan[b], and Supriyo Chakraborty[c]

[a]IBM Research, Hursley, UK
[b]Army Research Laboratory, Adelphi, USA
[c]IBM Research, Yorktown, USA

ABSTRACT

Future military coalition operations will increasingly rely on machine learning (ML) methods to improve situational awareness. The coalition context presents unique challenges for ML: the tactical environment creates significant computing and communications limitations while also having to deal with an adversarial presence. Further, coalition operations must operate in a distributed manner, while coping with the constraints posed by the operational environment. Envisioned ML deployments in military assets must be resilient to these challenges. Here, we focus on the susceptibility of ML models to be poisoned (during training) or fooled (after training) by adversarial inputs. We review recent work on distributed adversarial ML, and present new results from our own investigations into model poisoning attacks on distributed learning systems without a central parameter aggregation node.

Keywords: distributed learning, deep learning, adversarial machine learning, model poisoning, peer-to-peer

1. INTRODUCTION

We are interested in attacks against distributed machine learning systems. Our motivation for this comes from considering the needs of future military coalition operations. Situational awareness (SA) and understanding (SU) are crucial for military leaders to make good decisions.[1] Gaining SU is hampered by the sparseness and uncertainty of available information. Militaries are increasingly turning to technology to improve communications and information collection and fusion for SU, and recent advances in analytics and edge computing have given rise to the concept of the Internet of Battlefield Things (IoBT).[2,3] In this vision, future operational theatres will be populated by many small, low-power, communicating devices with varying degrees of intelligence and autonomy. Machine learning (ML), including deep learning (DL), will become increasingly important for extracting insights from data collected by these devices.[4] However, data transfer speeds will still be quite limited compared with computational capability, and the network may be unreliable, meaning that keeping data and learning at the network edge will be much more efficient than transferring data back to centralized servers or the cloud. Additionally, data privacy policies between coalition partners may prevent direct sharing of certain kinds of data. Finally, the IoBT is an adversarial environment, with malicious actors attempting to disrupt coalition communications and analytics.

In this paper we focus specifically on this adversarial aspect in relation to distributed deep learning. First we briefly survey related literature on distributed DL, various attacks against ML systems, and particularly attacks on distributed DL. We then describe some experiments implementing targeted model poisoning attacks[5] in a peer-to-peer DL setting. We end with a discussion of these results, some suggestions for improving the attack by exploiting aspects of the peer-to-peer learning setting, and proposals for future work in this area.

Further author information:
R.T.: rjtomsett@uk.ibm.com

2. PRIOR WORK

2.1 Distributed deep learning

Deep learning is computationally intensive and data hungry, and most researchers are not constrained to the IoBT environment described above. Therefore, the majority of research on distributing DL has aimed to improve learning speed and efficiency by exploiting parallelism in modern high-performance computing systems. Earlier efforts in this vein focused on *model parallelism*, in which a single model is split between many nodes that communicate during training.[6] Model parallelism allows extremely large models to be trained by increasing the amount of memory available for storing parameters and gradients. *Data parallelism* instead speeds up training by splitting different batches of data between many workers and training on these batches simultaneously.[7] This has increased in popularity as more memory has become available at each compute node, while methods incorporating both data and model parallelism allow for the rapid training of extremely large models for complex tasks.[8]

However, these approaches are not suitable for our motivating scenario in which communication is slow and unreliable, and data sharing is highly constrained. Federated learning[9] was recently proposed as a method for training a model using large numbers of unreliable nodes that cannot share data, for logistical and/or data privacy reasons. In the federated setup, each worker node trains a copy of the global model on its own data, and stochastically sends parameter updates to a central aggregation server. The aggregation server combines the updates from many nodes, and broadcasts the new global model parameters back to the workers. In this way, worker nodes can benefit from knowledge learned by other workers without ever communicating with them directly, or seeing any of their training data. Such a learning scheme can be tuned to optimize the usage of the available compute resources and communication bandwidth,[10] making it suitable for resource-constrained deployments as long as a central aggregation server is available.

A different approach to distributed training without data sharing is to use peer-to-peer communication to transmit parameter updates between workers. This approach still allows for infrequent and stochastic updates between nodes, but also obviates the need for a centralized parameter server. It is therefore a particularly attractive learning paradigm for the military coalition setting described above. "Gossip" averaging has been shown to be an effective method for performing peer-to-peer model learning, including recently for adapting stochastic gradient descent for training neural networks.[11–14]

2.2 Adversarial machine learning

Adversarial ML has been studied in various forms for well over a decade.[15] In an adversarial ML setting, an adversary attempts to deceive a model; for example, fooling it into making incorrect predictions, extracting private information from the model, or causing the model to learn incorrect mappings from inputs to outputs. Huang et al.[16] classify such attacks along three axes:

- *Influence*: attacks that manipulate training data (*causative* attacks), or probe previously-trained models (*exploratory* attacks)
- *Security violation*: attacks causing a system to incorrectly classify inputs (*integrity* violations), or making the system unavailable (*availability* violations), or extracting private data from the model (*privacy* violations)
- *Specificity*: attacks that are *targeted* towards particular inputs, or *indiscriminate* and created to reduce performance on a broad variety of inputs

Early work on adversarial ML arose in response to email spammers attempting to circumvent spam filters[17,18] using targeted, exploratory attacks to cause integrity violations. Interest in "adversarial examples" against deep neural networks was stoked by work showing that very accurate image classifiers could reliably be fooled into making incorrect classifications by modifying inputs with near-imperceptible amounts of adversarial noise.[19,20] Work on so-called poisoning attacks — causative attacks causing integrity violations — identified ways in which models could be manipulated by altering their training data.[21] Koh et al. were among the first to show that

neural network performance could be severely degraded by poisoning during training using a small number of minimally manipulated training data points,[22] while Chen et al. demonstrated how "backdoors" can be inserted into neural network-based biometric authentication systems via data poisoning attacks.[23]

2.3 Attacks against distributed machine learning systems

Several groups have recently begun research on adversarial ML in a distributed setting. Distributed adversarial ML is related to the concept of Byzantine fault tolerance in distributed systems. Byzantine-resilient systems can tolerate arbitrary failures from any of their constituent parts — in the distributed ML setting, this means learning cannot disrupted or poisoned by data or parameters shared by any of the nodes in the system, regardless of the data/parameter values. Several researchers have developed Byzantine-tolerant gradient descent algorithms for resilient distributed learning.[24–26]

Data poisoning attacks on distributed ML were studied by Zhang et al. in the context of an adversary with access to the distributed nodes' training data .[27,28] They formulate the scenario using game theory to calculate the optimal actions of the attacker manipulating training data at the nodes, and defender attempting to learn an accurate model. Model poisoning attacks have also been studied in the context of federated learning. Fung et al. developed a method to defend against two kinds of poisoning attack (label-flipping and backdoor engineering) in federated learning, where the poisoning is enacted by multiple colluding agents,[29] while Bagdasaryan et al. demonstrate a powerful poisoning attack in which one agent is able to make the global model 100% accurate on a backdoor task with only a single update to the central parameter server.[30] Bhagjoi et al. have also demonstrated the possibility of targeted model poisoning in federated learning, as well as introducing notions of attack stealth (high stealth means that malicious nodes are difficult to detect).[5]

3. MODEL POISONING IN PEER-TO-PEER MACHINE LEARNING

We extend prior work on poisoning distributed machine learning to the peer-to-peer learning arrangement. As described above, peer-to-peer learning seems particularly well suited to the military coalition setting, but as far as we are aware there has not yet been any work looking at attacks on peer-to-peer learning. The learning dynamics of peer-to-peer training methods are different from those that rely on parameter aggregation servers such as federated learning, and so we expect the effects of any particular model poisoning attack to be different as well. In this preliminary study, we chose to focus on targeted model poisoning when learning using gossiping stochastic gradient descent.[11,14]

3.1 Methods

In our learning setup, each peer has access to an IID subset of size S/K of the full data set, where S is the total number of training data points and K is the number of peers. Every peer starts with a copy of a neural network model with the same architecture and initial parameters, and attempts to minimize its loss on an unseen test data set by training using its local data and sharing parameters with other peers.

Peers are fully connected — each peer is able to send data to every other peer. We use random gossip to share parameters between peers, specifically the GoSGD algorithm.[11,14] This algorithm consists of two steps: the gradient update, where each peer k updates its model parameters (weights and biases) w_k using SGD on one randomly chosen mini-batch from its local data; and the mixing step, where each k transmits its parameters to at most one other peer, with probability p_k. If k transmits its parameters, it picks a random peer with uniform probability (i.e. $1/K$) to send them to. It also sends its mixing factor α_k, which is used to adjust the weight given to its parameters when combining them with other peers' parameters. Prior to transmitting, the mixing factor is updated by setting $\alpha_k \leftarrow \alpha_k/2$. Thus α_k shrinks every time k shares its parameters.

Prior to each gradient update, each peer k processes any messages from other peers — if it has received > 1 message, it processes them in the order they arrived. Processing a message from peer j means merging the received parameters w_j from j with k's own parameters w_k according to the mixing factors α_j, α_k using the rule $w_k \leftarrow \alpha_k w_k/(\alpha_j + \alpha_k) + \alpha_j w_j/(\alpha_j + \alpha_k)$. The mixing factor is also updated on processing a message, setting $\alpha_k \leftarrow \alpha_k + \alpha_j$. Thus α_k increases every time k receives parameters. All mixing factors are initialized as $\alpha_k \leftarrow 1/K$.

We adapt the targeted model poisoning attack described by Bhagoji et al.[5] to this peer-to-peer learning setup. In this attack, the single malicious peer m attempts to trick the benign peers to learn to classify a set of data points $\{\mathbf{x_i}\}_{i=1}^{r}$ with specific, incorrect labels $\{\tau_i\}_{i=1}^{r}$ instead of their true labels $\{y_i\}_{i=1}^{r}$. This set is available to the malicious peer, but not the benign peers, during training. As in Bhagoji et al.,[5] we choose to learn to mis-classify a single example, i.e. $r = 1$.

We use Bhagoji et al.'s alternating minimization attack strategy:[5] the malicious peer m updates its own model weights by first minimizing an adversarial objective (minimizing the loss on $\{\mathbf{x_i}\}_{i=1}^{r}$) and updating its parameters, then minimizing the global loss using a mini-batch from its allocated training data, and updating its parameters. In the original federated learning setting, this approach was designed to allow the malicious agent to achieve its malicious objective while maintaining stealth (i.e. the parameter updates it sends to the aggregation server should be distributed similarly to those sent by benign agents, and should not cause a drop in global model accuracy). We hypothesized that this training approach would achieve a similar goal in the peer-to-peer learning case.

All our experiments were run with 9 benign peers and 1 malicious peer (10 peers in total). As in Bhagoji et al.,[5] we used the Fashion-MNIST dataset,[31] a replacement for MNIST[32] made up of greyscale 28×28 pixel images, split into a training set of 60 000 and a test set of 10 000 images, and categorized into 10 classes. With our 10 peers, this means that each peer has access to a random subset of 6 000 images from the training set, and is tested at each step on the full 10 000–image test-set. For our adversarial objective, we want the peers to learn to misclassify an (unseen) example in class 5 ("sandle") as being in class 8 ("bag").

The system was implemented in Python, with each peer running two threads: one to handle the learning loop, and another to handle receiving parameters from other peers. This allows peers to run their learning loop continuously, without having to wait for other peers when sending/receiving parameters.[11] Peers communicate via sockets. The neural network at each peer was implemented using PyTorch,[33] and consisted of two convolutional layers followed by two fully connected layers, trained with dropout for 1600 training steps (1 mini-batch per step) using a mini-batch size of 300 (this corresponds to about 80 conventional training epochs). We trained using the standard stochastic gradient descent algorithm with momentum of 0.9 and a learning rate of 0.01. For ease of coordination, the 10 peers were run simultaneously on the same multicore machine using GNU Parallel.[34]

3.2 Results

Figure 1 (left column) shows the test accuracy and loss at each peer over training time with three different exchange probabilities: $p_k = 0.02$, $p_k = 0.05$, and $p_k = 1$. The choice of $p_k = 0.02$ was described by Blot et al. as being small, but still high enough to ensure adequate mixing during training.[11] $p_k = 0.05$ corresponds to, on average, about 1 exchange per 6 000 training samples, i.e. 1 conventional training epoch at each peer. $p_k = 1$ means that every peer transmits its parameters to 1 other peer every training step. In each case, the test accuracy and loss at each peer match each other closely during training, meaning that the malicious peer is hard to detect if only monitoring accuracy changes from its parameter updates.

Figure 1 also shows the confidence for the malicious test image to be wrongly classified as class 8, "bag" (the malicious objective), and the predicted label for this image, at each peer (middle and right columns, respectively). The malicious peer reliably classifies the image as class 8 ("bag") with very high confidence from early on during training. With $p_k = 0.02$, the malicious peer takes many iterations to reliably poison the benign peers: the majority of peers continue to classify the image as class 5 ("sandal") until iteration 1300. The benign peers' mean confidence that the malicious test image is class 8 also jumps to almost 0.8 at this iteration. With $p_k = 0.05$, the increase in mean confidence occurs more rapidly and steadily, and the majority of peers are poisoned reliably by iteration 1000. Finally when $p_k = 1$, all benign peers are reliably poisoned by iteration 160.

The malicious node does not need to abide by the same exchange rules as the other peers. For example, the malicious peer could set its exchange probability p_m higher than that of the benign peers in an effort to poison them more rapidly. We give an example of this in the top row of figure 2, which shows the learning when benign peers exchange with $p_k = 0.02$ while the malicious peer sets $p_m = 0.04$. Surprisingly for this training run, the reliability of poisoning *decreases* compared to the baseline case where $p_k = p_m = 0.02$ (top row in figure 1). This is because the malicious peer still performs the update $\alpha_m \leftarrow \alpha_m/2$ every time it transmits its parameters,

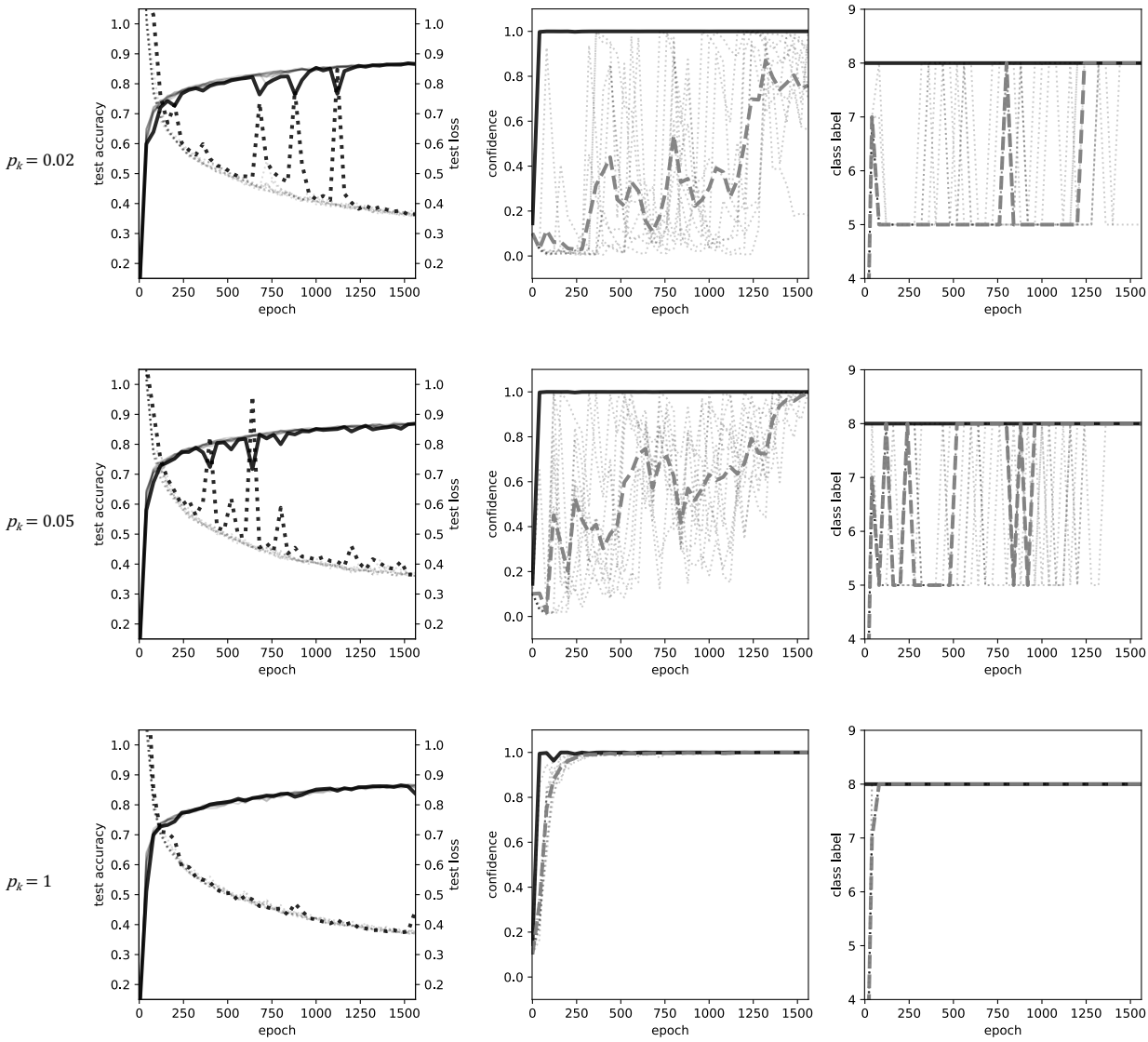

Figure 1. Targeted poisoning attacks in peer-to-peer learning with gossip (GoSGD[11]). Rows show results for different probabilities of exchanging parameters at each training iteration, p_k. *Left column:* Test accuracy (solid grey lines) and loss (dotted grey lines) for the benign peers, with the accuracy and loss for the malicious peer overlaid in black. *Middle column:* confidence for the benign peers that the malicious test image is in class 8, "bag" (dotted grey lines), with the confidence for the malicious peer overlaid in solid black. The dashed grey line shows the mean confidence for the benign peers. *Right column:* Class label for the malicious test image output by the benign peers (dotted grey lines), with the label output by the malicious peer overlaid in solid black. The dashed grey line shows the mode output label of the benign peers.

causing its mixing parameter α_m to shrink at twice the rate of the benign peers' mixing parameters and reducing the influence it has each time it transmits. To mitigate this effect, we re-ran this training experiment with a malicious transmission-mixing rule, setting $\alpha_m \leftarrow \alpha_m/\sqrt{2}$ prior to each transmission (keeping the original halving rule at the benign peers). The results using this rule are shown in the bottom row of figure 2. It has the effect of increasing the variability of the malicious peer's test accuracy and loss over the course of training, which could it make it more easily detectable by monitoring accuracy changes. However, it also strengthens the attack,

causing the benign nodes to increase their confidence in the malicious objective more rapidly. Interestingly, it also causes the variance of the benign peers' confidences in the malicious objective to decrease, even compared to the case where $p_k = p_m = 0.02$ (top row figure 1). We are currently conducting further investigations into modifying the rules for how to use and modify the mixing parameters on the malicious node to see if we can improve the attack strength for low values of p_k and p_m.

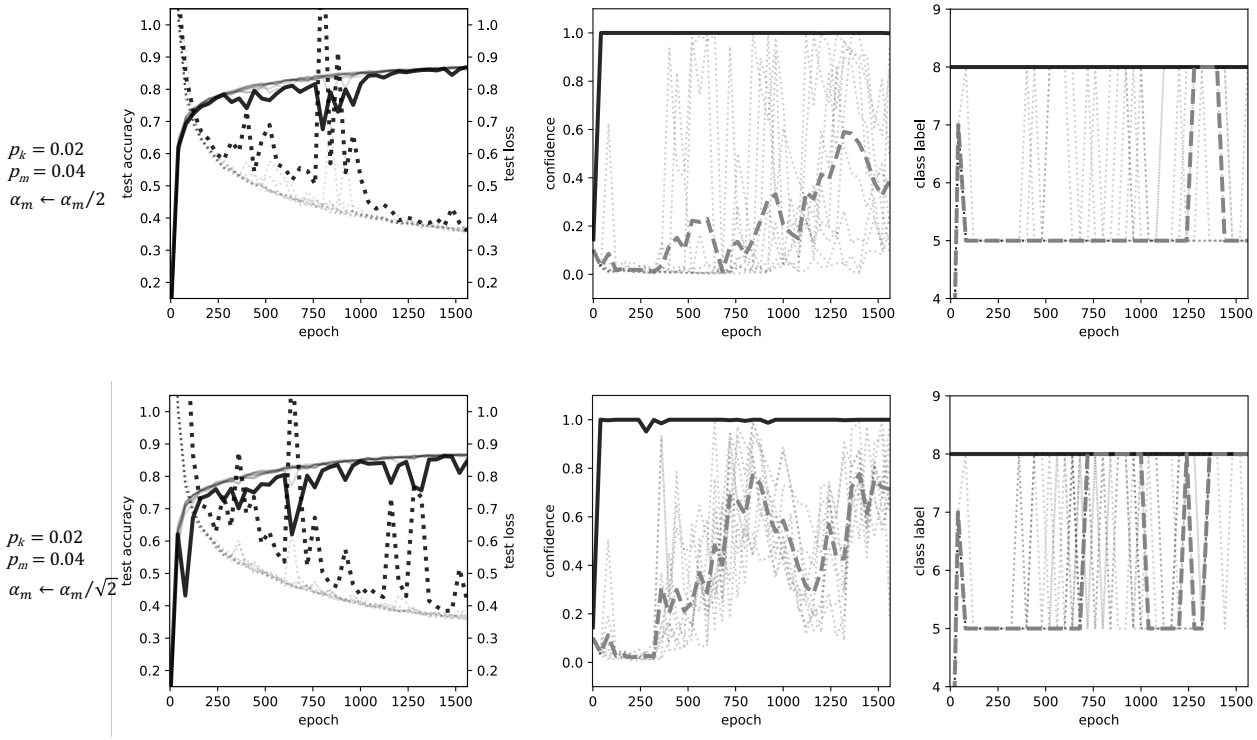

Figure 2. As figure 1, but with mismatched exchange probabilities. The benign peers use a probability of $p_k = 0.02$ while the malicious peer uses probability $p_m = 0.04$. *Top row:* results when the malicious peer updates its mixing factor α_m using the standard rule on transmission. *Bottom row:* results when the malicious peer updates its mixing factor using the rule $\alpha_m \leftarrow \alpha_m/\sqrt{2}$ to account for the increased number of transmissions it makes compared to the benign peers.

In addition to considering the change in accuracy caused by exchanging model parameters to attempt to spot malicious updates, peers can also compare the received parameter distributions from other peers to its own parameter distribution. As a malicious peer is optimizing a different objective from the benign peers, its weight distribution may be noticeably different, despite it using a stealthy procedure to update its weights. We visualize the model weight distributions in figure 3, both early and late during training. The histograms show the distributions as being almost indistinguishable on visual inspection. However, if we pick a training step where the malicious peer shows a decrease in test accuracy compared with the benign peers, as shown in figure 4, we can see a very distinctive difference in the weight distributions between the malicious and a benign peer. The malicious peer's weight distribution in this case has much smaller tails than the benign peer's, so is concentrated more densely around 0.

A possible method to detect a malicious peer would therefore be to monitor their parameter updates, and look for weight distributions that differ greatly from the local weight distribution. This can be tested using, for example, a two-sample Kolmogorov–Smirnoff (K–S) test. Applying this test to the distribution comparisons illustrated in figures 3 and 4 indicates that the sets of benign weights at all examined training steps are likely to be drawn from the same distribution (p-values of 0.85, 1.00 and 0.92 at step 120, 680 and 1200 respectively), while the malicious weights are not likely to be drawn from the same distribution as the benign weights ($p-$values of 0.12, 0.00 and 0.00 at step 120, 680 and 1200 respectively). The K–S test yields low $p-$values even for weight

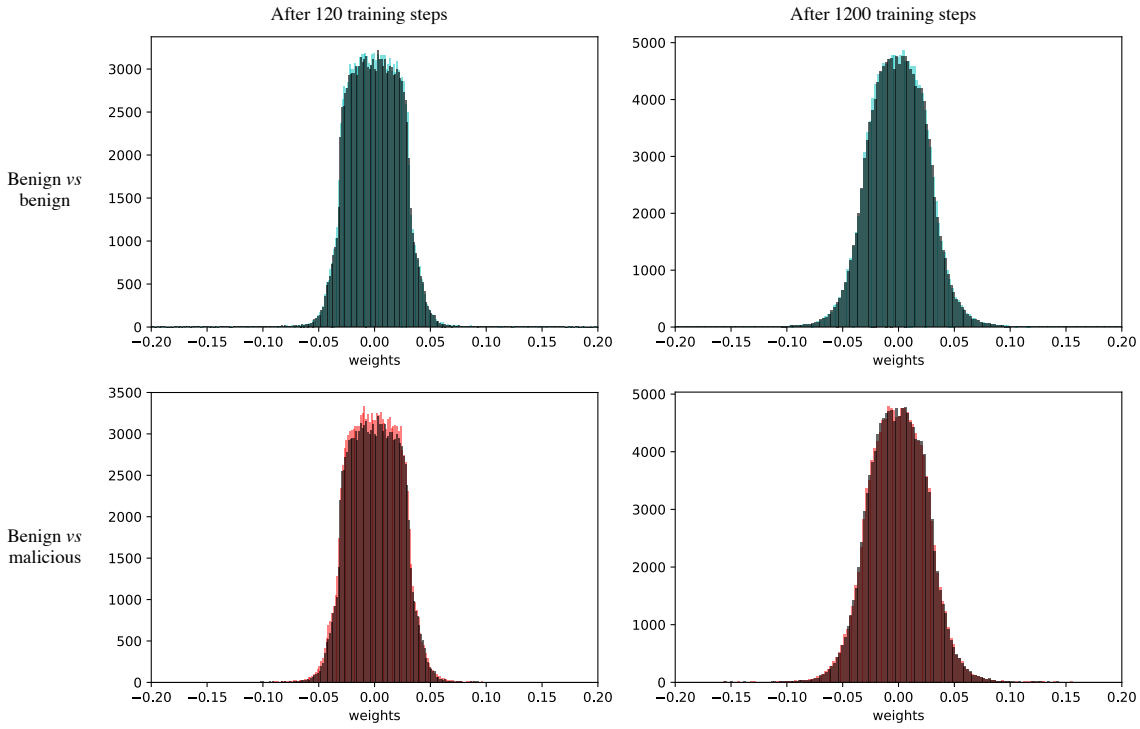

Figure 3. Weight distributions after 120 training steps (left column) and after 1200 training steps (right column) for two peers: in the top row, we compare weight distributions from two benign peers; in the bottom row, the weight distributions from a benign peer and the malicious peer (malicious distribution in red). Weights taken from the training run with $p_k = p_m = 0.02$ and the standard mixing parameter update rule.

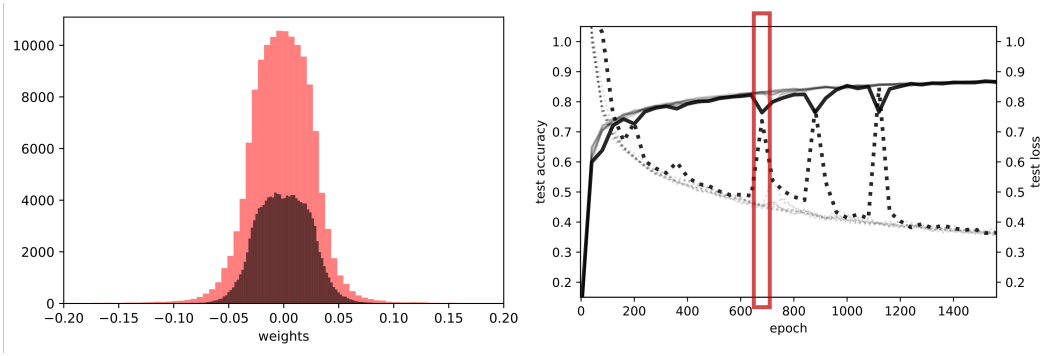

Figure 4. Weight distributions after 680 training steps (left column) for a benign peer and the malicious peer. Right column shows test accuracy/loss curve, highlighting the relevant training step and the large drop in test accuracy/increase in loss at the malicious node at this step. Weights taken from the training run with $p_k = p_m = 0.02$ and the standard mixing parameter update rule.

distributions that appeared visually similar, indicating that it may be a suitable test for spotting malicious updates. However, the $p-$values we report here are not corrected for multiple comparisons, and care needs to be taken when choosing a threshold for deciding whether a shared parameter set is malicious.

4. DISCUSSION

Our preliminary exploration of model poisoning attacks on peer-to-peer distributed DL has shown that it is possible to perform targeted model poisoning using a single malicious peer, and investigated several aspects of the learning setup that affect the poisoning effectiveness. However, at the low transmission probabilities likely to be used in real-world peer-to-peer learning scenarios, reliably poisoning the other peers takes many training steps — generally more than the number of epochs required to successfully poison the model in the federated learning case.[5] Our results suggest that increasing the transmission probability and modifying the mixing parameter update rules at the malicious peer could be a promising approach for improving its poisoning ability. However, the relationship between these factors is non-trivial and will require further research to tease-out. Finally, we propose that using a statistical test of the equality of probability distributions, such as the 2-sample K–S test, could be an effective method for spotting adversarial parameter updates. However, further work on improving the poisoning method, including adapting the malicious objective function, is needed to determine if malicious parameters can be learned that are statistically indistinguishable from benign parameters.

Our results are of concern to future military coalition operations in which edge nodes learn locally and make peer-to-peer parameter exchanges. We have demonstrated that, in principle, it is possible for adversaries to perform targeted model poisoning in such a scenario if they are able to insert a malicious peer, or gain control of a formerly benign peer. However, such attacks are unlikely to be practical in the near future, as more research is required on peer-to-peer learning systems in general before they are ready for real-world deployments in military operations. Further research should look deeper into the theory of learning using gossip-like schemes, as well as potentially develop other methods for peer-to-peer learning. In real deployments, the nodes are unlikely to be fully connected, and data transmission will be unreliable and noisy: what are the impacts of these factors on learning? How do they affect the resilience of the system to attacks? What happens when the training data is not IID between the peers, or when some peers only have data from a few target classes?

Practical next steps building on our preliminary results are to develop stronger theory around peer-to-peer model poisoning with gossiping peers, and to use that theory both to investigate better attacks and improve defenses. In particular, making the malicious peer more intelligent by allowing it to modify its parameter updating and sharing behaviour differently for each other peer could allow it to poison the other nodes more effectively, while maintaining stealth. Additionally, further experiments exploring different local training hyperparameters, datasets and model architectures are needed to fully explore these methods with more challenging learning problems.

ACKNOWLEDGMENTS

This research was sponsored by the U.S. Army Research Laboratory and the UK Ministry of Defence under Agreement Number W911NF–16–3–0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the UK Ministry of Defence or the UK Government. The U.S. and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copy-right notation hereon.

REFERENCES

[1] Endsley, M. R., "Toward a theory of situation awareness in dynamic systems," *Human factors* **37**(1), 32–64 (1995).
[2] Kott, A., Swami, A., and West, B. J., "The internet of battle things," *Computer* **49**(12), 70–75 (2016).
[3] Suri, N., Tortonesi, M., Michaelis, J., Budulas, P., Benincasa, G., Russell, S., Stefanelli, C., and Winkler, R., "Analyzing the applicability of internet of things to the battlefield environment," in [*Military Communications and Information Systems (ICMCIS), 2016 International Conference on*], 1–8, IEEE (2016).
[4] Chakraborty, S., Preece, A., Alzantot, M., Xing, T., Braines, D., and Srivastava, M., "Deep learning for situational understanding," in [*Information Fusion (FUSION), 2017 20th International Conference on*], IEEE (2017).
[5] Bhagoji, A. N., Chakraborty, S., Mittal, P., and Calo, S. B., "Analyzing federated learning through an adversarial lens," *arXiv:1811.12470* (2018).
[6] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., aurelio Ranzato, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., and Ng, A. Y., "Large scale distributed deep networks," in [*Advances in Neural Information Processing Systems 25*], 1223–1231 (2012).

[7] Chen, J., Monga, R., Bengio, S., and Józefowicz, R., "Revisiting distributed synchronous SGD," *arXiv:1604.00981* (2016).

[8] Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q. V., Hinton, G. E., and Dean, J., "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv:1701.06538* (2017).

[9] McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A., "Communication-Efficient Learning of Deep Networks from Decentralized Data," in [*Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*], *Proceedings of Machine Learning Research* **54**, 1273–1282, PMLR (20–22 Apr 2017).

[10] Wang, S., Tuor, T., Salonidis, T., Leung, K. K., Makaya, C., He, T., and Chan, K., "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in [*IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*], 63–71 (April 2018).

[11] Blot, M., Picard, D., Cord, M., and Thome, N., "Gossip training for deep learning," *arXiv:1611.09726* (2016).

[12] Jin, P. H., Yuan, Q., Iandola, F. N., and Keutzer, K., "How to scale distributed deep learning?," *arXiv:1611.04581* (2016).

[13] Pramod, S., "Elastic gossip: Distributing neural network training using gossip-like protocols," *arXiv:1812.02407* (2018).

[14] Blot, M., Picard, D., Thome, N., and Cord, M., "Distributed optimization for deep learning with gossip exchange," *Neurocomputing* **330**, 287 – 296 (2019).

[15] Biggio, B. and Roli, F., "Wild Patterns: Ten years after the rise of adversarial machine learning," *arXiv:1712.03141* (2017).

[16] Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B. I., and Tygar, J. D., "Adversarial machine learning," in [*Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*], *AISec '11*, 43–58 (2011).

[17] Dalvi, N., Domingos, P., Mausam, Sanghai, S., and Verma, D., "Adversarial classification," in [*Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*], *KDD '04*, 99–108 (2004).

[18] Lowd, D. and Meek, C., "Adversarial learning," in [*Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*], *KDD '05*, 641–647 (2005).

[19] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R., "Intriguing properties of neural networks," *arXiv:1312.6199* (2013).

[20] Goodfellow, I., Shlens, J., and Szegedy, C., "Explaining and harnessing adversarial examples," in [*International Conference on Learning Representations*], (2015).

[21] Biggio, B., Nelson, B., and Laskov, P., "Poisoning attacks against support vector machines," in [*Proceedings of the 29th International Coference on International Conference on Machine Learning*], *ICML'12*, 1467–1474 (2012).

[22] Koh, P. W. and Liang, P., "Understanding black-box predictions via influence functions," in [*Proceedings of the 34th International Conference on Machine Learning - Volume 70*], *ICML'17*, 1885–1894 (2017).

[23] Chen, X., Liu, C., Li, B., Lu, K., and Song, D., "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv:1712.05526* (2017).

[24] Chen, Y., Su, L., and Xu, J., "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *arXiv:1705.05491* (2017).

[25] Blanchard, P., El Mhamdi, E. M., Guerraoui, R., and Stainer, J., "Machine learning with adversaries: Byzantine tolerant gradient descent," in [*Advances in Neural Information Processing Systems 30*], 119–129 (2017).

[26] El Mhamdi, E. M., Guerraoui, R., and Rouault, S., "The hidden vulnerability of distributed learning in Byzantium," in [*Proceedings of the 35th International Conference on Machine Learning*], *Proceedings of Machine Learning Research* **80**, 3521–3530 (2018).

[27] Zhang, R. and Zhu, Q., "Secure and resilient distributed machine learning under adversarial environments," in [*2015 18th International Conference on Information Fusion (Fusion)*], 644–651 (July 2015).

[28] Zhang, R. and Zhu, Q., "A game-theoretic analysis of label flipping attacks on distributed support vector machines," in [*2017 51st Annual Conference on Information Sciences and Systems (CISS)*], 1–6 (March 2017).

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

[30] Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., and Shmatikov, V., "How to backdoor federated learning," *arXiv:1807.00459* (2018).

[31] Xiao, H., Rasul, K., and Vollgraf, R., "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *arXiv:1708.07747* (2017).

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

[33] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A., "Automatic differentiation in pytorch," in [*NIPS-W*], (2017).

[34] Tange, O., [*GNU Parallel 2018*], Ole Tange (2018).